

NVIDIA TensorRT 11.0.1 API Reference

Release 7.2.5 for NVIDIA DriveOS

NVIDIA Corporation

June 23, 2026

TensorRT API Reference

11.0.1

June 2026

Table 1 Revision history

Date	Summary of change
06-23-2026	Initial draft
06-23-2026	Review
06-23-2026	Approval review

Contents

1 Standard and Safe **1**

2 TensorRT **3**

3 Deprecated List **5**

4 Namespace Index **9**
4.1 Namespace List 9

5 Hierarchical Index **11**
5.1 Class Hierarchy 11

6 Class Index **17**
6.1 Class List 17

7 File Index **27**
7.1 File List 27

8 Namespace Documentation **29**
8.1 nvinfer1 Namespace Reference 29
8.1.1 Detailed Description 39
8.1.2 Typedef Documentation 39
8.1.2.1 AllocatorFlags 39
8.1.2.2 AsciiChar 40
8.1.2.3 BuilderFlags 40
8.1.2.4 char_t 40
8.1.2.5 Dims 40
8.1.2.6 IDebugListener 40
8.1.2.7 IErrorRecorder 40
8.1.2.8 IGpuAllocator 41
8.1.2.9 IGpuAsyncAllocator 41
8.1.2.10 ILogger 41
8.1.2.11 ILoggerFinder 41
8.1.2.12 InterfaceKind 41
8.1.2.13 IOutputAllocator 41
8.1.2.14 IPluginCapability 41
8.1.2.15 IPluginCreator 42
8.1.2.16 IPluginCreatorInterface 42
8.1.2.17 IPluginCreatorV3One 42
8.1.2.18 IPluginResource 42
8.1.2.19 IPluginV3 42
8.1.2.20 IPluginV3OneBuild 42
8.1.2.21 IPluginV3OneBuildV2 42

8.1.2.22 IPluginV3OneCore . . . 43
8.1.2.23 IPluginV3OneRuntime . . . 43
8.1.2.24 IProfiler . . . 43
8.1.2.25 IProgressMonitor . . . 43
8.1.2.26 IStreamReader . . . 43
8.1.2.27 IStreamReaderV2 . . . 43
8.1.2.28 IStreamWriter . . . 43
8.1.2.29 NetworkDefinitionCreationFlags . . . 44
8.1.2.30 PluginFormat . . . 44
8.1.2.31 SerializationFlags . . . 44
8.1.2.32 TacticSources . . . 44
8.1.2.33 TempfileControlFlags . . . 45
8.1.2.34 TensorFormats . . . 45
8.1.3 Enumeration Type Documentation . . . 45
8.1.3.1 ActivationType . . . 45
8.1.3.2 AllocatorFlag . . . 46
8.1.3.3 APILanguage . . . 46
8.1.3.4 AttentionIOForm . . . 46
8.1.3.5 AttentionNormalizationOp . . . 46
8.1.3.6 BoundingBoxFormat . . . 47
8.1.3.7 BuilderFlag . . . 47
8.1.3.8 CausalMaskKind . . . 50
8.1.3.9 CollectiveOperation . . . 51
8.1.3.10 CumulativeOperation . . . 51
8.1.3.11 DataType . . . 52
8.1.3.12 DeviceType . . . 52
8.1.3.13 DimensionOperation . . . 53
8.1.3.14 ElementWiseOperation . . . 53
8.1.3.15 EngineCapability . . . 54
8.1.3.16 EngineStat . . . 55
8.1.3.17 ErrorCode . . . 55
8.1.3.18 ExecutionContextAllocationStrategy . . . 56
8.1.3.19 FillOperation . . . 57
8.1.3.20 GatherMode . . . 57
8.1.3.21 HardwareCompatibilityLevel . . . 58
8.1.3.22 InterpolationMode . . . 59
8.1.3.23 KVCacheMode . . . 59
8.1.3.24 LayerInformationFormat . . . 59
8.1.3.25 LayerType . . . 60
8.1.3.26 LoopOutput . . . 61
8.1.3.27 MatrixOperation . . . 62
8.1.3.28 MemoryPoolType . . . 62
8.1.3.29 MoEActType . . . 63
8.1.3.30 NetworkDefinitionCreationFlag . . . 63
8.1.3.31 OptProfileSelector . . . 64
8.1.3.32 PaddingMode . . . 64
8.1.3.33 PluginCapabilityType . . . 67
8.1.3.34 PluginCreatorVersion . . . 67
8.1.3.35 PluginFieldType . . . 67
8.1.3.36 PluginVersion . . . 68
8.1.3.37 PoolingType . . . 68
8.1.3.38 PreviewFeature . . . 69
8.1.3.39 ProfilingVerbosity . . . 69
8.1.3.40 ReduceOperation . . . 69

8.1.3.41 ResizeCoordinateTransformation . . . 70
8.1.3.42 ResizeRoundMode . . . 71
8.1.3.43 ResizeSelector . . . 72
8.1.3.44 RuntimePlatform . . . 72
8.1.3.45 SampleMode . . . 73
8.1.3.46 ScaleMode . . . 73
8.1.3.47 ScatterMode . . . 74
8.1.3.48 SeekPosition . . . 74
8.1.3.49 SerializationFlag . . . 74
8.1.3.50 TacticSource . . . 75
8.1.3.51 TempfileControlFlag . . . 75
8.1.3.52 TensorFormat . . . 76
8.1.3.53 TensorIOMode . . . 77
8.1.3.54 TensorLocation . . . 78
8.1.3.55 TensorRTPhase . . . 78
8.1.3.56 TilingOptimizationLevel . . . 78
8.1.3.57 TopKOperation . . . 79
8.1.3.58 TripLimit . . . 79
8.1.3.59 UnaryOperation . . . 79
8.1.3.60 WeightsRole . . . 80
8.1.4 Function Documentation . . . 81
8.1.4.1 EnumMax() . . . 81
8.1.4.2 getBuilderPluginRegistry() . . . 81
8.1.4.3 getBuilderSafePluginRegistry() . . . 81
8.1.4.4 setInternalLibraryPath() . . . 81
8.2 nvinfer1::anonymous_namespace{NvInfer.h} Namespace Reference . . . 82
8.2.1 Function Documentation . . . 82
8.2.1.1 createInferBuilder() . . . 82
8.3 nvinfer1::anonymous_namespace{NvInferRuntime.h} Namespace Reference . . . 82
8.3.1 Function Documentation . . . 83
8.3.1.1 createInferRefitter() . . . 83
8.3.1.2 createInferRuntime() . . . 83
8.4 nvinfer1::impl Namespace Reference . . . 83
8.5 nvinfer1::plugin Namespace Reference . . . 85
8.6 nvinfer1::safe Namespace Reference . . . 85
8.7 nvinfer1::v_1_0 Namespace Reference . . . 85
8.7.1 Detailed Description . . . 86
8.8 nvinfer1::v_2_0 Namespace Reference . . . 86
8.9 nvinfer2 Namespace Reference . . . 87
8.10 nvinfer2::safe Namespace Reference . . . 87
8.10.1 Typedef Documentation . . . 89
8.10.1.1 AsciiChar . . . 89
8.10.1.2 DataType . . . 89
8.10.1.3 Dims . . . 89
8.10.1.4 ErrorCode . . . 90
8.10.1.5 ErrorDesc . . . 90
8.10.1.6 half_t . . . 90
8.10.1.7 IErrorRecorder . . . 90
8.10.1.8 InterfaceInfo . . . 90
8.10.1.9 IPluginCapability . . . 90
8.10.1.10 IPluginCreatorInterface . . . 90
8.10.1.11 IPluginResource . . . 91
8.10.1.12 IPluginV3 . . . 91
8.10.1.13 IVersionedInterface . . . 91

8.10.1.14 PluginField . . . 91
8.10.1.15 PluginFieldCollection . . . 91
8.10.1.16 RefCount . . . 91
8.10.1.17 TensorIOMode . . . 91
8.10.1.18 TensorRTPhase . . . 92
8.10.2 Enumeration Type Documentation . . . 92
8.10.2.1 MemoryPlacement . . . 92
8.10.2.2 MemoryUsage . . . 92
8.10.2.3 RuntimeErrorType . . . 92
8.10.2.4 Severity . . . 93
8.10.3 Function Documentation . . . 93
8.10.3.1 createTRTGraph() . . . 94
8.10.3.2 destroyTRTGraph() . . . 95
8.10.3.3 EnumMax() . . . 96
8.10.3.4 getSafePluginRegistry() . . . 96
8.10.3.5 operator"!=() . . . 96
8.10.3.6 operator==() . . . 97
8.10.4 Variable Documentation . . . 97
8.10.4.1 kMAXIMUM_ALLOC_SIZE . . . 97
8.11 nvinfer2::safe::consistency Namespace Reference . . . 97
8.11.1 Function Documentation . . . 97
8.11.1.1 createConsistencyChecker() . . . 98
8.12 nvinfer2::safe::impl Namespace Reference . . . 98
8.13 nvonnxparser Namespace Reference . . . 98
8.13.1 Detailed Description . . . 99
8.13.2 Typedef Documentation . . . 99
8.13.2.1 OnnxParserFlags . . . 99
8.13.3 Enumeration Type Documentation . . . 100
8.13.3.1 ErrorCode . . . 100
8.13.3.2 OnnxParserFlag . . . 100
8.13.4 Function Documentation . . . 101
8.13.4.1 createONNXConfig() . . . 101
8.13.4.2 EnumMax() . . . 101
8.13.4.3 EnumMax< ErrorCode >() . . . 102
8.13.4.4 EnumMax< OnnxParserFlag >() . . . 102
8.14 nvonnxparser::anonymous_namespace{NvOnnxParser.h} Namespace Reference . . . 102
8.14.1 Function Documentation . . . 102
8.14.1.1 createParser() . . . 102
8.14.1.2 createParserRefitter() . . . 103

9 Class Documentation **105**
9.1 Dims Class Reference . . . 105
9.1.1 Detailed Description . . . 105
9.2 nvinfer1::Dims2 Class Reference . . . 105
9.2.1 Detailed Description . . . 106
9.2.2 Constructor & Destructor Documentation . . . 106
9.2.2.1 Dims2() [1/2] . . . 106
9.2.2.2 Dims2() [2/2] . . . 106
9.3 nvinfer1::Dims3 Class Reference . . . 107
9.3.1 Detailed Description . . . 107
9.3.2 Constructor & Destructor Documentation . . . 107
9.3.2.1 Dims3() [1/2] . . . 107
9.3.2.2 Dims3() [2/2] . . . 107
9.4 nvinfer1::Dims4 Class Reference . . . 108

9.4.1 Detailed Description . . . 108
9.4.2 Constructor & Destructor Documentation . . . 109
9.4.2.1 Dims4() [1/2] . . . 109
9.4.2.2 Dims4() [2/2] . . . 109
9.5 nvinfer1::Dims64 Class Reference . . . 109
9.5.1 Member Data Documentation . . . 110
9.5.1.1 d . . . 110
9.5.1.2 MAX_DIMS . . . 110
9.5.1.3 nbDims . . . 111
9.6 nvinfer1::DimsExprs Class Reference . . . 111
9.6.1 Detailed Description . . . 111
9.6.2 Member Data Documentation . . . 111
9.6.2.1 d . . . 111
9.6.2.2 nbDims . . . 112
9.7 nvinfer1::DimsHW Class Reference . . . 112
9.7.1 Detailed Description . . . 112
9.7.2 Constructor & Destructor Documentation . . . 113
9.7.2.1 DimsHW() [1/2] . . . 113
9.7.2.2 DimsHW() [2/2] . . . 113
9.7.3 Member Function Documentation . . . 113
9.7.3.1 h() [1/2] . . . 113
9.7.3.2 h() [2/2] . . . 114
9.7.3.3 w() [1/2] . . . 114
9.7.3.4 w() [2/2] . . . 114
9.8 nvinfer1::DynamicPluginTensorDesc Struct Reference . . . 114
9.8.1 Detailed Description . . . 115
9.8.2 Member Data Documentation . . . 115
9.8.2.1 desc . . . 115
9.8.2.2 max . . . 115
9.8.2.3 min . . . 115
9.8.2.4 opt . . . 116
9.9 nvinfer1::impl::EnumMaxImpl< T > Struct Template Reference . . . 116
9.9.1 Detailed Description . . . 116
9.10 nvinfer2::safe::impl::EnumMaxImpl< T > Struct Template Reference . . . 116
9.10.1 Detailed Description . . . 116
9.11 nvinfer1::impl::EnumMaxImpl< ActivationType > Struct Reference . . . 117
9.11.1 Detailed Description . . . 117
9.11.2 Member Data Documentation . . . 117
9.11.2.1 kVALUE . . . 117
9.12 nvinfer1::impl::EnumMaxImpl< AllocatorFlag > Struct Reference . . . 117
9.12.1 Detailed Description . . . 118
9.12.2 Member Data Documentation . . . 118
9.12.2.1 kVALUE . . . 118
9.13 nvinfer1::impl::EnumMaxImpl< APILanguage > Struct Reference . . . 118
9.13.1 Detailed Description . . . 118
9.13.2 Member Data Documentation . . . 119
9.13.2.1 kVALUE . . . 119
9.14 nvinfer1::impl::EnumMaxImpl< AttentionIOForm > Struct Reference . . . 119
9.14.1 Detailed Description . . . 119
9.14.2 Member Data Documentation . . . 119
9.14.2.1 kVALUE . . . 119
9.15 nvinfer1::impl::EnumMaxImpl< AttentionNormalizationOp > Struct Reference . . . 120
9.15.1 Detailed Description . . . 120
9.15.2 Member Data Documentation . . . 120

9.15.2.1 kVALUE . . . 120
9.16 nvinfer1::impl::EnumMaxImpl< BoundingBoxFormat > Struct Reference . . . 120
9.16.1 Detailed Description . . . 121
9.16.2 Member Data Documentation . . . 121
9.16.2.1 kVALUE . . . 121
9.17 nvinfer1::impl::EnumMaxImpl< BuilderFlag > Struct Reference . . . 121
9.17.1 Detailed Description . . . 121
9.17.2 Member Data Documentation . . . 121
9.17.2.1 kVALUE . . . 122
9.18 nvinfer1::impl::EnumMaxImpl< CausalMaskKind > Struct Reference . . . 122
9.18.1 Detailed Description . . . 122
9.18.2 Member Data Documentation . . . 122
9.18.2.1 kVALUE . . . 122
9.19 nvinfer1::impl::EnumMaxImpl< CollectiveOperation > Struct Reference . . . 122
9.19.1 Detailed Description . . . 123
9.19.2 Member Data Documentation . . . 123
9.19.2.1 kVALUE . . . 123
9.20 nvinfer1::impl::EnumMaxImpl< CumulativeOperation > Struct Reference . . . 123
9.20.1 Detailed Description . . . 123
9.20.2 Member Data Documentation . . . 124
9.20.2.1 kVALUE . . . 124
9.21 nvinfer1::impl::EnumMaxImpl< DataType > Struct Reference . . . 124
9.21.1 Detailed Description . . . 124
9.21.2 Member Data Documentation . . . 124
9.21.2.1 kVALUE . . . 124
9.22 nvinfer1::impl::EnumMaxImpl< DeviceType > Struct Reference . . . 125
9.22.1 Detailed Description . . . 125
9.22.2 Member Data Documentation . . . 125
9.22.2.1 kVALUE . . . 125
9.23 nvinfer1::impl::EnumMaxImpl< DimensionOperation > Struct Reference . . . 125
9.23.1 Detailed Description . . . 126
9.23.2 Member Data Documentation . . . 126
9.23.2.1 kVALUE . . . 126
9.24 nvinfer1::impl::EnumMaxImpl< ElementWiseOperation > Struct Reference . . . 126
9.24.1 Detailed Description . . . 126
9.24.2 Member Data Documentation . . . 126
9.24.2.1 kVALUE . . . 127
9.25 nvinfer1::impl::EnumMaxImpl< EngineCapability > Struct Reference . . . 127
9.25.1 Detailed Description . . . 127
9.25.2 Member Data Documentation . . . 127
9.25.2.1 kVALUE . . . 127
9.26 nvinfer1::impl::EnumMaxImpl< EngineStat > Struct Reference . . . 128
9.26.1 Detailed Description . . . 128
9.26.2 Member Data Documentation . . . 128
9.26.2.1 kVALUE . . . 128
9.27 nvinfer1::impl::EnumMaxImpl< ErrorCode > Struct Reference . . . 128
9.27.1 Detailed Description . . . 129
9.27.2 Member Data Documentation . . . 129
9.27.2.1 kVALUE . . . 129
9.28 nvinfer1::impl::EnumMaxImpl< ExecutionContextAllocationStrategy > Struct Reference . . . 129
9.28.1 Detailed Description . . . 129
9.28.2 Member Data Documentation . . . 129
9.28.2.1 kVALUE . . . 130
9.29 nvinfer1::impl::EnumMaxImpl< FillOperation > Struct Reference . . . 130

9.29.1 Detailed Description . . . 130
9.29.2 Member Data Documentation . . . 130
9.29.2.1 kVALUE . . . 130
9.30 nvinfer1::impl::EnumMaxImpl< GatherMode > Struct Reference . . . 130
9.30.1 Detailed Description . . . 131
9.30.2 Member Data Documentation . . . 131
9.30.2.1 kVALUE . . . 131
9.31 nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel > Struct Reference . . . 131
9.31.1 Detailed Description . . . 131
9.31.2 Member Data Documentation . . . 132
9.31.2.1 kVALUE . . . 132
9.32 nvinfer1::impl::EnumMaxImpl< ILogger::Severity > Struct Reference . . . 132
9.32.1 Detailed Description . . . 132
9.32.2 Member Data Documentation . . . 132
9.32.2.1 kVALUE . . . 133
9.33 nvinfer1::impl::EnumMaxImpl< InterpolationMode > Struct Reference . . . 133
9.33.1 Detailed Description . . . 133
9.33.2 Member Data Documentation . . . 133
9.33.2.1 kVALUE . . . 133
9.34 nvinfer1::impl::EnumMaxImpl< KVCacheMode > Struct Reference . . . 134
9.34.1 Detailed Description . . . 134
9.34.2 Member Data Documentation . . . 134
9.34.2.1 kVALUE . . . 134
9.35 nvinfer1::impl::EnumMaxImpl< LayerInformationFormat > Struct Reference . . . 134
9.35.1 Detailed Description . . . 135
9.35.2 Member Data Documentation . . . 135
9.35.2.1 kVALUE . . . 135
9.36 nvinfer1::impl::EnumMaxImpl< LayerType > Struct Reference . . . 135
9.36.1 Detailed Description . . . 135
9.36.2 Member Data Documentation . . . 135
9.36.2.1 kVALUE . . . 136
9.37 nvinfer1::impl::EnumMaxImpl< LoopOutput > Struct Reference . . . 136
9.37.1 Detailed Description . . . 136
9.37.2 Member Data Documentation . . . 136
9.37.2.1 kVALUE . . . 136
9.38 nvinfer1::impl::EnumMaxImpl< MatrixOperation > Struct Reference . . . 136
9.38.1 Detailed Description . . . 137
9.38.2 Member Data Documentation . . . 137
9.38.2.1 kVALUE . . . 137
9.39 nvinfer1::impl::EnumMaxImpl< MemoryPoolType > Struct Reference . . . 137
9.39.1 Detailed Description . . . 137
9.39.2 Member Data Documentation . . . 138
9.39.2.1 kVALUE . . . 138
9.40 nvinfer1::impl::EnumMaxImpl< MoEActType > Struct Reference . . . 138
9.40.1 Detailed Description . . . 138
9.40.2 Member Data Documentation . . . 138
9.40.2.1 kVALUE . . . 138
9.41 nvinfer1::impl::EnumMaxImpl< NetworkDefinitionCreationFlag > Struct Reference . . . 139
9.41.1 Detailed Description . . . 139
9.41.2 Member Data Documentation . . . 139
9.41.2.1 kVALUE . . . 139
9.42 nvinfer1::impl::EnumMaxImpl< OptProfileSelector > Struct Reference . . . 139
9.42.1 Detailed Description . . . 140
9.42.2 Member Data Documentation . . . 140

9.42.2.1 kVALUE . . . 140
9.43 nvinfer1::impl::EnumMaxImpl< PaddingMode > Struct Reference . . . 140
9.43.1 Detailed Description . . . 140
9.43.2 Member Data Documentation . . . 140
9.43.2.1 kVALUE . . . 141
9.44 nvinfer1::impl::EnumMaxImpl< PoolingType > Struct Reference . . . 141
9.44.1 Detailed Description . . . 141
9.44.2 Member Data Documentation . . . 141
9.44.2.1 kVALUE . . . 141
9.45 nvinfer1::impl::EnumMaxImpl< PreviewFeature > Struct Reference . . . 141
9.45.1 Detailed Description . . . 142
9.45.2 Member Data Documentation . . . 142
9.45.2.1 kVALUE . . . 142
9.46 nvinfer1::impl::EnumMaxImpl< ProfilingVerbosity > Struct Reference . . . 142
9.46.1 Detailed Description . . . 142
9.46.2 Member Data Documentation . . . 143
9.46.2.1 kVALUE . . . 143
9.47 nvinfer1::impl::EnumMaxImpl< ReduceOperation > Struct Reference . . . 143
9.47.1 Detailed Description . . . 143
9.47.2 Member Data Documentation . . . 143
9.47.2.1 kVALUE . . . 143
9.48 nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation > Struct Reference . . . 144
9.48.1 Detailed Description . . . 144
9.48.2 Member Data Documentation . . . 144
9.48.2.1 kVALUE . . . 144
9.49 nvinfer1::impl::EnumMaxImpl< ResizeRoundMode > Struct Reference . . . 144
9.49.1 Detailed Description . . . 145
9.49.2 Member Data Documentation . . . 145
9.49.2.1 kVALUE . . . 145
9.50 nvinfer1::impl::EnumMaxImpl< ResizeSelector > Struct Reference . . . 145
9.50.1 Detailed Description . . . 145
9.50.2 Member Data Documentation . . . 145
9.50.2.1 kVALUE . . . 146
9.51 nvinfer1::impl::EnumMaxImpl< RuntimePlatform > Struct Reference . . . 146
9.51.1 Detailed Description . . . 146
9.51.2 Member Data Documentation . . . 146
9.51.2.1 kVALUE . . . 146
9.52 nvinfer1::impl::EnumMaxImpl< SampleMode > Struct Reference . . . 146
9.52.1 Detailed Description . . . 147
9.52.2 Member Data Documentation . . . 147
9.52.2.1 kVALUE . . . 147
9.53 nvinfer1::impl::EnumMaxImpl< ScaleMode > Struct Reference . . . 147
9.53.1 Detailed Description . . . 147
9.53.2 Member Data Documentation . . . 148
9.53.2.1 kVALUE . . . 148
9.54 nvinfer1::impl::EnumMaxImpl< ScatterMode > Struct Reference . . . 148
9.54.1 Detailed Description . . . 148
9.54.2 Member Data Documentation . . . 148
9.54.2.1 kVALUE . . . 148
9.55 nvinfer1::impl::EnumMaxImpl< SerializationFlag > Struct Reference . . . 149
9.55.1 Detailed Description . . . 149
9.55.2 Member Data Documentation . . . 149
9.55.2.1 kVALUE . . . 149
9.56 nvinfer2::safe::impl::EnumMaxImpl< Severity > Struct Reference . . . 149

9.56.1 Detailed Description . . . 150
9.56.2 Member Data Documentation . . . 150
9.56.2.1 kVALUE . . . 150
9.57 nvinfer1::impl::EnumMaxImpl< TacticSource > Struct Reference . . . 150
9.57.1 Detailed Description . . . 150
9.57.2 Member Data Documentation . . . 150
9.57.2.1 kVALUE . . . 151
9.58 nvinfer1::impl::EnumMaxImpl< TempfileControlFlag > Struct Reference . . . 151
9.58.1 Detailed Description . . . 151
9.58.2 Member Data Documentation . . . 151
9.58.2.1 kVALUE . . . 151
9.59 nvinfer1::impl::EnumMaxImpl< TensorFormat > Struct Reference . . . 152
9.59.1 Detailed Description . . . 152
9.59.2 Member Data Documentation . . . 152
9.59.2.1 kVALUE . . . 152
9.60 nvinfer1::impl::EnumMaxImpl< TensorIOMode > Struct Reference . . . 152
9.60.1 Detailed Description . . . 153
9.60.2 Member Data Documentation . . . 153
9.60.2.1 kVALUE . . . 153
9.61 nvinfer1::impl::EnumMaxImpl< TensorLocation > Struct Reference . . . 153
9.61.1 Detailed Description . . . 153
9.61.2 Member Data Documentation . . . 153
9.61.2.1 kVALUE . . . 154
9.62 nvinfer1::impl::EnumMaxImpl< TilingOptimizationLevel > Struct Reference . . . 154
9.62.1 Detailed Description . . . 154
9.62.2 Member Data Documentation . . . 154
9.62.2.1 kVALUE . . . 154
9.63 nvinfer1::impl::EnumMaxImpl< TopKOperation > Struct Reference . . . 154
9.63.1 Detailed Description . . . 155
9.63.2 Member Data Documentation . . . 155
9.63.2.1 kVALUE . . . 155
9.64 nvinfer1::impl::EnumMaxImpl< TripLimit > Struct Reference . . . 155
9.64.1 Detailed Description . . . 155
9.64.2 Member Data Documentation . . . 156
9.64.2.1 kVALUE . . . 156
9.65 nvinfer1::impl::EnumMaxImpl< UnaryOperation > Struct Reference . . . 156
9.65.1 Detailed Description . . . 156
9.65.2 Member Data Documentation . . . 156
9.65.2.1 kVALUE . . . 156
9.66 nvinfer1::impl::EnumMaxImpl< WeightsRole > Struct Reference . . . 157
9.66.1 Detailed Description . . . 157
9.66.2 Member Data Documentation . . . 157
9.66.2.1 kVALUE . . . 157
9.67 nvinfer1::plugin::GridAnchorParameters Struct Reference . . . 157
9.67.1 Detailed Description . . . 158
9.67.2 Member Data Documentation . . . 158
9.67.2.1 aspectRatios . . . 158
9.67.2.2 H . . . 158
9.67.2.3 maxSize . . . 159
9.67.2.4 minSize . . . 159
9.67.2.5 numAspectRatios . . . 159
9.67.2.6 variance . . . 159
9.67.2.7 W . . . 159
9.68 nvinfer1::IActivationLayer Class Reference . . . 160

9.68.1 Detailed Description . . . 160
9.68.2 Constructor & Destructor Documentation . . . 161
9.68.2.1 ~IActivationLayer() . . . 161
9.68.3 Member Function Documentation . . . 161
9.68.3.1 getActivationType() . . . 161
9.68.3.2 getAlpha() . . . 161
9.68.3.3 getBeta() . . . 162
9.68.3.4 setActivationType() . . . 162
9.68.3.5 setAlpha() . . . 162
9.68.3.6 setBeta() . . . 163
9.68.4 Member Data Documentation . . . 163
9.68.4.1 mImpl . . . 163
9.69 nvinfer1::IAssertionLayer Class Reference . . . 163
9.69.1 Detailed Description . . . 164
9.69.2 Constructor & Destructor Documentation . . . 164
9.69.2.1 ~IAssertionLayer() . . . 164
9.69.3 Member Function Documentation . . . 164
9.69.3.1 getMessage() . . . 165
9.69.3.2 setMessage() . . . 165
9.69.4 Member Data Documentation . . . 165
9.69.4.1 mImpl . . . 165
9.70 nvinfer1::IAttention Class Reference . . . 165
9.70.1 Detailed Description . . . 167
9.70.2 Constructor & Destructor Documentation . . . 168
9.70.2.1 ~IAttention() . . . 169
9.70.3 Member Function Documentation . . . 169
9.70.3.1 getCausal() . . . 169
9.70.3.2 getCausalKind() . . . 169
9.70.3.3 getDecomposable() . . . 170
9.70.3.4 getInput() . . . 170
9.70.3.5 getKeyValueForm() . . . 170
9.70.3.6 getKeyValueLengths() . . . 171
9.70.3.7 getMask() . . . 171
9.70.3.8 getMetadata() . . . 171
9.70.3.9 getName() . . . 172
9.70.3.10 getNbInputs() . . . 172
9.70.3.11 getNbOutputs() . . . 172
9.70.3.12 getNbRanks() . . . 172
9.70.3.13 getNormalizationOperation() . . . 173
9.70.3.14 getNormalizationQuantizeScale() . . . 173
9.70.3.15 getNormalizationQuantizeToType() . . . 173
9.70.3.16 getOutput() . . . 173
9.70.3.17 getQueryForm() . . . 174
9.70.3.18 getQueryLengths() . . . 174
9.70.3.19 setCausal() . . . 174
9.70.3.20 setCausalKind() . . . 175
9.70.3.21 setDecomposable() . . . 176
9.70.3.22 setInput() . . . 176
9.70.3.23 setKeyValueForm() . . . 176
9.70.3.24 setKeyValueLengths() . . . 177
9.70.3.25 setMask() . . . 178
9.70.3.26 setMetadata() . . . 178
9.70.3.27 setName() . . . 179
9.70.3.28 setNbRanks() . . . 179

9.70.3.29 setNormalizationOperation() . . . 180
9.70.3.30 setNormalizationQuantizeScale() . . . 180
9.70.3.31 setNormalizationQuantizeToType() . . . 181
9.70.3.32 setQueryForm() . . . 181
9.70.3.33 setQueryLengths() . . . 182
9.70.4 Member Data Documentation . . . 182
9.70.4.1 mImpl . . . 182
9.71 nvinfer1::IAttentionBoundaryLayer Class Reference . . . 183
9.71.1 Detailed Description . . . 183
9.71.2 Constructor & Destructor Documentation . . . 183
9.71.2.1 ~IAttentionBoundaryLayer() . . . 184
9.71.3 Member Function Documentation . . . 184
9.71.3.1 getAttention() . . . 184
9.71.4 Member Data Documentation . . . 184
9.71.4.1 mBoundary . . . 184
9.72 nvinfer1::IAttentionInputLayer Class Reference . . . 184
9.72.1 Detailed Description . . . 185
9.72.2 Constructor & Destructor Documentation . . . 185
9.72.2.1 ~IAttentionInputLayer() . . . 185
9.72.3 Member Function Documentation . . . 185
9.72.3.1 setInput() . . . 185
9.72.4 Member Data Documentation . . . 186
9.72.4.1 mImpl . . . 186
9.73 nvinfer1::IAttentionOutputLayer Class Reference . . . 186
9.73.1 Detailed Description . . . 187
9.73.2 Constructor & Destructor Documentation . . . 187
9.73.2.1 ~IAttentionOutputLayer() . . . 187
9.73.3 Member Data Documentation . . . 187
9.73.3.1 mImpl . . . 187
9.74 nvinfer1::IBuilder Class Reference . . . 188
9.74.1 Detailed Description . . . 189
9.74.2 Constructor & Destructor Documentation . . . 189
9.74.2.1 ~IBuilder() . . . 189
9.74.3 Member Function Documentation . . . 189
9.74.3.1 buildEngineWithConfig() . . . 189
9.74.3.2 buildSerializedNetwork() [1/2] . . . 190
9.74.3.3 buildSerializedNetwork() [2/2] . . . 191
9.74.3.4 buildSerializedNetworkToStream() . . . 191
9.74.3.5 createBuilderConfig() . . . 192
9.74.3.6 createNetworkV2() . . . 192
9.74.3.7 createOptimizationProfile() . . . 193
9.74.3.8 getErrorRecorder() . . . 193
9.74.3.9 getLogger() . . . 194
9.74.3.10 getMaxDLABatchSize() . . . 194
9.74.3.11 getMaxThreads() . . . 194
9.74.3.12 getNbDLACores() . . . 195
9.74.3.13 getPluginRegistry() . . . 195
9.74.3.14 isNetworkSupported() . . . 195
9.74.3.15 reset() . . . 196
9.74.3.16 setErrorRecorder() . . . 196
9.74.3.17 setGpuAllocator() . . . 196
9.74.3.18 setMaxThreads() . . . 197
9.74.4 Member Data Documentation . . . 197
9.74.4.1 mImpl . . . 197

9.75 nvinfer1::IBuilderConfig Class Reference . . . 198
9.75.1 Detailed Description . . . 200
9.75.2 Constructor & Destructor Documentation . . . 201
9.75.2.1 ~IBuilderConfig() . . . 201
9.75.3 Member Function Documentation . . . 201
9.75.3.1 addOptimizationProfile() . . . 201
9.75.3.2 canRunOnDLA() . . . 201
9.75.3.3 clearFlag() . . . 202
9.75.3.4 createTimingCache() . . . 202
9.75.3.5 getAvgTimingIterations() . . . 202
9.75.3.6 getBuilderOptimizationLevel() . . . 203
9.75.3.7 getDefaultDeviceType() . . . 203
9.75.3.8 getDeviceType() . . . 203
9.75.3.9 getDLACore() . . . 204
9.75.3.10 getEngineCapability() . . . 204
9.75.3.11 getFlag() . . . 204
9.75.3.12 getFlags() . . . 205
9.75.3.13 getHardwareCompatibilityLevel() . . . 205
9.75.3.14 getL2LimitForTiling() . . . 205
9.75.3.15 getMaxAuxStreams() . . . 206
9.75.3.16 getMaxNbTactics() . . . 206
9.75.3.17 getMemoryPoolLimit() . . . 206
9.75.3.18 getNbOptimizationProfiles() . . . 207
9.75.3.19 getNbPluginsToSerialize() . . . 207
9.75.3.20 getPluginToSerialize() . . . 207
9.75.3.21 getPreviewFeature() . . . 208
9.75.3.22 getProfileStream() . . . 208
9.75.3.23 getProfilingVerbosity() . . . 209
9.75.3.24 getProgressMonitor() . . . 209
9.75.3.25 getRemoteAutoTuningConfig() . . . 209
9.75.3.26 getRuntimePlatform() . . . 210
9.75.3.27 getTacticSources() . . . 210
9.75.3.28 getTilingOptimizationLevel() . . . 210
9.75.3.29 getTimingCache() . . . 211
9.75.3.30 isDeviceTypeSet() . . . 211
9.75.3.31 reset() . . . 211
9.75.3.32 resetDeviceType() . . . 211
9.75.3.33 setAvgTimingIterations() . . . 212
9.75.3.34 setBuilderOptimizationLevel() . . . 212
9.75.3.35 setDefaultDeviceType() . . . 213
9.75.3.36 setDeviceType() . . . 213
9.75.3.37 setDLACore() . . . 214
9.75.3.38 setEngineCapability() . . . 214
9.75.3.39 setFlag() . . . 214
9.75.3.40 setFlags() . . . 215
9.75.3.41 setHardwareCompatibilityLevel() . . . 216
9.75.3.42 setL2LimitForTiling() . . . 216
9.75.3.43 setMaxAuxStreams() . . . 217
9.75.3.44 setMaxNbTactics() . . . 218
9.75.3.45 setMemoryPoolLimit() . . . 218
9.75.3.46 setPluginsToSerialize() . . . 219
9.75.3.47 setPreviewFeature() . . . 219
9.75.3.48 setProfileStream() . . . 219
9.75.3.49 setProfilingVerbosity() . . . 220

9.75.3.50 setProgressMonitor() . . . 220
9.75.3.51 setRemoteAutoTuningConfig() . . . 221
9.75.3.52 setRuntimePlatform() . . . 221
9.75.3.53 setTacticSources() . . . 221
9.75.3.54 setTilingOptimizationLevel() . . . 222
9.75.3.55 setTimingCache() . . . 222
9.75.4 Member Data Documentation . . . 223
9.75.4.1 mImpl . . . 223
9.76 nvinfer1::ICastLayer Class Reference . . . 223
9.76.1 Detailed Description . . . 224
9.76.2 Constructor & Destructor Documentation . . . 224
9.76.2.1 ~ICastLayer() . . . 224
9.76.3 Member Function Documentation . . . 224
9.76.3.1 getToType() . . . 224
9.76.3.2 setToType() . . . 224
9.76.4 Member Data Documentation . . . 225
9.76.4.1 mImpl . . . 225
9.77 nvinfer1::IConcatenationLayer Class Reference . . . 225
9.77.1 Detailed Description . . . 226
9.77.2 Constructor & Destructor Documentation . . . 226
9.77.2.1 ~IConcatenationLayer() . . . 226
9.77.3 Member Function Documentation . . . 226
9.77.3.1 getAxis() . . . 226
9.77.3.2 setAxis() . . . 227
9.77.4 Member Data Documentation . . . 227
9.77.4.1 mImpl . . . 227
9.78 nvinfer1::IConditionLayer Class Reference . . . 227
9.78.1 Detailed Description . . . 228
9.78.2 Constructor & Destructor Documentation . . . 228
9.78.2.1 ~IConditionLayer() . . . 228
9.78.3 Member Data Documentation . . . 228
9.78.3.1 mImpl . . . 228
9.79 nvinfer2::safe::consistency::IConsistencyChecker Class Reference . . . 228
9.79.1 Detailed Description . . . 229
9.79.2 Constructor & Destructor Documentation . . . 229
9.79.2.1 ~IConsistencyChecker() . . . 229
9.79.2.2 IConsistencyChecker() [1/3] . . . 230
9.79.2.3 IConsistencyChecker() [2/3] . . . 230
9.79.2.4 IConsistencyChecker() [3/3] . . . 230
9.79.3 Member Function Documentation . . . 231
9.79.3.1 operator=() [1/2] . . . 231
9.79.3.2 operator=() [2/2] . . . 231
9.79.3.3 validate() . . . 231
9.80 nvinfer1::IConstantLayer Class Reference . . . 232
9.80.1 Detailed Description . . . 232
9.80.2 Constructor & Destructor Documentation . . . 233
9.80.2.1 ~IConstantLayer() . . . 233
9.80.3 Member Function Documentation . . . 233
9.80.3.1 getDimensions() . . . 233
9.80.3.2 getWeights() . . . 233
9.80.3.3 setDimensions() . . . 233
9.80.3.4 setWeights() . . . 234
9.80.4 Member Data Documentation . . . 234
9.80.4.1 mImpl . . . 234

9.81 nvinfer1::IConvolutionLayer Class Reference . . . 234
9.81.1 Detailed Description . . . 236
9.81.2 Constructor & Destructor Documentation . . . 236
9.81.2.1 ~IConvolutionLayer() . . . 236
9.81.3 Member Function Documentation . . . 236
9.81.3.1 getBiasWeights() . . . 236
9.81.3.2 getDilationNd() . . . 237
9.81.3.3 getKernelSizeNd() . . . 237
9.81.3.4 getKernelWeights() . . . 237
9.81.3.5 getNbGroups() . . . 237
9.81.3.6 getNbOutputMaps() . . . 238
9.81.3.7 getPaddingMode() . . . 238
9.81.3.8 getPaddingNd() . . . 238
9.81.3.9 getPostPadding() . . . 238
9.81.3.10 getPrePadding() . . . 239
9.81.3.11 getStrideNd() . . . 239
9.81.3.12 setBiasWeights() . . . 239
9.81.3.13 setDilationNd() . . . 240
9.81.3.14 setInput() . . . 240
9.81.3.15 setKernelSizeNd() . . . 240
9.81.3.16 setKernelWeights() . . . 241
9.81.3.17 setNbGroups() . . . 241
9.81.3.18 setNbOutputMaps() . . . 242
9.81.3.19 setPaddingMode() . . . 242
9.81.3.20 setPaddingNd() . . . 242
9.81.3.21 setPostPadding() . . . 243
9.81.3.22 setPrePadding() . . . 243
9.81.3.23 setStrideNd() . . . 243
9.81.4 Member Data Documentation . . . 244
9.81.4.1 mImpl . . . 244
9.82 nvinfer1::ICudaEngine Class Reference . . . 244
9.82.1 Detailed Description . . . 247
9.82.2 Constructor & Destructor Documentation . . . 247
9.82.2.1 ~ICudaEngine() . . . 247
9.82.3 Member Function Documentation . . . 247
9.82.3.1 createEngineInspector() . . . 247
9.82.3.2 createExecutionContext() [1/2] . . . 248
9.82.3.3 createExecutionContext() [2/2] . . . 248
9.82.3.4 createRuntimeConfig() . . . 248
9.82.3.5 createSerializationConfig() . . . 249
9.82.3.6 getAliasedInputTensor() . . . 249
9.82.3.7 getDeviceMemorySizeForProfileV2() . . . 250
9.82.3.8 getDeviceMemorySizeV2() . . . 250
9.82.3.9 getEngineCapability() . . . 250
9.82.3.10 getEngineStat() . . . 250
9.82.3.11 getErrorRecorder() . . . 251
9.82.3.12 getHardwareCompatibilityLevel() . . . 252
9.82.3.13 getIOTensorName() . . . 252
9.82.3.14 getName() . . . 252
9.82.3.15 getNbAuxStreams() . . . 253
9.82.3.16 getNbIOTensors() . . . 253
9.82.3.17 getNbLayers() . . . 253
9.82.3.18 getNbOptimizationProfiles() . . . 254
9.82.3.19 getProfileShape() . . . 254

9.82.3.20 getProfileTensorValuesV2() . . . 254
9.82.3.21 getProfilingVerbosity() . . . 255
9.82.3.22 getStreamableWeightsSize() . . . 255
9.82.3.23 getTacticSources() . . . 256
9.82.3.24 getTensorBytesPerComponent() [1/2] . . . 256
9.82.3.25 getTensorBytesPerComponent() [2/2] . . . 256
9.82.3.26 getTensorComponentsPerElement() [1/2] . . . 257
9.82.3.27 getTensorComponentsPerElement() [2/2] . . . 258
9.82.3.28 getTensorDataType() . . . 258
9.82.3.29 getTensorFormat() [1/2] . . . 259
9.82.3.30 getTensorFormat() [2/2] . . . 259
9.82.3.31 getTensorFormatDesc() [1/2] . . . 259
9.82.3.32 getTensorFormatDesc() [2/2] . . . 260
9.82.3.33 getTensorIOMode() . . . 260
9.82.3.34 getTensorLocation() . . . 261
9.82.3.35 getTensorShape() . . . 261
9.82.3.36 getTensorVectorizedDim() [1/2] . . . 262
9.82.3.37 getTensorVectorizedDim() [2/2] . . . 262
9.82.3.38 getWeightStreamingAutomaticBudget() . . . 263
9.82.3.39 getWeightStreamingBudgetV2() . . . 263
9.82.3.40 getWeightStreamingScratchMemorySize() . . . 264
9.82.3.41 isDebugTensor() . . . 265
9.82.3.42 isRefittable() . . . 265
9.82.3.43 isShapeInferenceIO() . . . 265
9.82.3.44 serialize() . . . 266
9.82.3.45 serializeWithConfig() . . . 266
9.82.3.46 setErrorRecorder() . . . 266
9.82.3.47 setWeightStreamingBudgetV2() . . . 267
9.82.4 Member Data Documentation . . . 268
9.82.4.1 mImpl . . . 268
9.83 nvinfer1::ICumulativeLayer Class Reference . . . 268
9.83.1 Detailed Description . . . 269
9.83.2 Constructor & Destructor Documentation . . . 269
9.83.2.1 ~ICumulativeLayer() . . . 269
9.83.3 Member Function Documentation . . . 269
9.83.3.1 getExclusive() . . . 270
9.83.3.2 getOperation() . . . 270
9.83.3.3 getReverse() . . . 270
9.83.3.4 setExclusive() . . . 270
9.83.3.5 setOperation() . . . 271
9.83.3.6 setReverse() . . . 271
9.83.4 Member Data Documentation . . . 272
9.83.4.1 mImpl . . . 272
9.84 IDebugListener Class Reference . . . 272
9.84.1 Detailed Description . . . 272
9.85 nvinfer1::v_1_0::IDebugListener Class Reference . . . 272
9.85.1 Constructor & Destructor Documentation . . . 273
9.85.1.1 ~IDebugListener() . . . 273
9.85.2 Member Function Documentation . . . 273
9.85.2.1 getInterfaceInfo() . . . 273
9.85.2.2 processDebugTensor() . . . 273
9.86 nvinfer1::IDeconvolutionLayer Class Reference . . . 274
9.86.1 Detailed Description . . . 276
9.86.2 Constructor & Destructor Documentation . . . 276

9.86.2.1 ~IDeconvolutionLayer() . . . 276
9.86.3 Member Function Documentation . . . 276
9.86.3.1 getBiasWeights() . . . 276
9.86.3.2 getDilationNd() . . . 276
9.86.3.3 getKernelSizeNd() . . . 277
9.86.3.4 getKernelWeights() . . . 277
9.86.3.5 getNbGroups() . . . 277
9.86.3.6 getNbOutputMaps() . . . 277
9.86.3.7 getPaddingMode() . . . 278
9.86.3.8 getPaddingNd() . . . 278
9.86.3.9 getPostPadding() . . . 278
9.86.3.10 getPrePadding() . . . 278
9.86.3.11 getStrideNd() . . . 279
9.86.3.12 setBiasWeights() . . . 279
9.86.3.13 setDilationNd() . . . 279
9.86.3.14 setInput() . . . 279
9.86.3.15 setKernelSizeNd() . . . 280
9.86.3.16 setKernelWeights() . . . 280
9.86.3.17 setNbGroups() . . . 281
9.86.3.18 setNbOutputMaps() . . . 281
9.86.3.19 setPaddingMode() . . . 281
9.86.3.20 setPaddingNd() . . . 282
9.86.3.21 setPostPadding() . . . 282
9.86.3.22 setPrePadding() . . . 282
9.86.3.23 setStrideNd() . . . 283
9.86.4 Member Data Documentation . . . 283
9.86.4.1 mImpl . . . 283
9.87 nvinfer1::IDequantizeLayer Class Reference . . . 283
9.87.1 Detailed Description . . . 284
9.87.2 Constructor & Destructor Documentation . . . 285
9.87.2.1 ~IDequantizeLayer() . . . 285
9.87.3 Member Function Documentation . . . 285
9.87.3.1 getAxis() . . . 286
9.87.3.2 getBlockShape() . . . 286
9.87.3.3 getToType() . . . 286
9.87.3.4 setAxis() . . . 286
9.87.3.5 setBlockShape() . . . 286
9.87.3.6 setToType() . . . 287
9.87.4 Member Data Documentation . . . 287
9.87.4.1 mImpl . . . 287
9.88 nvinfer1::IDimensionExpr Class Reference . . . 288
9.88.1 Detailed Description . . . 288
9.88.2 Constructor & Destructor Documentation . . . 289
9.88.2.1 ~IDimensionExpr() . . . 289
9.88.3 Member Function Documentation . . . 289
9.88.3.1 getConstantValue() . . . 289
9.88.3.2 isConstant() . . . 289
9.88.3.3 isSizeTensor() . . . 289
9.88.4 Member Data Documentation . . . 289
9.88.4.1 mImpl . . . 290
9.89 nvinfer1::IDistCollectiveLayer Class Reference . . . 290
9.89.1 Detailed Description . . . 290
9.89.2 Constructor & Destructor Documentation . . . 290
9.89.2.1 ~IDistCollectiveLayer() . . . 291

9.89.3 Member Data Documentation . . . 291
9.89.3.1 mImpl . . . 291
9.90 nvinfer1::IDynamicQuantizeLayer Class Reference . . . 291
9.90.1 Detailed Description . . . 292
9.90.2 Constructor & Destructor Documentation . . . 293
9.90.2.1 ~IDynamicQuantizeLayer() . . . 293
9.90.3 Member Function Documentation . . . 293
9.90.3.1 getAxis() . . . 293
9.90.3.2 getBlockShape() . . . 293
9.90.3.3 getBlockSize() . . . 294
9.90.3.4 getScaleType() . . . 294
9.90.3.5 getToType() . . . 294
9.90.3.6 setAxis() . . . 294
9.90.3.7 setBlockShape() . . . 295
9.90.3.8 setBlockSize() . . . 295
9.90.3.9 setInput() . . . 295
9.90.3.10 setScaleType() . . . 296
9.90.3.11 setToType() . . . 296
9.90.4 Member Data Documentation . . . 296
9.90.4.1 mImpl . . . 296
9.91 nvinfer1::IEinsumLayer Class Reference . . . 297
9.91.1 Detailed Description . . . 297
9.91.2 Constructor & Destructor Documentation . . . 298
9.91.2.1 ~IEinsumLayer() . . . 298
9.91.3 Member Function Documentation . . . 298
9.91.3.1 getEquation() . . . 298
9.91.3.2 setEquation() . . . 298
9.91.4 Member Data Documentation . . . 299
9.91.4.1 mImpl . . . 299
9.92 nvinfer1::IElementWiseLayer Class Reference . . . 299
9.92.1 Detailed Description . . . 300
9.92.2 Constructor & Destructor Documentation . . . 300
9.92.2.1 ~IElementWiseLayer() . . . 300
9.92.3 Member Function Documentation . . . 300
9.92.3.1 getOperation() . . . 300
9.92.3.2 setOperation() . . . 301
9.92.4 Member Data Documentation . . . 301
9.92.4.1 mImpl . . . 301
9.93 nvinfer1::IEngineInspector Class Reference . . . 301
9.93.1 Detailed Description . . . 302
9.93.2 Constructor & Destructor Documentation . . . 302
9.93.2.1 ~IEngineInspector() . . . 303
9.93.3 Member Function Documentation . . . 303
9.93.3.1 getEngineInformation() . . . 303
9.93.3.2 getErrorRecorder() . . . 303
9.93.3.3 getExecutionContext() . . . 304
9.93.3.4 getLayerInformation() . . . 304
9.93.3.5 setErrorRecorder() . . . 305
9.93.3.6 setExecutionContext() . . . 305
9.93.4 Member Data Documentation . . . 306
9.93.4.1 mImpl . . . 306
9.94 IErrorRecorder Class Reference . . . 306
9.94.1 Detailed Description . . . 306
9.95 nvinfer1::v_1_0::IErrorRecorder Class Reference . . . 307

9.95.1 Member Typedef Documentation . . . 308
9.95.1.1 ErrorDesc . . . 308
9.95.1.2 RefCount . . . 308
9.95.2 Constructor & Destructor Documentation . . . 308
9.95.2.1 IErrorRecorder() . . . 308
9.95.2.2 ~IErrorRecorder() . . . 308
9.95.3 Member Function Documentation . . . 309
9.95.3.1 clear() . . . 309
9.95.3.2 decRefCount() . . . 309
9.95.3.3 getErrorCode() . . . 309
9.95.3.4 getErrorDesc() . . . 310
9.95.3.5 getInterfaceInfo() . . . 311
9.95.3.6 getNbErrors() . . . 311
9.95.3.7 hasOverflowed() . . . 312
9.95.3.8 incRefCount() . . . 312
9.95.3.9 reportError() . . . 312
9.95.4 Member Data Documentation . . . 313
9.95.4.1 kMAX_DESC_LENGTH . . . 313
9.96 nvinfer1::IExecutionContext Class Reference . . . 314
9.96.1 Detailed Description . . . 316
9.96.2 Constructor & Destructor Documentation . . . 317
9.96.2.1 ~IExecutionContext() . . . 317
9.96.3 Member Function Documentation . . . 317
9.96.3.1 allInputDimensionsSpecified() . . . 317
9.96.3.2 enqueueV3() . . . 317
9.96.3.3 executeV2() . . . 318
9.96.3.4 getDebugListener() . . . 318
9.96.3.5 getDebugState() . . . 319
9.96.3.6 getDebugSync() . . . 320
9.96.3.7 getEngine() . . . 320
9.96.3.8 getEnqueueEmitsProfile() . . . 320
9.96.3.9 getErrorRecorder() . . . 321
9.96.3.10 getInputConsumedEvent() . . . 321
9.96.3.11 getMaxOutputSize() . . . 321
9.96.3.12 getName() . . . 322
9.96.3.13 getNvtxVerbosity() . . . 322
9.96.3.14 getOptimizationProfile() . . . 322
9.96.3.15 getOutputAllocator() . . . 322
9.96.3.16 getOutputTensorAddress() . . . 323
9.96.3.17 getPersistentCacheLimit() . . . 323
9.96.3.18 getProfiler() . . . 324
9.96.3.19 getRuntimeConfig() . . . 324
9.96.3.20 getTemporaryStorageAllocator() . . . 324
9.96.3.21 getTensorAddress() . . . 324
9.96.3.22 getTensorShape() . . . 325
9.96.3.23 getTensorStrides() . . . 326
9.96.3.24 getUnfusedTensorsDebugState() . . . 326
9.96.3.25 inferShapes() . . . 326
9.96.3.26 reportToProfiler() . . . 327
9.96.3.27 setAllTensorsDebugState() . . . 328
9.96.3.28 setAuxStreams() . . . 328
9.96.3.29 setCommunicator() . . . 329
9.96.3.30 setDebugListener() . . . 329
9.96.3.31 setDebugSync() . . . 330

9.96.3.32 setDeviceMemory() . . . 330
9.96.3.33 setDeviceMemoryV2() . . . 331
9.96.3.34 setEnqueueEmitsProfile() . . . 331
9.96.3.35 setErrorRecorder() . . . 331
9.96.3.36 setInputConsumedEvent() . . . 332
9.96.3.37 setInputShape() . . . 332
9.96.3.38 setInputTensorAddress() . . . 333
9.96.3.39 setName() . . . 333
9.96.3.40 setNvtxVerbosity() . . . 334
9.96.3.41 setOptimizationProfileAsync() . . . 334
9.96.3.42 setOutputAllocator() . . . 335
9.96.3.43 setOutputTensorAddress() . . . 336
9.96.3.44 setPersistentCacheLimit() . . . 337
9.96.3.45 setProfiler() . . . 337
9.96.3.46 setTemporaryStorageAllocator() . . . 337
9.96.3.47 setTensorAddress() . . . 338
9.96.3.48 setTensorDebugState() . . . 339
9.96.3.49 setUnfusedTensorsDebugState() . . . 340
9.96.3.50 updateDeviceMemorySizeForShapes() . . . 340
9.96.4 Member Data Documentation . . . 340
9.96.4.1 mImpl . . . 341
9.97 nvinfer1::IExprBuilder Class Reference . . . 341
9.97.1 Detailed Description . . . 341
9.97.2 Constructor & Destructor Documentation . . . 342
9.97.2.1 ~IExprBuilder() . . . 342
9.97.3 Member Function Documentation . . . 342
9.97.3.1 constant() . . . 342
9.97.3.2 declareSizeTensor() . . . 342
9.97.3.3 operation() . . . 343
9.97.4 Member Data Documentation . . . 343
9.97.4.1 mImpl . . . 343
9.98 nvinfer1::IFillLayer Class Reference . . . 344
9.98.1 Detailed Description . . . 345
9.98.2 Constructor & Destructor Documentation . . . 346
9.98.2.1 ~IFillLayer() . . . 346
9.98.3 Member Function Documentation . . . 346
9.98.3.1 getAlpha() . . . 346
9.98.3.2 getAlphaInt64() . . . 346
9.98.3.3 getBeta() . . . 347
9.98.3.4 getBetaInt64() . . . 347
9.98.3.5 getDimensions() . . . 347
9.98.3.6 getOperation() . . . 348
9.98.3.7 getToType() . . . 348
9.98.3.8 isAlphaBetaInt64() . . . 348
9.98.3.9 setAlpha() . . . 348
9.98.3.10 setAlphaInt64() . . . 349
9.98.3.11 setBeta() . . . 349
9.98.3.12 setBetaInt64() . . . 350
9.98.3.13 setDimensions() . . . 351
9.98.3.14 setInput() . . . 351
9.98.3.15 setOperation() . . . 352
9.98.3.16 setToType() . . . 352
9.98.4 Member Data Documentation . . . 353
9.98.4.1 mImpl . . . 353

9.99 nvinfer1::IGatherLayer Class Reference . . . 353
9.99.1 Detailed Description . . . 354
9.99.2 Constructor & Destructor Documentation . . . 356
9.99.2.1 ~IGatherLayer() . . . 356
9.99.3 Member Function Documentation . . . 356
9.99.3.1 getGatherAxis() . . . 356
9.99.3.2 getMode() . . . 356
9.99.3.3 getNbElementWiseDims() . . . 357
9.99.3.4 setGatherAxis() . . . 357
9.99.3.5 setMode() . . . 357
9.99.3.6 setNbElementWiseDims() . . . 357
9.99.4 Member Data Documentation . . . 358
9.99.4.1 mImpl . . . 358
9.100 IGpuAllocator Class Reference . . . 358
9.100.1 Detailed Description . . . 359
9.101 nvinfer1::v_1_0::IGpuAllocator Class Reference . . . 359
9.101.1 Constructor & Destructor Documentation . . . 360
9.101.1.1 ~IGpuAllocator() . . . 360
9.101.1.2 IGpuAllocator() . . . 360
9.101.2 Member Function Documentation . . . 360
9.101.2.1 allocate() . . . 360
9.101.2.2 allocateAsync() . . . 361
9.101.2.3 deallocate() . . . 362
9.101.2.4 deallocateAsync() . . . 363
9.101.2.5 getInterfaceInfo() . . . 363
9.101.2.6 reallocate() . . . 364
9.102 IGpuAsyncAllocator Class Reference . . . 365
9.102.1 Detailed Description . . . 365
9.103 nvinfer1::v_1_0::IGpuAsyncAllocator Class Reference . . . 365
9.103.1 Constructor & Destructor Documentation . . . 366
9.103.1.1 IGpuAsyncAllocator() . . . 366
9.103.1.2 ~IGpuAsyncAllocator() . . . 366
9.103.2 Member Function Documentation . . . 366
9.103.2.1 allocate() . . . 366
9.103.2.2 allocateAsync() . . . 367
9.103.2.3 deallocate() . . . 368
9.103.2.4 deallocateAsync() . . . 369
9.103.2.5 getInterfaceInfo() . . . 369
9.104 nvinfer1::IGridSampleLayer Class Reference . . . 370
9.104.1 Detailed Description . . . 370
9.104.2 Constructor & Destructor Documentation . . . 371
9.104.2.1 ~IGridSampleLayer() . . . 371
9.104.3 Member Function Documentation . . . 371
9.104.3.1 getAlignCorners() . . . 371
9.104.3.2 getInterpolationMode() . . . 371
9.104.3.3 getSampleMode() . . . 372
9.104.3.4 setAlignCorners() . . . 372
9.104.3.5 setInterpolationMode() . . . 372
9.104.3.6 setSampleMode() . . . 373
9.104.4 Member Data Documentation . . . 373
9.104.4.1 mImpl . . . 373
9.105 nvinfer1::IHostMemory Class Reference . . . 373
9.105.1 Detailed Description . . . 374
9.105.2 Constructor & Destructor Documentation . . . 374

9.105.2.1 ~IHostMemory() . . . 374
9.105.3 Member Function Documentation . . . 374
9.105.3.1 data() . . . 374
9.105.3.2 size() . . . 375
9.105.3.3 type() . . . 375
9.105.4 Member Data Documentation . . . 375
9.105.4.1 mImpl . . . 375
9.106 nvinfer1::IIdentityLayer Class Reference . . . 375
9.106.1 Detailed Description . . . 376
9.106.2 Constructor & Destructor Documentation . . . 377
9.106.2.1 ~IIdentityLayer() . . . 377
9.106.3 Member Data Documentation . . . 377
9.106.3.1 mImpl . . . 377
9.107 nvinfer1::IIfConditional Class Reference . . . 377
9.107.1 Detailed Description . . . 378
9.107.2 Constructor & Destructor Documentation . . . 378
9.107.2.1 ~IIfConditional() . . . 379
9.107.3 Member Function Documentation . . . 379
9.107.3.1 addInput() . . . 379
9.107.3.2 addOutput() . . . 379
9.107.3.3 getName() . . . 380
9.107.3.4 setCondition() . . . 380
9.107.3.5 setName() . . . 380
9.107.4 Member Data Documentation . . . 381
9.107.4.1 mImpl . . . 381
9.108 nvinfer1::IIfConditionalBoundaryLayer Class Reference . . . 381
9.108.1 Detailed Description . . . 382
9.108.2 Constructor & Destructor Documentation . . . 382
9.108.2.1 ~IIfConditionalBoundaryLayer() . . . 382
9.108.3 Member Function Documentation . . . 382
9.108.3.1 getConditional() . . . 382
9.108.4 Member Data Documentation . . . 382
9.108.4.1 mBoundary . . . 383
9.109 nvinfer1::IIfConditionalInputLayer Class Reference . . . 383
9.109.1 Detailed Description . . . 383
9.109.2 Constructor & Destructor Documentation . . . 383
9.109.2.1 ~IIfConditionalInputLayer() . . . 384
9.109.3 Member Data Documentation . . . 384
9.109.3.1 mImpl . . . 384
9.110 nvinfer1::IIfConditionalOutputLayer Class Reference . . . 384
9.110.1 Detailed Description . . . 385
9.110.2 Constructor & Destructor Documentation . . . 385
9.110.2.1 ~IIfConditionalOutputLayer() . . . 385
9.110.3 Member Data Documentation . . . 385
9.110.3.1 mImpl . . . 385
9.111 nvinfer1::IIteratorLayer Class Reference . . . 385
9.111.1 Detailed Description . . . 386
9.111.2 Constructor & Destructor Documentation . . . 386
9.111.2.1 ~IIteratorLayer() . . . 386
9.111.3 Member Function Documentation . . . 386
9.111.3.1 getAxis() . . . 387
9.111.3.2 getReverse() . . . 387
9.111.3.3 setAxis() . . . 387
9.111.3.4 setReverse() . . . 387

9.111.4 Member Data Documentation . . . 387
9.111.4.1 mImpl . . . 388
9.112 nvinfer1::IKVCacheUpdateLayer Class Reference . . . 388
9.112.1 Detailed Description . . . 389
9.112.2 Constructor & Destructor Documentation . . . 389
9.112.2.1 ~IKVCacheUpdateLayer() . . . 389
9.112.3 Member Function Documentation . . . 389
9.112.3.1 getCacheMode() . . . 389
9.112.3.2 getUpdateForm() . . . 390
9.112.3.3 getUpdateLengths() . . . 390
9.112.3.4 setCacheMode() . . . 390
9.112.3.5 setInput() . . . 391
9.112.3.6 setUpdateForm() . . . 391
9.112.3.7 setUpdateLengths() . . . 392
9.112.4 Member Data Documentation . . . 392
9.112.4.1 mImpl . . . 392
9.113 nvinfer1::ILayer Class Reference . . . 393
9.113.1 Detailed Description . . . 394
9.113.2 Constructor & Destructor Documentation . . . 394
9.113.2.1 ~ILayer() . . . 394
9.113.3 Member Function Documentation . . . 394
9.113.3.1 getInput() . . . 395
9.113.3.2 getMetadata() . . . 396
9.113.3.3 getName() . . . 396
9.113.3.4 getNbInputs() . . . 396
9.113.3.5 getNbOutputs() . . . 397
9.113.3.6 getNbRanks() . . . 397
9.113.3.7 getOutput() . . . 397
9.113.3.8 getOutputType() . . . 397
9.113.3.9 getType() . . . 398
9.113.3.10 setInput() . . . 398
9.113.3.11 setMetadata() . . . 399
9.113.3.12 setName() . . . 399
9.113.3.13 setNbRanks() . . . 399
9.113.4 Member Data Documentation . . . 400
9.113.4.1 mLayer . . . 400
9.114 nvinfer1::v_1_0::ILogger Class Reference . . . 400
9.114.1 Detailed Description . . . 401
9.114.2 Member Enumeration Documentation . . . 401
9.114.2.1 Severity . . . 401
9.114.3 Constructor & Destructor Documentation . . . 401
9.114.3.1 ILogger() . . . 402
9.114.3.2 ~ILogger() . . . 402
9.114.4 Member Function Documentation . . . 402
9.114.4.1 getInterfaceInfo() . . . 402
9.114.4.2 log() . . . 402
9.115 nvinfer1::v_1_0::ILoggerFinder Class Reference . . . 403
9.115.1 Detailed Description . . . 404
9.115.2 Constructor & Destructor Documentation . . . 404
9.115.2.1 ~ILoggerFinder() . . . 404
9.115.3 Member Function Documentation . . . 404
9.115.3.1 findLogger() . . . 404
9.115.3.2 getInterfaceInfo() . . . 404
9.116 nvinfer1::ILoop Class Reference . . . 405

9.116.1 Detailed Description . . . 405
9.116.2 Constructor & Destructor Documentation . . . 406
9.116.2.1 ~ILoop() . . . 406
9.116.3 Member Function Documentation . . . 406
9.116.3.1 addIterator() . . . 406
9.116.3.2 addLoopOutput() . . . 406
9.116.3.3 addRecurrence() . . . 406
9.116.3.4 addTripLimit() . . . 407
9.116.3.5 getName() . . . 407
9.116.3.6 setName() . . . 407
9.116.4 Member Data Documentation . . . 408
9.116.4.1 mImpl . . . 408
9.117 nvinfer1::ILoopBoundaryLayer Class Reference . . . 408
9.117.1 Detailed Description . . . 409
9.117.2 Constructor & Destructor Documentation . . . 409
9.117.2.1 ~ILoopBoundaryLayer() . . . 409
9.117.3 Member Function Documentation . . . 409
9.117.3.1 getLoop() . . . 409
9.117.4 Member Data Documentation . . . 409
9.117.4.1 mBoundary . . . 410
9.118 nvinfer1::ILoopOutputLayer Class Reference . . . 410
9.118.1 Detailed Description . . . 411
9.118.2 Constructor & Destructor Documentation . . . 411
9.118.2.1 ~ILoopOutputLayer() . . . 411
9.118.3 Member Function Documentation . . . 411
9.118.3.1 getAxis() . . . 411
9.118.3.2 getLoopOutput() . . . 411
9.118.3.3 setAxis() . . . 412
9.118.3.4 setInput() . . . 412
9.118.4 Member Data Documentation . . . 412
9.118.4.1 mImpl . . . 412
9.119 nvinfer1::ILRNLayer Class Reference . . . 413
9.119.1 Detailed Description . . . 413
9.119.2 Constructor & Destructor Documentation . . . 414
9.119.2.1 ~ILRNLayer() . . . 414
9.119.3 Member Function Documentation . . . 414
9.119.3.1 getAlpha() . . . 414
9.119.3.2 getBeta() . . . 414
9.119.3.3 getK() . . . 415
9.119.3.4 getWindowSize() . . . 415
9.119.3.5 setAlpha() . . . 415
9.119.3.6 setBeta() . . . 415
9.119.3.7 setK() . . . 416
9.119.3.8 setWindowSize() . . . 416
9.119.4 Member Data Documentation . . . 416
9.119.4.1 mImpl . . . 416
9.120 nvinfer1::IMatrixMultiplyLayer Class Reference . . . 417
9.120.1 Detailed Description . . . 417
9.120.2 Constructor & Destructor Documentation . . . 418
9.120.2.1 ~IMatrixMultiplyLayer() . . . 418
9.120.3 Member Function Documentation . . . 418
9.120.3.1 getOperation() . . . 418
9.120.3.2 setOperation() . . . 418
9.120.4 Member Data Documentation . . . 419

9.120.4.1 mImpl . . . 419
9.121 nvinfer1::IMoELayer Class Reference . . . 419
9.121.1 Detailed Description . . . 421
9.121.2 Constructor & Destructor Documentation . . . 423
9.121.2.1 ~IMoELayer() . . . 423
9.121.3 Member Function Documentation . . . 423
9.121.3.1 getActivationType() . . . 423
9.121.3.2 getDynQOutputScaleType() . . . 424
9.121.3.3 getQuantizationBlockShape() . . . 424
9.121.3.4 getQuantizationToType() . . . 424
9.121.3.5 getSwigluParamAlpha() . . . 425
9.121.3.6 getSwigluParamBeta() . . . 425
9.121.3.7 getSwigluParamLimit() . . . 425
9.121.3.8 setActivationType() . . . 425
9.121.3.9 setDynQOutputScaleType() . . . 426
9.121.3.10 setGatedBiases() . . . 426
9.121.3.11 setGatedWeights() . . . 427
9.121.3.12 setInput() [1/2] . . . 427
9.121.3.13 setInput() [2/2] . . . 428
9.121.3.14 setQuantizationBlockShape() . . . 428
9.121.3.15 setQuantizationDynamicDblQ() . . . 428
9.121.3.16 setQuantizationStatic() . . . 429
9.121.3.17 setQuantizationToType() . . . 430
9.121.3.18 setSwigluParamAlpha() . . . 430
9.121.3.19 setSwigluParamBeta() . . . 431
9.121.3.20 setSwigluParamLimit() . . . 431
9.121.3.21 setSwigluParams() . . . 432
9.121.4 Member Data Documentation . . . 432
9.121.4.1 mImpl . . . 432
9.122 nvinfer1::INetworkDefinition Class Reference . . . 433
9.122.1 Detailed Description . . . 437
9.122.2 Constructor & Destructor Documentation . . . 438
9.122.2.1 ~INetworkDefinition() . . . 438
9.122.3 Member Function Documentation . . . 438
9.122.3.1 addActivation() . . . 438
9.122.3.2 addAssertion() . . . 439
9.122.3.3 addAttention() . . . 439
9.122.3.4 addAttentionV2() . . . 440
9.122.3.5 addCast() . . . 441
9.122.3.6 addConcatenation() . . . 441
9.122.3.7 addConstant() . . . 442
9.122.3.8 addConvolutionNd() . . . 443
9.122.3.9 addCumulative() . . . 443
9.122.3.10 addDeconvolutionNd() . . . 444
9.122.3.11 addDequantize() . . . 445
9.122.3.12 addDistCollective() . . . 446
9.122.3.13 addDynamicQuantize() . . . 447
9.122.3.14 addDynamicQuantizeV2() . . . 447
9.122.3.15 addEinsum() . . . 448
9.122.3.16 addElementWise() . . . 449
9.122.3.17 addFill() . . . 449
9.122.3.18 addGather() . . . 450
9.122.3.19 addGatherV2() . . . 451
9.122.3.20 addGridSample() . . . 451

9.122.3.21addIdentity() . . . 452
9.122.3.22addIfConditional() . . . 452
9.122.3.23addInput() . . . 453
9.122.3.24addKVCacheUpdate() . . . 454
9.122.3.25addLoop() . . . 455
9.122.3.26addLRN() . . . 455
9.122.3.27addMatrixMultiply() . . . 456
9.122.3.28addMoE() . . . 456
9.122.3.29addNMS() [1/2] . . . 457
9.122.3.30addNMS() [2/2] . . . 458
9.122.3.31addNonZero() [1/2] . . . 458
9.122.3.32addNonZero() [2/2] . . . 459
9.122.3.33addNormalization() . . . 459
9.122.3.34addNormalizationV2() . . . 461
9.122.3.35addOneHot() . . . 462
9.122.3.36addPaddingNd() . . . 463
9.122.3.37addParametricReLU() . . . 463
9.122.3.38addPluginV2() . . . 464
9.122.3.39addPluginV3() . . . 464
9.122.3.40addPoolingNd() . . . 465
9.122.3.41addQuantize() . . . 466
9.122.3.42addRaggedSoftMax() . . . 466
9.122.3.43addReduce() . . . 467
9.122.3.44addResize() . . . 468
9.122.3.45addReverseSequence() . . . 468
9.122.3.46addRotaryEmbedding() . . . 469
9.122.3.47addScale() . . . 470
9.122.3.48addScaleNd() . . . 470
9.122.3.49addScatter() . . . 471
9.122.3.50addSelect() . . . 472
9.122.3.51addShape() . . . 473
9.122.3.52addShuffle() . . . 473
9.122.3.53addSlice() . . . 474
9.122.3.54addSoftMax() . . . 475
9.122.3.55addSqueeze() . . . 475
9.122.3.56addTopK() [1/2] . . . 476
9.122.3.57addTopK() [2/2] . . . 476
9.122.3.58addUnary() . . . 477
9.122.3.59addUnsqueeze() . . . 478
9.122.3.60areWeightsMarkedRefittable() . . . 479
9.122.3.61getBuilder() . . . 479
9.122.3.62getErrorRecorder() . . . 479
9.122.3.63getFlag() . . . 480
9.122.3.64getFlags() . . . 480
9.122.3.65getInput() . . . 480
9.122.3.66getLayer() . . . 481
9.122.3.67getName() . . . 481
9.122.3.68getNbInputs() . . . 482
9.122.3.69getNbLayers() . . . 482
9.122.3.70getNbOutputs() . . . 482
9.122.3.71getOutput() . . . 482
9.122.3.72hasImplicitBatchDimension() . . . 483
9.122.3.73isDebugTensor() . . . 483
9.122.3.74markDebug() . . . 483

9.122.3.75 markOutput() . . . 484
9.122.3.76 markOutputForShapes() . . . 484
9.122.3.77 markUnfusedTensorsAsDebugTensors() . . . 485
9.122.3.78 markWeightsRefittable() . . . 485
9.122.3.79 removeTensor() . . . 486
9.122.3.80 setErrorRecorder() . . . 486
9.122.3.81 setName() . . . 486
9.122.3.82 setWeightsName() . . . 487
9.122.3.83 unmarkDebug() . . . 488
9.122.3.84 unmarkOutput() . . . 488
9.122.3.85 unmarkOutputForShapes() . . . 489
9.122.3.86 unmarkUnfusedTensorsAsDebugTensors() . . . 489
9.122.3.87 unmarkWeightsRefittable() . . . 489
9.122.4 Member Data Documentation . . . 490
9.122.4.1 mImpl . . . 490
9.123 nvinfer1::INMSLayer Class Reference . . . 490
9.123.1 Detailed Description . . . 491
9.123.2 Constructor & Destructor Documentation . . . 492
9.123.2.1 ~INMSLayer() . . . 492
9.123.3 Member Function Documentation . . . 492
9.123.3.1 getBoundingBoxFormat() . . . 492
9.123.3.2 getIndicesType() . . . 492
9.123.3.3 getTopKBoxLimit() . . . 493
9.123.3.4 setBoundingBoxFormat() . . . 493
9.123.3.5 setIndicesType() . . . 493
9.123.3.6 setInput() . . . 494
9.123.3.7 setTopKBoxLimit() . . . 494
9.123.4 Member Data Documentation . . . 494
9.123.4.1 mImpl . . . 495
9.124 nvinfer1::INoCopy Class Reference . . . 495
9.124.1 Detailed Description . . . 496
9.124.2 Constructor & Destructor Documentation . . . 496
9.124.2.1 INoCopy() [1/3] . . . 496
9.124.2.2 ~INoCopy() . . . 496
9.124.2.3 INoCopy() [2/3] . . . 496
9.124.2.4 INoCopy() [3/3] . . . 496
9.124.3 Member Function Documentation . . . 497
9.124.3.1 operator=() [1/2] . . . 497
9.124.3.2 operator=() [2/2] . . . 497
9.125 INonZero Class Reference . . . 497
9.125.1 Detailed Description . . . 497
9.126 nvinfer1::INonZeroLayer Class Reference . . . 498
9.126.1 Constructor & Destructor Documentation . . . 498
9.126.1.1 ~INonZeroLayer() . . . 498
9.126.2 Member Function Documentation . . . 498
9.126.2.1 getIndicesType() . . . 499
9.126.2.2 setIndicesType() . . . 499
9.126.3 Member Data Documentation . . . 499
9.126.3.1 mImpl . . . 499
9.127 nvinfer1::INormalizationLayer Class Reference . . . 500
9.127.1 Detailed Description . . . 500
9.127.2 Constructor & Destructor Documentation . . . 501
9.127.2.1 ~INormalizationLayer() . . . 501
9.127.3 Member Function Documentation . . . 501

9.127.3.1 getAxes() . . . 501
9.127.3.2 getEpsilon() . . . 501
9.127.3.3 getNbGroups() . . . 502
9.127.3.4 isV2() . . . 502
9.127.3.5 setAxes() . . . 502
9.127.3.6 setEpsilon() . . . 502
9.127.3.7 setNbGroups() . . . 503
9.127.4 Member Data Documentation . . . 503
9.127.4.1 mImpl . . . 503
9.128 nvinfer1::InterfaceInfo Class Reference . . . 504
9.128.1 Detailed Description . . . 504
9.128.2 Member Data Documentation . . . 504
9.128.2.1 kind . . . 504
9.128.2.2 major . . . 504
9.128.2.3 minor . . . 504
9.129 nvinfer1::IOneHotLayer Class Reference . . . 505
9.129.1 Detailed Description . . . 505
9.129.2 Constructor & Destructor Documentation . . . 506
9.129.2.1 ~IOneHotLayer() . . . 506
9.129.3 Member Function Documentation . . . 506
9.129.3.1 getAxis() . . . 506
9.129.3.2 setAxis() . . . 506
9.129.4 Member Data Documentation . . . 506
9.129.4.1 mImpl . . . 507
9.130 nvonnxparser::IOnnxConfig Class Reference . . . 507
9.130.1 Detailed Description . . . 508
9.130.2 Member Typedef Documentation . . . 508
9.130.2.1 Verbosity . . . 508
9.130.3 Constructor & Destructor Documentation . . . 508
9.130.3.1 ~IOnnxConfig() . . . 508
9.130.4 Member Function Documentation . . . 508
9.130.4.1 addVerbosity() . . . 508
9.130.4.2 getFullTextFileName() . . . 509
9.130.4.3 getModelDtype() . . . 509
9.130.4.4 getModelFileName() . . . 509
9.130.4.5 getPrintLayerInfo() . . . 510
9.130.4.6 getTextFileName() . . . 510
9.130.4.7 getVerbosityLevel() . . . 510
9.130.4.8 reduceVerbosity() . . . 511
9.130.4.9 setFullTextFileName() . . . 511
9.130.4.10 setModelDtype() . . . 511
9.130.4.11 setModelFileName() . . . 512
9.130.4.12 setPrintLayerInfo() . . . 512
9.130.4.13 setTextFileName() . . . 512
9.130.4.14 setVerbosityLevel() . . . 513
9.131 nvinfer1::IOptimizationProfile Class Reference . . . 513
9.131.1 Detailed Description . . . 514
9.131.2 Constructor & Destructor Documentation . . . 515
9.131.2.1 ~IOptimizationProfile() . . . 515
9.131.3 Member Function Documentation . . . 515
9.131.3.1 getDimensions() . . . 515
9.131.3.2 getExtraMemoryTarget() . . . 515
9.131.3.3 getNbShapeValues() . . . 515
9.131.3.4 getShapeValuesV2() . . . 516

9.131.3.5 isValid() . . . 516
9.131.3.6 setDimensions() . . . 517
9.131.3.7 setExtraMemoryTarget() . . . 518
9.131.3.8 setShapeValuesV2() . . . 518
9.131.4 Member Data Documentation . . . 520
9.131.4.1 mImpl . . . 520
9.132IOutputAllocator Class Reference . . . 520
9.132.1 Detailed Description . . . 520
9.133nvinfer1::v_1_0::IOutputAllocator Class Reference . . . 520
9.133.1 Member Function Documentation . . . 521
9.133.1.1 getInterfaceInfo() . . . 521
9.133.1.2 notifyShape() . . . 521
9.133.1.3 reallocateOutput() . . . 522
9.133.1.4 reallocateOutputAsync() . . . 522
9.134nvinfer1::IPaddingLayer Class Reference . . . 523
9.134.1 Detailed Description . . . 524
9.134.2 Constructor & Destructor Documentation . . . 524
9.134.2.1 ~IPaddingLayer() . . . 524
9.134.3 Member Function Documentation . . . 524
9.134.3.1 getPostPaddingNd() . . . 524
9.134.3.2 getPrePaddingNd() . . . 525
9.134.3.3 setPostPaddingNd() . . . 525
9.134.3.4 setPrePaddingNd() . . . 525
9.134.4 Member Data Documentation . . . 526
9.134.4.1 mImpl . . . 526
9.135nvinfer1::IParametricReLULayer Class Reference . . . 526
9.135.1 Detailed Description . . . 526
9.135.2 Constructor & Destructor Documentation . . . 527
9.135.2.1 ~IParametricReLULayer() . . . 527
9.135.3 Member Data Documentation . . . 527
9.135.3.1 mImpl . . . 527
9.136nvonnxparser::IParser Class Reference . . . 527
9.136.1 Detailed Description . . . 529
9.136.2 Constructor & Destructor Documentation . . . 529
9.136.2.1 ~IParser() . . . 529
9.136.3 Member Function Documentation . . . 529
9.136.3.1 clearErrors() . . . 529
9.136.3.2 clearFlag() . . . 530
9.136.3.3 getError() . . . 530
9.136.3.4 getFlag() . . . 530
9.136.3.5 getFlags() . . . 531
9.136.3.6 getLayerOutputTensor() . . . 531
9.136.3.7 getNbErrors() . . . 531
9.136.3.8 getNbSubgraphs() . . . 532
9.136.3.9 getSubgraphNodes() . . . 532
9.136.3.10 getUsedVCPluginLibraries() . . . 532
9.136.3.11 isSubgraphSupported() . . . 533
9.136.3.12 loadInitializer() . . . 533
9.136.3.13 loadModelProto() . . . 534
9.136.3.14 parse() . . . 535
9.136.3.15 parseFromFile() . . . 535
9.136.3.16 parseModelProto() . . . 536
9.136.3.17 setBuilderConfig() . . . 536
9.136.3.18 setFlag() . . . 536

9.136.3.19 setFlags() . . . 536
9.136.3.20 supportsModelV2() . . . 537
9.136.3.21 supportsOperator() . . . 537
9.137 nvonnxparser::IParserError Class Reference . . . 538
9.137.1 Detailed Description . . . 539
9.137.2 Constructor & Destructor Documentation . . . 539
9.137.2.1 ~IParserError() . . . 539
9.137.3 Member Function Documentation . . . 539
9.137.3.1 code() . . . 539
9.137.3.2 desc() . . . 539
9.137.3.3 file() . . . 539
9.137.3.4 func() . . . 540
9.137.3.5 line() . . . 540
9.137.3.6 localFunctionStack() . . . 540
9.137.3.7 localFunctionStackSize() . . . 540
9.137.3.8 node() . . . 540
9.137.3.9 nodeName() . . . 541
9.137.3.10 nodeOperator() . . . 541
9.138 nvonnxparser::IParserRefitter Class Reference . . . 541
9.138.1 Detailed Description . . . 541
9.138.2 Constructor & Destructor Documentation . . . 542
9.138.2.1 ~IParserRefitter() . . . 542
9.138.3 Member Function Documentation . . . 542
9.138.3.1 clearErrors() . . . 542
9.138.3.2 getError() . . . 542
9.138.3.3 getNbErrors() . . . 543
9.138.3.4 loadInitializer() . . . 543
9.138.3.5 loadModelProto() . . . 544
9.138.3.6 refitFromBytes() . . . 544
9.138.3.7 refitFromFile() . . . 545
9.138.3.8 refitModelProto() . . . 545
9.139 IPluginCapability Class Reference . . . 546
9.139.1 Detailed Description . . . 546
9.140 nvinfer1::v_1_0::IPluginCapability Class Reference . . . 546
9.141 nvinfer2::safe::consistency::IPluginChecker Class Reference . . . 547
9.141.1 Detailed Description . . . 547
9.141.2 Constructor & Destructor Documentation . . . 547
9.141.2.1 ~IPluginChecker() . . . 547
9.141.2.2 IPluginChecker() [1/3] . . . 548
9.141.2.3 IPluginChecker() [2/3] . . . 548
9.141.2.4 IPluginChecker() [3/3] . . . 548
9.141.3 Member Function Documentation . . . 549
9.141.3.1 operator=() [1/2] . . . 549
9.141.3.2 operator=() [2/2] . . . 549
9.141.3.3 validate() . . . 549
9.142 IPluginCreator Class Reference . . . 550
9.142.1 Detailed Description . . . 550
9.143 nvinfer1::v_1_0::IPluginCreator Class Reference . . . 550
9.143.1 Constructor & Destructor Documentation . . . 551
9.143.1.1 IPluginCreator() . . . 551
9.143.1.2 ~IPluginCreator() . . . 551
9.143.2 Member Function Documentation . . . 551
9.143.2.1 createPlugin() . . . 551
9.143.2.2 deserializePlugin() . . . 552

9.143.2.3 getFieldNames() . . . 552
9.143.2.4 getInterfaceInfo() . . . 553
9.143.2.5 getPluginName() . . . 553
9.143.2.6 getPluginNamespace() . . . 553
9.143.2.7 getPluginVersion() . . . 554
9.143.2.8 setPluginNamespace() . . . 554
9.144 IPluginCreatorInterface Class Reference . . . 555
9.144.1 Detailed Description . . . 555
9.145 nvinfer1::v_1_0::IPluginCreatorInterface Class Reference . . . 555
9.145.1 Constructor & Destructor Documentation . . . 556
9.145.1.1 ~IPluginCreatorInterface() . . . 556
9.145.1.2 IPluginCreatorInterface() [1/3] . . . 556
9.145.1.3 IPluginCreatorInterface() [2/3] . . . 556
9.145.1.4 IPluginCreatorInterface() [3/3] . . . 556
9.145.2 Member Function Documentation . . . 557
9.145.2.1 operator=() [1/2] . . . 557
9.145.2.2 operator=() [2/2] . . . 557
9.146 IPluginCreatorV3One Class Reference . . . 557
9.146.1 Detailed Description . . . 557
9.147 nvinfer1::v_1_0::IPluginCreatorV3One Class Reference . . . 558
9.147.1 Constructor & Destructor Documentation . . . 558
9.147.1.1 IPluginCreatorV3One() [1/3] . . . 559
9.147.1.2 ~IPluginCreatorV3One() . . . 559
9.147.1.3 IPluginCreatorV3One() [2/3] . . . 559
9.147.1.4 IPluginCreatorV3One() [3/3] . . . 559
9.147.2 Member Function Documentation . . . 559
9.147.2.1 createPlugin() . . . 559
9.147.2.2 getFieldNames() . . . 560
9.147.2.3 getInterfaceInfo() . . . 560
9.147.2.4 getPluginName() . . . 560
9.147.2.5 getPluginNamespace() . . . 561
9.147.2.6 getPluginVersion() . . . 561
9.147.2.7 operator=() [1/2] . . . 561
9.147.2.8 operator=() [2/2] . . . 561
9.148 nvinfer1::IPluginRegistry Class Reference . . . 562
9.148.1 Detailed Description . . . 563
9.148.2 Member Typedef Documentation . . . 563
9.148.2.1 PluginLibraryHandle . . . 563
9.148.3 Constructor & Destructor Documentation . . . 563
9.148.3.1 ~IPluginRegistry() . . . 563
9.148.4 Member Function Documentation . . . 563
9.148.4.1 acquirePluginResource() . . . 563
9.148.4.2 deregisterCreator() . . . 564
9.148.4.3 deregisterLibrary() . . . 564
9.148.4.4 getAllCreators() . . . 565
9.148.4.5 getAllCreatorsRecursive() . . . 565
9.148.4.6 getCreator() . . . 566
9.148.4.7 getErrorRecorder() . . . 566
9.148.4.8 isParentSearchEnabled() . . . 567
9.148.4.9 loadLibrary() . . . 567
9.148.4.10 registerCreator() . . . 567
9.148.4.11 releasePluginResource() . . . 568
9.148.4.12 setErrorRecorder() . . . 568
9.148.4.13 setParentSearchEnabled() . . . 569

9.149 IPluginResource Class Reference . . . 569
9.149.1 Detailed Description . . . 569
9.150 nvinfer1::v_1_0::IPluginResource Class Reference . . . 570
9.150.1 Constructor & Destructor Documentation . . . 570
9.150.1.1 ~IPluginResource() . . . 570
9.150.1.2 IPluginResource() [1/3] . . . 570
9.150.1.3 IPluginResource() [2/3] . . . 571
9.150.1.4 IPluginResource() [3/3] . . . 571
9.150.2 Member Function Documentation . . . 571
9.150.2.1 clone() . . . 571
9.150.2.2 getInterfaceInfo() . . . 571
9.150.2.3 operator=() [1/2] . . . 572
9.150.2.4 operator=() [2/2] . . . 572
9.150.2.5 release() . . . 572
9.151 nvinfer1::IPluginResourceContext Class Reference . . . 572
9.151.1 Detailed Description . . . 573
9.151.2 Constructor & Destructor Documentation . . . 573
9.151.2.1 ~IPluginResourceContext() . . . 573
9.151.2.2 IPluginResourceContext() [1/3] . . . 573
9.151.2.3 IPluginResourceContext() [2/3] . . . 574
9.151.2.4 IPluginResourceContext() [3/3] . . . 574
9.151.3 Member Function Documentation . . . 574
9.151.3.1 getErrorRecorder() . . . 574
9.151.3.2 getGpuAllocator() . . . 574
9.151.3.3 operator=() [1/2] . . . 575
9.151.3.4 operator=() [2/2] . . . 575
9.152 nvinfer1::IPluginV2 Class Reference . . . 575
9.152.1 Detailed Description . . . 576
9.152.2 Member Function Documentation . . . 576
9.152.2.1 clone() . . . 577
9.152.2.2 configureWithFormat() . . . 577
9.152.2.3 destroy() . . . 578
9.152.2.4 enqueue() . . . 578
9.152.2.5 getNbOutputs() . . . 579
9.152.2.6 getOutputDimensions() . . . 579
9.152.2.7 getPluginNamespace() . . . 580
9.152.2.8 getPluginType() . . . 580
9.152.2.9 getPluginVersion() . . . 581
9.152.2.10 getSerializationSize() . . . 581
9.152.2.11 getTensorRTVersion() . . . 582
9.152.2.12 getWorkspaceSize() . . . 582
9.152.2.13 initialize() . . . 583
9.152.2.14 serialize() . . . 583
9.152.2.15 setPluginNamespace() . . . 583
9.152.2.16 supportsFormat() . . . 584
9.152.2.17 terminate() . . . 585
9.153 nvinfer1::IPluginV2DynamicExt Class Reference . . . 585
9.153.1 Detailed Description . . . 586
9.153.2 Constructor & Destructor Documentation . . . 587
9.153.2.1 ~IPluginV2DynamicExt() . . . 587
9.153.3 Member Function Documentation . . . 587
9.153.3.1 clone() . . . 587
9.153.3.2 configurePlugin() . . . 588
9.153.3.3 enqueue() . . . 589

9.153.3.4 getOutputDimensions() . . . 589
9.153.3.5 getTensorRTVersion() . . . 590
9.153.3.6 getWorkspaceSize() . . . 590
9.153.3.7 supportsFormatCombination() . . . 591
9.153.4 Member Data Documentation . . . 591
9.153.4.1 kFORMAT_COMBINATION_LIMIT . . . 591
9.154 nvinfer1::IPluginV2Ext Class Reference . . . 592
9.154.1 Detailed Description . . . 593
9.154.2 Constructor & Destructor Documentation . . . 593
9.154.2.1 IPluginV2Ext() . . . 593
9.154.2.2 ~IPluginV2Ext() . . . 593
9.154.3 Member Function Documentation . . . 593
9.154.3.1 attachToContext() . . . 593
9.154.3.2 clone() . . . 594
9.154.3.3 configurePlugin() . . . 595
9.154.3.4 configureWithFormat() . . . 596
9.154.3.5 detachFromContext() . . . 596
9.154.3.6 getOutputDataType() . . . 596
9.154.3.7 getTensorRTVersion() . . . 597
9.155 nvinfer1::IPluginV2IOExt Class Reference . . . 598
9.155.1 Detailed Description . . . 598
9.155.2 Member Function Documentation . . . 599
9.155.2.1 configurePlugin() . . . 599
9.155.2.2 getTensorRTVersion() . . . 599
9.155.2.3 supportsFormatCombination() . . . 600
9.156 nvinfer1::IPluginV2Layer Class Reference . . . 601
9.156.1 Detailed Description . . . 601
9.156.2 Constructor & Destructor Documentation . . . 602
9.156.2.1 ~IPluginV2Layer() . . . 602
9.156.3 Member Function Documentation . . . 602
9.156.3.1 getPlugin() . . . 602
9.156.4 Member Data Documentation . . . 602
9.156.4.1 mImpl . . . 602
9.157 IPluginV3 Class Reference . . . 602
9.157.1 Detailed Description . . . 603
9.158 nvinfer1::v_1_0::IPluginV3 Class Reference . . . 603
9.158.1 Member Function Documentation . . . 603
9.158.1.1 clone() . . . 604
9.158.1.2 getCapabilityInterface() . . . 604
9.158.1.3 getInterfaceInfo() . . . 604
9.159 nvinfer1::IPluginV3Layer Class Reference . . . 605
9.159.1 Detailed Description . . . 605
9.159.2 Constructor & Destructor Documentation . . . 605
9.159.2.1 ~IPluginV3Layer() . . . 606
9.159.3 Member Function Documentation . . . 606
9.159.3.1 getPlugin() . . . 606
9.159.4 Member Data Documentation . . . 606
9.159.4.1 mImpl . . . 606
9.160 IPluginV3OneBuild Class Reference . . . 606
9.160.1 Detailed Description . . . 607
9.161 nvinfer1::v_1_0::IPluginV3OneBuild Class Reference . . . 607
9.161.1 Member Function Documentation . . . 608
9.161.1.1 configurePlugin() . . . 608
9.161.1.2 getFormatCombinationLimit() . . . 609

9.161.1.3 getInterfaceInfo() . . . 609
9.161.1.4 getMetadataString() . . . 609
9.161.1.5 getNbOutputs() . . . 610
9.161.1.6 getNbTactics() . . . 610
9.161.1.7 getOutputDataTypes() . . . 610
9.161.1.8 getOutputShapes() . . . 611
9.161.1.9 getTimingCacheID() . . . 612
9.161.1.10 getValidTactics() . . . 612
9.161.1.11 getWorkspaceSize() . . . 613
9.161.1.12 supportsFormatCombination() . . . 613
9.161.2 Member Data Documentation . . . 614
9.161.2.1 kDEFAULT_FORMAT_COMBINATION_LIMIT . . . 614
9.162 nvinfer1::v_2_0::IPluginV3OneBuild Class Reference . . . 615
9.162.1 Member Function Documentation . . . 615
9.162.1.1 getAliasedInput() . . . 615
9.162.1.2 getInterfaceInfo() . . . 616
9.163 IPluginV3OneBuildV2 Class Reference . . . 616
9.163.1 Detailed Description . . . 616
9.164 IPluginV3OneCore Class Reference . . . 617
9.164.1 Detailed Description . . . 617
9.165 nvinfer1::v_1_0::IPluginV3OneCore Class Reference . . . 617
9.165.1 Member Function Documentation . . . 618
9.165.1.1 getInterfaceInfo() . . . 618
9.165.1.2 getPluginName() . . . 618
9.165.1.3 getPluginNamespace() . . . 618
9.165.1.4 getPluginVersion() . . . 619
9.166 IPluginV3OneRuntime Class Reference . . . 619
9.166.1 Detailed Description . . . 619
9.167 nvinfer1::v_1_0::IPluginV3OneRuntime Class Reference . . . 620
9.167.1 Member Function Documentation . . . 620
9.167.1.1 attachToContext() . . . 620
9.167.1.2 enqueue() . . . 621
9.167.1.3 getFieldsToSerialize() . . . 622
9.167.1.4 getInterfaceInfo() . . . 622
9.167.1.5 onShapeChange() . . . 622
9.167.1.6 setTactic() . . . 623
9.168 nvinfer2::safe::IPluginV3OneSafeBuild Class Reference . . . 623
9.168.1 Detailed Description . . . 625
9.168.2 Constructor & Destructor Documentation . . . 625
9.168.2.1 IPluginV3OneSafeBuild() [1/3] . . . 625
9.168.2.2 IPluginV3OneSafeBuild() [2/3] . . . 625
9.168.2.3 ~IPluginV3OneSafeBuild() . . . 626
9.168.2.4 IPluginV3OneSafeBuild() [3/3] . . . 626
9.168.3 Member Function Documentation . . . 626
9.168.3.1 configurePlugin() . . . 626
9.168.3.2 getFormatCombinationLimit() . . . 627
9.168.3.3 getInterfaceInfo() . . . 627
9.168.3.4 getMetadataString() . . . 628
9.168.3.5 getNbOutputs() . . . 628
9.168.3.6 getNbTactics() . . . 628
9.168.3.7 getOutputDataTypes() . . . 628
9.168.3.8 getOutputShapes() . . . 629
9.168.3.9 getTimingCacheID() . . . 630
9.168.3.10 getValidTactics() . . . 630

9.168.3.11 getWorkspaceSize() . . . 631
9.168.3.12 operator=() [1/2] . . . 631
9.168.3.13 operator=() [2/2] . . . 632
9.168.3.14 supportsFormatCombination() . . . 632
9.168.4 Member Data Documentation . . . 633
9.168.4.1 kDEFAULT_FORMAT_COMBINATION_LIMIT . . . 633
9.169 nvinfer2::safe::IPluginV3OneSafeBuildMSS Class Reference . . . 633
9.169.1 Detailed Description . . . 634
9.169.2 Constructor & Destructor Documentation . . . 634
9.169.2.1 IPluginV3OneSafeBuildMSS() [1/3] . . . 634
9.169.2.2 IPluginV3OneSafeBuildMSS() [2/3] . . . 635
9.169.2.3 ~IPluginV3OneSafeBuildMSS() . . . 635
9.169.2.4 IPluginV3OneSafeBuildMSS() [3/3] . . . 635
9.169.3 Member Function Documentation . . . 635
9.169.3.1 getInterfaceInfo() . . . 635
9.169.3.2 getOutputShapes() . . . 635
9.169.3.3 getSymbolicOutputShapes() . . . 636
9.169.3.4 operator=() [1/2] . . . 637
9.169.3.5 operator=() [2/2] . . . 637
9.170 nvinfer2::safe::IPluginV3OneSafeCore Class Reference . . . 637
9.170.1 Detailed Description . . . 638
9.170.2 Constructor & Destructor Documentation . . . 638
9.170.2.1 IPluginV3OneSafeCore() [1/3] . . . 638
9.170.2.2 IPluginV3OneSafeCore() [2/3] . . . 639
9.170.2.3 ~IPluginV3OneSafeCore() . . . 639
9.170.2.4 IPluginV3OneSafeCore() [3/3] . . . 639
9.170.3 Member Function Documentation . . . 639
9.170.3.1 getInterfaceInfo() . . . 639
9.170.3.2 getPluginName() . . . 639
9.170.3.3 getPluginNamespace() . . . 640
9.170.3.4 getPluginVersion() . . . 640
9.170.3.5 operator=() [1/2] . . . 641
9.170.3.6 operator=() [2/2] . . . 641
9.171 nvinfer2::safe::IPluginV3OneSafeRuntime Class Reference . . . 641
9.171.1 Detailed Description . . . 642
9.171.2 Constructor & Destructor Documentation . . . 642
9.171.2.1 IPluginV3OneSafeRuntime() [1/3] . . . 643
9.171.2.2 IPluginV3OneSafeRuntime() [2/3] . . . 643
9.171.2.3 ~IPluginV3OneSafeRuntime() . . . 643
9.171.2.4 IPluginV3OneSafeRuntime() [3/3] . . . 643
9.171.3 Member Function Documentation . . . 643
9.171.3.1 clone() . . . 643
9.171.3.2 enqueue() . . . 644
9.171.3.3 getFieldsToSerialize() . . . 644
9.171.3.4 getInterfaceInfo() . . . 645
9.171.3.5 initResource() . . . 645
9.171.3.6 operator=() [1/2] . . . 645
9.171.3.7 operator=() [2/2] . . . 646
9.171.3.8 setTactic() . . . 646
9.172 nvinfer1::IPoolingLayer Class Reference . . . 646
9.172.1 Detailed Description . . . 648
9.172.2 Constructor & Destructor Documentation . . . 648
9.172.2.1 ~IPoolingLayer() . . . 648
9.172.3 Member Function Documentation . . . 648

9.172.3.1 getAverageCountExcludesPadding() . . . 648
9.172.3.2 getBlendFactor() . . . 649
9.172.3.3 getPaddingMode() . . . 649
9.172.3.4 getPaddingNd() . . . 649
9.172.3.5 getPoolingType() . . . 650
9.172.3.6 getPostPadding() . . . 650
9.172.3.7 getPrePadding() . . . 650
9.172.3.8 getStrideNd() . . . 650
9.172.3.9 getWindowSizeNd() . . . 651
9.172.3.10 setAverageCountExcludesPadding() . . . 651
9.172.3.11 setBlendFactor() . . . 651
9.172.3.12 setPaddingMode() . . . 652
9.172.3.13 setPaddingNd() . . . 652
9.172.3.14 setPoolingType() . . . 652
9.172.3.15 setPostPadding() . . . 653
9.172.3.16 setPrePadding() . . . 653
9.172.3.17 setStrideNd() . . . 653
9.172.3.18 setWindowSizeNd() . . . 654
9.172.4 Member Data Documentation . . . 654
9.172.4.1 mImpl . . . 654
9.173 IProfiler Class Reference . . . 654
9.173.1 Detailed Description . . . 654
9.174 nvinfer1::v_1_0::IProfiler Class Reference . . . 655
9.174.1 Constructor & Destructor Documentation . . . 655
9.174.1.1 ~IProfiler() . . . 655
9.174.2 Member Function Documentation . . . 655
9.174.2.1 getInterfaceInfo() . . . 655
9.174.2.2 reportLayerTime() . . . 655
9.175 IProgressMonitor Class Reference . . . 656
9.175.1 Detailed Description . . . 656
9.176 nvinfer1::v_1_0::IProgressMonitor Class Reference . . . 657
9.176.1 Constructor & Destructor Documentation . . . 657
9.176.1.1 IProgressMonitor() . . . 657
9.176.1.2 ~IProgressMonitor() . . . 657
9.176.2 Member Function Documentation . . . 658
9.176.2.1 getInterfaceInfo() . . . 658
9.176.2.2 phaseFinish() . . . 658
9.176.2.3 phaseStart() . . . 658
9.176.2.4 stepComplete() . . . 659
9.177 nvinfer1::IQuantizeLayer Class Reference . . . 660
9.177.1 Detailed Description . . . 661
9.177.2 Constructor & Destructor Documentation . . . 662
9.177.2.1 ~IQuantizeLayer() . . . 662
9.177.3 Member Function Documentation . . . 662
9.177.3.1 getAxis() . . . 662
9.177.3.2 getBlockShape() . . . 663
9.177.3.3 getToType() . . . 663
9.177.3.4 setAxis() . . . 663
9.177.3.5 setBlockShape() . . . 663
9.177.3.6 setToType() . . . 663
9.177.4 Member Data Documentation . . . 664
9.177.4.1 mImpl . . . 664
9.178 nvinfer1::IRaggedSoftMaxLayer Class Reference . . . 664
9.178.1 Detailed Description . . . 665

9.178.2 Constructor & Destructor Documentation . . . 665
9.178.2.1 ~IRaggedSoftMaxLayer() . . . 665
9.178.3 Member Data Documentation . . . 665
9.178.3.1 mImpl . . . 665
9.179 nvinfer1::IRecurrenceLayer Class Reference . . . 666
9.179.1 Detailed Description . . . 666
9.179.2 Constructor & Destructor Documentation . . . 666
9.179.2.1 ~IRecurrenceLayer() . . . 667
9.179.3 Member Function Documentation . . . 667
9.179.3.1 setInput() . . . 667
9.179.4 Member Data Documentation . . . 667
9.179.4.1 mImpl . . . 667
9.180 nvinfer1::IReduceLayer Class Reference . . . 668
9.180.1 Detailed Description . . . 668
9.180.2 Constructor & Destructor Documentation . . . 669
9.180.2.1 ~IReduceLayer() . . . 669
9.180.3 Member Function Documentation . . . 669
9.180.3.1 getKeepDimensions() . . . 669
9.180.3.2 getOperation() . . . 669
9.180.3.3 getReduceAxes() . . . 670
9.180.3.4 setKeepDimensions() . . . 670
9.180.3.5 setOperation() . . . 670
9.180.3.6 setReduceAxes() . . . 670
9.180.4 Member Data Documentation . . . 671
9.180.4.1 mImpl . . . 671
9.181 nvinfer1::IRefitter Class Reference . . . 671
9.181.1 Detailed Description . . . 672
9.181.2 Constructor & Destructor Documentation . . . 672
9.181.2.1 ~IRefitter() . . . 673
9.181.3 Member Function Documentation . . . 673
9.181.3.1 getAll() . . . 673
9.181.3.2 getAllWeights() . . . 673
9.181.3.3 getErrorRecorder() . . . 674
9.181.3.4 getLogger() . . . 674
9.181.3.5 getMaxThreads() . . . 674
9.181.3.6 getMissing() . . . 675
9.181.3.7 getMissingWeights() . . . 675
9.181.3.8 getNamedWeights() . . . 676
9.181.3.9 getWeightsLocation() . . . 676
9.181.3.10 getWeightsPrototype() . . . 677
9.181.3.11 getWeightsValidation() . . . 677
9.181.3.12 refitCudaEngine() . . . 677
9.181.3.13 refitCudaEngineAsync() . . . 677
9.181.3.14 setErrorRecorder() . . . 678
9.181.3.15 setMaxThreads() . . . 678
9.181.3.16 setNamedWeights() [1/2] . . . 679
9.181.3.17 setNamedWeights() [2/2] . . . 679
9.181.3.18 setWeights() . . . 680
9.181.3.19 setWeightsValidation() . . . 681
9.181.3.20 unsetNamedWeights() . . . 681
9.181.4 Member Data Documentation . . . 681
9.181.4.1 mImpl . . . 682
9.182 nvinfer1::IResizeLayer Class Reference . . . 682
9.182.1 Detailed Description . . . 683

9.182.2 Constructor & Destructor Documentation . . . 684
9.182.2.1 ~IResizeLayer() . . . 684
9.182.3 Member Function Documentation . . . 684
9.182.3.1 getCoordinateTransformation() . . . 684
9.182.3.2 getCubicCoeff() . . . 685
9.182.3.3 getExcludeOutside() . . . 685
9.182.3.4 getNearestRounding() . . . 685
9.182.3.5 getOutputDimensions() . . . 685
9.182.3.6 getResizeMode() . . . 686
9.182.3.7 getScales() . . . 686
9.182.3.8 getSelectorForSinglePixel() . . . 686
9.182.3.9 setCoordinateTransformation() . . . 687
9.182.3.10 setCubicCoeff() . . . 687
9.182.3.11 setExcludeOutside() . . . 687
9.182.3.12 setInput() . . . 687
9.182.3.13 setNearestRounding() . . . 688
9.182.3.14 setOutputDimensions() . . . 688
9.182.3.15 setResizeMode() . . . 689
9.182.3.16 setScales() . . . 689
9.182.3.17 setSelectorForSinglePixel() . . . 690
9.182.4 Member Data Documentation . . . 690
9.182.4.1 mImpl . . . 690
9.183 nvinfer1::IReverseSequenceLayer Class Reference . . . 691
9.183.1 Detailed Description . . . 691
9.183.2 Constructor & Destructor Documentation . . . 692
9.183.2.1 ~IReverseSequenceLayer() . . . 692
9.183.3 Member Function Documentation . . . 692
9.183.3.1 getBatchAxis() . . . 692
9.183.3.2 getSequenceAxis() . . . 692
9.183.3.3 setBatchAxis() . . . 693
9.183.3.4 setSequenceAxis() . . . 693
9.183.4 Member Data Documentation . . . 693
9.183.4.1 mImpl . . . 693
9.184 nvinfer1::IRotaryEmbeddingLayer Class Reference . . . 694
9.184.1 Detailed Description . . . 694
9.184.2 Constructor & Destructor Documentation . . . 695
9.184.2.1 ~IRotaryEmbeddingLayer() . . . 695
9.184.3 Member Function Documentation . . . 695
9.184.3.1 getInterleaved() . . . 695
9.184.3.2 getRotaryEmbeddingDim() . . . 695
9.184.3.3 setInput() . . . 695
9.184.3.4 setInterleaved() . . . 696
9.184.3.5 setRotaryEmbeddingDim() . . . 696
9.184.4 Member Data Documentation . . . 696
9.184.4.1 mImpl . . . 696
9.185 nvinfer1::IRuntime Class Reference . . . 697
9.185.1 Detailed Description . . . 698
9.185.2 Constructor & Destructor Documentation . . . 698
9.185.2.1 ~IRuntime() . . . 698
9.185.3 Member Function Documentation . . . 698
9.185.3.1 deserializeCudaEngine() [1/2] . . . 698
9.185.3.2 deserializeCudaEngine() [2/2] . . . 699
9.185.3.3 getDLACore() . . . 699
9.185.3.4 getEngineHostCodeAllowed() . . . 700

9.185.3.5 getErrorRecorder() . . . 700
9.185.3.6 getLogger() . . . 700
9.185.3.7 getMaxThreads() . . . 701
9.185.3.8 getNbDLACores() . . . 701
9.185.3.9 getPluginRegistry() . . . 701
9.185.3.10 getTempfileControlFlags() . . . 701
9.185.3.11 getTemporaryDirectory() . . . 702
9.185.3.12 loadRuntime() . . . 702
9.185.3.13 setDLACore() . . . 702
9.185.3.14 setEngineHostCodeAllowed() . . . 703
9.185.3.15 setErrorRecorder() . . . 703
9.185.3.16 setGpuAllocator() . . . 704
9.185.3.17 setMaxThreads() . . . 704
9.185.3.18 setTempfileControlFlags() . . . 704
9.185.3.19 setTemporaryDirectory() . . . 705
9.185.4 Member Data Documentation . . . 706
9.185.4.1 mImpl . . . 706
9.186 nvinfer1::IRuntimeConfig Class Reference . . . 706
9.186.1 Detailed Description . . . 707
9.186.2 Constructor & Destructor Documentation . . . 707
9.186.2.1 ~IRuntimeConfig() . . . 707
9.186.3 Member Function Documentation . . . 707
9.186.3.1 getExecutionContextAllocationStrategy() . . . 707
9.186.3.2 setExecutionContextAllocationStrategy() . . . 707
9.186.4 Member Data Documentation . . . 708
9.186.4.1 mImpl . . . 708
9.187 nvinfer2::safe::ISafeMemAllocator Class Reference . . . 708
9.187.1 Detailed Description . . . 709
9.187.2 Constructor & Destructor Documentation . . . 709
9.187.2.1 ~ISafeMemAllocator() . . . 709
9.187.2.2 ISafeMemAllocator() [1/3] . . . 709
9.187.2.3 ISafeMemAllocator() [2/3] . . . 710
9.187.2.4 ISafeMemAllocator() [3/3] . . . 710
9.187.3 Member Function Documentation . . . 710
9.187.3.1 allocate() . . . 710
9.187.3.2 deallocate() . . . 711
9.187.3.3 operator=() [1/2] . . . 712
9.187.3.4 operator=() [2/2] . . . 713
9.188 nvinfer2::safe::ISafePluginCreatorV3One Class Reference . . . 713
9.188.1 Detailed Description . . . 714
9.188.2 Constructor & Destructor Documentation . . . 714
9.188.2.1 ISafePluginCreatorV3One() [1/3] . . . 714
9.188.2.2 ~ISafePluginCreatorV3One() . . . 714
9.188.2.3 ISafePluginCreatorV3One() [2/3] . . . 714
9.188.2.4 ISafePluginCreatorV3One() [3/3] . . . 715
9.188.3 Member Function Documentation . . . 715
9.188.3.1 createPlugin() . . . 715
9.188.3.2 getFieldNames() . . . 715
9.188.3.3 getInterfaceInfo() . . . 716
9.188.3.4 getPluginName() . . . 716
9.188.3.5 getPluginNamespace() . . . 716
9.188.3.6 getPluginVersion() . . . 717
9.188.3.7 getSafeRecorder() . . . 717
9.188.3.8 operator=() [1/2] . . . 717

9.188.3.9 operator=() [2/2] . . . 718
9.188.3.10 setSafeRecorder() . . . 718
9.189 nvinfer2::safe::ISafePluginRegistry Class Reference . . . 718
9.189.1 Detailed Description . . . 719
9.189.2 Constructor & Destructor Documentation . . . 719
9.189.2.1 ISafePluginRegistry() [1/3] . . . 719
9.189.2.2 ISafePluginRegistry() [2/3] . . . 720
9.189.2.3 ISafePluginRegistry() [3/3] . . . 720
9.189.2.4 ~ISafePluginRegistry() . . . 720
9.189.3 Member Function Documentation . . . 720
9.189.3.1 deregisterCreator() . . . 720
9.189.3.2 getAllCreators() . . . 721
9.189.3.3 getCreator() . . . 722
9.189.3.4 getSafeRecorder() . . . 722
9.189.3.5 operator=() [1/2] . . . 723
9.189.3.6 operator=() [2/2] . . . 724
9.189.3.7 registerCreator() . . . 724
9.189.3.8 setSafeRecorder() . . . 725
9.190 nvinfer2::safe::ISafePluginResourceContext Class Reference . . . 726
9.190.1 Detailed Description . . . 726
9.190.2 Constructor & Destructor Documentation . . . 727
9.190.2.1 ~ISafePluginResourceContext() . . . 727
9.190.2.2 ISafePluginResourceContext() [1/3] . . . 727
9.190.2.3 ISafePluginResourceContext() [2/3] . . . 728
9.190.2.4 ISafePluginResourceContext() [3/3] . . . 728
9.190.3 Member Function Documentation . . . 728
9.190.3.1 getRuntimeErrorInformation() . . . 728
9.190.3.2 getSafeMemAllocator() . . . 729
9.190.3.3 getSafeRecorder() . . . 730
9.190.3.4 operator=() [1/2] . . . 730
9.190.3.5 operator=() [2/2] . . . 731
9.191 nvinfer2::safe::ISafeRecorder Class Reference . . . 731
9.191.1 Detailed Description . . . 733
9.191.2 Constructor & Destructor Documentation . . . 733
9.191.2.1 ISafeRecorder() [1/3] . . . 733
9.191.2.2 ISafeRecorder() [2/3] . . . 733
9.191.2.3 ISafeRecorder() [3/3] . . . 734
9.191.2.4 ~ISafeRecorder() . . . 734
9.191.3 Member Function Documentation . . . 734
9.191.3.1 getId() . . . 735
9.191.3.2 getSeverity() . . . 735
9.191.3.3 operator=() [1/2] . . . 736
9.191.3.4 operator=() [2/2] . . . 736
9.191.3.5 reportDebug() . . . 736
9.191.3.6 reportInfo() . . . 737
9.191.3.7 reportVerbose() . . . 738
9.191.3.8 reportWarn() . . . 738
9.191.3.9 setSeverity() . . . 739
9.191.4 Member Data Documentation . . . 739
9.191.4.1 kMAX_SAFE_DESC_LENGTH . . . 740
9.191.4.2 mId . . . 740
9.191.4.3 mSeverity . . . 740
9.192 nvinfer1::IScaleLayer Class Reference . . . 740
9.192.1 Detailed Description . . . 741

9.192.2 Constructor & Destructor Documentation . . . 742
9.192.2.1 ~IScaleLayer() . . . 742
9.192.3 Member Function Documentation . . . 742
9.192.3.1 getChannelAxis() . . . 742
9.192.3.2 getMode() . . . 743
9.192.3.3 getPower() . . . 743
9.192.3.4 getScale() . . . 743
9.192.3.5 getShift() . . . 743
9.192.3.6 setChannelAxis() . . . 744
9.192.3.7 setMode() . . . 744
9.192.3.8 setPower() . . . 744
9.192.3.9 setScale() . . . 745
9.192.3.10 setShift() . . . 745
9.192.4 Member Data Documentation . . . 745
9.192.4.1 mImpl . . . 745
9.193 nvinfer1::IScatterLayer Class Reference . . . 745
9.193.1 Detailed Description . . . 746
9.193.2 Constructor & Destructor Documentation . . . 747
9.193.2.1 ~IScatterLayer() . . . 747
9.193.3 Member Function Documentation . . . 747
9.193.3.1 getAxis() . . . 747
9.193.3.2 getMode() . . . 748
9.193.3.3 setAxis() . . . 748
9.193.3.4 setMode() . . . 748
9.193.4 Member Data Documentation . . . 748
9.193.4.1 mImpl . . . 748
9.194 nvinfer1::ISelectLayer Class Reference . . . 749
9.194.1 Detailed Description . . . 749
9.194.2 Constructor & Destructor Documentation . . . 749
9.194.2.1 ~ISelectLayer() . . . 749
9.194.3 Member Data Documentation . . . 750
9.194.3.1 mImpl . . . 750
9.195 nvinfer1::ISerializationConfig Class Reference . . . 750
9.195.1 Detailed Description . . . 751
9.195.2 Constructor & Destructor Documentation . . . 751
9.195.2.1 ~ISerializationConfig() . . . 751
9.195.3 Member Function Documentation . . . 751
9.195.3.1 clearFlag() . . . 751
9.195.3.2 getFlag() . . . 752
9.195.3.3 getFlags() . . . 752
9.195.3.4 setFlag() . . . 752
9.195.3.5 setFlags() . . . 752
9.195.4 Member Data Documentation . . . 753
9.195.4.1 mImpl . . . 753
9.196 nvinfer1::IShapeLayer Class Reference . . . 753
9.196.1 Detailed Description . . . 754
9.196.2 Constructor & Destructor Documentation . . . 754
9.196.2.1 ~IShapeLayer() . . . 754
9.196.3 Member Data Documentation . . . 754
9.196.3.1 mImpl . . . 754
9.197 nvinfer1::IShuffleLayer Class Reference . . . 755
9.197.1 Detailed Description . . . 756
9.197.2 Constructor & Destructor Documentation . . . 756
9.197.2.1 ~IShuffleLayer() . . . 756

9.197.3 Member Function Documentation . . . 756
9.197.3.1 getFirstTranspose() . . . 756
9.197.3.2 getReshapeDimensions() . . . 757
9.197.3.3 getSecondTranspose() . . . 757
9.197.3.4 getZeroIsPlaceholder() . . . 757
9.197.3.5 setFirstTranspose() . . . 757
9.197.3.6 setInput() . . . 758
9.197.3.7 setReshapeDimensions() . . . 759
9.197.3.8 setSecondTranspose() . . . 759
9.197.3.9 setZeroIsPlaceholder() . . . 760
9.197.4 Member Data Documentation . . . 760
9.197.4.1 mImpl . . . 760
9.198 nvinfer1::ISliceLayer Class Reference . . . 760
9.198.1 Detailed Description . . . 761
9.198.2 Constructor & Destructor Documentation . . . 762
9.198.2.1 ~ISliceLayer() . . . 763
9.198.3 Member Function Documentation . . . 763
9.198.3.1 getAxes() . . . 763
9.198.3.2 getMode() . . . 763
9.198.3.3 getSize() . . . 764
9.198.3.4 getStart() . . . 764
9.198.3.5 getStride() . . . 764
9.198.3.6 setAxes() . . . 764
9.198.3.7 setInput() . . . 765
9.198.3.8 setMode() . . . 766
9.198.3.9 setSize() . . . 766
9.198.3.10 setStart() . . . 766
9.198.3.11 setStride() . . . 767
9.198.4 Member Data Documentation . . . 767
9.198.4.1 mImpl . . . 767
9.199 nvinfer1::ISoftMaxLayer Class Reference . . . 767
9.199.1 Detailed Description . . . 768
9.199.2 Constructor & Destructor Documentation . . . 768
9.199.2.1 ~ISoftMaxLayer() . . . 768
9.199.3 Member Function Documentation . . . 769
9.199.3.1 getAxes() . . . 769
9.199.3.2 setAxes() . . . 769
9.199.4 Member Data Documentation . . . 769
9.199.4.1 mImpl . . . 769
9.200 nvinfer1::ISqueezeLayer Class Reference . . . 770
9.200.1 Detailed Description . . . 770
9.200.2 Constructor & Destructor Documentation . . . 770
9.200.2.1 ~ISqueezeLayer() . . . 771
9.200.3 Member Function Documentation . . . 771
9.200.3.1 setInput() . . . 771
9.200.4 Member Data Documentation . . . 771
9.200.4.1 mImpl . . . 771
9.201 IStreamReader Class Reference . . . 772
9.201.1 Detailed Description . . . 772
9.202 nvinfer1::v_1_0::IStreamReader Class Reference . . . 772
9.202.1 Detailed Description . . . 773
9.202.2 Constructor & Destructor Documentation . . . 773
9.202.2.1 ~IStreamReader() . . . 773
9.202.2.2 IStreamReader() [1/3] . . . 773

9.202.2.3 IStreamReader() [2/3] . . . 773
9.202.2.4 IStreamReader() [3/3] . . . 773
9.202.3 Member Function Documentation . . . 773
9.202.3.1 getInterfaceInfo() . . . 774
9.202.3.2 operator=() [1/2] . . . 774
9.202.3.3 operator=() [2/2] . . . 774
9.202.3.4 read() . . . 774
9.203 IStreamReaderV2 Class Reference . . . 775
9.203.1 Detailed Description . . . 775
9.204 nvinfer1::v_1_0::IStreamReaderV2 Class Reference . . . 775
9.204.1 Constructor & Destructor Documentation . . . 776
9.204.1.1 ~IStreamReaderV2() . . . 776
9.204.1.2 IStreamReaderV2() [1/3] . . . 776
9.204.1.3 IStreamReaderV2() [2/3] . . . 776
9.204.1.4 IStreamReaderV2() [3/3] . . . 776
9.204.2 Member Function Documentation . . . 776
9.204.2.1 getInterfaceInfo() . . . 777
9.204.2.2 operator=() [1/2] . . . 777
9.204.2.3 operator=() [2/2] . . . 777
9.204.2.4 read() . . . 777
9.204.2.5 seek() . . . 778
9.205 IStreamWriter Class Reference . . . 778
9.205.1 Detailed Description . . . 778
9.206 nvinfer1::v_1_0::IStreamWriter Class Reference . . . 779
9.206.1 Constructor & Destructor Documentation . . . 779
9.206.1.1 ~IStreamWriter() . . . 779
9.206.1.2 IStreamWriter() [1/3] . . . 779
9.206.1.3 IStreamWriter() [2/3] . . . 780
9.206.1.4 IStreamWriter() [3/3] . . . 780
9.206.2 Member Function Documentation . . . 780
9.206.2.1 getInterfaceInfo() . . . 780
9.206.2.2 operator=() [1/2] . . . 780
9.206.2.3 operator=() [2/2] . . . 780
9.206.2.4 write() . . . 780
9.207 nvinfer1::ITensor Class Reference . . . 781
9.207.1 Detailed Description . . . 782
9.207.2 Constructor & Destructor Documentation . . . 782
9.207.2.1 ~ITensor() . . . 783
9.207.3 Member Function Documentation . . . 783
9.207.3.1 getAllowedFormats() . . . 783
9.207.3.2 getBroadcastAcrossBatch() . . . 783
9.207.3.3 getDimensionName() . . . 783
9.207.3.4 getDimensions() . . . 784
9.207.3.5 getLocation() . . . 784
9.207.3.6 getName() . . . 785
9.207.3.7 getType() . . . 785
9.207.3.8 isExecutionTensor() . . . 785
9.207.3.9 isNetworkInput() . . . 786
9.207.3.10 isNetworkOutput() . . . 786
9.207.3.11 isShapeTensor() . . . 786
9.207.3.12 setAllowedFormats() . . . 787
9.207.3.13 setBroadcastAcrossBatch() . . . 788
9.207.3.14 setDimensionName() . . . 788
9.207.3.15 setDimensions() . . . 790

9.207.3.16 setLocation() . . . 790
9.207.3.17 setName() . . . 791
9.207.4 Member Data Documentation . . . 791
9.207.4.1 mImpl . . . 792
9.208 nvinfer1::ITimingCache Class Reference . . . 792
9.208.1 Detailed Description . . . 792
9.208.2 Constructor & Destructor Documentation . . . 793
9.208.2.1 ~ITimingCache() . . . 793
9.208.3 Member Function Documentation . . . 793
9.208.3.1 combine() . . . 793
9.208.3.2 query() . . . 794
9.208.3.3 queryKeys() . . . 794
9.208.3.4 reset() . . . 795
9.208.3.5 serialize() . . . 795
9.208.3.6 update() . . . 795
9.208.4 Member Data Documentation . . . 796
9.208.4.1 mImpl . . . 796
9.209 nvinfer1::ITopKLayer Class Reference . . . 796
9.209.1 Detailed Description . . . 797
9.209.2 Constructor & Destructor Documentation . . . 797
9.209.2.1 ~ITopKLayer() . . . 798
9.209.3 Member Function Documentation . . . 798
9.209.3.1 getIndicesType() . . . 798
9.209.3.2 getK() . . . 798
9.209.3.3 getOperation() . . . 798
9.209.3.4 getReduceAxes() . . . 799
9.209.3.5 setIndicesType() . . . 799
9.209.3.6 setInput() . . . 799
9.209.3.7 setK() . . . 800
9.209.3.8 setOperation() . . . 800
9.209.3.9 setReduceAxes() . . . 800
9.209.4 Member Data Documentation . . . 801
9.209.4.1 mImpl . . . 801
9.210 nvinfer1::ITripLimitLayer Class Reference . . . 801
9.210.1 Detailed Description . . . 802
9.210.2 Constructor & Destructor Documentation . . . 802
9.210.2.1 ~ITripLimitLayer() . . . 802
9.210.3 Member Function Documentation . . . 802
9.210.3.1 getTripLimit() . . . 802
9.210.4 Member Data Documentation . . . 802
9.210.4.1 mImpl . . . 802
9.211 nvinfer2::safe::ITRTGraph Class Reference . . . 803
9.211.1 Detailed Description . . . 804
9.211.2 Constructor & Destructor Documentation . . . 804
9.211.2.1 ITRTGraph() [1/3] . . . 805
9.211.2.2 ITRTGraph() [2/3] . . . 805
9.211.2.3 ~ITRTGraph() . . . 805
9.211.2.4 ITRTGraph() [3/3] . . . 805
9.211.3 Member Function Documentation . . . 805
9.211.3.1 clone() . . . 805
9.211.3.2 executeAsync() . . . 806
9.211.3.3 getErrorBuffer() . . . 806
9.211.3.4 getInputConsumedEvent() . . . 807
9.211.3.5 getIOProfile() . . . 807

9.211.3.6 getIOTensorAddress() [1/2] . . . 808
9.211.3.7 getIOTensorAddress() [2/2] . . . 808
9.211.3.8 getIOTensorDescriptor() [1/2] . . . 809
9.211.3.9 getIOTensorDescriptor() [2/2] . . . 809
9.211.3.10 getIOTensorName() . . . 810
9.211.3.11 getNbAuxStreams() . . . 810
9.211.3.12 getNbIOProfiles() . . . 811
9.211.3.13 getNbIOTensors() . . . 811
9.211.3.14 getSafeRecorder() . . . 812
9.211.3.15 getScratchMemory() . . . 812
9.211.3.16 getScratchMemorySize() . . . 813
9.211.3.17 getTRTManagedScratch() . . . 813
9.211.3.18 operator=() [1/2] . . . 814
9.211.3.19 operator=() [2/2] . . . 814
9.211.3.20 setAuxStreams() . . . 814
9.211.3.21 setInputConsumedEvent() . . . 815
9.211.3.22 setIOProfile() . . . 815
9.211.3.23 setIOTensorAddress() [1/2] . . . 816
9.211.3.24 setIOTensorAddress() [2/2] . . . 816
9.211.3.25 setScratchMemory() . . . 817
9.211.3.26 sync() . . . 818
9.212 nvinfer1::IUnaryLayer Class Reference . . . 818
9.212.1 Detailed Description . . . 819
9.212.2 Constructor & Destructor Documentation . . . 819
9.212.2.1 ~IUnaryLayer() . . . 819
9.212.3 Member Function Documentation . . . 819
9.212.3.1 getOperation() . . . 819
9.212.3.2 setOperation() . . . 820
9.212.4 Member Data Documentation . . . 820
9.212.4.1 mImpl . . . 820
9.213 nvinfer1::IUnsqueezeLayer Class Reference . . . 820
9.213.1 Detailed Description . . . 821
9.213.2 Constructor & Destructor Documentation . . . 821
9.213.2.1 ~IUnsqueezeLayer() . . . 821
9.213.3 Member Function Documentation . . . 821
9.213.3.1 setInput() . . . 821
9.213.4 Member Data Documentation . . . 822
9.213.4.1 mImpl . . . 822
9.214 nvinfer1::IVersionedInterface Class Reference . . . 822
9.214.1 Detailed Description . . . 823
9.214.2 Constructor & Destructor Documentation . . . 824
9.214.2.1 ~IVersionedInterface() . . . 824
9.214.2.2 IVersionedInterface() [1/3] . . . 824
9.214.2.3 IVersionedInterface() [2/3] . . . 824
9.214.2.4 IVersionedInterface() [3/3] . . . 824
9.214.3 Member Function Documentation . . . 824
9.214.3.1 getAPILanguage() . . . 824
9.214.3.2 getInterfaceInfo() . . . 825
9.214.3.3 operator=() [1/2] . . . 825
9.214.3.4 operator=() [2/2] . . . 825
9.215 nvinfer1::Permutation Struct Reference . . . 825
9.215.1 Detailed Description . . . 826
9.215.2 Member Data Documentation . . . 826
9.215.2.1 order . . . 826

9.216 nvinfer2::safe::PhysicalDims Class Reference . . . 826
9.216.1 Detailed Description . . . 827
9.216.2 Member Data Documentation . . . 827
9.216.2.1 d . . . 827
9.216.2.2 MAX_DIMS . . . 827
9.216.2.3 nbDims . . . 827
9.217 nvinfer1::PluginField Class Reference . . . 827
9.217.1 Detailed Description . . . 828
9.217.2 Constructor & Destructor Documentation . . . 828
9.217.2.1 PluginField() . . . 828
9.217.3 Member Data Documentation . . . 828
9.217.3.1 data . . . 828
9.217.3.2 length . . . 829
9.217.3.3 name . . . 829
9.217.3.4 type . . . 829
9.218 nvinfer1::PluginFieldCollection Struct Reference . . . 829
9.218.1 Detailed Description . . . 829
9.218.2 Member Data Documentation . . . 830
9.218.2.1 fields . . . 830
9.218.2.2 nbFields . . . 830
9.219 nvinfer1::PluginRegistrar< T > Class Template Reference . . . 830
9.219.1 Detailed Description . . . 830
9.219.2 Constructor & Destructor Documentation . . . 831
9.219.2.1 PluginRegistrar() . . . 831
9.220 nvinfer1::PluginTensorDesc Struct Reference . . . 831
9.220.1 Detailed Description . . . 831
9.220.2 Member Data Documentation . . . 832
9.220.2.1 dims . . . 832
9.220.2.2 format . . . 832
9.220.2.3 scale . . . 832
9.220.2.4 type . . . 832
9.221 PluginVersion Struct Reference . . . 832
9.221.1 Detailed Description . . . 833
9.222 nvinfer1::plugin::PriorBoxParameters Struct Reference . . . 833
9.222.1 Detailed Description . . . 834
9.222.2 Member Data Documentation . . . 834
9.222.2.1 aspectRatios . . . 834
9.222.2.2 clip . . . 834
9.222.2.3 flip . . . 834
9.222.2.4 imgH . . . 834
9.222.2.5 imgW . . . 835
9.222.2.6 maxSize . . . 835
9.222.2.7 minSize . . . 835
9.222.2.8 numAspectRatios . . . 835
9.222.2.9 numMaxSize . . . 835
9.222.2.10 numMinSize . . . 835
9.222.2.11 offset . . . 836
9.222.2.12 stepH . . . 836
9.222.2.13 stepW . . . 836
9.222.2.14 variance . . . 836
9.223 nvinfer1::plugin::RegionParameters Struct Reference . . . 836
9.223.1 Detailed Description . . . 837
9.223.2 Member Data Documentation . . . 837
9.223.2.1 classes . . . 837

9.223.2.2 coords . . . 837
9.223.2.3 num . . . 837
9.223.2.4 smTree . . . 838
9.224 nvinfer1::plugin::RPROIParams Struct Reference . . . 838
9.224.1 Detailed Description . . . 838
9.224.2 Member Data Documentation . . . 839
9.224.2.1 anchorsRatioCount . . . 839
9.224.2.2 anchorsScaleCount . . . 839
9.224.2.3 featureStride . . . 839
9.224.2.4 iouThreshold . . . 839
9.224.2.5 minBoxSize . . . 839
9.224.2.6 nmsMaxOut . . . 840
9.224.2.7 poolingH . . . 840
9.224.2.8 poolingW . . . 840
9.224.2.9 preNmsTop . . . 840
9.224.2.10 spatialScale . . . 840
9.225 nvinfer2::safe::RuntimeErrorInformation Struct Reference . . . 841
9.225.1 Detailed Description . . . 841
9.225.2 Member Data Documentation . . . 841
9.225.2.1 bitMask . . . 841
9.226 nvinfer1::plugin::softmaxTree Struct Reference . . . 841
9.226.1 Detailed Description . . . 842
9.226.2 Member Data Documentation . . . 842
9.226.2.1 child . . . 842
9.226.2.2 group . . . 842
9.226.2.3 groupOffset . . . 842
9.226.2.4 groups . . . 842
9.226.2.5 groupSize . . . 843
9.226.2.6 leaf . . . 843
9.226.2.7 n . . . 843
9.226.2.8 name . . . 843
9.226.2.9 parent . . . 843
9.227 nvinfer2::safe::TensorDescriptor Struct Reference . . . 843
9.227.1 Detailed Description . . . 844
9.227.2 Member Data Documentation . . . 844
9.227.2.1 bytesPerComponent . . . 844
9.227.2.2 componentsPerVector . . . 845
9.227.2.3 dataType . . . 845
9.227.2.4 ioMode . . . 845
9.227.2.5 memPlacement . . . 845
9.227.2.6 shape . . . 845
9.227.2.7 sizeInBytes . . . 846
9.227.2.8 stride . . . 846
9.227.2.9 strideOrder . . . 846
9.227.2.10 tensorName . . . 846
9.227.2.11 userShape . . . 846
9.227.2.12 vectorizedDim . . . 846
9.228 nvinfer1::v_1_0::TimingCacheKey Struct Reference . . . 847
9.228.1 Detailed Description . . . 847
9.228.2 Member Data Documentation . . . 847
9.228.2.1 data . . . 847
9.229 nvinfer1::v_1_0::TimingCacheValue Struct Reference . . . 847
9.229.1 Member Data Documentation . . . 848
9.229.1.1 kINVALID_TACTIC_HASH . . . 848

9.229.1.2 tacticHash . . . 848
9.229.1.3 timingMSec . . . 848
9.230 nvinfer2::safe::TypedArray Class Reference . . . 849
9.230.1 Detailed Description . . . 850
9.230.2 Constructor & Destructor Documentation . . . 850
9.230.2.1 TypedArray() [1/10] . . . 850
9.230.2.2 TypedArray() [2/10] . . . 850
9.230.2.3 TypedArray() [3/10] . . . 850
9.230.2.4 ~TypedArray() . . . 850
9.230.2.5 TypedArray() [4/10] . . . 851
9.230.2.6 TypedArray() [5/10] . . . 851
9.230.2.7 TypedArray() [6/10] . . . 851
9.230.2.8 TypedArray() [7/10] . . . 851
9.230.2.9 TypedArray() [8/10] . . . 851
9.230.2.10 TypedArray() [9/10] . . . 852
9.230.2.11 TypedArray() [10/10] . . . 852
9.230.3 Member Function Documentation . . . 852
9.230.3.1 getBool() . . . 852
9.230.3.2 getData() . . . 852
9.230.3.3 getFloat() . . . 853
9.230.3.4 getHalf() . . . 853
9.230.3.5 getInt32() . . . 853
9.230.3.6 getInt64() . . . 853
9.230.3.7 getInt8() . . . 853
9.230.3.8 getSize() . . . 853
9.230.3.9 getType() . . . 854
9.230.3.10 getUint8() . . . 854
9.230.3.11 operator=() [1/2] . . . 854
9.230.3.12 operator=() [2/2] . . . 854
9.231 Value Struct Reference . . . 854
9.231.1 Detailed Description . . . 855
9.232 nvinfer1::Weights Class Reference . . . 855
9.232.1 Detailed Description . . . 855
9.232.2 Member Data Documentation . . . 855
9.232.2.1 count . . . 856
9.232.2.2 type . . . 856
9.232.2.3 values . . . 856

10 File Documentation **857**
10.1 NvInfer.h File Reference . . . 857
10.1.1 Detailed Description . . . 865
10.2 NvInfer.h . . . 865
10.3 NvInferImpl.h File Reference . . . 916
10.4 NvInferImpl.h . . . 916
10.5 NvInferLegacyDims.h File Reference . . . 934
10.5.1 Detailed Description . . . 934
10.5.2 Macro Definition Documentation . . . 934
10.5.2.1 NV_INFER_INTERNAL_INCLUDE . . . 935
10.6 NvInferLegacyDims.h . . . 935
10.7 NvInferPlugin.h File Reference . . . 936
10.7.1 Detailed Description . . . 936
10.7.2 Function Documentation . . . 937
10.7.2.1 initLibNvInferPlugins() . . . 937
10.8 NvInferPlugin.h . . . 937

10.9 NvInferPluginBase.h File Reference . . . 937
10.9.1 Macro Definition Documentation . . . 939
10.9.1.1 NV_INFER_INTERNAL_INCLUDE . . . 939
10.10 NvInferPluginBase.h . . . 939
10.11 NvInferPluginUtils.h File Reference . . . 941
10.11.1 Detailed Description . . . 941
10.12 NvInferPluginUtils.h . . . 942
10.13 NvInferRuntime.h File Reference . . . 943
10.13.1 Detailed Description . . . 947
10.13.2 Macro Definition Documentation . . . 947
10.13.2.1 NV_INFER_INTERNAL_INCLUDE . . . 948
10.13.2.2 REGISTER_TENSORRT_PLUGIN . . . 948
10.13.3 Function Documentation . . . 948
10.13.3.1 getInferLibBuildVersion() . . . 948
10.13.3.2 getInferLibMajorVersion() . . . 948
10.13.3.3 getInferLibMinorVersion() . . . 948
10.13.3.4 getInferLibPatchVersion() . . . 948
10.13.3.5 getLogger() . . . 949
10.13.3.6 getPluginRegistry() . . . 949
10.14 NvInferRuntime.h . . . 949
10.15 NvInferRuntimeBase.h File Reference . . . 972
10.15.1 Detailed Description . . . 974
10.15.2 Macro Definition Documentation . . . 974
10.15.2.1 NV_TENSORRT_VERSION . . . 974
10.15.2.2 NV_TENSORRT_VERSION_INT . . . 975
10.15.2.3 TENSORRTAPI . . . 975
10.15.2.4 TRT_DEPRECATED . . . 975
10.15.2.5 TRT_DEPRECATED_API . . . 975
10.15.2.6 TRT_DEPRECATED_BECAUSE . . . 975
10.15.2.7 TRT_DEPRECATED_ENUM . . . 975
10.15.2.8 TRT_NODISCARD . . . 976
10.15.2.9 TRT_NODISCARD_BECAUSE . . . 976
10.15.2.10 TRTNOEXCEPT . . . 976
10.15.3 Function Documentation . . . 976
10.15.3.1 getInferLibVersion() . . . 976
10.16 NvInferRuntimeBase.h . . . 977
10.17 NvInferRuntimeCommon.h File Reference . . . 981
10.17.1 Detailed Description . . . 981
10.17.2 Macro Definition Documentation . . . 981
10.17.2.1 NV_INFER_INTERNAL_INCLUDE . . . 981
10.18 NvInferRuntimeCommon.h . . . 982
10.19 NvInferRuntimePlugin.h File Reference . . . 983
10.19.1 Detailed Description . . . 984
10.19.2 Macro Definition Documentation . . . 984
10.19.2.1 NV_INFER_INTERNAL_INCLUDE . . . 984
10.20 NvInferRuntimePlugin.h . . . 984
10.21 NvInferVersion.h File Reference . . . 987
10.21.1 Detailed Description . . . 988
10.21.2 Macro Definition Documentation . . . 988
10.21.2.1 NV_TENSORRT_BUILD . . . 988
10.21.2.2 NV_TENSORRT_LWS_MAJOR . . . 989
10.21.2.3 NV_TENSORRT_LWS_MINOR . . . 989
10.21.2.4 NV_TENSORRT_LWS_PATCH . . . 989
10.21.2.5 NV_TENSORRT_MAJOR . . . 989

10.21.2.6 NV_TENSORRT_MINOR . . . 989
10.21.2.7 NV_TENSORRT_PATCH . . . 989
10.21.2.8 NV_TENSORRT_RELEASE_TYPE . . . 990
10.21.2.9 NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS . . . 990
10.21.2.10 NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY . . . 990
10.21.2.11 NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE . . . 990
10.21.2.12 TRT_BUILD_ENTERPRISE . . . 990
10.21.2.13 TRT_MAJOR_ENTERPRISE . . . 990
10.21.2.14 TRT_MINOR_ENTERPRISE . . . 991
10.21.2.15 TRT_PATCH_ENTERPRISE . . . 991
10.22 NvInferVersion.h . . . 991
10.23 NvOnnxConfig.h File Reference . . . 991
10.23.1 Detailed Description . . . 992
10.24 NvOnnxConfig.h . . . 992
10.25 NvOnnxParser.h File Reference . . . 993
10.25.1 Detailed Description . . . 994
10.25.2 Macro Definition Documentation . . . 995
10.25.2.1 NV_ONNX_PARSER_MAJOR . . . 995
10.25.2.2 NV_ONNX_PARSER_MINOR . . . 995
10.25.2.3 NV_ONNX_PARSER_PATCH . . . 995
10.25.3 Function Documentation . . . 995
10.25.3.1 createNvOnnxParser_INTERNAL() . . . 995
10.25.3.2 createNvOnnxParserRefitter_INTERNAL() . . . 995
10.25.3.3 getNvOnnxParserVersion() . . . 996
10.26 NvOnnxParser.h . . . 996
10.27 NvInferConsistency.h File Reference . . . 998
10.28 NvInferConsistency.h . . . 999
10.29 NvInferForwardDecl.h File Reference . . . 1000
10.29.1 Macro Definition Documentation . . . 1001
10.29.1.1 NV_INFER_INTERNAL_INCLUDE . . . 1001
10.30 NvInferForwardDecl.h . . . 1001
10.31 NvInferSafeMemAllocator.h File Reference . . . 1002
10.32 NvInferSafeMemAllocator.h . . . 1003
10.33 NvInferSafePlugin.h File Reference . . . 1004
10.34 NvInferSafePlugin.h . . . 1005
10.35 NvInferSafeRecorder.h File Reference . . . 1009
10.36 NvInferSafeRecorder.h . . . 1010
10.37 NvInferSafeRuntime.h File Reference . . . 1012
10.38 NvInferSafeRuntime.h . . . 1013

Index **1017**

Chapter 1

Standard and Safe

This document covers both the standard TensorRT release and the safe TensorRT release. Interfaces supported in the safe runtime are exclusively defined in the following interface files:

NvInferConsistency.h . . . 998
NvInferForwardDecl.h . . . 1000
NvInferPluginBase.h . . . 937
NvInferRuntimeBase.h . . . 972
NvInferSafeMemAllocator.h . . . 1002
NvInferSafePlugin.h . . . 1004
NvInferSafeRecorder.h . . . 1009
NvInferSafeRuntime.h . . . 1012
NvInferVersion.h . . . 987

See the *TensorRT Safety Developer Guide Supplement* for more details on the Standard/Safe split. The safety runtime is designed to support automotive Safety Integrity Level B (ASIL-B) for safety flows. Applications using the safety runtime must follow the *NVIDIA DriveOS 7.2 Safety Manual*. The safety runtime makes use of CUDA® and is designed to work with applications using CUDA. The *NVIDIA Deep Learning Inference SEooC 2.2 Safety Manual* is the authoritative source for all usage restrictions, assumptions, and integration conditions. All adapted restrictions for the certified state of TensorRT are documented in the Safety Manual, and users are required to follow those instructions when deploying the safety runtime in ASIL-rated systems.

Chapter 2

TensorRT

This is the API documentation for the NVIDIA TensorRT library. It provides information on individual functions, classes and methods. Use the index on the left to navigate the documentation.

Please see the accompanying user guide and samples for higher-level information and general advice on using TensorRT.

TensorRT Versioning follows Semantic Versioning Guidelines specified here: https://semver.org/

Chapter 3

Deprecated List

Class IPluginCreator

Deprecated in TensorRT 10.0. Please implement IPluginCreatorV3One along with IPluginV3 plugins instead.

Class IStreamReader

Deprecated in TensorRT 11.0. Superseded by IStreamReaderV2.

Member nvinfer1::IAttention::getCausal () const noexcept

Deprecated in TensorRT 10.16. Superseded by getCausalKind.

Member nvinfer1::IAttention::setCausal (bool isCausal) noexcept

Deprecated in TensorRT 10.16. Superseded by setCausalKind.

Member nvinfer1::IExecutionContext::setDeviceMemory (void ∗memory) noexcept

Deprecated in TensorRT 10.1. Superseded by setDeviceMemoryV2().

Member nvinfer1::INetworkDefinition::addAttention (ITensor &query, ITensor &key, ITensor &value, AttentionNormalizationOp normOp, bool causal) noexcept

Deprecated in TensorRT 10.16. Superseded by addAttentionV2 with CausalMaskKind parameter.

Member nvinfer1::INetworkDefinition::addNMS (ITensor &boxes, ITensor &scores, ITensor &maxOutput↩ BoxesPerClass) noexcept

Deprecated in TensorRT 10.14. Superseded by four-argument addNMS.

Member nvinfer1::INetworkDefinition::addNonZero (ITensor &input) noexcept

Deprecated in TensorRT 10.14. Superseded by two-argument addNonZero.

Member nvinfer1::INetworkDefinition::addNormalization (ITensor &input, ITensor &scale, ITensor &bias, uint32_t axesMask) noexcept

Deprecated in TensorRT 10.15. Superseded by addNormalizationV2().

Member nvinfer1::INetworkDefinition::addPluginV2 (ITensor ∗const ∗inputs, int32_t nbInputs, IPluginV2 &plugin) noexcept

Deprecated in TensorRT 10.8. Superseded by addPluginV3.

Member nvinfer1::INetworkDefinition::addTopK (ITensor &input, TopKOperation op, int32_t k, uint32_↩ t reduceAxes) noexcept

Deprecated in TensorRT 10.14. Superseded by five-argument addTopK.

Member nvinfer1::INetworkDefinition::hasImplicitBatchDimension () const noexcept

Deprecated in TensorRT 10.0. Implicit batch is not supported since TensorRT 10.0.

Class nvinfer1::IPluginV2

Deprecated in TensorRT 8.5. Implement IPluginV3 instead.

Class nvinfer1::IPluginV2DynamicExt

Deprecated in TensorRT 10.0. Please implement IPluginV3 instead.

Class nvinfer1::IPluginV2Ext

Deprecated in TensorRT 8.5. Implement IPluginV3 instead.

Class nvinfer1::IPluginV2IOExt

Deprecated in TensorRT 10.0. Implement IPluginV3 instead.

Class nvinfer1::IPluginV2Layer

Deprecated in TensorRT 10.8. Superseded by IPluginV3Layer.

Member nvinfer1::ITensor::getBroadcastAcrossBatch () const noexcept

Deprecated in TensorRT 10.0. Implicit batch is not supported since TensorRT 10.0.

Member nvinfer1::ITensor::setBroadcastAcrossBatch (bool broadcastAcrossBatch) noexcept

Deprecated in TensorRT 10.0. Implicit batch is not supported since TensorRT 10.0.

Member nvinfer1::ITensor::setLocation (TensorLocation location) noexcept

Deprecated in TensorRT 10.0. RNNv2 is not supported and the location must always be TensorLocation::kDEVICE since TensorRT 10.0.

Member nvinfer1::kDIRECT_IO

Deprecated in TensorRT 10.7. Unneeded API.

Member nvinfer1::kSAFETY_SCOPE

Deprecated in TensorRT 10.16. In EngineCapability::kSTANDARD flow, safety restrictions are no longer supported. In EngineCapability::kSAFETY and EngineCapability::kDLA_STANDALONE flows, restrictions are enforced natively. This flag is retained for API compatibility but is ignored.

Member nvinfer1::kSTRONGLY_TYPED

Deprecated in TensorRT 11.0. Strongly typed mode is always enabled. This flag is retained for API compatibility but is ignored.

Member nvinfer1::PluginCreatorVersion

Deprecated in TensorRT 10.10. PluginCreatorVersion is used only in relation to plugin creators based off IPluginCreator, which is deprecated.

Member nvinfer1::v_1_0::IGpuAllocator::allocate (uint64_t const size, uint64_t const alignment, AllocatorFlags const flags) noexcept=0

Deprecated in TensorRT 10.0. Superseded by allocateAsync

Member nvinfer1::v_1_0::IGpuAllocator::deallocate (void *const memory) noexcept=0

Deprecated in TensorRT 10.0. Superseded by deallocateAsync

Member nvinfer1::v_1_0::IGpuAsyncAllocator::allocate (uint64_t const size, uint64_t const alignment, AllocatorFlags const flags) noexcept override

Deprecated in TensorRT 10.0. Superseded by allocateAsync

Member nvinfer1::v_1_0::IGpuAsyncAllocator::deallocate (void *const memory) noexcept override

Deprecated in TensorRT 10.0. Superseded by deallocateAsync

Member nvinfer1::v_1_0::IOutputAllocator::reallocateOutput (char const *, void *, uint64_t, uint64_t) noexcept

Deprecated in TensorRT 10.0. Superseded by reallocateOutputAsync with cudaStream_t argument

Class nvinfer1::v_1_0::IStreamReader

Deprecated in TensorRT 11.0. Superseded by IStreamReaderV2.

Class PluginVersion

Deprecated in TensorRT 10.10. PluginVersion is used only in relation to IPluginV2-descendent plugin interfaces, which are all deprecated.

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

nvinfer1
    The TensorRT API version 1 namespace . . . 29
nvinfer1::anonymous_namespace{NvInfer.h} . . . 82
nvinfer1::anonymous_namespace{NvInferRuntime.h} . . . 82
nvinfer1::impl . . . 83
nvinfer1::plugin . . . 85
nvinfer1::safe . . . 85
nvinfer1::v_1_0
    Forward declare IErrorRecorder and ILogger for use in other interfaces . . . 85
nvinfer1::v_2_0 . . . 86
nvinfer2 . . . 87
nvinfer2::safe . . . 87
nvinfer2::safe::consistency . . . 97
nvinfer2::safe::impl . . . 98
nvonnxparser
    The TensorRT ONNX parser API namespace . . . 98
nvonnxparser::anonymous_namespace{NvOnnxParser.h} . . . 102

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Dims . . . 105
nvinfer1::Dims64 . . . 109
  nvinfer1::Dims2 . . . 105
    nvinfer1::Dims3 . . . 107
      nvinfer1::Dims4 . . . 108
    nvinfer1::DimsHW . . . 112
nvinfer1::DimsExprs . . . 111
nvinfer1::DynamicPluginTensorDesc . . . 114
nvinfer1::impl::EnumMaxImpl< T > . . . 116
nvinfer2::safe::impl::EnumMaxImpl< T > . . . 116
nvinfer1::impl::EnumMaxImpl< ActivationType > . . . 117
nvinfer1::impl::EnumMaxImpl< AllocatorFlag > . . . 117
nvinfer1::impl::EnumMaxImpl< APILanguage > . . . 118
nvinfer1::impl::EnumMaxImpl< AttentionIOForm > . . . 119
nvinfer1::impl::EnumMaxImpl< AttentionNormalizationOp > . . . 120
nvinfer1::impl::EnumMaxImpl< BoundingBoxFormat > . . . 120
nvinfer1::impl::EnumMaxImpl< BuilderFlag > . . . 121
nvinfer1::impl::EnumMaxImpl< CausalMaskKind > . . . 122
nvinfer1::impl::EnumMaxImpl< CollectiveOperation > . . . 122
nvinfer1::impl::EnumMaxImpl< CumulativeOperation > . . . 123
nvinfer1::impl::EnumMaxImpl< DataType > . . . 124
nvinfer1::impl::EnumMaxImpl< DeviceType > . . . 125
nvinfer1::impl::EnumMaxImpl< DimensionOperation > . . . 125
nvinfer1::impl::EnumMaxImpl< ElementWiseOperation > . . . 126
nvinfer1::impl::EnumMaxImpl< EngineCapability > . . . 127
nvinfer1::impl::EnumMaxImpl< EngineStat > . . . 128
nvinfer1::impl::EnumMaxImpl< ErrorCode > . . . 128
nvinfer1::impl::EnumMaxImpl< ExecutionContextAllocationStrategy > . . . 129
nvinfer1::impl::EnumMaxImpl< FillOperation > . . . 130
nvinfer1::impl::EnumMaxImpl< GatherMode > . . . 130
nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel > . . . 131

nvinfer1::impl::EnumMaxImpl< ILogger::Severity > . . . 132
nvinfer1::impl::EnumMaxImpl< InterpolationMode > . . . 133
nvinfer1::impl::EnumMaxImpl< KVCacheMode > . . . 134
nvinfer1::impl::EnumMaxImpl< LayerInformationFormat > . . . 134
nvinfer1::impl::EnumMaxImpl< LayerType > . . . 135
nvinfer1::impl::EnumMaxImpl< LoopOutput > . . . 136
nvinfer1::impl::EnumMaxImpl< MatrixOperation > . . . 136
nvinfer1::impl::EnumMaxImpl< MemoryPoolType > . . . 137
nvinfer1::impl::EnumMaxImpl< MoEActType > . . . 138
nvinfer1::impl::EnumMaxImpl< NetworkDefinitionCreationFlag > . . . 139
nvinfer1::impl::EnumMaxImpl< OptProfileSelector > . . . 139
nvinfer1::impl::EnumMaxImpl< PaddingMode > . . . 140
nvinfer1::impl::EnumMaxImpl< PoolingType > . . . 141
nvinfer1::impl::EnumMaxImpl< PreviewFeature > . . . 141
nvinfer1::impl::EnumMaxImpl< ProfilingVerbosity > . . . 142
nvinfer1::impl::EnumMaxImpl< ReduceOperation > . . . 143
nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation > . . . 144
nvinfer1::impl::EnumMaxImpl< ResizeRoundMode > . . . 144
nvinfer1::impl::EnumMaxImpl< ResizeSelector > . . . 145
nvinfer1::impl::EnumMaxImpl< RuntimePlatform > . . . 146
nvinfer1::impl::EnumMaxImpl< SampleMode > . . . 146
nvinfer1::impl::EnumMaxImpl< ScaleMode > . . . 147
nvinfer1::impl::EnumMaxImpl< ScatterMode > . . . 148
nvinfer1::impl::EnumMaxImpl< SerializationFlag > . . . 149
nvinfer2::safe::impl::EnumMaxImpl< Severity > . . . 149
nvinfer1::impl::EnumMaxImpl< TacticSource > . . . 150
nvinfer1::impl::EnumMaxImpl< TempfileControlFlag > . . . 151
nvinfer1::impl::EnumMaxImpl< TensorFormat > . . . 152
nvinfer1::impl::EnumMaxImpl< TensorIOMode > . . . 152
nvinfer1::impl::EnumMaxImpl< TensorLocation > . . . 153
nvinfer1::impl::EnumMaxImpl< TilingOptimizationLevel > . . . 154
nvinfer1::impl::EnumMaxImpl< TopKOperation > . . . 154
nvinfer1::impl::EnumMaxImpl< TripLimit > . . . 155
nvinfer1::impl::EnumMaxImpl< UnaryOperation > . . . 156
nvinfer1::impl::EnumMaxImpl< WeightsRole > . . . 157
nvinfer1::plugin::GridAnchorParameters . . . 157
nvinfer2::safe::consistency::IConsistencyChecker . . . 228
IDebugListener . . . 272
IErrorRecorder . . . 306
IGpuAllocator . . . 358
IGpuAsyncAllocator . . . 365
nvinfer1::INoCopy . . . 495
  nvinfer1::IAttention . . . 165
  nvinfer1::IBuilder . . . 188
  nvinfer1::IBuilderConfig . . . 198
  nvinfer1::ICudaEngine . . . 244
  nvinfer1::IDimensionExpr . . . 288
  nvinfer1::IEngineInspector . . . 301
  nvinfer1::IExecutionContext . . . 314
  nvinfer1::IExprBuilder . . . 341
  nvinfer1::IHostMemory . . . 373
  nvinfer1::IIfConditional . . . 377
  nvinfer1::ILayer . . . 393
    nvinfer1::IActivationLayer . . . 160

nvinfer1::IAssertionLayer . . . 163
nvinfer1::IAttentionBoundaryLayer . . . 183
nvinfer1::IAttentionInputLayer . . . 184
nvinfer1::IAttentionOutputLayer . . . 186
nvinfer1::ICastLayer . . . 223
nvinfer1::IConcatenationLayer . . . 225
nvinfer1::IConstantLayer . . . 232
nvinfer1::IConvolutionLayer . . . 234
nvinfer1::ICumulativeLayer . . . 268
nvinfer1::IDeconvolutionLayer . . . 274
nvinfer1::IDequantizeLayer . . . 283
nvinfer1::IDistCollectiveLayer . . . 290
nvinfer1::IDynamicQuantizeLayer . . . 291
nvinfer1::IEinsumLayer . . . 297
nvinfer1::IElementWiseLayer . . . 299
nvinfer1::IFillLayer . . . 344
nvinfer1::IGatherLayer . . . 353
nvinfer1::IGridSampleLayer . . . 370
nvinfer1::IIdentityLayer . . . 375
nvinfer1::IIfConditionalBoundaryLayer . . . 381
nvinfer1::IConditionLayer . . . 227
nvinfer1::IIfConditionalInputLayer . . . 383
nvinfer1::IIfConditionalOutputLayer . . . 384
nvinfer1::IKVCacheUpdateLayer . . . 388
nvinfer1::ILRNLayer . . . 413
nvinfer1::ILoopBoundaryLayer . . . 408
nvinfer1::IIteratorLayer . . . 385
nvinfer1::ILoopOutputLayer . . . 410
nvinfer1::IRecurrenceLayer . . . 666
nvinfer1::ITripLimitLayer . . . 801
nvinfer1::IMatrixMultiplyLayer . . . 417
nvinfer1::IMoELayer . . . 419
nvinfer1::INMSLayer . . . 490
nvinfer1::INonZeroLayer . . . 498
nvinfer1::INormalizationLayer . . . 500
nvinfer1::IOneHotLayer . . . 505
nvinfer1::IPaddingLayer . . . 523
nvinfer1::IParametricReLULayer . . . 526
nvinfer1::IPluginV2Layer . . . 601
nvinfer1::IPluginV3Layer . . . 605
nvinfer1::IPoolingLayer . . . 646
nvinfer1::IQuantizeLayer . . . 660
nvinfer1::IRaggedSoftMaxLayer . . . 664
nvinfer1::IReduceLayer . . . 668
nvinfer1::IResizeLayer . . . 682
nvinfer1::IReverseSequenceLayer . . . 691
nvinfer1::IRotaryEmbeddingLayer . . . 694
nvinfer1::IScaleLayer . . . 740
nvinfer1::IScatterLayer . . . 745
nvinfer1::ISelectLayer . . . 749
nvinfer1::IShapeLayer . . . 753
nvinfer1::IShuffleLayer . . . 755
nvinfer1::ISliceLayer . . . 760
nvinfer1::ISoftMaxLayer . . . 767

nvinfer1::ISqueezeLayer . . . 770
nvinfer1::ITopKLayer . . . 796
nvinfer1::IUnaryLayer . . . 818
nvinfer1::IUnsqueezeLayer . . . 820
nvinfer1::ILoop . . . 405
nvinfer1::INetworkDefinition . . . 433
nvinfer1::IOptimizationProfile . . . 513
nvinfer1::IRefitter . . . 671
nvinfer1::IRuntime . . . 697
nvinfer1::IRuntimeConfig . . . 706
nvinfer1::ISerializationConfig . . . 750
nvinfer1::ITensor . . . 781
nvinfer1::ITimingCache . . . 792
INonZero . . . 497
nvinfer1::InterfaceInfo . . . 504
nvonnxparser::IOnnxConfig . . . 507
IOutputAllocator . . . 520
nvonnxparser::IParser . . . 527
nvonnxparser::IParserError . . . 538
nvonnxparser::IParserRefitter . . . 541
IPluginCapability . . . 546
nvinfer2::safe::consistency::IPluginChecker . . . 547
IPluginCreator . . . 550
IPluginCreatorInterface . . . 555
IPluginCreatorV3One . . . 557
nvinfer1::IPluginRegistry . . . 562
IPluginResource . . . 569
nvinfer1::IPluginResourceContext . . . 572
nvinfer1::IPluginV2 . . . 575
nvinfer1::IPluginV2Ext . . . 592
nvinfer1::IPluginV2DynamicExt . . . 585
nvinfer1::IPluginV2IOExt . . . 598
IPluginV3 . . . 602
IPluginV3OneBuild . . . 606
IPluginV3OneBuildV2 . . . 616
IPluginV3OneCore . . . 617
IPluginV3OneRuntime . . . 619
IProfiler . . . 654
IProgressMonitor . . . 656
nvinfer2::safe::ISafeMemAllocator . . . 708
nvinfer2::safe::ISafePluginRegistry . . . 718
nvinfer2::safe::ISafePluginResourceContext . . . 726
IStreamReader . . . 772
IStreamReaderV2 . . . 775
IStreamWriter . . . 778
nvinfer2::safe::ITRTGraph . . . 803
nvinfer1::IVersionedInterface . . . 822
nvinfer1::v_1_0::IDebugListener . . . 272
nvinfer1::v_1_0::IErrorRecorder . . . 307
nvinfer2::safe::ISafeRecorder . . . 731
nvinfer1::v_1_0::IGpuAllocator . . . 359
nvinfer1::v_1_0::IGpuAsyncAllocator . . . 365
nvinfer1::v_1_0::ILogger . . . 400

nvinfer1::v_1_0::ILoggerFinder . . . 403
nvinfer1::v_1_0::IOutputAllocator . . . 520
nvinfer1::v_1_0::IPluginCapability . . . 546
nvinfer1::v_1_0::IPluginV3OneBuild . . . 607
nvinfer1::v_2_0::IPluginV3OneBuild . . . 615
nvinfer1::v_1_0::IPluginV3OneCore . . . 617
nvinfer1::v_1_0::IPluginV3OneRuntime . . . 620
nvinfer2::safe::IPluginV3OneSafeBuild . . . 623
nvinfer2::safe::IPluginV3OneSafeBuildMSS . . . 633
nvinfer2::safe::IPluginV3OneSafeCore . . . 637
nvinfer2::safe::IPluginV3OneSafeRuntime . . . 641
nvinfer1::v_1_0::IPluginCreatorInterface . . . 555
nvinfer1::v_1_0::IPluginCreator . . . 550
nvinfer1::v_1_0::IPluginCreatorV3One . . . 558
nvinfer2::safe::ISafePluginCreatorV3One . . . 713
nvinfer1::v_1_0::IPluginResource . . . 570
nvinfer1::v_1_0::IPluginV3 . . . 603
nvinfer1::v_1_0::IProfiler . . . 655
nvinfer1::v_1_0::IProgressMonitor . . . 657
nvinfer1::v_1_0::IStreamReader . . . 772
nvinfer1::v_1_0::IStreamReaderV2 . . . 775
nvinfer1::v_1_0::IStreamWriter . . . 779
nvinfer1::Permutation . . . 825
nvinfer2::safe::PhysicalDims . . . 826
nvinfer1::PluginField . . . 827
nvinfer1::PluginFieldCollection . . . 829
nvinfer1::PluginRegistrar< T > . . . 830
nvinfer1::PluginTensorDesc . . . 831
PluginVersion . . . 832
nvinfer1::plugin::PriorBoxParameters . . . 833
nvinfer1::plugin::RegionParameters . . . 836
nvinfer1::plugin::RPROIParams . . . 838
nvinfer2::safe::RuntimeErrorInformation . . . 841
nvinfer1::plugin::softmaxTree . . . 841
nvinfer2::safe::TensorDescriptor . . . 843
nvinfer1::v_1_0::TimingCacheKey . . . 847
nvinfer1::v_1_0::TimingCacheValue . . . 847
nvinfer2::safe::TypedArray . . . 849
Value . . . 854
nvinfer1::Weights . . . 855

Chapter 6

Class Index

6.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Dims
: Structure to define the dimensions of a tensor . . . 105

nvinfer1::Dims2
: Descriptor for two-dimensional data . . . 105

nvinfer1::Dims3
: Descriptor for three-dimensional data . . . 107

nvinfer1::Dims4
: Descriptor for four-dimensional data . . . 108

nvinfer1::Dims64 . . . 109

nvinfer1::DimsExprs
: Analog of class Dims with expressions instead of constants for the dimensions . . . 111

nvinfer1::DimsHW
: Descriptor for two-dimensional spatial data . . . 112

nvinfer1::DynamicPluginTensorDesc
: Summarizes tensors that a plugin might see for an input or output . . . 114

nvinfer1::impl::EnumMaxImpl< T >
: Declaration of EnumMaxImpl struct to store the exclusive upper bound of an enumeration type . 116

nvinfer2::safe::impl::EnumMaxImpl< T >
: Declaration of EnumMaxImpl struct to store maximum number of elements in an enumeration type 116

nvinfer1::impl::EnumMaxImpl< ActivationType > . . . 117

nvinfer1::impl::EnumMaxImpl< AllocatorFlag >
: Maximum number of elements in AllocatorFlag enum . . . 117

nvinfer1::impl::EnumMaxImpl< APILanguage >
: One greater than the maximum value of APILanguage enum . . . 118

nvinfer1::impl::EnumMaxImpl< AttentionIOForm > . . . 119

nvinfer1::impl::EnumMaxImpl< AttentionNormalizationOp > . . . 120

nvinfer1::impl::EnumMaxImpl< BoundingBoxFormat > . . . 120

nvinfer1::impl::EnumMaxImpl< BuilderFlag > . . . 121

nvinfer1::impl::EnumMaxImpl< CausalMaskKind > . . . 122

nvinfer1::impl::EnumMaxImpl< CollectiveOperation > . . . 122

nvinfer1::impl::EnumMaxImpl< CumulativeOperation >
: Maximum number of elements in CumulativeOperation enum . . . 123

nvinfer1::impl::EnumMaxImpl< DataType >
One greater than the maximum value of DataType enum . . . 124
nvinfer1::impl::EnumMaxImpl< DeviceType >
Maximum number of elements in DeviceType enum . . . 125
nvinfer1::impl::EnumMaxImpl< DimensionOperation >
Maximum number of elements in DimensionOperation enum . . . 125
nvinfer1::impl::EnumMaxImpl< ElementWiseOperation > . . . 126
nvinfer1::impl::EnumMaxImpl< EngineCapability >
Maximum number of elements in EngineCapability enum . . . 127
nvinfer1::impl::EnumMaxImpl< EngineStat >
Maximum number of engine statistic kinds in EngineStat enum . . . 128
nvinfer1::impl::EnumMaxImpl< ErrorCode >
One greater than the maximum value of ErrorCode enum . . . 128
nvinfer1::impl::EnumMaxImpl< ExecutionContextAllocationStrategy > . . . 129
nvinfer1::impl::EnumMaxImpl< FillOperation > . . . 130
nvinfer1::impl::EnumMaxImpl< GatherMode > . . . 130
nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel > . . . 131
nvinfer1::impl::EnumMaxImpl< ILogger::Severity >
Maximum number of elements in ILogger::Severity enum . . . 132
nvinfer1::impl::EnumMaxImpl< InterpolationMode > . . . 133
nvinfer1::impl::EnumMaxImpl< KVCacheMode > . . . 134
nvinfer1::impl::EnumMaxImpl< LayerInformationFormat >
Maximum number of layer information formats in LayerInformationFormat enum . . . 134
nvinfer1::impl::EnumMaxImpl< LayerType > . . . 135
nvinfer1::impl::EnumMaxImpl< LoopOutput > . . . 136
nvinfer1::impl::EnumMaxImpl< MatrixOperation > . . . 136
nvinfer1::impl::EnumMaxImpl< MemoryPoolType > . . . 137
nvinfer1::impl::EnumMaxImpl< MoEActType > . . . 138
nvinfer1::impl::EnumMaxImpl< NetworkDefinitionCreationFlag > . . . 139
nvinfer1::impl::EnumMaxImpl< OptProfileSelector >
Number of different values of OptProfileSelector enum . . . 139
nvinfer1::impl::EnumMaxImpl< PaddingMode > . . . 140
nvinfer1::impl::EnumMaxImpl< PoolingType > . . . 141
nvinfer1::impl::EnumMaxImpl< PreviewFeature > . . . 141
nvinfer1::impl::EnumMaxImpl< ProfilingVerbosity >
Maximum number of profile verbosity levels in ProfilingVerbosity enum . . . 142
nvinfer1::impl::EnumMaxImpl< ReduceOperation > . . . 143
nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation > . . . 144
nvinfer1::impl::EnumMaxImpl< ResizeRoundMode > . . . 144
nvinfer1::impl::EnumMaxImpl< ResizeSelector > . . . 145
nvinfer1::impl::EnumMaxImpl< RuntimePlatform > . . . 146
nvinfer1::impl::EnumMaxImpl< SampleMode > . . . 146
nvinfer1::impl::EnumMaxImpl< ScaleMode > . . . 147
nvinfer1::impl::EnumMaxImpl< ScatterMode > . . . 148
nvinfer1::impl::EnumMaxImpl< SerializationFlag >
Maximum number of serialization flags in SerializationFlag enum . . . 149
nvinfer2::safe::impl::EnumMaxImpl< Severity >
Provides a static constant for the maximum value of the Severity enum . . . 149
nvinfer1::impl::EnumMaxImpl< TacticSource >
Maximum number of tactic sources in TacticSource enum . . . 150
nvinfer1::impl::EnumMaxImpl< TempfileControlFlag >
Maximum number of elements in TempfileControlFlag enum . . . 151
nvinfer1::impl::EnumMaxImpl< TensorFormat >
Maximum number of elements in TensorFormat enum . . . 152

nvinfer1::impl::EnumMaxImpl< TensorIOMode >
One greater than the maximum value of TensorIOMode enum . . . 152
nvinfer1::impl::EnumMaxImpl< TensorLocation >
Maximum number of elements in TensorLocation enum . . . 153
nvinfer1::impl::EnumMaxImpl< TilingOptimizationLevel > . . . 154
nvinfer1::impl::EnumMaxImpl< TopKOperation > . . . 154
nvinfer1::impl::EnumMaxImpl< TripLimit > . . . 155
nvinfer1::impl::EnumMaxImpl< UnaryOperation > . . . 156
nvinfer1::impl::EnumMaxImpl< WeightsRole >
Maximum number of elements in WeightsRole enum . . . 157
nvinfer1::plugin::GridAnchorParameters
The Anchor Generator plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions (H x W). GridAnchorParameters defines a set of parameters for creating the plugin layer for all feature maps . . . 157
nvinfer1::IActivationLayer
An Activation layer in a network definition . . . 160
nvinfer1::IAssertionLayer
An assertion layer in a network . . . 163
nvinfer1::IAttention
Helper for constructing an attention that consumes query, key and value tensors . . . 165
nvinfer1::IAttentionBoundaryLayer
This is a base class for Attention boundary layers . . . 183
nvinfer1::IAttentionInputLayer
This layer represents an input to an attention subgraph . . . 184
nvinfer1::IAttentionOutputLayer
This layer represents an output of an IAttention . . . 186
nvinfer1::IBuilder
Builds an engine from a network definition . . . 188
nvinfer1::IBuilderConfig
Holds properties for configuring a builder to produce an engine . . . 198
nvinfer1::ICastLayer
A cast layer in a network . . . 223
nvinfer1::IConcatenationLayer
A concatenation layer in a network definition . . . 225
nvinfer1::IConditionLayer
This layer represents a condition input to an IIfConditional . . . 227
nvinfer2::safe::consistency::IConsistencyChecker
Validates a TRT engine (Serialized binary blob) . . . 228
nvinfer1::IConstantLayer
Layer that represents a constant value . . . 232
nvinfer1::IConvolutionLayer
A convolution layer in a network definition . . . 234
nvinfer1::ICudaEngine
An engine for executing inference on a built network, with functionally unsafe features . . . 244
nvinfer1::ICumulativeLayer
Layer that represents a cumulative operation across a tensor . . . 268
IDebugListener
User-implemented callback for notification when value of a debug tensor is updated . . . 272
nvinfer1::v_1_0::IDebugListener . . . 272
nvinfer1::IDeconvolutionLayer
A deconvolution layer in a network definition . . . 274
nvinfer1::IDequantizeLayer
A Dequantize layer in a network definition . . . 283

nvinfer1::IDimensionExpr
An IDimensionExpr represents an integer expression constructed from constants, input dimensions, and binary operations. These expressions are can be used in overrides of IPluginV2DynamicExt::getOutputDimensions or IPluginV3OneBuild::getOutputShapes() to define output dimensions in terms of input dimensions . . . 288
nvinfer1::IDistCollectiveLayer . . . 290
nvinfer1::IDynamicQuantizeLayer
A network layer to perform dynamic quantization . . . 291
nvinfer1::IEinsumLayer
An Einsum layer in a network . . . 297
nvinfer1::IElementWiseLayer
A elementwise layer in a network definition . . . 299
nvinfer1::IEngineInspector
An engine inspector which prints out the layer information of an engine or an execution context . 301
IErrorRecorder
Reference counted application-implemented error reporting interface for TensorRT objects . . . 306
nvinfer1::v_1_0::IErrorRecorder . . . 307
nvinfer1::IExecutionContext
Context for executing inference using an engine, with functionally unsafe features . . . 314
nvinfer1::IExprBuilder
Object for constructing IDimensionExpr . . . 341
nvinfer1::IFillLayer
Generate a tensor according to a specified mode . . . 344
nvinfer1::IGatherLayer
A Gather layer in a network definition. Supports several kinds of gathering . . . 353
IGpuAllocator
Application-implemented class for controlling allocation on the GPU . . . 358
nvinfer1::v_1_0::IGpuAllocator . . . 359
IGpuAsyncAllocator
Application-implemented class for controlling asynchronous (stream ordered) memory allocation on the GPU . . . 365
nvinfer1::v_1_0::IGpuAsyncAllocator . . . 365
nvinfer1::IGridSampleLayer
A GridSample layer in a network definition . . . 370
nvinfer1::IHostMemory
Class to handle library allocated memory that is accessible to the user . . . 373
nvinfer1::IIdentityLayer
A layer that represents the identity function . . . 375
nvinfer1::IIfConditional
Helper for constructing conditionally-executed subgraphs . . . 377
nvinfer1::IIfConditionalBoundaryLayer
This is a base class for Conditional boundary layers . . . 381
nvinfer1::IIfConditionalInputLayer
This layer represents an input to an IIfConditional . . . 383
nvinfer1::IIfConditionalOutputLayer
This layer represents an output of an IIfConditional . . . 384
nvinfer1::IIteratorLayer
A layer to do iterations . . . 385
nvinfer1::IKVCacheUpdateLayer
Layer that represents a KVCacheUpdate operation . . . 388
nvinfer1::ILayer
Base class for all layer classes in a network definition . . . 393
nvinfer1::v_1_0::ILogger
Application-implemented logging interface for the builder, refitter and runtime . . . 400

nvinfer1::v_1_0::ILoggerFinder
A virtual base class to find a logger. Allows a plugin to find an instance of a logger if it needs to emit a log message. A pointer to an instance of this class is passed to a plugin shared library on initialization when that plugin is serialized as part of a version-compatible plan. See the plugin chapter in the developer guide for details . . . 403
nvinfer1::ILoop
Helper for creating a recurrent subgraph . . . 405
nvinfer1::ILoopBoundaryLayer
This is a base class for Loop boundary layers . . . 408
nvinfer1::ILoopOutputLayer
An ILoopOutputLayer is the sole way to get output from a loop . . . 410
nvinfer1::ILRNLayer
A LRN layer in a network definition . . . 413
nvinfer1::IMatrixMultiplyLayer
Layer that represents a Matrix Multiplication . . . 417
nvinfer1::IMoELayer
A MoE layer in a network definition. Mixture of Experts (MoE) is a collection of experts with each expert specializing in processing different subsets of input data. The key innovation lies in using a Router that selectively activates only the specific experts needed for a given input, rather than engaging the entire neural network for every task . . . 419
nvinfer1::INetworkDefinition
A network definition for input to the builder . . . 433
nvinfer1::INMSLayer
A non-maximum suppression layer in a network definition . . . 490
nvinfer1::INoCopy
Forward declaration of IEngineInspector for use by other interfaces . . . 495
INonZero
A NonZero layer in a network . . . 497
nvinfer1::INonZeroLayer . . . 498
nvinfer1::INormalizationLayer
A normalization layer in a network definition . . . 500
nvinfer1::InterfaceInfo
Version information associated with a TRT interface . . . 504
nvinfer1::IOneHotLayer
A OneHot layer in a network definition . . . 505
nvonnxparser::IOnnxConfig
Configuration Manager Class . . . 507
nvinfer1::IOptimizationProfile
Optimization profile for dynamic input dimensions and shape tensors . . . 513
IOutputAllocator
Callback from ExecutionContext::enqueueV3() . . . 520
nvinfer1::v_1_0::IOutputAllocator . . . 520
nvinfer1::IPaddingLayer
Layer that represents a padding operation . . . 523
nvinfer1::IParametricReLULayer
Layer that represents a parametric ReLU operation . . . 526
nvonnxparser::IParser
Object for parsing ONNX models into a TensorRT network definition . . . 527
nvonnxparser::IParserError
Object containing information about an error . . . 538
nvonnxparser::IParserRefitter
An interface designed to refit weights from an ONNX model . . . 541
IPluginCapability
Base class for plugin capability interfaces . . . 546

nvinfer1::v_1_0::IPluginCapability . . . 546
nvinfer2::safe::consistency::IPluginChecker
Interface for validating a plugin used in a TensorRT engine . . . 547
IPluginCreator
Plugin creator class for user implemented layers . . . 550
nvinfer1::v_1_0::IPluginCreator . . . 550
IPluginCreatorInterface
Base class for all plugin creator versions . . . 555
nvinfer1::v_1_0::IPluginCreatorInterface . . . 555
IPluginCreatorV3One
A plugin creator class capable of producing IPluginV3 objects . . . 557
nvinfer1::v_1_0::IPluginCreatorV3One . . . 558
nvinfer1::IPluginRegistry
Single registration point for all plugins in an application. It is used to find plugin implementations during engine deserialization. Internally, the plugin registry is considered to be a singleton so all plugins in an application are part of the same global registry. Note that the plugin registry is only supported for plugins of type IPluginV2 and should also have a corresponding IPluginCreator implementation . . . 562
IPluginResource
Interface for plugins to define custom resources that could be shared through the plugin registry . 569
nvinfer1::v_1_0::IPluginResource . . . 570
nvinfer1::IPluginResourceContext
Interface for plugins to access per context resources provided by TensorRT . . . 572
nvinfer1::IPluginV2
Plugin class for user-implemented layers . . . 575
nvinfer1::IPluginV2DynamicExt
Similar to IPluginV2Ext, but with support for dynamic shapes . . . 585
nvinfer1::IPluginV2Ext
Plugin class for user-implemented layers . . . 592
nvinfer1::IPluginV2IOExt
Plugin class for user-implemented layers . . . 598
nvinfer1::IPluginV2Layer
Layer type for pluginV2 . . . 601
IPluginV3
Plugin class for the V3 generation of user-implemented layers . . . 602
nvinfer1::v_1_0::IPluginV3 . . . 603
nvinfer1::IPluginV3Layer
Layer type for V3 plugins . . . 605
IPluginV3OneBuild
A plugin capability interface that enables the build capability (PluginCapabilityType::kBUILD). Exposes methods that allow the expression of the build time properties and behavior of a plugin . 606
nvinfer1::v_1_0::IPluginV3OneBuild . . . 607
nvinfer1::v_2_0::IPluginV3OneBuild . . . 615
IPluginV3OneBuildV2
A plugin capability interface that extends IPluginV3OneBuild by providing I/O aliasing functionality . . . 616
IPluginV3OneCore
A plugin capability interface that enables the core capability (PluginCapabilityType::kCORE) . . 617
nvinfer1::v_1_0::IPluginV3OneCore . . . 617
IPluginV3OneRuntime
A plugin capability interface that enables the runtime capability (PluginCapabilityType::k↩RUNTIME). Exposes methods that allow the expression of the runtime properties and behavior of a plugin . . . 619
nvinfer1::v_1_0::IPluginV3OneRuntime . . . 620

nvinfer2::safe::IPluginV3OneSafeBuild
Interface for plugins used during the TensorRT build phase . . . 623
nvinfer2::safe::IPluginV3OneSafeBuildMSS
This class provides build capability and let output shapes be calculated by symbolic expression to support multiple static shape . . . 633
nvinfer2::safe::IPluginV3OneSafeCore
Interface specifying the base plugin capability . . . 637
nvinfer2::safe::IPluginV3OneSafeRuntime
Runtime component of the pluginV3 system . . . 641
nvinfer1::IPoolingLayer
A Pooling layer in a network definition . . . 646
IProfiler
Application-implemented interface for profiling . . . 654
nvinfer1::v_1_0::IProfiler . . . 655
IProgressMonitor
Application-implemented progress reporting interface for TensorRT . . . 656
nvinfer1::v_1_0::IProgressMonitor . . . 657
nvinfer1::IQuantizeLayer
A Quantize layer in a network definition . . . 660
nvinfer1::IRaggedSoftMaxLayer
A RaggedSoftmax layer in a network definition . . . 664
nvinfer1::IRecurrenceLayer
A recurrence layer in a network definition . . . 666
nvinfer1::IReduceLayer
Layer that represents a reduction across a non-bool tensor . . . 668
nvinfer1::IRefitter
Updates weights in an engine . . . 671
nvinfer1::IResizeLayer
A resize layer in a network definition . . . 682
nvinfer1::IReverseSequenceLayer
A ReverseSequence layer in a network definition . . . 691
nvinfer1::IRotaryEmbeddingLayer
Layer that implements Rotary Position Embedding (RoPE) (https://arxiv.org/abs/2104.↩ 09864) . . . 694
nvinfer1::IRuntime
Allows a serialized functionally unsafe engine to be deserialized . . . 697
nvinfer1::IRuntimeConfig
A class for runtime configuration. This class is used during execution context creation . . . 706
nvinfer2::safe::ISafeMemAllocator
Application-implemented class for controlling memory allocation on the GPU/CPU . . . 708
nvinfer2::safe::ISafePluginCreatorV3One
The main interface to be implemented by all plugin creator classes . . . 713
nvinfer2::safe::ISafePluginRegistry
PluginRegistry interface class that contains all the methods user can use to interface with the registry object . . . 718
nvinfer2::safe::ISafePluginResourceContext
Interface for plugins to access per context resources provided by TensorRT . . . 726
nvinfer2::safe::ISafeRecorder
Interface for extended recorder which allows error, warn, debug, or info messages to be recorded . 731
nvinfer1::IScaleLayer
A Scale layer in a network definition . . . 740
nvinfer1::IScatterLayer
A scatter layer in a network definition. Supports several kinds of scattering . . . 745

nvinfer1::ISelectLayer
Select elements from two data tensors based on a condition tensor . . . 749
nvinfer1::ISerializationConfig
Holds properties for configuring an engine to serialize the binary . . . 750
nvinfer1::IShapeLayer
Layer type for getting shape of a tensor . . . 753
nvinfer1::IShuffleLayer
Layer type for shuffling data . . . 755
nvinfer1::ISliceLayer
Slices an input tensor into an output tensor based on the offset and strides . . . 760
nvinfer1::ISoftMaxLayer
A Softmax layer in a network definition . . . 767
nvinfer1::ISqueezeLayer
Layer that represents a squeeze operation, removing unit dimensions of the first input tensor on a set of axes specified by the second input tensor . . . 770
IStreamReader
Application-implemented class for reading data in a stream-based manner . . . 772
nvinfer1::v_1_0::IStreamReader . . . 772
IStreamReaderV2
Application-implemented class for reading data in a stream-based manner asynchronously. Intended for use with the GDS API for optimizing load times . . . 775
nvinfer1::v_1_0::IStreamReaderV2 . . . 775
IStreamWriter
Application-implemented class for writing data in a stream-based manner . . . 778
nvinfer1::v_1_0::IStreamWriter . . . 779
nvinfer1::ITensor
A tensor in a network definition . . . 781
nvinfer1::ITimingCache
Class to handle tactic timing info collected from builder . . . 792
nvinfer1::ITopKLayer
Layer that represents a TopK reduction . . . 796
nvinfer1::ITripLimitLayer
A layer that represents a trip-count limiter . . . 801
nvinfer2::safe::ITRTGraph
Abstract Interface for a functionally safe graph for executing inference on a built network . . . 803
nvinfer1::IUnaryLayer
Layer that represents an unary operation . . . 818
nvinfer1::IUnsqueezeLayer
Layer that represents an unsqueeze operation, which reshapes the first input tensor by inserting unit-length dimensions to the output at the axes specified by the second input tensor . . . 820
nvinfer1::IVersionedInterface
An Interface class for version control . . . 822
nvinfer1::Permutation
Represents a permutation of dimensions . . . 825
nvinfer2::safe::PhysicalDims
Structure to define the physical dimensions of a tensor with support for up to 9 dimensions . . . 826
nvinfer1::PluginField
Structure containing plugin attribute field names and associated data This information can be parsed to decode necessary plugin metadata . . . 827
nvinfer1::PluginFieldCollection
Plugin field collection struct . . . 829

nvinfer1::PluginRegistrar< T > Register the plugin creator to the registry The static registry object will be instantiated when the plugin library is loaded. This static object will register all creators available in the library to the registry . . . 830
nvinfer1::PluginTensorDesc Fields that a plugin might see for an input or output . . . 831
PluginVersion Definition of plugin versions . . . 832
nvinfer1::plugin::PriorBoxParameters The PriorBox plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions (H x W) . . . 833
nvinfer1::plugin::RegionParameters The Region plugin layer performs region proposal calculation . . . 836
nvinfer1::plugin::RPROIParams RPROIParams is used to create the RPROIPlugin instance . . . 838
nvinfer2::safe::RuntimeErrorInformation Holds information about runtime errors that occur during asynchronous kernel execution . . . 841
nvinfer1::plugin::softmaxTree When performing yolo9000, softmaxTree is helping to do softmax on confidence scores, for element to get the precise classification through word-tree structured classification definition . . . 841
nvinfer2::safe::TensorDescriptor A simple record summarizing various properties of a network IO Tensor . . . 843
nvinfer1::v_1_0::TimingCacheKey The key to retrieve timing cache entries . . . 847
nvinfer1::v_1_0::TimingCacheValue . . . 847
nvinfer2::safe::TypedArray A standard_layout and trivially_copyable typed array that knows the data type and size it is holding 849
Value The values in the cache entry . . . 854
nvinfer1::Weights An array of weights used as a layer parameter . . . 855

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

NvInfer.h . . . 857
NvInferImpl.h . . . 916
NvInferLegacyDims.h . . . 934
NvInferPlugin.h . . . 936
NvInferPluginBase.h . . . 937
NvInferPluginUtils.h . . . 941
NvInferRuntime.h . . . 943
NvInferRuntimeBase.h . . . 972
NvInferRuntimeCommon.h . . . 981
NvInferRuntimePlugin.h . . . 983
NvInferVersion.h . . . 987
NvOnnxConfig.h . . . 991
NvOnnxParser.h . . . 993
NvInferConsistency.h . . . 998
NvInferForwardDecl.h . . . 1000
NvInferSafeMemAllocator.h . . . 1002
NvInferSafePlugin.h . . . 1004
NvInferSafeRecorder.h . . . 1009
NvInferSafeRuntime.h . . . 1012

Chapter 8

Namespace Documentation

8.1 nvinfer1 Namespace Reference

The TensorRT API version 1 namespace.

Namespaces

- namespace anonymous_namespace{NvInfer.h}
- namespace anonymous_namespace{NvInferRuntime.h}
- namespace impl
- namespace plugin
- namespace safe
- namespace v_1_0

  Forward declare IErrorRecorder and ILogger for use in other interfaces.
- namespace v_2_0

Classes

- class Dims2

  Descriptor for two-dimensional data.
- class Dims3

  Descriptor for three-dimensional data.
- class Dims4

  Descriptor for four-dimensional data.
- class Dims64
- class DimsExprs

  Analog of class Dims with expressions instead of constants for the dimensions.
- class DimsHW

  Descriptor for two-dimensional spatial data.
- struct DynamicPluginTensorDesc

  Summarizes tensors that a plugin might see for an input or output.

- class IActivationLayer

  An Activation layer in a network definition.
- class IAssertionLayer

  An assertion layer in a network.
- class IAttention

  Helper for constructing an attention that consumes query, key and value tensors.
- class IAttentionBoundaryLayer

  This is a base class for Attention boundary layers.
- class IAttentionInputLayer

  This layer represents an input to an attention subgraph.
- class IAttentionOutputLayer

  This layer represents an output of an IAttention.
- class IBuilder

  Builds an engine from a network definition.
- class IBuilderConfig

  Holds properties for configuring a builder to produce an engine.
- class ICastLayer

  A cast layer in a network.
- class IConcatenationLayer

  A concatenation layer in a network definition.
- class IConditionLayer

  This layer represents a condition input to an IIfConditional.
- class IConstantLayer

  Layer that represents a constant value.
- class IConvolutionLayer

  A convolution layer in a network definition.
- class ICudaEngine

  An engine for executing inference on a built network, with functionally unsafe features.
- class ICumulativeLayer

  Layer that represents a cumulative operation across a tensor.
- class IDeconvolutionLayer

  A deconvolution layer in a network definition.
- class IDequantizeLayer

  A Dequantize layer in a network definition.
- class IDimensionExpr

  An IDimensionExpr represents an integer expression constructed from constants, input dimensions, and binary operations. These expressions are can be used in overrides of IPluginV2DynamicExt::getOutputDimensions or IPluginV3OneBuild::getOutputShapes() to define output dimensions in terms of input dimensions.
- class IDistCollectiveLayer
- class IDynamicQuantizeLayer

  A network layer to perform dynamic quantization.
- class IEinsumLayer

  An Einsum layer in a network.
- class IElementWiseLayer

  A elementwise layer in a network definition.
- class IEngineInspector

  An engine inspector which prints out the layer information of an engine or an execution context.

- class IExecutionContext

  Context for executing inference using an engine, with functionally unsafe features.
- class IExprBuilder

  Object for constructing IDimensionExpr.
- class IFillLayer

  Generate a tensor according to a specified mode.
- class IGatherLayer

  A Gather layer in a network definition. Supports several kinds of gathering.
- class IGridSampleLayer

  A GridSample layer in a network definition.
- class IHostMemory

  Class to handle library allocated memory that is accessible to the user.
- class IIdentityLayer

  A layer that represents the identity function.
- class IIfConditional

  Helper for constructing conditionally-executed subgraphs.
- class IIfConditionalBoundaryLayer

  This is a base class for Conditional boundary layers.
- class IIfConditionalInputLayer

  This layer represents an input to an IIfConditional.
- class IIfConditionalOutputLayer

  This layer represents an output of an IIfConditional.
- class IIteratorLayer

  A layer to do iterations.
- class IKVCacheUpdateLayer

  Layer that represents a KVCacheUpdate operation.
- class ILayer

  Base class for all layer classes in a network definition.
- class ILoop

  Helper for creating a recurrent subgraph.
- class ILoopBoundaryLayer

  This is a base class for Loop boundary layers.
- class ILoopOutputLayer

  An ILoopOutputLayer is the sole way to get output from a loop.
- class ILRNLayer

  A LRN layer in a network definition.
- class IMatrixMultiplyLayer

  Layer that represents a Matrix Multiplication.
- class IMoELayer

  A MoE layer in a network definition. Mixture of Experts (MoE) is a collection of experts with each expert specializing in processing different subsets of input data. The key innovation lies in using a Router that selectively activates only the specific experts needed for a given input, rather than engaging the entire neural network for every task.
- class INetworkDefinition

  A network definition for input to the builder.
- class INMSLayer

  A non-maximum suppression layer in a network definition.
- class INoCopy

Forward declaration of IEngineInspector for use by other interfaces.

- class INonZeroLayer
- class INormalizationLayer

  A normalization layer in a network definition.
- class InterfaceInfo

  Version information associated with a TRT interface.
- class IOneHotLayer

  A OneHot layer in a network definition.
- class IOptimizationProfile

  Optimization profile for dynamic input dimensions and shape tensors.
- class IPaddingLayer

  Layer that represents a padding operation.
- class IParametricReLULayer

  Layer that represents a parametric ReLU operation.
- class IPluginRegistry

  Single registration point for all plugins in an application. It is used to find plugin implementations during engine deserialization. Internally, the plugin registry is considered to be a singleton so all plugins in an application are part of the same global registry. Note that the plugin registry is only supported for plugins of type IPluginV2 and should also have a corresponding IPluginCreator implementation.
- class IPluginResourceContext

  Interface for plugins to access per context resources provided by TensorRT.
- class IPluginV2

  Plugin class for user-implemented layers.
- class IPluginV2DynamicExt

  Similar to IPluginV2Ext, but with support for dynamic shapes.
- class IPluginV2Ext

  Plugin class for user-implemented layers.
- class IPluginV2IOExt

  Plugin class for user-implemented layers.
- class IPluginV2Layer

  Layer type for pluginV2.
- class IPluginV3Layer

  Layer type for V3 plugins.
- class IPoolingLayer

  A Pooling layer in a network definition.
- class IQuantizeLayer

  A Quantize layer in a network definition.
- class IRaggedSoftMaxLayer

  A RaggedSoftmax layer in a network definition.
- class IRecurrenceLayer

  A recurrence layer in a network definition.
- class IReduceLayer

  Layer that represents a reduction across a non-bool tensor.
- class IRefitter

  Updates weights in an engine.
- class IResizeLayer

  A resize layer in a network definition.
- class IReverseSequenceLayer

A ReverseSequence layer in a network definition.

- class IRotaryEmbeddingLayer

  Layer that implements Rotary Position Embedding (RoPE) (https://arxiv.org/abs/2104.09864).
- class IRuntime

  Allows a serialized functionally unsafe engine to be deserialized.
- class IRuntimeConfig

  A class for runtime configuration. This class is used during execution context creation.
- class IScaleLayer

  A Scale layer in a network definition.
- class IScatterLayer

  A scatter layer in a network definition. Supports several kinds of scattering.
- class ISelectLayer

  Select elements from two data tensors based on a condition tensor.
- class ISerializationConfig

  Holds properties for configuring an engine to serialize the binary.
- class IShapeLayer

  Layer type for getting shape of a tensor.
- class IShuffleLayer

  Layer type for shuffling data.
- class ISliceLayer

  Slices an input tensor into an output tensor based on the offset and strides.
- class ISoftMaxLayer

  A Softmax layer in a network definition.
- class ISqueezeLayer

  Layer that represents a squeeze operation, removing unit dimensions of the first input tensor on a set of axes specified by the second input tensor.
- class ITensor

  A tensor in a network definition.
- class ITimingCache

  Class to handle tactic timing info collected from builder.
- class ITopKLayer

  Layer that represents a TopK reduction.
- class ITripLimitLayer

  A layer that represents a trip-count limiter.
- class IUnaryLayer

  Layer that represents an unary operation.
- class IUnsqueezeLayer

  Layer that represents an unsqueeze operation, which reshapes the first input tensor by inserting unit-length dimensions to the output at the axes specified by the second input tensor.
- class IVersionedInterface

  An Interface class for version control.
- struct Permutation

  Represents a permutation of dimensions.
- class PluginField

  Structure containing plugin attribute field names and associated data This information can be parsed to decode necessary plugin metadata.
- struct PluginFieldCollection

  Plugin field collection struct.

- class PluginRegistrar

  Register the plugin creator to the registry The static registry object will be instantiated when the plugin library is loaded. This static object will register all creators available in the library to the registry.
- struct PluginTensorDesc

  Fields that a plugin might see for an input or output.
- class Weights

  An array of weights used as a layer parameter.

Typedefs

- using IPluginResource = v_1_0::IPluginResource
- using IPluginCreatorInterface = v_1_0::IPluginCreatorInterface
- using IPluginV3 = v_1_0::IPluginV3
- using IPluginCapability = v_1_0::IPluginCapability
- using char_t = char

  char_t is the type used by TensorRT to represent all valid characters.
- using AsciiChar = char_t
- using IErrorRecorder = v_1_0::IErrorRecorder
- using ILogger = v_1_0::ILogger
- using Dims = Dims64
- using InterfaceKind = char const *
- using TensorFormats = uint32_t

  It is capable of representing one or more TensorFormat by binary OR operations, e.g., 1U << TensorFormat::kCHW4 | 1U << TensorFormat::kCHW32.
- using BuilderFlags = uint32_t

  Represents one or more BuilderFlag values using binary OR operations, e.g., 1U << BuilderFlag::kDEBUG.
- using IProgressMonitor = v_1_0::IProgressMonitor
- using NetworkDefinitionCreationFlags = uint32_t

  Represents one or more NetworkDefinitionCreationFlag flags using binary OR operations. e.g., 1U << NetworkDefinitionCreationFlag::kPREFER_JIT_PYTHON_PLUGINS.
- using IStreamReader = v_1_0::IStreamReader
- using IStreamWriter = v_1_0::IStreamWriter
- using IStreamReaderV2 = v_1_0::IStreamReaderV2
- using IPluginV3OneCore = v_1_0::IPluginV3OneCore
- using IPluginV3OneBuild = v_1_0::IPluginV3OneBuild
- using IPluginV3OneRuntime = v_1_0::IPluginV3OneRuntime
- using IPluginV3OneBuildV2 = v_2_0::IPluginV3OneBuild
- using IProfiler = v_1_0::IProfiler
- using TempfileControlFlags = uint32_t

  Represents a collection of one or more TempfileControlFlag values combined using bitwise-OR operations.
- using AllocatorFlags = uint32_t
- using IGpuAllocator = v_1_0::IGpuAllocator
- using TacticSources = uint32_t

  Represents a collection of one or more TacticSource values combine using bitwise-OR operations.
- using SerializationFlags = uint32_t

  Represents one or more SerializationFlag values using binary OR operations, e.g., 1U << SerializationFlag::kEXCLUDE_LEAN_RUNTIME
- using IOutputAllocator = v_1_0::IOutputAllocator
- using IDebugListener = v_1_0::IDebugListener
- using ILoggerFinder = v_1_0::ILoggerFinder

- using IGpuAsyncAllocator = v_1_0::IGpuAsyncAllocator
- using IPluginCreatorV3One = v_1_0::IPluginCreatorV3One
- using PluginFormat = TensorFormat

  PluginFormat is reserved for backward compatibility.
- using IPluginCreator = v_1_0::IPluginCreator

Enumerations

- enum class PluginFieldType : int32_t {
  kFLOAT16 = 0 , kFLOAT32 = 1 , kFLOAT64 = 2 , kINT8 = 3 ,
  kINT16 = 4 , kINT32 = 5 , kCHAR = 6 , kDIMS = 7 ,
  kUNKNOWN = 8 , kBF16 = 9 , kINT64 = 10 , kFP8 = 11 ,
  kINT4 = 12 , kFP4 = 13 }

  The possible field types for custom layer.
- enum class TensorRTPhase : int32_t { kBUILD = 0 , kRUNTIME = 1 }

  Indicates a phase of operation of TensorRT.
- enum class PluginCapabilityType : int32_t { kCORE = 0 , kBUILD = 1 , kRUNTIME = 2 }

  Enumerates the different capability types a IPluginV3 object may have.
- enum class DataType : int32_t {
  kFLOAT = 0 , kHALF = 1 , kINT8 = 2 , kINT32 = 3 ,
  kBOOL = 4 , kUINT8 = 5 , kFP8 = 6 , kBF16 = 7 ,
  kINT64 = 8 , kINT4 = 9 , kFP4 = 10 , kE8M0 = 11 }

  The type of weights and tensors. The datatypes other than kBOOL, kINT32, and kINT64 are "activation datatypes," as they often represent values corresponding to inference results.
- enum class APILanguage : int32_t { kCPP = 0 , kPYTHON = 1 }

  Programming language used in the implementation of a TRT interface.
- enum class ErrorCode : int32_t {
  kSUCCESS = 0 , kUNSPECIFIED_ERROR = 1 , kINTERNAL_ERROR = 2 , kINVALID_ARGUMENT = 3 ,
  kINVALID_CONFIG = 4 , kFAILED_ALLOCATION = 5 , kFAILED_INITIALIZATION = 6 , kFAILED_EXECUTION = 7 ,
  kFAILED_COMPUTATION = 8 , kINVALID_STATE = 9 , kUNSUPPORTED_STATE = 10 }

  Error codes that can be returned by TensorRT during execution.
- enum class TensorIOMode : int32_t { kNONE = 0 , kINPUT = 1 , kOUTPUT = 2 }

  Definition of tensor IO Mode.
- enum class LayerType : int32_t {
  kCONVOLUTION = 0 , kCAST = 1 , kACTIVATION = 2 , kPOOLING = 3 ,
  kLRN = 4 , kSCALE = 5 , kSOFTMAX = 6 , kDECONVOLUTION = 7 ,
  kCONCATENATION = 8 , kELEMENTWISE = 9 , kPLUGIN = 10 , kUNARY = 11 ,
  kPADDING = 12 , kSHUFFLE = 13 , kREDUCE = 14 , kTOPK = 15 ,
  kGATHER = 16 , kMATRIX_MULTIPLY = 17 , kRAGGED_SOFTMAX = 18 , kCONSTANT = 19 ,
  kIDENTITY = 20 , kPLUGIN_V2 = 21 , kSLICE = 22 , kSHAPE = 23 ,
  kPARAMETRIC_RELU = 24 , kRESIZE = 25 , kTRIP_LIMIT = 26 , kRECURRENCE = 27 ,
  kITERATOR = 28 , kLOOP_OUTPUT = 29 , kSELECT = 30 , kFILL = 31 ,
  kQUANTIZE = 32 , kDEQUANTIZE = 33 , kCONDITION = 34 , kCONDITIONAL_INPUT = 35 ,
  kCONDITIONAL_OUTPUT = 36 , kSCATTER = 37 , kEINSUM = 38 , kASSERTION = 39 ,
  kONE_HOT = 40 , kNON_ZERO = 41 , kGRID_SAMPLE = 42 , kNMS = 43 ,
  kREVERSE_SEQUENCE = 44 , kNORMALIZATION = 45 , kPLUGIN_V3 = 46 , kSQUEEZE = 47 ,
  kUNSQUEEZE = 48 , kCUMULATIVE = 49 , kDYNAMIC_QUANTIZE = 50 , kATTENTION_INPUT = 51 ,
  kATTENTION_OUTPUT = 52 , kROTARY_EMBEDDING = 53 , kKVCACHE_UPDATE = 54 , kMOE = 55 ,
  kDIST_COLLECTIVE = 56 }

The type values of layer classes.

- enum class ActivationType : int32_t {
  kRELU = 0 , kSIGMOID = 1 , kTANH = 2 , kLEAKY_RELU = 3 ,
  kELU = 4 , kSELU = 5 , kSOFTSIGN = 6 , kSOFTPLUS = 7 ,
  kCLIP = 8 , kHARD_SIGMOID = 9 , kSCALED_TANH = 10 , kTHRESHOLDED_RELU = 11 ,
  kGELU_ERF = 12 , kGELU_TANH = 13 }

  Enumerates the types of activation to perform in an activation layer.
- enum class PaddingMode : int32_t { kEXPLICIT_ROUND_DOWN = 0 , kEXPLICIT_ROUND_UP = 1 , kSAME_UPPER = 2 , kSAME_LOWER = 3 }

  Enumerates the modes of padding to perform in convolution, deconvolution and pooling layer, padding mode takes precedence if setPaddingMode() and setPrePadding() are also used.
- enum class PoolingType : int32_t { kMAX = 0 , kAVERAGE = 1 , kMAX_AVERAGE_BLEND = 2 }

  The type of pooling to perform in a pooling layer.
- enum class ScaleMode : int32_t { kUNIFORM = 0 , kCHANNEL = 1 , kELEMENTWISE = 2 }

  Controls how shift, scale and power are applied in a Scale layer.
- enum class ElementWiseOperation : int32_t {
  kSUM = 0 , kPROD = 1 , kMAX = 2 , kMIN = 3 ,
  kSUB = 4 , kDIV = 5 , kPOW = 6 , kFLOOR_DIV = 7 ,
  kAND = 8 , kOR = 9 , kXOR = 10 , kEQUAL = 11 ,
  kGREATER = 12 , kLESS = 13 }

  Enumerates the binary operations that may be performed by an ElementWise layer.
- enum class GatherMode : int32_t { kDEFAULT = 0 , kELEMENT = 1 , kND = 2 }

  Control form of IGatherLayer.
- enum class UnaryOperation : int32_t {
  kEXP = 0 , kLOG = 1 , kSQRT = 2 , kRECIP = 3 ,
  kABS = 4 , kNEG = 5 , kSIN = 6 , kCOS = 7 ,
  kTAN = 8 , kSINH = 9 , kCOSH = 10 , kASIN = 11 ,
  kACOS = 12 , kATAN = 13 , kASINH = 14 , kACOSH = 15 ,
  kATANH = 16 , kCEIL = 17 , kFLOOR = 18 , kERF = 19 ,
  kNOT = 20 , kSIGN = 21 , kROUND = 22 , kISINF = 23 ,
  kISNAN = 24 }

  Enumerates the unary operations that may be performed by a Unary layer.
- enum class ReduceOperation : int32_t {
  kSUM = 0 , kPROD = 1 , kMAX = 2 , kMIN = 3 ,
  kAVG = 4 , kNONE = 5 }

  Enumerates the reduce operations that may be performed by a Reduce layer.
- enum class CollectiveOperation : int32_t {
  kALL_REDUCE = 0 , kALL_GATHER = 1 , kBROADCAST = 2 , kREDUCE = 3 ,
  kREDUCE_SCATTER = 4 , kALL_TO_ALL = 5 , kGATHER = 6 , kSCATTER = 7 }

  Enumerates the collective operations that may be performed by a DistCollective layer.
- enum class SampleMode : int32_t {
  kSTRICT_BOUNDS = 0 , kWRAP = 1 , kCLAMP = 2 , kFILL = 3 ,
  kREFLECT = 4 }

  Controls how ISliceLayer and IGridSample handle out-of-bounds coordinates.
- enum class TopKOperation : int32_t { kMAX = 0 , kMIN = 1 }

  Enumerates the operations that may be performed by a TopK layer.
- enum class MatrixOperation : int32_t { kNONE = 0 , kTRANSPOSE = 1 , kVECTOR = 2 }

  Enumerates the operations that may be performed on a tensor by IMatrixMultiplyLayer before multiplication.
- enum class InterpolationMode : int32_t { kNEAREST = 0 , kLINEAR = 1 , kCUBIC = 2 }

  Enumerates various modes of interpolation.

- enum class ResizeCoordinateTransformation : int32_t { kALIGN_CORNERS = 0 , kASYMMETRIC = 1 , kHALF_PIXEL = 2 }

  The resize coordinate transformation function.
- enum class ResizeSelector : int32_t { kFORMULA = 0 , kUPPER = 1 }

  The coordinate selector when resize to single pixel output.
- enum class ResizeRoundMode : int32_t { kHALF_UP = 0 , kHALF_DOWN = 1 , kFLOOR = 2 , kCEIL = 3 }

  The rounding mode for nearest neighbor resize.
- enum class LoopOutput : int32_t { kLAST_VALUE = 0 , kCONCATENATE = 1 , kREVERSE = 2 }

  Enum that describes kinds of loop outputs.
- enum class TripLimit : int32_t { kCOUNT = 0 , kWHILE = 1 }

  Enum that describes kinds of trip limits.
- enum class FillOperation : int32_t { kLINSPACE = 0 , kRANDOM_UNIFORM = 1 , kRANDOM_NORMAL = 2 }

  Enumerates the tensor fill operations that may performed by a fill layer.
- enum class ScatterMode : int32_t { kELEMENT = 0 , kND = 1 }

  Control form of IScatterLayer.
- enum class BoundingBoxFormat : int32_t { kCORNER_PAIRS = 0 , kCENTER_SIZES = 1 }

  Representation of bounding box data used for the Boxes input tensor in INMSLayer.
- enum class CumulativeOperation : int32_t { kSUM = 0 }

  Enumerates the cumulative operations that may be performed by a Cumulative layer.
- enum class AttentionNormalizationOp : int32_t { kNONE = 0 , kSOFTMAX = 1 }

  Enumerates the operations that may be performed by the normalization in the attention subgraph.
- enum class CausalMaskKind : int32_t { kNONE = 0 , kUPPER_LEFT = 1 , kLOWER_RIGHT = 2 }

  Enumerates the causal mask alignment orientation for the attention.
- enum class AttentionIOForm : int32_t { kPADDED_BHND = 0 , kPACKED_NHD = 1 }

  Enumerates the layout of the input/output tensors in an Attention layer.
- enum class KVCacheMode : int32_t { kLINEAR = 0 }

  Enumerates the KVCache modes that may be performed by a KVCacheUpdate layer.
- enum class MoEActType : int32_t { kNONE = 0 , kSILU = 1 }

  Enumerates the activation type for the MoE layer.
- enum class RuntimePlatform : int32_t { kSAME_AS_BUILD = 0 , kWINDOWS_AMD64 = 1 }

  Describes the intended runtime platform (operating system and CPU architecture) for the execution of the TensorRT engine. TensorRT provides support for cross-platform engine compatibility when the target runtime platform is different from the build platform.
- enum class BuilderFlag : int32_t {
  kDEBUG = 2 , kGPU_FALLBACK = 3 , kREFIT = 4 , kDISABLE_TIMING_CACHE = 5 ,
  kTF32 = 6 , kSPARSE_WEIGHTS = 7 , kSAFETY_SCOPE = 8 , kDIRECT_IO = 11 ,
  kVERSION_COMPATIBLE = 12 , kEXCLUDE_LEAN_RUNTIME = 13 , kERROR_ON_TIMING_CACHE_MISS = 15 , kDISABLE_COMPILATION_CACHE = 17 ,
  kSTRIP_PLAN = 18 , kREFIT_IDENTICAL = 19 , kWEIGHT_STREAMING = 20 , kREFIT_INDIVIDUAL = 22 ,
  kSTRICT_NANS = 23 , kMONITOR_MEMORY = 24 , kEDITABLE_TIMING_CACHE = 26 , kDISTRIBUTIVE_INDEPENDEN = 27 }

  List of valid modes that the builder can enable when creating an engine from a network definition.
- enum class MemoryPoolType : int32_t {
  kWORKSPACE = 0 , kDLA_MANAGED_SRAM = 1 , kDLA_LOCAL_DRAM = 2 , kDLA_GLOBAL_DRAM = 3 ,
  kTACTIC_DRAM = 4 , kTACTIC_SHARED_MEMORY = 5 }

  The type for memory pools used by TensorRT.

- enum class PreviewFeature : int32_t { kALIASED_PLUGIN_IO_10_03 = 0 , kRUNTIME_ACTIVATION_RESIZE_10_10 = 1 }

  Define preview features.
- enum class HardwareCompatibilityLevel : int32_t { kNONE = 0 , kAMPERE_PLUS = 1 , kSAME_COMPUTE_CAPABILITY = 2 }

  Describes requirements of compatibility with GPU architectures other than that of the GPU on which the engine was built.
- enum class TilingOptimizationLevel : int32_t { kNONE = 0 , kFAST = 1 , kMODERATE = 2 , kFULL = 3 }

  Define the optimization levels for Tiling.
- enum class NetworkDefinitionCreationFlag : int32_t { kSTRONGLY_TYPED = 0 , kPREFER_JIT_PYTHON_PLUGINS = 1 , kPREFER_AOT_PYTHON_PLUGINS = 2 }

  List of immutable network properties expressed at network creation time. NetworkDefinitionCreationFlag is used with createNetworkV2() to specify immutable properties of the network.
- enum class EngineCapability : int32_t { kSTANDARD = 0 , kSAFETY = 1 , kDLA_STANDALONE = 2 }

  List of supported engine capability flows.
- enum class DimensionOperation : int32_t {
  kSUM = 0 , kPROD = 1 , kMAX = 2 , kMIN = 3 ,
  kSUB = 4 , kEQUAL = 5 , kLESS = 6 , kFLOOR_DIV = 7 ,
  kCEIL_DIV = 8 }

  An operation on two IDimensionExpr, which represent integer expressions used in dimension computations.
- enum class TensorLocation : int32_t { kDEVICE = 0 , kHOST = 1 }

  The location for tensor data storage, device or host.
- enum class SeekPosition : int32_t { kSET = 0 , kCUR = 1 , kEND = 2 }

  Controls the seek mode of IStreamReaderV2.
- enum class WeightsRole : int32_t {
  kKERNEL = 0 , kBIAS = 1 , kSHIFT = 2 , kSCALE = 3 ,
  kCONSTANT = 4 , kANY = 5 }

  How a layer uses particular Weights.
- enum class DeviceType : int32_t { kGPU = 0 , kDLA = 1 }

  The device that this layer/network will execute on.
- enum class TempfileControlFlag : int32_t { kALLOW_IN_MEMORY_FILES = 0 , kALLOW_TEMPORARY_FILES = 1 }

  Flags used to control TensorRT's behavior when creating executable temporary files.
- enum class TensorFormat : int32_t {
  kLINEAR = 0 , kCHW2 = 1 , kHWC8 = 2 , kCHW4 = 3 ,
  kCHW16 = 4 , kCHW32 = 5 , kDHWC8 = 6 , kCDHW32 = 7 ,
  kHWC = 8 , kDLA_LINEAR = 9 , kDLA_HWC4 = 10 , kHWC16 = 11 ,
  kDHWC = 12 }

  Format of the input/output tensors.
- enum class AllocatorFlag : int32_t { kRESIZABLE = 0 }

  Allowed type of memory allocation.
- enum class OptProfileSelector : int32_t { kMIN = 0 , kOPT = 1 , kMAX = 2 }

  When setting or querying optimization profile parameters (such as shape tensor inputs or dynamic dimensions), select whether we are interested in the minimum, optimum, or maximum values for these parameters. The minimum and maximum specify the permitted range that is supported at runtime, while the optimum value is used for the kernel selection. This should be the "typical" value that is expected to occur at runtime.
- enum class TacticSource : int32_t { kEDGE_MASK_CONVOLUTIONS = 0 , kJIT_CONVOLUTIONS = 1 }

  List of tactic sources for TensorRT.
- enum class ProfilingVerbosity : int32_t { kLAYER_NAMES_ONLY = 0 , kNONE = 1 , kDETAILED = 2 }

  List of verbosity levels of layer information exposed in NVTX annotations and in IEngineInspector.

- enum class SerializationFlag : int32_t { kEXCLUDE_WEIGHTS = 0 , kEXCLUDE_LEAN_RUNTIME = 1 , kINCLUDE_REFIT = 2 }

  List of valid flags that the engine can enable when serializing the bytes.
- enum class ExecutionContextAllocationStrategy : int32_t { kSTATIC = 0 , kON_PROFILE_CHANGE = 1 , kUSER_MANAGED = 2 }

  Different memory allocation behaviors for IExecutionContext.
- enum class EngineStat : int32_t { kTOTAL_WEIGHTS_SIZE = 0 , kSTRIPPED_WEIGHTS_SIZE = 1 }

  The kind of engine statistics that queried from the ICudaEngine.
- enum class LayerInformationFormat : int32_t { kONELINE = 0 , kJSON = 1 }

  The format in which the IEngineInspector prints the layer information.
- enum class PluginVersion : uint8_t {
  kV2 = 0 , kV2_EXT = 1 , kV2_IOEXT = 2 , kV2_DYNAMICEXT = 3 ,
  kV2_DYNAMICEXT_PYTHON = kPLUGIN_VERSION_PYTHON_BIT | 3 }
- enum class PluginCreatorVersion : int32_t { kV1 = 0 , kV1_PYTHON = kPLUGIN_VERSION_PYTHON_BIT }

  Enum to identify version of the plugin creator.

Functions

- template<typename T >
  constexpr int32_t EnumMax () noexcept
- TENSORRTAPI nvinfer1::IPluginRegistry * getBuilderPluginRegistry (nvinfer1::EngineCapability capability) noexcept

  Return the plugin registry for building a Standard engine, or nullptr if no registry exists.
- TENSORRTAPI bool setInternalLibraryPath (AsciiChar const *path) noexcept

  Set a custom directory path for loading internal TensorRT libraries when building engines.
- TRT_DEPRECATED_API nvinfer1::safe::IPluginRegistry * getBuilderSafePluginRegistry (nvinfer1::EngineCapability capability) noexcept

  Return the plugin registry for building a Safety engine, or nullptr if no registry exists.

8.1.1 Detailed Description

The TensorRT API version 1 namespace.

8.1.2 Typedef Documentation

8.1.2.1 AllocatorFlags

```
using nvinfer1::AllocatorFlags = typedef uint32_t
```

8.1.2.2 AsciiChar

```
using nvinfer1::AsciiChar = typedef char_t
```

AsciiChar is the type used by TensorRT to represent valid ASCII characters. This type is widely used in automotive safety context.

8.1.2.3 BuilderFlags

```
using nvinfer1::BuilderFlags = typedef uint32_t
```

Represents one or more BuilderFlag values using binary OR operations, e.g., 1U << BuilderFlag::kDEBUG.

See also

IBuilderConfig::setFlags(), IBuilderConfig::getFlags()

8.1.2.4 char_t

```
using nvinfer1::char_t = typedef char
```

char_t is the type used by TensorRT to represent all valid characters.

8.1.2.5 Dims

```
using nvinfer1::Dims = typedef Dims64
```

Alias for Dims64.

8.1.2.6 IDebugListener

```
using nvinfer1::IDebugListener = typedef v_1_0::IDebugListener
```

8.1.2.7 IErrorRecorder

```
typedef v_1_0::IErrorRecorder nvinfer1::IErrorRecorder
```

8.1.2.8 IGpuAllocator

```
typedef v_1_0::IGpuAllocator nvinfer1::IGpuAllocator
```

8.1.2.9 IGpuAsyncAllocator

```
using nvinfer1::IGpuAsyncAllocator = typedef v_1_0::IGpuAsyncAllocator
```

8.1.2.10 ILogger

```
typedef v_1_0::ILogger nvinfer1::ILogger
```

8.1.2.11 ILoggerFinder

```
using nvinfer1::ILoggerFinder = typedef v_1_0::ILoggerFinder
```

8.1.2.12 InterfaceKind

```
using nvinfer1::InterfaceKind = typedef char const*
```

8.1.2.13 IOutputAllocator

```
using nvinfer1::IOutputAllocator = typedef v_1_0::IOutputAllocator
```

8.1.2.14 IPluginCapability

```
using nvinfer1::IPluginCapability = typedef v_1_0::IPluginCapability
```

8.1.2.15 IPluginCreator

```
using nvinfer1::IPluginCreator = typedef v_1_0::IPluginCreator
```

8.1.2.16 IPluginCreatorInterface

```
using nvinfer1::IPluginCreatorInterface = typedef v_1_0::IPluginCreatorInterface
```

8.1.2.17 IPluginCreatorV3One

```
using nvinfer1::IPluginCreatorV3One = typedef v_1_0::IPluginCreatorV3One
```

8.1.2.18 IPluginResource

```
using nvinfer1::IPluginResource = typedef v_1_0::IPluginResource
```

8.1.2.19 IPluginV3

```
using nvinfer1::IPluginV3 = typedef v_1_0::IPluginV3
```

8.1.2.20 IPluginV3OneBuild

```
using nvinfer1::IPluginV3OneBuild = typedef v_1_0::IPluginV3OneBuild
```

8.1.2.21 IPluginV3OneBuildV2

```
using nvinfer1::IPluginV3OneBuildV2 = typedef v_2_0::IPluginV3OneBuild
```

8.1.2.22 IPluginV3OneCore

```
using nvinfer1::IPluginV3OneCore = typedef v_1_0::IPluginV3OneCore
```

8.1.2.23 IPluginV3OneRuntime

```
using nvinfer1::IPluginV3OneRuntime = typedef v_1_0::IPluginV3OneRuntime
```

8.1.2.24 IProfiler

```
using nvinfer1::IProfiler = typedef v_1_0::IProfiler
```

8.1.2.25 IProgressMonitor

```
using nvinfer1::IProgressMonitor = typedef v_1_0::IProgressMonitor
```

8.1.2.26 IStreamReader

```
using nvinfer1::IStreamReader = typedef v_1_0::IStreamReader
```

8.1.2.27 IStreamReaderV2

```
using nvinfer1::IStreamReaderV2 = typedef v_1_0::IStreamReaderV2
```

8.1.2.28 IStreamWriter

```
using nvinfer1::IStreamWriter = typedef v_1_0::IStreamWriter
```

8.1.2.29 NetworkDefinitionCreationFlags

```
using nvinfer1::NetworkDefinitionCreationFlags = typedef uint32_t
```

Represents one or more NetworkDefinitionCreationFlag flags using binary OR operations. e.g., 1U << NetworkDefinitionCreationFlag::kPREFER_JIT_PYTHON_PLUGINS.

See also

IBuilder::createNetworkV2

8.1.2.30 PluginFormat

```
using nvinfer1::PluginFormat = typedef TensorFormat
```

PluginFormat is reserved for backward compatibility.

See also

IPluginV2::supportsFormat()

8.1.2.31 SerializationFlags

```
using nvinfer1::SerializationFlags = typedef uint32_t
```

Represents one or more SerializationFlag values using binary OR operations, e.g., 1U << SerializationFlag::kEXCLUDE_LEAN_RUNT

See also

ISerializationConfig::setFlags(), ISerializationConfig::getFlags()

8.1.2.32 TacticSources

```
using nvinfer1::TacticSources = typedef uint32_t
```

Represents a collection of one or more TacticSource values combine using bitwise-OR operations.

See also

IBuilderConfig::setTacticSources(), IBuilderConfig::getTacticSources()

8.1.2.33 TempfileControlFlags

```
using nvinfer1::TempfileControlFlags = typedef uint32_t
```

Represents a collection of one or more TempfileControlFlag values combined using bitwise-OR operations.

See also

TempfileControlFlag, IRuntime::setTempfileControlFlags(), IRuntime::getTempfileControlFlags()

8.1.2.34 TensorFormats

```
using nvinfer1::TensorFormats = typedef uint32_t
```

It is capable of representing one or more TensorFormat by binary OR operations, e.g., 1U << TensorFormat::kCHW4 | 1U << TensorFormat::kCHW32.

See also

ITensor::getAllowedFormats(), ITensor::setAllowedFormats(),

8.1.3 Enumeration Type Documentation

8.1.3.1 ActivationType

```
enum class nvinfer1::ActivationType : int32_t [strong]
```

Enumerates the types of activation to perform in an activation layer.

Enumerator

kRELU	Rectified linear activation.
kSIGMOID	Sigmoid activation.
kTANH	TanH activation.
kLEAKY_RELU	LeakyRelu activation: x>=0 ? x : alpha * x.
kELU	Elu activation: x>=0 ? x : alpha * (exp(x) - 1).
kSELU	Selu activation: x>0 ? beta * x : beta * (alpha*exp(x) - alpha)
kSOFTSIGN	Softsign activation: x / (1+|x|)
kSOFTPLUS	Parametric softplus activation: alpha*log(exp(beta*x)+1)
kCLIP	Clip activation: max(alpha, min(beta, x))
kHARD_SIGMOID	Hard sigmoid activation: max(0, min(1, alpha*x+beta))
kSCALED_TANH	Scaled tanh activation: alpha*tanh(beta*x)
kTHRESHOLDED_RELU	Thresholded ReLU activation: x>alpha ? x : 0.
kGELU_ERF	GELU erf activation: 0.5 * x * (1 + erf(sqrt(0.5) * x))
kGELU_TANH	GELU tanh activation: 0.5 * x * (1 + tanh(sqrt(2/pi) * (0.044715F * pow(x, 3) + x)))

8.1.3.2 AllocatorFlag

`enum class nvinfer1::AllocatorFlag : int32_t [strong]`

Allowed type of memory allocation.

Enumerator

kRESIZABLE	TensorRT may call realloc() on this allocation.

8.1.3.3 APILanguage

`enum class nvinfer1::APILanguage : int32_t [strong]`

Programming language used in the implementation of a TRT interface.

Enumerator

kCPP	
kPYTHON	

8.1.3.4 AttentionIOForm

`enum class nvinfer1::AttentionIOForm : int32_t [strong]`

Enumerates the layout of the input/output tensors in an Attention layer.

Enumerator

kPADDED_BHND	All batches padded to maximum length. Shape: [batch_size, num_heads, num_tokens, head_dim]
kPACKED_NHD	All batches concatenated without padding. Shape: [total_tokens, num_heads, head_dim]

8.1.3.5 AttentionNormalizationOp

`enum class nvinfer1::AttentionNormalizationOp : int32_t [strong]`

Enumerates the operations that may be performed by the normalization in the attention subgraph.

Enumerator

kNONE	Apply no normalization on the attention scores. Must be used with decomposable=True on pre-Blackwell GPUs.
kSOFTMAX	Apply softmax normalization on the attention scores on the `s_kv` dimension.

8.1.3.6 BoundingBoxFormat

`enum class nvinfer1::BoundingBoxFormat : int32_t [strong]`

Representation of bounding box data used for the Boxes input tensor in INMSLayer.

See also

INMSLayer

Enumerator

kCORNER_PAIRS	(x1, y1, x2, y2) where (x1, y1) and (x2, y2) are any pair of diagonal corners
kCENTER_SIZES	(x_center, y_center, width, height) where (x_center, y_center) is the center point of the box

8.1.3.7 BuilderFlag

`enum class nvinfer1::BuilderFlag : int32_t [strong]`

List of valid modes that the builder can enable when creating an engine from a network definition.

See also

IBuilderConfig::setFlags(), IBuilderConfig::getFlags()

Enumerator

kDEBUG	Enable debugging of layers via synchronizing after every layer.
kGPU_FALLBACK	Enable layers marked to execute on GPU if layer cannot execute on DLA.
kREFIT	Enable building a refittable engine.
kDISABLE_TIMING_CACHE	Disable reuse of timing information across identical layers.

Enumerator

kTF32	Allow (but not require) computations on tensors of type DataType::kFLOAT to use TF32. TF32 computes inner products by rounding the inputs to 10-bit mantissas before multiplying, but accumulates the sum using 23-bit mantissas. Enabled by default.
kSPARSE_WEIGHTS	Allow the builder to examine weights and use optimized functions when weights have suitable sparsity.
kSAFETY_SCOPE	Change the allowed parameters in the EngineCapability::kSTANDARD flow to match the restrictions that EngineCapability::kSAFETY check against for DeviceType::kGPU and EngineCapability::kDLA_STANDALONE check against the DeviceType::kDLA case. This flag is forced to true if EngineCapability::kSAFETY at build time if it is unset. This flag is only supported in NVIDIA Drive(R) products. **Deprecated** Deprecated in TensorRT 10.16. In EngineCapability::kSTANDARD flow, safety restrictions are no longer supported. In EngineCapability::kSAFETY and EngineCapability::kDLA_STANDALONE flows, restrictions are enforced natively. This flag is retained for API compatibility but is ignored.
kDIRECT_IO	Require that no reformats be inserted between a layer and a network I/O tensor for which ITensor::setAllowedFormats was called. Build fails if a reformat is required for functional correctness. **Deprecated** Deprecated in TensorRT 10.7. Unneeded API.
kVERSION_COMPATIBLE	Restrict to lean runtime operators to provide version forward compatibility for the plan. This flag is only supported by NVIDIA Volta and later GPUs. This flag is not supported in NVIDIA Drive(R) products.
kEXCLUDE_LEAN_RUNTIME	Exclude lean runtime from the plan when version forward compatability is enabled. By default, this flag is unset, so the lean runtime will be included in the plan. If BuilderFlag::kVERSION_COMPATIBLE is not set then the value of this flag will be ignored.
kERROR_ON_TIMING_CACHE_MISS	Emit error when a tactic being timed is not present in the timing cache. This flag has an effect only when IBuilderConfig has an associated ITimingCache.
kDISABLE_COMPILATION_CACHE	Disable caching of JIT-compilation results during engine build. By default, JIT-compiled code will be serialized as part of the timing cache, which may significantly increase the cache size. Setting this flag prevents the code from being serialized. This flag has an effect only when BuilderFlag::DISABLE_TIMING_CACHE is not set.
kSTRIP_PLAN	Strip the refittable weights from the engine plan file.

Enumerator

kREFIT_IDENTICAL	Create a refittable engine under the assumption that the refit weights will be identical to those provided at build time. The resulting engine will have the same performance as a non-refittable one. All refittable weights can be refitted through the refit API, but if the refit weights are not identical to the build-time weights, behavior is undefined. When used alongside 'kSTRIP_PLAN', this flag will result in a small plan file for which weights are later supplied via refitting. This enables use of a single set of weights with different inference backends, or with TensorRT plans for multiple GPU architectures.
kWEIGHT_STREAMING	Enable weight streaming for the current engine. Weight streaming from the host enables execution of models that do not fit in GPU memory by allowing TensorRT to intelligently stream network weights from the CPU DRAM. Please see ICudaEngine::getMinimumWeightStreamingBudget for the default memory budget when this flag is enabled. Enabling this feature changes the behavior of IRuntime::deserializeCudaEngine to allocate the entire network's weights on the CPU DRAM instead of GPU memory. Then, ICudaEngine::createExecutionContext will determine the optimal split of weights between the CPU and GPU and place weights accordingly. Future TensorRT versions may enable this flag by default. Warning Enabling this flag may marginally increase build time. Enabling this feature will significantly increase the latency of ICudaEngine::createExecutionContext. See also IRuntime::deserializeCudaEngine, ICudaEngine::getMinimumWeightStreamingBudget, ICudaEngine::setWeightStreamingBudget
kREFIT_INDIVIDUAL	Enable building a refittable engine and provide fine-grained control. This allows control over which weights are refittable or not using INetworkDefinition::markWeightsRefittable and INetworkDefinition::unmarkWeightsRefittable. By default, all weights are non-refittable when this flag is enabled. This flag cannot be used together with kREFIT or kREFIT_IDENTICAL.

Enumerator

kSTRICT_NANS	Disable floating-point optimizations: 0∗x => 0, x-x => 0, or x/x => 1. These identities are not true when x is a NaN or Inf, and thus might hide propagation or generation of NaNs. This flag is typically used in combination with kSPARSE_WEIGHTS. There are three valid sparsity configurations. 1. Disable all sparsity. Both kSPARSE_WEIGHTS and kSTRICT_NANS are unset 2. Enable sparsity only where it does not affect propagation/generation of NaNs. Both kSPARSE_WEIGHTS and kSTRICT_NANS are set 3. Enable all sparsity. kSPARSE_WEIGHTS is set and kSTRICT_NANS is unset
kMONITOR_MEMORY	Enable memory monitor during build time.
kEDITABLE_TIMING_CACHE	Enable editable timing cache.
kDISTRIBUTIVE_INDEPENDENCE	Enable distributive independence. When BuilderFlag::kDISTRIBUTIVE_INDEPENDENCE is set and a layer documents axis i of an output as a distributive axis, then the layer behaves exactly as if each evaluation across axis i was done using identical operations. The definition of distributive axis is as follows: For IMatrixMultiplyLayer: All axes that are not one of the vector or matrix dimensions are distributive axes. For layers that perform reduction: All non-reduction axes are distributive axes. For layers that perform einsum: Let n be the leftmost reduction axis. The axes to the left of n are distributive axes.

8.1.3.8 CausalMaskKind

```
enum class nvinfer1::CausalMaskKind : int32_t [strong]
```

Enumerates the causal mask alignment orientation for the attention.

When s_q == s_kv, both kUPPER_LEFT and kLOWER_RIGHT produce identical triangular masks. When s_q != s_kv (e.g., during LLM generation where s_q=1 and s_kv grows):

- kUPPER_LEFT: Diagonal anchored at top-left corner (j <= i). Query tokens attend only to the earliest cache positions.
- kLOWER_RIGHT: Diagonal anchored at bottom-right corner (j <= i + (s_kv - s_q)). Query tokens attend to all preceding context, which is the correct behavior for autoregressive generation.

See also

IAttention::setCausalKind(), IAttention::getCausalKind()

Enumerator

kNONE	No causal masking applied.
kUPPER_LEFT	Diagonal anchored at top-left corner (legacy default when causal=true).
kLOWER_RIGHT	Diagonal anchored at bottom-right corner (decode-aligned semantics).

8.1.3.9 CollectiveOperation

```
enum class nvinfer1::CollectiveOperation : int32_t [strong]
```

Enumerates the collective operations that may be performed by a DistCollective layer.

See also

IDistCollectiveLayer

Enumerator

kALL_REDUCE	All reduce.
kALL_GATHER	All gather.
kBROADCAST	Broadcast.
kREDUCE	Reduce.
kREDUCE_SCATTER	Reduce scatter.
kALL_TO_ALL	All-to-all exchange.
kGATHER	Gather to root.
kSCATTER	Scatter from root.

8.1.3.10 CumulativeOperation

```
enum class nvinfer1::CumulativeOperation : int32_t [strong]
```

Enumerates the cumulative operations that may be performed by a Cumulative layer.

The table shows the initial value of each Cumulative operation.

Operation	kFLOAT, kHALF, kBF16	kINT32, kINT64
kSUM	+0.0	0

Enumerator

kSUM	Calculate cumulative sum.

8.1.3.11 DataType

```
enum class nvinfer1::DataType : int32_t [strong]
```

The type of weights and tensors. The datatypes other than kBOOL, kINT32, and kINT64 are "activation datatypes," as they often represent values corresponding to inference results.

Enumerator

kFLOAT	32-bit floating point format.
kHALF	IEEE 16-bit floating-point format – has a 5 bit exponent and 11 bit significand.
kINT8	Signed 8-bit integer representing a quantized floating-point value.
kINT32	Signed 32-bit integer format.
kBOOL	8-bit boolean. 0 = false, 1 = true, other values undefined.
kUINT8	Unsigned 8-bit integer format. Cannot be used to represent quantized floating-point values. Use the IdentityLayer to convert kUINT8 network-level inputs to {kFLOAT, kHALF} prior to use with other TensorRT layers, or to convert intermediate output before kUINT8 network-level outputs from {kFLOAT, kHALF} to kUINT8. kUINT8 conversions are only supported for {kFLOAT, kHALF}. kUINT8 to {kFLOAT, kHALF} conversion will convert the integer values to equivalent floating point values. {kFLOAT, kHALF} to kUINT8 conversion will convert the floating point values to integer values by truncating towards zero. This conversion has undefined behavior for floating point values outside the range [0.0F, 256.0F) after truncation. kUINT8 conversions are not supported for {kINT8, kINT32, kBOOL}.
kFP8	Signed 8-bit floating point with 1 sign bit, 4 exponent bits, 3 mantissa bits, and exponent-bias 7.
kBF16	Brain float – has an 8 bit exponent and 8 bit significand.
kINT64	Signed 64-bit integer type.
kINT4	Signed 4-bit integer type.
kFP4	4-bit floating point type 1 bit sign, 2 bit exponent, 1 bit mantissa
kE8M0	Unsigned representation of exponent-only 8-bit floating point type for quantization scales.

8.1.3.12 DeviceType

```
enum class nvinfer1::DeviceType : int32_t [strong]
```

The device that this layer/network will execute on.

Enumerator

kGPU	GPU Device.
kDLA	DLA Core.

8.1.3.13 DimensionOperation

```
enum class nvinfer1::DimensionOperation : int32_t [strong]
```

An operation on two IDimensionExpr, which represent integer expressions used in dimension computations.

For example, given two IDimensionExpr x and y and an IExprBuilder& eb, eb.operation(DimensionOperation::kSUM, x, y) creates a representation of x+y.

See also

IDimensionExpr, IExprBuilder

Enumerator

kSUM	Sum of the two operands.
kPROD	Product of the two operands.
kMAX	Maximum of the two operands.
kMIN	Minimum of the two operands.
kSUB	Substract the second element from the first.
kEQUAL	1 if operands are equal, 0 otherwise.
kLESS	1 if first operand is less than second operand, 0 otherwise.
kFLOOR_DIV	Floor division of the first element by the second.
kCEIL_DIV	Division rounding up.

8.1.3.14 ElementWiseOperation

```
enum class nvinfer1::ElementWiseOperation : int32_t [strong]
```

Enumerates the binary operations that may be performed by an ElementWise layer.

Operations kAND, kOR, and kXOR must have inputs of DataType::kBOOL.

All other operations must have inputs of floating-point type, DataType::kINT8, DataType::kINT32, or DataType::kINT64.

See also

IElementWiseLayer

Enumerator

kSUM	Sum of the two elements.
kPROD	Product of the two elements.
kMAX	Maximum of the two elements.
kMIN	Minimum of the two elements.
kSUB	Subtract the second element from the first.
kDIV	Divide the first element by the second.
kPOW	The first element to the power of the second element.
kFLOOR_DIV	Floor division of the first element by the second.
kAND	Logical AND of two elements.
kOR	Logical OR of two elements.
kXOR	Logical XOR of two elements.
kEQUAL	Check if two elements are equal.
kGREATER	Check if element in first tensor is greater than corresponding element in second tensor.
kLESS	Check if element in first tensor is less than corresponding element in second tensor.

8.1.3.15 EngineCapability

```
enum class nvinfer1::EngineCapability : int32_t [strong]
```

List of supported engine capability flows.

The EngineCapability determines the restrictions of a network during build time and what runtime it targets. EngineCapability::kSTANDARD does not provide any restrictions on functionality and the resulting serialized engine can be executed with TensorRT's standard runtime APIs in the nvinfer1 namespace. EngineCapability::kSAFETY provides a restricted subset of network operations that are safety certified and the resulting serialized engine can be executed with TensorRT's safe runtime APIs in the nvinfer2::safe namespace. EngineCapability::kDLA_STANDALONE provides a restricted subset of network operations that are DLA compatible and the resulting serialized engine can be executed using standalone DLA runtime APIs. See sampleCudla for an example of integrating cuDLA APIs with TensorRT APIs.

Enumerator

kSTANDARD	Standard: TensorRT flow without targeting the safety runtime. This flow supports both DeviceType::kGPU and DeviceType::kDLA.
kSAFETY	Safety: TensorRT flow with restrictions targeting the safety runtime. See safety documentation for list of supported layers and formats. This flow supports only DeviceType::kGPU. This flag is only supported in NVIDIA Drive(R) products.
kDLA_STANDALONE	DLA Standalone: TensorRT flow with restrictions targeting external, to TensorRT, DLA runtimes. See DLA documentation for list of supported layers and formats. This flow supports only DeviceType::kDLA.

8.1.3.16 EngineStat

```
enum class nvinfer1::EngineStat : int32_t [strong]
```

The kind of engine statistics that queried from the ICudaEngine.

See also

ICudaEngine::getEngineStat()

BuilderFlag::kSTRIP_PLAN

Enumerator

kTOTAL_WEIGHTS_SIZE	Return the total weight size in bytes.
kSTRIPPED_WEIGHTS_SIZE	Return the stripped weight size in bytes for engines built with BuilderFlag::kSTRIP_PLAN.

8.1.3.17 ErrorCode

```
enum class nvinfer1::ErrorCode : int32_t [strong]
```

Error codes that can be returned by TensorRT during execution.

Enumerator

kSUCCESS	Execution completed successfully.
kUNSPECIFIED_ERROR	An error that does not fall into any other category. This error is included for forward compatibility.
kINTERNAL_ERROR	A non-recoverable TensorRT error occurred. TensorRT is in an invalid internal state when this error is emitted and any further calls to TensorRT will result in undefined behavior.
kINVALID_ARGUMENT	An argument passed to the function is invalid in isolation. This is a violation of the API contract.
kINVALID_CONFIG	An error occurred when comparing the state of an argument relative to other arguments. For example, the dimensions for concat differ between two tensors outside of the channel dimension. This error is triggered when an argument is correct in isolation, but not relative to other arguments. This is to help to distinguish from the simple errors from the more complex errors. This is a violation of the API contract.
kFAILED_ALLOCATION	An error occurred when performing an allocation of memory on the host or the device. A memory allocation error is normally fatal, but in the case where the application provided its own memory allocation routine, it is possible to increase the pool of available memory and resume execution.
kFAILED_INITIALIZATION	One, or more, of the components that TensorRT relies on did not initialize correctly. This is a system setup issue.

Enumerator

kFAILED_EXECUTION	An error occurred during execution that caused TensorRT to end prematurely, either an asynchronous error, user cancellation, or other execution errors reported by CUDA/DLA. In a dynamic system, the data can be thrown away and the next frame can be processed or execution can be retried. This is either an execution error or a memory error.
kFAILED_COMPUTATION	An error occurred during execution that caused the data to become corrupted, but execution finished. Examples of this error are NaN squashing or integer overflow. In a dynamic system, the data can be thrown away and the next frame can be processed or execution can be retried. This is either a data corruption error, an input error, or a range error. This is not used in safety but may be used in standard.
kINVALID_STATE	TensorRT was put into a bad state by incorrect sequence of function calls. An example of an invalid state is specifying a layer to be DLA only without GPU fallback, and that layer is not supported by DLA. This can occur in situations where a service is optimistically executing networks for multiple different configurations without checking proper error configurations, and instead throwing away bad configurations caught by TensorRT. This is a violation of the API contract, but can be recoverable. Example of a recovery: GPU fallback is disabled and conv layer with large filter(63x63) is specified to run on DLA. This will fail due to DLA not supporting the large kernel size. This can be recovered by either turning on GPU fallback or setting the layer to run on the GPU.
kUNSUPPORTED_STATE	An error occurred due to the network not being supported on the device due to constraints of the hardware or system. An example is running an unsafe layer in a safety certified context, or a resource requirement for the current network is greater than the capabilities of the target device. The network is otherwise correct, but the network and hardware combination is problematic. This can be recoverable. Examples: • Scratch space requests larger than available device memory and can be recovered by increasing allowed workspace size. • Tensor size exceeds the maximum element count and can be recovered by reducing the maximum batch size.

8.1.3.18 ExecutionContextAllocationStrategy

```
enum class nvinfer1::ExecutionContextAllocationStrategy : int32_t [strong]
```

Different memory allocation behaviors for IExecutionContext.

IExecutionContext requires a block of device memory for internal activation tensors during inference. The user can either let the execution context manage the memory in various ways or allocate the memory themselves.

See also

ICudaEngine::createExecutionContext()

IExecutionContext::setDeviceMemory()

Enumerator

kSTATIC	Default static allocation with the maximum size across all profiles.
kON_PROFILE_CHANGE	Reallocate for a profile when it's selected.
kUSER_MANAGED	The user supplies custom allocation to the execution context.

8.1.3.19 FillOperation

```
enum class nvinfer1::FillOperation : int32_t [strong]
```

Enumerates the tensor fill operations that may performed by a fill layer.

See also

IFillLayer

Enumerator

kLINSPACE	Compute each value via an affine function of its indices. For example, suppose the parameters for the IFillLayer are: • Dimensions = [3,4] • Alpha = 1 • Beta = [100,10] Element [i,j] of the output is Alpha + Beta[0]∗i + Beta[1]∗j. Thus the output matrix is: `1 11 21 31` 101 111 121 131 201 211 221 231 A static beta b is implicitly a 1D tensor, i.e. Beta = [b]. Output type must be INT32, INT64, or FLOAT.
kRANDOM_UNIFORM	Randomly draw values from a uniform distribution. Output type must be FLOAT or HALF.
kRANDOM_NORMAL	Randomly draw values from a normal distribution. Output type must be FLOAT or HALF.

8.1.3.20 GatherMode

```
enum class nvinfer1::GatherMode : int32_t [strong]
```

Control form of IGatherLayer.

See also

IGatherLayer

Enumerator

kDEFAULT	Similar to ONNX Gather.
kELEMENT	Similar to ONNX GatherElements.
kND	Similar to ONNX GatherND.

8.1.3.21 HardwareCompatibilityLevel

```
enum class nvinfer1::HardwareCompatibilityLevel : int32_t [strong]
```

Describes requirements of compatibility with GPU architectures other than that of the GPU on which the engine was built.

Warning

Note that compatibility with future hardware depends on CUDA forward compatibility support.

Enumerator

kNONE	Do not require hardware compatibility with GPU architectures other than that of the GPU on which the engine was built.
kAMPERE_PLUS	Require that the engine is compatible with Ampere and newer GPUs. This will limit the combined usage of driver reserved and backend kernel max shared memory to 48KiB, may reduce the number of available tactics for each layer, and may prevent some fusions from occurring. Thus this can decrease the performance, especially for tf32 models. This option will disable cuDNN, cuBLAS, and cuBLASLt as tactic sources. This option is only supported for engines built on NVIDIA Ampere and later GPUs. The driver reserved shared memory can be queried from cuDeviceGetAttribute(&reservedShmem, CU_DEVICE_ATTRIBUTE_↩ RESERVED_SHARED_MEMORY_PER_BLOCK).
kSAME_COMPUTE_CAPABILITY	Require that the engine is compatible with GPUs that have the same Compute Capability (https://developer.nvidia.com/cuda-gpus) as the one it was built on. This may decrease the performance compared to an engine with no compatibility. This option will disable cuDNN, cuBLAS, and cuBLASLt as tactic sources. This option is only supported for engines built on NVIDIA Turing and later GPUs.

8.1.3.22 InterpolationMode

```
enum class nvinfer1::InterpolationMode : int32_t [strong]
```

Enumerates various modes of interpolation.

Enumerator

kNEAREST	ND (0 < N <= 8) nearest neighbor resizing.
kLINEAR	Supports linear (1D), bilinear (2D), and trilinear (3D) interpolation.
kCUBIC	Supports bicubic (2D) interpolation.

8.1.3.23 KVCacheMode

```
enum class nvinfer1::KVCacheMode : int32_t [strong]
```

Enumerates the KVCache modes that may be performed by a KVCacheUpdate layer.

Enumerator

kLINEAR	Linear mode.

8.1.3.24 LayerInformationFormat

```
enum class nvinfer1::LayerInformationFormat : int32_t [strong]
```

The format in which the IEngineInspector prints the layer information.

See also

IEngineInspector::getLayerInformation(), IEngineInspector::getEngineInformation()

Enumerator

kONELINE	Print layer information in one line per layer.
kJSON	Print layer information in JSON format.

8.1.3.25 LayerType

```
enum class nvinfer1::LayerType : int32_t [strong]
```

The type values of layer classes.

See also

ILayer::getType()

Enumerator

kCONVOLUTION	Convolution layer.
kCAST	Cast layer.
kACTIVATION	Activation layer.
kPOOLING	Pooling layer.
kLRN	LRN layer.
kSCALE	Scale layer.
kSOFTMAX	SoftMax layer.
kDECONVOLUTION	Deconvolution layer.
kCONCATENATION	Concatenation layer.
kELEMENTWISE	Elementwise layer.
kPLUGIN	Plugin layer.
kUNARY	UnaryOp operation Layer.
kPADDING	Padding layer.
kSHUFFLE	Shuffle layer.
kREDUCE	Reduce layer.
kTOPK	TopK layer.
kGATHER	Gather layer.
kMATRIX_MULTIPLY	Matrix multiply layer.
kRAGGED_SOFTMAX	Ragged softmax layer.
kCONSTANT	Constant layer.
kIDENTITY	Identity layer.
kPLUGIN_V2	PluginV2 layer.
kSLICE	Slice layer.
kSHAPE	Shape layer.
kPARAMETRIC_RELU	Parametric ReLU layer.
kRESIZE	Resize Layer.
kTRIP_LIMIT	Loop Trip limit layer.
kRECURRENCE	Loop Recurrence layer.
kITERATOR	Loop Iterator layer.
kLOOP_OUTPUT	Loop output layer.
kSELECT	Select layer.
kFILL	Fill layer.
kQUANTIZE	Quantize layer.

Enumerator

kDEQUANTIZE	Dequantize layer.
kCONDITION	Condition layer.
kCONDITIONAL_INPUT	Conditional Input layer.
kCONDITIONAL_OUTPUT	Conditional Output layer.
kSCATTER	Scatter layer.
kEINSUM	Einsum layer.
kASSERTION	Assertion layer.
kONE_HOT	OneHot layer.
kNON_ZERO	NonZero layer.
kGRID_SAMPLE	Grid sample layer.
kNMS	NMS layer.
kREVERSE_SEQUENCE	Reverse sequence layer.
kNORMALIZATION	Normalization layer.
kPLUGIN_V3	PluginV3 layer.
kSQUEEZE	Squeeze Layer.
kUNSQUEEZE	Unsqueeze Layer.
kCUMULATIVE	Cumulative layer.
kDYNAMIC_QUANTIZE	Dynamic Quantize layer.
kATTENTION_INPUT	Attention Input.
kATTENTION_OUTPUT	Attention Output.
kROTARY_EMBEDDING	Rotary Embedding layer.
kKVCACHE_UPDATE	KV Cache Update layer.
kMOE	MoE layer.
kDIST_COLLECTIVE	DistCollective layer.

8.1.3.26 LoopOutput

```
enum class nvinfer1::LoopOutput : int32_t [strong]
```

Enum that describes kinds of loop outputs.

Enumerator

kLAST_VALUE	Output value is value of tensor for last iteration.
kCONCATENATE	Output value is concatenation of values of tensor for each iteration, in forward order.
kREVERSE	Output value is concatenation of values of tensor for each iteration, in reverse order.

8.1.3.27 MatrixOperation

```
enum class nvinfer1::MatrixOperation : int32_t [strong]
```

Enumerates the operations that may be performed on a tensor by IMatrixMultiplyLayer before multiplication.

Enumerator

kNONE	Treat x as a matrix if it has two dimensions, or as a collection of matrices if x has more than two dimensions, where the last two dimensions are the matrix dimensions. x must have at least two dimensions.
kTRANSPOSE	Like kNONE, but transpose the matrix dimensions.
kVECTOR	Treat x as a vector if it has one dimension, or as a collection of vectors if x has more than one dimension. x must have at least one dimension. The first input tensor with dimensions [M,K] used with MatrixOperation::kVECTOR is equivalent to a tensor with dimensions [M, 1, K] with MatrixOperation::kNONE, i.e. is treated as M row vectors of length K, or dimensions [M, K, 1] with MatrixOperation::kTRANSPOSE. The second input tensor with dimensions [M,K] used with MatrixOperation::kVECTOR is equivalent to a tensor with dimensions [M, K, 1] with MatrixOperation::kNONE, i.e. is treated as M column vectors of length K, or dimensions [M, 1, K] with MatrixOperation::kTRANSPOSE.

8.1.3.28 MemoryPoolType

```
enum class nvinfer1::MemoryPoolType : int32_t [strong]
```

The type for memory pools used by TensorRT.

See also

IBuilderConfig::setMemoryPoolLimit, IBuilderConfig::getMemoryPoolLimit

Enumerator

kWORKSPACE	kWORKSPACE is used by TensorRT to store intermediate buffers within an operation. This defaults to max device memory. Set to a smaller value to restrict tactics whose workspace usage exceeds the threshold en masse.
kDLA_MANAGED_SRAM	kDLA_MANAGED_SRAM is a fast software managed RAM used by DLA to communicate within a layer. The size of this pool must be at least 4 KiB and must be a power of 2. This defaults to 1 MiB. Orin has capacity of 1 MiB per core.
kDLA_LOCAL_DRAM	kDLA_LOCAL_DRAM is host RAM used by DLA to share intermediate tensor data across operations. The size of this pool must be at least 4 KiB and must be a power of 2. This defaults to 1 GiB.
kDLA_GLOBAL_DRAM	kDLA_GLOBAL_DRAM is host RAM used by DLA to store weights and metadata for execution. The size of this pool must be at least 4 KiB and must be a power of 2. This defaults to 512 MiB.

Enumerator

kTACTIC_DRAM	kTACTIC_DRAM is the device DRAM used by the optimizer to run tactics. On embedded devices, where host and device memory are unified, this includes all host memory required by TensorRT to build the network up to the point of each memory allocation. This defaults to 75% of totalGlobalMem as reported by cudaGetDeviceProperties when cudaGetDeviceProperties.embedded is true, and 100% otherwise.
kTACTIC_SHARED_MEMORY	kTACTIC_SHARED_MEMORY defines the maximum sum of shared memory reserved by the driver and used for executing CUDA kernels. Adjust this value to restrict tactics that exceed the specified threshold en masse. The default value is device max capability. This value must be less than 1GiB. The driver reserved shared memory can be queried from cuDeviceGetAttribute(&reservedShmem, CU_DEVICE_ATTRIBUTE_↩ RESERVED_SHARED_MEMORY_PER_BLOCK). Updating this flag will override the shared memory limit set by HardwareCompatibilityLevel, which defaults to 48KiB - reservedShmem.

8.1.3.29 MoEActType

```
enum class nvinfer1::MoEActType :  int32_t  [strong]
```

Enumerates the activation type for the MoE layer.

Enumerator

kNONE	
kSILU	

8.1.3.30 NetworkDefinitionCreationFlag

```
enum class nvinfer1::NetworkDefinitionCreationFlag :  int32_t  [strong]
```

List of immutable network properties expressed at network creation time. NetworkDefinitionCreationFlag is used with createNetworkV2() to specify immutable properties of the network.

See also

IBuilder::createNetworkV2

Enumerator

kSTRONGLY_TYPED	Mark the network to be strongly typed. Every tensor in the network has a data type defined in the network following only type inference rules and the inputs/operator annotations. Setting layer precision and layer output types is not allowed, and the network output types will be inferred based on the input types and the type inference rules. **Deprecated** Deprecated in TensorRT 11.0. Strongly typed mode is always enabled. This flag is retained for API compatibility but is ignored.
kPREFER_JIT_PYTHON_PLUGINS	If set, for a Python plugin with both AOT and JIT implementations, the JIT implementation will be used. Any plugin-specific JIT/↩ AOT specification may override this. Cannot be used in conjunction with NetworkDefinitionCreationFlag::kPREFER_AOT_PYTHON_PLUGINS.
kPREFER_AOT_PYTHON_PLUGINS	If set, for a Python plugin with both AOT and JIT implementations, the AOT implementation will be used. Any plugin-specific JIT/AOT specification may override this. Cannot be used in conjunction with NetworkDefinitionCreationFlag::kPREFER_JIT_PYTHON_PLUGINS.

8.1.3.31 OptProfileSelector

```
enum class nvinfer1::OptProfileSelector :  int32_t  [strong]
```

When setting or querying optimization profile parameters (such as shape tensor inputs or dynamic dimensions), select whether we are interested in the minimum, optimum, or maximum values for these parameters. The minimum and maximum specify the permitted range that is supported at runtime, while the optimum value is used for the kernel selection. This should be the ”typical” value that is expected to occur at runtime.

See also

IOptimizationProfile::setDimensions(), IOptimizationProfile::setShapeValuesV2()

Enumerator

kMIN	This is used to set or get the minimum permitted value for dynamic dimensions etc.
kOPT	This is used to set or get the value that is used in the optimization (kernel selection).
kMAX	This is used to set or get the maximum permitted value for dynamic dimensions etc.

8.1.3.32 PaddingMode

```
enum class nvinfer1::PaddingMode :  int32_t  [strong]
```

Enumerates the modes of padding to perform in convolution, deconvolution and pooling layer, padding mode takes precedence if setPaddingMode() and setPrePadding() are also used.

There are two padding styles, EXPLICIT and SAME with each style having two variants. The EXPLICIT style determine if the final sampling location is used or not. The SAME style determine if the asymmetry in the padding is on the pre or post padding.

```
Shorthand:
    I = dimensions of input image.
    B = prePadding, before the image data.
    A = postPadding, after the image data.
    P = delta between input and output
    S = stride
    F = filter
    O = output
    D = dilation
    M = I + B + A ; The image data plus any padding
    DK = 1 + D * (F - 1)
```

Formulas for Convolution:

- EXPLICIT_ROUND_DOWN:
  ```
  O = floor((M - DK) / S) + 1
  ```
- EXPLICIT_ROUND_UP:
  ```
  O = ceil((M - DK) / S) + 1
  ```
- SAME_UPPER:
  ```
  O = ceil(I / S)
  P = floor((I - 1) / S) * S + DK - I;
  B = floor(P / 2)
  A = P - B
  ```
- SAME_LOWER:
  ```
  O = ceil(I / S)
  P = floor((I - 1) / S) * S + DK - I;
  A = floor(P / 2)
  B = P - A
  ```

Formulas for Deconvolution:

- EXPLICIT_ROUND_DOWN:
- EXPLICIT_ROUND_UP:
  ```
  O = (I - 1) * S + DK - (B + A)
  ```
- SAME_UPPER:
  ```
  O = min(I * S, (I - 1) * S + DK)
  P = max(DK - S, 0)
  B = floor(P / 2)
  A = P - B
  ```
- SAME_LOWER:
  ```
  O = min(I * S, (I - 1) * S + DK)
  P = max(DK - S, 0)
  A = floor(P / 2)
  B = P - A
  ```

Formulas for Pooling:

- EXPLICIT_ROUND_DOWN:
  ```
  O = floor((M - F) / S) + 1
  ```
- EXPLICIT_ROUND_UP:
  ```
  O = ceil((M - F) / S) + 1
  ```

- SAME_UPPER:
```
O = ceil(I / S)
P = floor((I - 1) / S) * S + F - I;
B = floor(P / 2)
A = P - B
```

- SAME_LOWER:
```
O = ceil(I / S)
P = floor((I - 1) / S) * S + F - I;
A = floor(P / 2)
B = P - A
```

Pooling Example 1:
```
Given I = {6, 6}, B = {3, 3}, A = {2, 2}, S = {2, 2}, F = {3, 3}.  What is O?
(B, A can be calculated for SAME_UPPER and SAME_LOWER mode)
```

- EXPLICIT_ROUND_DOWN:
```
Computation:
    M = {6, 6} + {3, 3} + {2, 2} ==> {11, 11}
    O ==> floor((M - F) / S) + 1
      ==> floor(({11, 11} - {3, 3}) / {2, 2}) + {1, 1}
      ==> floor({8, 8} / {2, 2}) + {1, 1}
      ==> {5, 5}
```

- EXPLICIT_ROUND_UP:
```
Computation:
    M = {6, 6} + {3, 3} + {2, 2} ==> {11, 11}
    O ==> ceil((M - F) / S) + 1
      ==> ceil(({11, 11} - {3, 3}) / {2, 2}) + {1, 1}
      ==> ceil({8, 8} / {2, 2}) + {1, 1}
      ==> {5, 5}
```

  The sample points are $\{0, 2, 4, 6, 8\}$ in each dimension.

- SAME_UPPER:
```
Computation:
    I = {6, 6}
    S = {2, 2}
    O = ceil(I / S) = {3, 3}
    P = floor((I - 1) / S) * S + F - I
        ==> floor(({6, 6} - {1, 1}) / {2, 2}) * {2, 2} + {3, 3} - {6, 6}
        ==> {4, 4} + {3, 3} - {6, 6}
        ==> {1, 1}
    B = floor({1, 1} / {2, 2})
        ==> {0, 0}
    A = {1, 1} - {0, 0}
        ==> {1, 1}
```

- SAME_LOWER:
```
Computation:
    I = {6, 6}
    S = {2, 2}
    O = ceil(I / S) = {3, 3}
    P = floor((I - 1) / S) * S + F - I
      ==> {1, 1}
    A = floor({1, 1} / {2, 2})
      ==> {0, 0}
    B = {1, 1} - {0, 0}
      ==> {1, 1}
```

  The sample pointers are $\{0, 2, 4\}$ in each dimension. SAMPLE_UPPER has $\{O0, O1, O2, pad\}$ in output in each dimension. SAMPLE_LOWER has $\{pad, O0, O1, O2\}$ in output in each dimension.

Pooling Example 2:
```
Given I = {6, 6}, B = {3, 3}, A = {3, 3}, S = {2, 2}, F = {3, 3}.  What is O?
```

Enumerator

kEXPLICIT_ROUND_DOWN	Use explicit padding, rounding output size down.
kEXPLICIT_ROUND_UP	Use explicit padding, rounding output size up.
~~kSAME_UPPER~~	~~Use SAME padding, with prePadding <= postPadding.~~
kSAME_LOWER	Use SAME padding, with prePadding >= postPadding.

8.1.3.33 PluginCapabilityType

```
enum class nvinfer1::PluginCapabilityType : int32_t [strong]
```

Enumerates the different capability types a IPluginV3 object may have.

Enumerator

kCORE	Core capability. Every IPluginV3 object must have this.
kBUILD	Build capability. IPluginV3 objects provided to TensorRT build phase must have this.
kRUNTIME	Runtime capability. IPluginV3 objects provided to TensorRT build and execution phases must have this.

8.1.3.34 PluginCreatorVersion

```
enum class nvinfer1::PluginCreatorVersion : int32_t [strong]
```

Enum to identify version of the plugin creator.

Deprecated Deprecated in TensorRT 10.10. PluginCreatorVersion is used only in relation to plugin creators based off IPluginCreator, which is deprecated.

Enumerator

kV1	IPluginCreator.
kV1_PYTHON	IPluginCreator-based Python plugin creators.

8.1.3.35 PluginFieldType

```
enum class nvinfer1::PluginFieldType : int32_t [strong]
```

The possible field types for custom layer.

Enumerator

kFLOAT16	FP16 field type.
kFLOAT32	FP32 field type.

Enumerator

kFLOAT64	FP64 field type.
kINT8	INT8 field type.
kINT16	INT16 field type.
kINT32	INT32 field type.
kCHAR	char field type.
kDIMS	nvinfer1::Dims field type.
kUNKNOWN	Unknown field type.
kBF16	BF16 field type.
kINT64	INT64 field type.
kFP8	FP8 field type.
kINT4	INT4 field type.
kFP4	FP4 field type.

8.1.3.36 PluginVersion

`enum class nvinfer1::PluginVersion : uint8_t [strong]`

Enumerator

kV2	IPluginV2.
kV2_EXT	IPluginV2Ext.
kV2_IOEXT	IPluginV2IOExt.
kV2_DYNAMICEXT	IPluginV2DynamicExt.
kV2_DYNAMICEXT_PYTHON	IPluginV2DynamicExt-based Python plugins.

8.1.3.37 PoolingType

`enum class nvinfer1::PoolingType : int32_t [strong]`

The type of pooling to perform in a pooling layer.

Enumerator

kMAX	Maximum over elements.
kAVERAGE	Average over elements. If the tensor is padded, the count includes the padding.
kMAX_AVERAGE_BLEND	Blending between max and average pooling: (1-blendFactor)∗maxPool + blendFactor∗avgPool.

8.1.3.38 PreviewFeature

`enum class nvinfer1::PreviewFeature : int32_t [strong]`

Define preview features.

Preview Features have been fully tested but are not yet as stable as other features in TensorRT. They are provided as opt-in features for at least one release.

Enumerator

kALIASED_PLUGIN_IO_10_03	Allows plugin I/O to be aliased when using IPluginV3OneBuildV2
kRUNTIME_ACTIVATION_RESIZE_10_10	Allows IExecutionContext::updateDeviceMemorySizeForShapes to resize runner internal activation memory. Using this feature can reduce runtime memory requirement when the actual input tensor shapes are smaller than the maximum input tensor dimensions.

8.1.3.39 ProfilingVerbosity

`enum class nvinfer1::ProfilingVerbosity : int32_t [strong]`

List of verbosity levels of layer information exposed in NVTX annotations and in IEngineInspector.

See also

IBuilderConfig::setProfilingVerbosity(), IBuilderConfig::getProfilingVerbosity(), IEngineInspector

Enumerator

kLAYER_NAMES_ONLY	Print only the layer names. This is the default setting.
kNONE	Do not print any layer information.
kDETAILED	Print detailed layer information including layer names and layer parameters.

8.1.3.40 ReduceOperation

`enum class nvinfer1::ReduceOperation : int32_t [strong]`

Enumerates the reduce operations that may be performed by a Reduce layer.

The table shows the result of reducing across an empty volume of a given type.

Operation	kFLOAT and kHALF	kINT32	kINT8
kSUM	0	0	0
kPROD	1	1	1
kMAX	negative infinity	INT_MIN	-128
kMIN	positive infinity	INT_MAX	127
kAVG	NaN	0	-128
kNONE	Undefined	Undefined	Undefined

The current version of TensorRT usually performs reduction for kINT8 via kFLOAT or kHALF. The kINT8 values show the quantized representations of the floating-point values.

Note

kNONE is a reduce operation which does not modify the input tensor. This is applicable to Multi-Device mode only, as a reduce operation is not mandatory for certain collective operations. See INetworkDefinition::addDistCollective for more details.

Enumerator

kSUM	Sum of the elements.
kPROD	Product of the elements.
kMAX	Maximum of the elements.
kMIN	Minimum of the elements.
kAVG	Average of the elements.
kNONE	No reduction.

8.1.3.41 ResizeCoordinateTransformation

```
enum class nvinfer1::ResizeCoordinateTransformation : int32_t [strong]
```

The resize coordinate transformation function.

See also

IResizeLayer::setCoordinateTransformation()

Enumerator

kALIGN_CORNERS	Think of each value in the tensor as a unit volume, and the coordinate is a point inside this volume. The coordinate point is drawn as a star `(*)` in the below diagram, and multiple values range has a length. Define `x_origin` as the coordinate of axis x in the input tensor, `x_resized` as the coordinate of axis x in the output tensor, `length_origin` as length of the input tensor in axis x, and `length_resize` as length of the output tensor in axis x. This transformation maps coordinates as: `|<----------length---------->|` `| 0	1	2	3 |` `* * * *` `x_origin = x_resized * (length_origin - 1) / (length_resize - 1)`
kASYMMETRIC	This transformation maps coordinates as: `|<----------length---------->|` `| 0	1	2	3 |` `* * * *` `x_origin = x_resized * (length_origin / length_resize)`
kHALF_PIXEL	This transformation maps coordinates as: `|<----------length---------->|` `| 0	1	2	3 |` `* * * *` `x_origin = (x_resized + 0.5) * (length_origin / length_resize) - 0.5`

8.1.3.42 ResizeRoundMode

`enum class nvinfer1::ResizeRoundMode : int32_t [strong]`

The rounding mode for nearest neighbor resize.

See also

IResizeLayer::setNearestRounding()

Enumerator

kHALF_UP	Round half up.
kHALF_DOWN	Round half down.
kFLOOR	Round to floor.
kCEIL	Round to ceil.

8.1.3.43 ResizeSelector

```
enum class nvinfer1::ResizeSelector : int32_t [strong]
```

The coordinate selector when resize to single pixel output.

See also

IResizeLayer::setSelectorForSinglePixel()

Enumerator

kFORMULA	Use formula to map the original index.
kUPPER	Select the upper left pixel.

8.1.3.44 RuntimePlatform

```
enum class nvinfer1::RuntimePlatform : int32_t [strong]
```

Describes the intended runtime platform (operating system and CPU architecture) for the execution of the TensorRT engine. TensorRT provides support for cross-platform engine compatibility when the target runtime platform is different from the build platform.

Note

The cross-platform engine will not be able to run on the host platform it was built on.

When building a cross-platform engine that also requires version forward compatibility, kEXCLUDE_LEAN_RUNTIME must be set to exclude the target platform lean runtime.

The cross-platform engine might have performance differences compared to the natively built engine on the target platform.

See also

IBuilderConfig::setRuntimePlatform(), IBuilderConfig::getRuntimePlatform()

Enumerator

kSAME_AS_BUILD	No requirement for cross-platform compatibility. The engine constructed by TensorRT can only run on the identical platform it was built on.
kWINDOWS_AMD64	Designates the target platform for engine execution as Windows AMD64 system. Currently this flag can only be enabled when building engines on Linux AMD64 platforms.

8.1.3.45 SampleMode

`enum class nvinfer1::SampleMode : int32_t [strong]`

Controls how ISliceLayer and IGridSample handle out-of-bounds coordinates.

See also

ISliceLayer and IGridSample

Enumerator

kSTRICT_BOUNDS	Fail with error when the coordinates are out of bounds.
kWRAP	Coordinates wrap around periodically.
kCLAMP	Out of bounds indices are clamped to bounds.
kFILL	Use fill input value when coordinates are out of bounds.
kREFLECT	Coordinates reflect. The axis of reflection is the middle of the perimeter pixel and the reflections are repeated indefinitely within the padded regions. Repeats values for a single pixel and throws error for zero pixels.

8.1.3.46 ScaleMode

`enum class nvinfer1::ScaleMode : int32_t [strong]`

Controls how shift, scale and power are applied in a Scale layer.

See also

IScaleLayer

Enumerator

kUNIFORM	Identical coefficients across all elements of the tensor.
kCHANNEL	Per-channel coefficients.
kELEMENTWISE	Elementwise coefficients.

8.1.3.47 ScatterMode

```
enum class nvinfer1::ScatterMode : int32_t [strong]
```

Control form of IScatterLayer.

See also

IScatterLayer

Enumerator

kELEMENT	Similar to ONNX ScatterElements.
kND	Similar to ONNX ScatterND.

8.1.3.48 SeekPosition

```
enum class nvinfer1::SeekPosition : int32_t [strong]
```

Controls the seek mode of IStreamReaderV2.

Enumerator

kSET	From the beginning of the file.
kCUR	From the current position of the file.
kEND	From the tail of the file.

8.1.3.49 SerializationFlag

```
enum class nvinfer1::SerializationFlag : int32_t [strong]
```

List of valid flags that the engine can enable when serializing the bytes.

See also

ISerializationConfig::setFlags(), ISerializationConfig::getFlags()

Enumerator

kEXCLUDE_WEIGHTS	Exclude the weights that can be refitted.
kEXCLUDE_LEAN_RUNTIME	Exclude the lean runtime.
kINCLUDE_REFIT	Remain refittable if originally so.

8.1.3.50 TacticSource

```
enum class nvinfer1::TacticSource : int32_t [strong]
```

List of tactic sources for TensorRT.

See also

TacticSources, IBuilderConfig::setTacticSources(), IBuilderConfig::getTacticSources()

Enumerator

kEDGE_MASK_CONVOLUTIONS	Enables convolution tactics implemented with edge mask tables. These tactics tradeoff memory for performance by consuming additional memory space proportional to the input size. Enabled by default.
kJIT_CONVOLUTIONS	Enables convolution tactics implemented with source-code JIT fusion. The engine building time may increase when this is enabled. Enabled by default.

8.1.3.51 TempfileControlFlag

```
enum class nvinfer1::TempfileControlFlag : int32_t [strong]
```

Flags used to control TensorRT's behavior when creating executable temporary files.

On some platforms the TensorRT runtime may need to create files in a temporary directory or use platform-specific APIs to create files in-memory to load temporary DLLs that implement runtime code. These flags allow the application to explicitly control TensorRT's use of these files. This will preclude the use of certain TensorRT APIs for deserializing and loading lean runtimes.

Enumerator

kALLOW_IN_MEMORY_FILES	Allow creating and loading files in-memory (or unnamed files).
kALLOW_TEMPORARY_FILES	Allow creating and loading named files in a temporary directory on the filesystem. \see IRuntime::setTemporaryDirectory()

8.1.3.52 TensorFormat

```
enum class nvinfer1::TensorFormat : int32_t [strong]
```

Format of the input/output tensors.

This enum is used by both plugins and network I/O tensors.

See also

IPluginV2::supportsFormat(), safe::ICudaEngine::getBindingFormat()

Many of the formats are **vector-major** or **vector-minor**. These formats specify a *vector dimension* and *scalars per vector*. For example, suppose that the tensor has has dimensions [M,N,C,H,W], the vector dimension is C and there are V scalars per vector.

- A **vector-major** format splits the vectorized dimension into two axes in the memory layout. The vectorized dimension is replaced by an axis of length ceil(C/V) and a new dimension of length V is appended. For the example tensor, the memory layout is equivalent to an array with dimensions [M][N][ceil(C/V)][H][W][V]. Tensor coordinate (m,n,c,h,w) maps to array location [m][n][c/V][h][w][c%V].
- A **vector-minor** format moves the vectorized dimension to become the last axis in the memory layout. For the example tensor, the memory layout is equivalent to an array with dimensions [M][N][H][W][ceil(C/V)*V]. Tensor coordinate (m,n,c,h,w) maps array location subscript [m][n][h][w][c].

In interfaces that refer to "components per element", that's the value of V above.

For more information about data formats, see the topic "Data Format Description" located in the TensorRT Developer Guide. https://docs.nvidia.com/deeplearning/tensorrt/latest/inference-library/advanced.html#i-o-formats

Enumerator

kLINEAR	Memory layout is similar to an array in C or C++. The stride of each dimension is the product of the dimensions after it. The last dimension has unit stride. This format supports all TensorRT types. For DLA usage, the tensor sizes are limited to C,H,W in the range [1,8192].
kCHW2	Vector-major format with two scalars per vector. Vector dimension is third to last. This format requires FP16 or BF16 and at least three dimensions.
kHWC8	Vector-minor format with eight scalars per vector. Vector dimension is third to last. This format requires FP16 or BF16 and at least three dimensions.
kCHW4	Vector-major format with four scalars per vector. Vector dimension is third to last. This format requires INT8 and at least three dimensions. For INT8, the length of the vector dimension must be a build-time constant. Deprecated usage: If running on the DLA, this format can be used for acceleration with the caveat that C must be less than or equal to 4. If used as DLA input and the build option kGPU_FALLBACK is not specified, it needs to meet line stride requirement of DLA format. Column stride in bytes must be a multiple of 64 on Orin.

Enumerator

kCHW16	Vector-major format with 16 scalars per vector. Vector dimension is third to last. This format is only supported by DLA and requires FP16 and at least three dimensions. This format maps to the native feature format for FP16, and the tensor sizes are limited to C,H,W in the range [1,8192].
kCHW32	Vector-major format with 32 scalars per vector. Vector dimension is third to last. This format requires INT8, FP32, or FP16 and at least three dimensions. For DLA usage, this format maps to the native feature format for INT8, and the tensor sizes are limited to C,H,W in the range [1,8192].
kDHWC8	Vector-minor format with eight scalars per vector. Vector dimension is fourth to last. This format requires FP16 or BF16 and at least four dimensions.
kCDHWC32	Vector-major format with 32 scalars per vector. Vector dimension is fourth to last. This format requires FP16 or INT8 and at least four dimensions.
kHWC	Vector-minor format where channel dimension is third to last and unpadded. `This format requires either FP32 or UINT8 and at least three dimensions.`
kDLA_LINEAR	DLA planar format. For a tensor with dimension {N, C, H, W}, the W axis always has unit stride. The stride for stepping along the H axis is rounded up to 64 bytes. The memory layout is equivalent to a C array with dimensions [N][C][H][roundUp(W, 64/elementSize)] where elementSize is 2 for FP16 and 1 for Int8, with the tensor coordinates (n, c, h, w) mapping to array subscript [n][c][h][w].
kDLA_HWC4	DLA image format. For a tensor with dimension {N, C, H, W} the C axis always has unit stride. The stride for stepping along the H axis is rounded up to 64 bytes on Orin. C can only be 1, 3 or 4. If C == 1, it will map to grayscale format. If C == 3 or C == 4, it will map to color image format. And if C == 3, the stride for stepping along the W axis needs to be padded to 4 in elements. When C is {1, 3, 4}, then C' is {1, 4, 4} respectively, the memory layout is equivalent to a C array with dimensions [N][H][roundUp(W, 64/C'/elementSize)][C'] on Orin where elementSize is 2 for FP16 and 1 for Int8. The tensor coordinates (n, c, h, w) mapping to array subscript [n][h][w][c].
kHWC16	Vector-minor format with 16 scalars per vector. Vector dimension is third to last. This requires FP16, INT8 or FP8 and at least three dimensions.
kDHWC	Vector-minor format with one scalar per vector. Vector dimension is fourth to last. This format requires FP32 and at least four dimensions.

8.1.3.53 TensorIOMode

`enum class nvinfer1::TensorIOMode : int32_t [strong]`

Definition of tensor IO Mode.

Enumerator

kNONE	Tensor is not an input or output.
kINPUT	Tensor is input to the engine.
kOUTPUT	Tensor is output by the engine.

8.1.3.54 TensorLocation

```
enum class nvinfer1::TensorLocation : int32_t [strong]
```

The location for tensor data storage, device or host.

Enumerator

kDEVICE	Data stored on device.
kHOST	Data stored on host.

8.1.3.55 TensorRTPhase

```
enum class nvinfer1::TensorRTPhase : int32_t [strong]
```

Indicates a phase of operation of TensorRT.

Enumerator

kBUILD	Build phase of TensorRT.
kRUNTIME	Execution phase of TensorRT.

8.1.3.56 TilingOptimizationLevel

```
enum class nvinfer1::TilingOptimizationLevel : int32_t [strong]
```

Define the optimization levels for Tiling.

TensorRT will try tiling optimization for on-chip caching if non-zero level is set. This level determines how much effort TensorRT would take to find a better solution for performance.

Enumerator

kNONE	Do not apply any tiling strategy.
kFAST	Use a fast algorithm and heuristic based strategy. Slightly increases engine build time.
kMODERATE	Increase search space and use a mixed heuristic/profiling strategy. Moderately increases engine build time.
kFULL	Increase search space even wider. Significantly increases engine build time.

8.1.3.57 TopKOperation

`enum class nvinfer1::TopKOperation : int32_t [strong]`

Enumerates the operations that may be performed by a TopK layer.

Enumerator

kMAX	Maximum of the elements.
kMIN	Minimum of the elements.

8.1.3.58 TripLimit

`enum class nvinfer1::TripLimit : int32_t [strong]`

Enum that describes kinds of trip limits.

Enumerator

kCOUNT	Tensor is a scalar of type kINT32 or kINT64 that contains the trip count.
kWHILE	Tensor is a scalar of type kBOOL. Loop terminates when value is false.

8.1.3.59 UnaryOperation

`enum class nvinfer1::UnaryOperation : int32_t [strong]`

Enumerates the unary operations that may be performed by a Unary layer.

Operations kNOT must have inputs of DataType::kBOOL.

Operation kSIGN and kABS must have inputs of floating-point type, DataType::kINT8, DataType::kINT32 or DataType::kINT64.

Operation kISINF must have inputs of floating-point type.

All other operations must have inputs of floating-point type.

See also

IUnaryLayer

Enumerator

kEXP	Exponentiation.
kLOG	Log (base e).
kSQRT	Square root.
kRECIP	Reciprocal.
kABS	Absolute value.
kNEG	Negation.
kSIN	Sine.
kCOS	Cosine.
kTAN	Tangent.
kSINH	Hyperbolic sine.
kCOSH	Hyperbolic cosine.
kASIN	Inverse sine.
kACOS	Inverse cosine.
kATAN	Inverse tangent.
kASINH	Inverse hyperbolic sine.
kACOSH	Inverse hyperbolic cosine.
kATANH	Inverse hyperbolic tangent.
kCEIL	Ceiling.
kFLOOR	Floor.
kERF	Gauss error function.
kNOT	Logical NOT.
kSIGN	Sign, If input > 0, output 1; if input < 0, output -1; if input == 0, output 0.
kROUND	Round to nearest even for floating-point data type.
kISINF	Return true if input value equals +/- infinity for floating-point data type.
kISNAN	Return true if input value is a NaN for floating-point data type.

8.1.3.60 WeightsRole

```
enum class nvinfer1::WeightsRole : int32_t [strong]
```

How a layer uses particular Weights.

The power weights of an IScaleLayer are omitted. Refitting those is not supported.

Enumerator

kKERNEL	kernel for IConvolutionLayer or IDeconvolutionLayer
kBIAS	bias for IConvolutionLayer or IDeconvolutionLayer
kSHIFT	shift part of IScaleLayer
kSCALE	scale part of IScaleLayer
kCONSTANT	weights for IConstantLayer
kANY	Any other weights role.

8.1.4 Function Documentation

8.1.4.1 EnumMax()

```
template<typename T >
constexpr int32_t nvinfer1::EnumMax ( )  [constexpr], [noexcept]
```

One greater than the maximum value of enumeration type T. For example, if the highest enumerator in T has value 5, then EnumMax<T>() returns 6.

8.1.4.2 getBuilderPluginRegistry()

```
TENSORRTAPI nvinfer1::IPluginRegistry * nvinfer1::getBuilderPluginRegistry (
            nvinfer1::EngineCapability capability )  [noexcept]
```

Return the plugin registry for building a Standard engine, or nullptr if no registry exists.

Also return nullptr if the input argument is not EngineCapability::kSTANDARD. Engine capabilities EngineCapability::kSTANDARD and EngineCapability::kSAFETY have distinct plugin registries. Use IPluginRegistry::registerCreator from the registry to register plugins. Plugins registered in a registry associated with a specific engine capability are only available when building engines with that engine capability.

There is no plugin registry for EngineCapability::kDLA_STANDALONE.

8.1.4.3 getBuilderSafePluginRegistry()

```
TRT_DEPRECATED_API nvinfer1::safe::IPluginRegistry * nvinfer1::getBuilderSafePluginRegistry (
            nvinfer1::EngineCapability capability )  [noexcept]
```

Return the plugin registry for building a Safety engine, or nullptr if no registry exists.

Also return nullptr if the input argument is not EngineCapability::kSAFETY. When building a Standard engine, use nvinfer1::getBuilderPluginRegistry(). Use safe::IPluginRegistry::registerCreator from the registry to register plugins.

8.1.4.4 setInternalLibraryPath()

```
TENSORRTAPI bool nvinfer1::setInternalLibraryPath (
            AsciiChar const * path )  [noexcept]
```

Set a custom directory path for loading internal TensorRT libraries when building engines.

Parameters

path	The path to prepend to internal library names.

Returns

true if the path was successfully set, updated, or reset (by passing empty string). false if it could not be set due to null pointer or an invalid path.

This API take effect globally for all engines built by the current process.

8.2 nvinfer1::anonymous_namespace{NvInfer.h} Namespace Reference

Functions

- IBuilder * createInferBuilder (ILogger &logger) noexcept

  Create an instance of an IBuilder class.

8.2.1 Function Documentation

8.2.1.1 createInferBuilder()

```
IBuilder * nvinfer1::anonymous_namespace{NvInfer.h}::createInferBuilder (
            ILogger & logger )  [inline], [noexcept]
```

Create an instance of an IBuilder class.

Parameters

logger	The logging class for the builder.

unnamed namespace avoids linkage surprises when linking objects built with different versions of this header.

8.3 nvinfer1::anonymous_namespace{NvInferRuntime.h} Namespace Reference

Functions

- IRuntime * createInferRuntime (ILogger &logger) noexcept

Create an instance of an IRuntime class.

- IRefitter * createInferRefitter (ICudaEngine &engine, ILogger &logger) noexcept

  Create an instance of an IRefitter class.

8.3.1 Function Documentation

8.3.1.1 createInferRefitter()

```
IRefitter * nvinfer1::anonymous_namespace{NvInferRuntime.h}::createInferRefitter (
            ICudaEngine & engine,
            ILogger & logger )  [inline], [noexcept]
```

Create an instance of an IRefitter class.

Parameters

engine	The engine class for the refitter.
logger	The logging class for the refitter.

8.3.1.2 createInferRuntime()

```
IRuntime * nvinfer1::anonymous_namespace{NvInferRuntime.h}::createInferRuntime (
            ILogger & logger )  [inline], [noexcept]
```

Create an instance of an IRuntime class.

Parameters

logger	The logging class for the runtime.

8.4 nvinfer1::impl Namespace Reference

Classes

- struct EnumMaxImpl

  Declaration of EnumMaxImpl struct to store the exclusive upper bound of an enumeration type.
- struct EnumMaxImpl< ActivationType >
- struct EnumMaxImpl< AllocatorFlag >

Maximum number of elements in AllocatorFlag enum.

- struct EnumMaxImpl< APILanguage >

  One greater than the maximum value of APILanguage enum.
- struct EnumMaxImpl< AttentionIOForm >
- struct EnumMaxImpl< AttentionNormalizationOp >
- struct EnumMaxImpl< BoundingBoxFormat >
- struct EnumMaxImpl< BuilderFlag >
- struct EnumMaxImpl< CausalMaskKind >
- struct EnumMaxImpl< CollectiveOperation >
- struct EnumMaxImpl< CumulativeOperation >

  Maximum number of elements in CumulativeOperation enum.
- struct EnumMaxImpl< DataType >

  One greater than the maximum value of DataType enum.
- struct EnumMaxImpl< DeviceType >

  Maximum number of elements in DeviceType enum.
- struct EnumMaxImpl< DimensionOperation >

  Maximum number of elements in DimensionOperation enum.
- struct EnumMaxImpl< ElementWiseOperation >
- struct EnumMaxImpl< EngineCapability >

  Maximum number of elements in EngineCapability enum.
- struct EnumMaxImpl< EngineStat >

  Maximum number of engine statistic kinds in EngineStat enum.
- struct EnumMaxImpl< ErrorCode >

  One greater than the maximum value of ErrorCode enum.
- struct EnumMaxImpl< ExecutionContextAllocationStrategy >
- struct EnumMaxImpl< FillOperation >
- struct EnumMaxImpl< GatherMode >
- struct EnumMaxImpl< HardwareCompatibilityLevel >
- struct EnumMaxImpl< ILogger::Severity >

  Maximum number of elements in ILogger::Severity enum.
- struct EnumMaxImpl< InterpolationMode >
- struct EnumMaxImpl< KVCacheMode >
- struct EnumMaxImpl< LayerInformationFormat >

  Maximum number of layer information formats in LayerInformationFormat enum.
- struct EnumMaxImpl< LayerType >
- struct EnumMaxImpl< LoopOutput >
- struct EnumMaxImpl< MatrixOperation >
- struct EnumMaxImpl< MemoryPoolType >
- struct EnumMaxImpl< MoEActType >
- struct EnumMaxImpl< NetworkDefinitionCreationFlag >
- struct EnumMaxImpl< OptProfileSelector >

  Number of different values of OptProfileSelector enum.
- struct EnumMaxImpl< PaddingMode >
- struct EnumMaxImpl< PoolingType >
- struct EnumMaxImpl< PreviewFeature >
- struct EnumMaxImpl< ProfilingVerbosity >

  Maximum number of profile verbosity levels in ProfilingVerbosity enum.
- struct EnumMaxImpl< ReduceOperation >
- struct EnumMaxImpl< ResizeCoordinateTransformation >

- struct EnumMaxImpl< ResizeRoundMode >
- struct EnumMaxImpl< ResizeSelector >
- struct EnumMaxImpl< RuntimePlatform >
- struct EnumMaxImpl< SampleMode >
- struct EnumMaxImpl< ScaleMode >
- struct EnumMaxImpl< ScatterMode >
- struct EnumMaxImpl< SerializationFlag >

  Maximum number of serialization flags in SerializationFlag enum.
- struct EnumMaxImpl< TacticSource >

  Maximum number of tactic sources in TacticSource enum.
- struct EnumMaxImpl< TempfileControlFlag >

  Maximum number of elements in TempfileControlFlag enum.
- struct EnumMaxImpl< TensorFormat >

  Maximum number of elements in TensorFormat enum.
- struct EnumMaxImpl< TensorIOMode >

  One greater than the maximum value of TensorIOMode enum.
- struct EnumMaxImpl< TensorLocation >

  Maximum number of elements in TensorLocation enum.
- struct EnumMaxImpl< TilingOptimizationLevel >
- struct EnumMaxImpl< TopKOperation >
- struct EnumMaxImpl< TripLimit >
- struct EnumMaxImpl< UnaryOperation >
- struct EnumMaxImpl< WeightsRole >

  Maximum number of elements in WeightsRole enum.

8.5 nvinfer1::plugin Namespace Reference

Classes

- struct GridAnchorParameters

  The Anchor Generator plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions (H x W). GridAnchorParameters defines a set of parameters for creating the plugin layer for all feature maps.
- struct PriorBoxParameters

  The PriorBox plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions (H x W).
- struct RegionParameters

  The Region plugin layer performs region proposal calculation.
- struct RPROIParams

  RPROIParams is used to create the RPROIPlugin instance.
- struct softmaxTree

  When performing yolo9000, softmaxTree is helping to do softmax on confidence scores, for element to get the precise classification through word-tree structured classification definition.

8.6 nvinfer1::safe Namespace Reference

8.7 nvinfer1::v_1_0 Namespace Reference

Forward declare IErrorRecorder and ILogger for use in other interfaces.

Classes

- class IDebugListener
- class IErrorRecorder
- class IGpuAllocator
- class IGpuAsyncAllocator
- class ILogger

  Application-implemented logging interface for the builder, refitter and runtime.
- class ILoggerFinder

  A virtual base class to find a logger. Allows a plugin to find an instance of a logger if it needs to emit a log message. A pointer to an instance of this class is passed to a plugin shared library on initialization when that plugin is serialized as part of a version-compatible plan. See the plugin chapter in the developer guide for details.
- class IOutputAllocator
- class IPluginCapability
- class IPluginCreator
- class IPluginCreatorInterface
- class IPluginCreatorV3One
- class IPluginResource
- class IPluginV3
- class IPluginV3OneBuild
- class IPluginV3OneCore
- class IPluginV3OneRuntime
- class IProfiler
- class IProgressMonitor
- class IStreamReader
- class IStreamReaderV2
- class IStreamWriter
- struct TimingCacheKey

  The key to retrieve timing cache entries.
- struct TimingCacheValue

8.7.1 Detailed Description

Forward declare IErrorRecorder and ILogger for use in other interfaces.

DO NOT REFER TO namespace v_1_0 IN CODE. ALWAYS USE nvinfer1 INSTEAD. The name v_1_0 may change in future versions of TensorRT.

8.8 nvinfer1::v_2_0 Namespace Reference

Classes

- class IPluginV3OneBuild

8.9 nvinfer2 Namespace Reference

Namespaces

- namespace safe

8.10 nvinfer2::safe Namespace Reference

Namespaces

- namespace consistency
- namespace impl

Classes

- class IPluginV3OneSafeBuild

  Interface for plugins used during the TensorRT build phase.
- class IPluginV3OneSafeBuildMSS

  This class provides build capability and let output shapes be calculated by symbolic expression to support multiple static shape.
- class IPluginV3OneSafeCore

  Interface specifying the base plugin capability.
- class IPluginV3OneSafeRuntime

  Runtime component of the pluginV3 system.
- class ISafeMemAllocator

  Application-implemented class for controlling memory allocation on the GPU/CPU.
- class ISafePluginCreatorV3One

  The main interface to be implemented by all plugin creator classes.
- class ISafePluginRegistry

  PluginRegistry interface class that contains all the methods user can use to interface with the registry object.
- class ISafePluginResourceContext

  Interface for plugins to access per context resources provided by TensorRT.
- class ISafeRecorder

  Interface for extended recorder which allows error, warn, debug, or info messages to be recorded.
- class ITRTGraph

  Abstract Interface for a functionally safe graph for executing inference on a built network.
- class PhysicalDims

  Structure to define the physical dimensions of a tensor with support for up to 9 dimensions.
- struct RuntimeErrorInformation

  Holds information about runtime errors that occur during asynchronous kernel execution.
- struct TensorDescriptor

  A simple record summarizing various properties of a network IO Tensor.
- class TypedArray

  A standard_layout and trivially_copyable typed array that knows the data type and size it is holding.

Typedefs

- using Dims = nvinfer1::Dims
- using IVersionedInterface = nvinfer1::IVersionedInterface
- using AsciiChar = nvinfer1::AsciiChar
- using InterfaceInfo = nvinfer1::InterfaceInfo
- using DataType = nvinfer1::DataType
- using TensorIOMode = nvinfer1::TensorIOMode
- using ErrorCode = nvinfer1::ErrorCode
- using IErrorRecorder = nvinfer1::IErrorRecorder
- using IPluginResource = nvinfer1::IPluginResource
- using IPluginCreatorInterface = nvinfer1::IPluginCreatorInterface
- using IPluginV3 = nvinfer1::IPluginV3
- using IPluginCapability = nvinfer1::IPluginCapability
- using PluginField = nvinfer1::PluginField
- using PluginFieldCollection = nvinfer1::PluginFieldCollection
- using TensorRTPhase = nvinfer1::TensorRTPhase
- using ErrorDesc = char const *

  Type alias for error description strings.
- using RefCount = int32_t

  Type alias for reference count.
- using half_t = __half

Enumerations

- enum class Severity : uint32_t {
  kINTERNAL_ERROR = 0 , kERROR = 1 , kWARNING = 2 , kINFO = 3 ,
  kVERBOSE = 4 , kDEBUG = 5 }

  Enumerates severity levels for messages issued by the message recorder.
- enum class RuntimeErrorType : uint64_t {
  kNAN_CONSUMED = 1ULL << 0 , kINF_CONSUMED = 1ULL << 1 , kGATHER_OOB = 1ULL << 2 ,
  kSCATTER_OOB = 1ULL << 3 ,
  kSCATTER_RACE = 1ULL << 4 , kDIV_ZERO = 1ULL << 5 }

  Enumerates types of runtime errors that can occur during kernel execution.
- enum class MemoryPlacement : uint32_t {
  kNONE = 0x0 , kMANAGED = 0x1 , kGPU = 0x2 , kCPU = 0x4 ,
  kCPU_PINNED = 0x80 }

  Enum to describe the placement of the memory region.
- enum class MemoryUsage : uint32_t { kGENERIC = 0x0 , kIMMUTABLE = 0x3 , kSCRATCH = 0x5 ,
  kIOTENSOR = 0x6 }

  Enum to describe the usage of memory region.

Functions

- template<typename T >
  constexpr int32_t EnumMax () noexcept

  Maximum number of elements in an enumeration type.
- nvinfer2::safe::ISafePluginRegistry * getSafePluginRegistry (nvinfer2::safe::ISafeRecorder &recorder) noexcept

  Get the SafePluginRegistry singleton instance and set the SafeRecorder of the safePluginRegistry.
- bool operator== (PhysicalDims const &d0, PhysicalDims const &d1) noexcept
- bool operator!= (PhysicalDims const &d0, PhysicalDims const &d1) noexcept
- ErrorCode createTRTGraph (ITRTGraph *&graph, void const *buffer, int64_t bufferSize, ISafeRecorder &recorder, bool trtManagedScratch=true, ISafeMemAllocator *allocator=nullptr) noexcept

  The C factory function which serves as an entry point to TRT that creates an instance of a ITRTGraph class. It will performs the following operations:
- ErrorCode destroyTRTGraph (ITRTGraph *&graph) noexcept

  Toplevel graph destructor.

Variables

- constexpr uint64_t kMAXIMUM_ALLOC_SIZE {17179869184U}

  Legacy constant (16 GiB) kept for TensorRT excessiveMemoryDetection test build compatibility.

8.10.1 Typedef Documentation

8.10.1.1 AsciiChar

```
using nvinfer2::safe::AsciiChar = typedef nvinfer1::AsciiChar
```

8.10.1.2 DataType

```
using nvinfer2::safe::DataType = typedef nvinfer1::DataType
```

8.10.1.3 Dims

```
using nvinfer2::safe::Dims = typedef nvinfer1::Dims
```

8.10.1.4 ErrorCode

```
using nvinfer2::safe::ErrorCode = typedef nvinfer1::ErrorCode
```

8.10.1.5 ErrorDesc

```
using nvinfer2::safe::ErrorDesc = typedef char const*
```

Type alias for error description strings.

8.10.1.6 half_t

```
using nvinfer2::safe::half_t = typedef __half
```

8.10.1.7 IErrorRecorder

```
using nvinfer2::safe::IErrorRecorder = typedef nvinfer1::IErrorRecorder
```

8.10.1.8 InterfaceInfo

```
using nvinfer2::safe::InterfaceInfo = typedef nvinfer1::InterfaceInfo
```

8.10.1.9 IPluginCapability

```
using nvinfer2::safe::IPluginCapability = typedef nvinfer1::IPluginCapability
```

8.10.1.10 IPluginCreatorInterface

```
using nvinfer2::safe::IPluginCreatorInterface = typedef nvinfer1::IPluginCreatorInterface
```

8.10.1.11 IPluginResource

```
using nvinfer2::safe::IPluginResource = typedef nvinfer1::IPluginResource
```

8.10.1.12 IPluginV3

```
using nvinfer2::safe::IPluginV3 = typedef nvinfer1::IPluginV3
```

8.10.1.13 IVersionedInterface

```
using nvinfer2::safe::IVersionedInterface = typedef nvinfer1::IVersionedInterface
```

8.10.1.14 PluginField

```
using nvinfer2::safe::PluginField = typedef nvinfer1::PluginField
```

8.10.1.15 PluginFieldCollection

```
using nvinfer2::safe::PluginFieldCollection = typedef nvinfer1::PluginFieldCollection
```

8.10.1.16 RefCount

```
using nvinfer2::safe::RefCount = typedef int32_t
```

Type alias for reference count.

8.10.1.17 TensorIOMode

```
using nvinfer2::safe::TensorIOMode = typedef nvinfer1::TensorIOMode
```

8.10.1.18 TensorRTPhase

```
using nvinfer2::safe::TensorRTPhase = typedef nvinfer1::TensorRTPhase
```

8.10.2 Enumeration Type Documentation

8.10.2.1 MemoryPlacement

```
enum class nvinfer2::safe::MemoryPlacement : uint32_t [strong]
```

Enum to describe the placement of the memory region.

Enumerator

kNONE	Invalid or unspecified placement (used for error checking)
kMANAGED	CUDA managed memory (not used for safety)
kGPU	Device memory allocated via cudaMalloc.
kCPU	Regular (paged) host memory allocated via malloc, or aligned_alloc.
kCPU_PINNED	Page-locked host memory allocated via cudaHostAlloc, mappable to device (for zero-copy)

8.10.2.2 MemoryUsage

```
enum class nvinfer2::safe::MemoryUsage : uint32_t [strong]
```

Enum to describe the usage of memory region.

Enumerator

kGENERIC	Uncommitted memory usage.
kIMMUTABLE	Memory (network weights) that is initialized once and can be shared across different graphs.
kSCRATCH	Scratchpad memory per context used for intermediate tensors.
kIOTENSOR	Memory used for IO Tensors.

8.10.2.3 RuntimeErrorType

```
enum class nvinfer2::safe::RuntimeErrorType : uint64_t [strong]
```

Enumerates types of runtime errors that can occur during kernel execution.

- kNAN_CONSUMED error occurs when a NAN value is stored in an INT8 or FP4 quantized datatype.
- kINF_CONSUMED error occurs when a +/-INF value is stored in an INT8 or FP4 quantized datatype.
- kGATHER_OOB error occurs when a gather index tensor contains a value that is outside of the data tensor.
- kSCATTER_OOB error occurs when a scatter index tensor contains a value that is outside of the data tensor.
- kSCATTER_RACE error occurs when a scatter index tensor contains duplicate indices with reduction mode kNONE.
- kDIV_ZERO error occurs when a division-by-zero happens and its output is of an integer type.

Enumerator

kNAN_CONSUMED	NaN floating-point value was silently consumed.
kINF_CONSUMED	Inf floating-point value was silently consumed.
kGATHER_OOB	Out-of-bounds access in gather operation.
kSCATTER_OOB	Out-of-bounds access in scatter operation.
kSCATTER_RACE	Race condition in scatter operation.
kDIV_ZERO	Division-by-zero in int division.

8.10.2.4 Severity

```
enum class nvinfer2::safe::Severity : uint32_t [strong]
```

Enumerates severity levels for messages issued by the message recorder.

Enumerator

kINTERNAL_ERROR	An internal error has occurred. Execution is unrecoverable.
kERROR	An application error has occurred.
kWARNING	An application error has been discovered, but TensorRT has recovered or fallen back to a default.
kINFO	Informational messages with instructional information.
kVERBOSE	Verbose messages with additional information.
kDEBUG	Trace messages with debugging information.

8.10.3 Function Documentation

8.10.3.1 createTRTGraph()

```
ErrorCode nvinfer2::safe::createTRTGraph (
            ITRTGraph *& graph,
            void const * buffer,
            int64_t bufferSize,
            ISafeRecorder & recorder,
            bool trtManagedScratch = true,
            ISafeMemAllocator * allocator = nullptr )  [noexcept]
```

The C factory function which serves as an entry point to TRT that creates an instance of a ITRTGraph class. It will performs the following operations:

- Deserialize the TRT engine (Serialized binary blob) from the buffer
- Allocate the persistent device memory (for const model weights) though the provided allocator (if not provided then using the default allocator), load the weights into device memory and store it as a shared_ptr member variable (for sharing with clone method)

Precondition

INIT API

Parameters

graph	A pointer to ITRTGraph, that will be initialized after this call.
buffer	A buffer that holds the serialized blob that describes the graph.
bufferSize	Size of the serialized blob
recorder	An ISafeRecorder that sets severity level and records the error messages during the inference. Note this does not include RuntimeError happening during kernel execution, see getErrorBuffer for RuntimeError.
trtManagedScratch	if this parameter is true (default), then TRT will use the memory allocator to allocate the scratch memory that holds the intermediate tensors. When cloning the graph, TRT will also allocate a new scratch memory automatically. If the parameter is set to false (for sharing scratch memory across multiple graphs), user needs to call setScratchMemory for the graph (and its clones) with a memory buffer size at least the size queried from getScratchMemorySize.
allocator	The memory allocator to be used by the runtime. All GPU memory acquired will use this allocator. If nullptr is passed, the default allocator will be used, which calls cudaMalloc and cudaFree.

Returns

ErrorCode

Return values

ErrorCode::kSUCCESS	Graph created successfully; `graph` is set to the new instance.

Return values

ErrorCode::kINVALID_ARGUMENT	`buffer` is nullptr or `bufferSize` is not positive.
ErrorCode::kINVALID_STATE	VM state does not allow this init API (e.g. NvDVMS check failed).
ErrorCode::kFAILED_INITIALIZATION	CUDA setup check failed, or graph construction recorded this error on `recorder`, or an exception occurred (e.g. allocation failure during graph creation).
Other	codes may be propagated from deserialize(), setManagedScratch(), or loadPersistent() (e.g. kINVALID_CONFIG, kFAILED_ALLOCATION, kINVALID_STATE).

8.10.3.2 destroyTRTGraph()

```
ErrorCode nvinfer2::safe::destroyTRTGraph (
            ITRTGraph *& graph ) [noexcept]
```

Toplevel graph destructor.

- Frees the shared persistent memory only if this is the last graph instance that holds a reference to it.
- Frees the scratch memory if it is managed by TRT
- Deletes the graph object and all instruction instances underneath it for all IO profiles

Precondition

DEINIT API

Parameters

graph	A pointer reference to a graph object which will be destroyed and reference will be set to nullptr.

Returns

ErrorCode

Return values

ErrorCode::kSUCCESS	Graph destroyed successfully; `graph` is set to nullptr.
ErrorCode::kINVALID_ARGUMENT	`graph` is nullptr.
ErrorCode::kINVALID_STATE	VM state does not allow this deinit API (e.g. NvDVMS check failed), or unloadPersistent() failed (e.g. resource deallocation or CUDA teardown failed).
ErrorCode::kFAILED_INITIALIZATION	An exception occurred during destruction (e.g. during graph cleanup).
ErrorCode::kFAILED_ALLOCATION	Region deallocation failed (e.g. allocator returned false).

8.10.3.3 EnumMax()

```
template<typename T >
constexpr int32_t nvinfer2::safe::EnumMax ( )  [constexpr], [noexcept]
```

Maximum number of elements in an enumeration type.

8.10.3.4 getSafePluginRegistry()

```
nvinfer2::safe::ISafePluginRegistry * nvinfer2::safe::getSafePluginRegistry (
            nvinfer2::safe::ISafeRecorder & recorder )  [noexcept]
```

Get the SafePluginRegistry singleton instance and set the SafeRecorder of the safePluginRegistry.

Precondition

INIT API

This function retrieves the singleton instance of the SafePluginRegistry and sets the provided SafeRecorder for error tracking. The SafePluginRegistry is used to manage plugin creators in a safe and thread-safe manner.

Parameters

recorder	The SafeRecorder to be set for the SafePluginRegistry instance.

Returns

A pointer to the ISafePluginRegistry interface of the SafePluginRegistry instance.

- Allowed context for the API call
  - Thread-safe: Yes

8.10.3.5 operator"!=()

```
bool nvinfer2::safe::operator!= (
            PhysicalDims const & d0,
            PhysicalDims const & d1 )  [inline], [noexcept]
```

8.10.3.6 operator==()

```
bool nvinfer2::safe::operator== (
            PhysicalDims const & d0,
            PhysicalDims const & d1 )  [inline], [noexcept]
```

8.10.4 Variable Documentation

8.10.4.1 kMAXIMUM_ALLOC_SIZE

```
constexpr uint64_t nvinfer2::safe::kMAXIMUM_ALLOC_SIZE {17179869184U}  [constexpr]
```

Legacy constant (16 GiB) kept for TensorRT excessiveMemoryDetection test build compatibility.

Remarks

TRTS-11093 removed the max alloc limit from the allocator; this constant is deprecated and will be removed once TensorRT updates its lweSafety tests. Do not use for new code.

8.11 nvinfer2::safe::consistency Namespace Reference

Classes

- class IConsistencyChecker

  Validates a TRT engine (Serialized binary blob)
- class IPluginChecker

  Interface for validating a plugin used in a TensorRT engine.

Functions

- nvinfer2::safe::ErrorCode createConsistencyChecker (IConsistencyChecker ∗&icc, nvinfer2::safe::ISafeRecorder &recorder, void const ∗blob, size_t size, std::vector< std::string > const &pluginBuildLibs={}) noexcept

  Create an instance of an IConsistencyChecker class.

8.11.1 Function Documentation

8.11.1.1 createConsistencyChecker()

```
nvinfer2::safe::ErrorCode nvinfer2::safe::consistency::createConsistencyChecker (
            IConsistencyChecker *& icc,
            nvinfer2::safe::ISafeRecorder & recorder,
            void const * blob,
            size_t size,
            std::vector< std::string > const & pluginBuildLibs = {} )  [noexcept]
```

Create an instance of an IConsistencyChecker class.

ISafeRecorder is the error recorder class for the consistency checker.

8.12 nvinfer2::safe::impl Namespace Reference

Classes

- struct EnumMaxImpl

  Declaration of EnumMaxImpl struct to store maximum number of elements in an enumeration type.
- struct EnumMaxImpl< Severity >

  Provides a static constant for the maximum value of the Severity enum.

8.13 nvonnxparser Namespace Reference

The TensorRT ONNX parser API namespace.

Namespaces

- namespace anonymous_namespace{NvOnnxParser.h}

Classes

- class IOnnxConfig

  Configuration Manager Class.
- class IParser

  an object for parsing ONNX models into a TensorRT network definition
- class IParserError

  an object containing information about an error
- class IParserRefitter

  An interface designed to refit weights from an ONNX model.

Typedefs

- using OnnxParserFlags = uint32_t

  Represents one or more OnnxParserFlag values using binary OR operations, e.g., 1U << OnnxParserFlag::kNATIVE_INSTANCENORM.

Enumerations

- enum class ErrorCode : int {
  kSUCCESS = 0 , kINTERNAL_ERROR = 1 , kMEM_ALLOC_FAILED = 2 , kMODEL_DESERIALIZE_FAILED = 3 ,
  kINVALID_VALUE = 4 , kINVALID_GRAPH = 5 , kINVALID_NODE = 6 , kUNSUPPORTED_GRAPH = 7 ,
  kUNSUPPORTED_NODE = 8 , kUNSUPPORTED_NODE_ATTR = 9 , kUNSUPPORTED_NODE_INPUT = 10 , kUNSUPPORTED_NODE_DATATYPE = 11 ,
  kUNSUPPORTED_NODE_DYNAMIC = 12 , kUNSUPPORTED_NODE_SHAPE = 13 , kREFIT_FAILED = 14 }

  The type of error that the parser or refitter may return.
- enum class OnnxParserFlag : int32_t {
  kNATIVE_INSTANCENORM = 0 , kENABLE_UINT8_AND_ASYMMETRIC_QUANTIZATION_DLA = 1 ,
  kREPORT_CAPABILITY_DLA = 2 , kENABLE_PLUGIN_OVERRIDE = 3 ,
  kADJUST_FOR_DLA = 4 }

Functions

- TENSORRTAPI IOnnxConfig * createONNXConfig ()
- template<typename T >
  constexpr int32_t EnumMax () noexcept=delete
- template<> constexpr int32_t EnumMax< ErrorCode > () noexcept

  Specialization. See `nvonnxparser::EnumMax()` for details.
- template<> constexpr int32_t EnumMax< OnnxParserFlag > () noexcept

  Specialization. See `nvonnxparser::EnumMax()` for details.

8.13.1 Detailed Description

The TensorRT ONNX parser API namespace.

8.13.2 Typedef Documentation

8.13.2.1 OnnxParserFlags

```
using nvonnxparser::OnnxParserFlags = typedef uint32_t
```

Represents one or more OnnxParserFlag values using binary OR operations, e.g., 1U << OnnxParserFlag::kNATIVE_INSTANCENORM

See also

IParser::setFlags() and IParser::getFlags()

8.13.3 Enumeration Type Documentation

8.13.3.1 ErrorCode

`enum class nvonnxparser::ErrorCode : int [strong]`

The type of error that the parser or refitter may return.

Enumerator

kSUCCESS	
kINTERNAL_ERROR	
kMEM_ALLOC_FAILED	
kMODEL_DESERIALIZE_FAILED	
kINVALID_VALUE	
kINVALID_GRAPH	
kINVALID_NODE	
kUNSUPPORTED_GRAPH	
kUNSUPPORTED_NODE	
kUNSUPPORTED_NODE_ATTR	
kUNSUPPORTED_NODE_INPUT	
kUNSUPPORTED_NODE_DATATYPE	
kUNSUPPORTED_NODE_DYNAMIC	
kUNSUPPORTED_NODE_SHAPE	
kREFIT_FAILED	

8.13.3.2 OnnxParserFlag

`enum class nvonnxparser::OnnxParserFlag : int32_t [strong]`

Enumerator

kNATIVE_INSTANCENORM	Parse the ONNX model into the INetworkDefinition with the intention of using TensorRT's native layer implementation over the plugin implementation for InstanceNormalization nodes. This flag is required when building version-compatible or hardware-compatible engines. This flag is set to be ON by default.
kENABLE_UINT8_AND_ASYMMETRIC_↩ QUANTIZATION_DLA	Enable UINT8 as a quantization data type and asymmetric quantization with non-zero zero-point values in Quantize and Dequantize nodes. This flag is set to be OFF by default. The resulting engine must be built targeting DLA version >= 3.16.

Enumerator

kREPORT_CAPABILITY_DLA	Parse the ONNX model with per-node validation for DLA. If the model is not fully supported by DLA, then parsing will fail. If this flag is set, isSubGraphSupported() will also return capability in the context of DLA support. When this flag is set, a valid IBuilderConfig must be provided to the parser via setBuilderConfig().
kENABLE_PLUGIN_OVERRIDE	Allow a loaded plugin with the same name as an ONNX operator type to override the default ONNX implementation, even if the plugin namespace attribute is not set. Useful for custom plugins that replace standard ONNX operators, such as alternative implementations for better performance. This flag is set to be OFF by default.
kADJUST_FOR_DLA	Opportunistically rewrite or modify layers to make them more amenable to running on DLA.

8.13.4 Function Documentation

8.13.4.1 createONNXConfig()

```
TENSORRTAPI IOnnxConfig * nvonnxparser::createONNXConfig ( )
```

8.13.4.2 EnumMax()

```
template<typename T >
constexpr int32_t nvonnxparser::EnumMax ( )  [constexpr], [delete], [noexcept]
```

Returns

the numerical value of the highest-valued enumerator for type T. It must be specialized for each enum type that uses it. {$nv-internal-release begin}

Note

This is different from nvinfer1::EnumMax<T>(), which is one more than that. {$nv-internal-release end}

8.13.4.3 EnumMax< ErrorCode >()

```
template<>
constexpr int32_t nvonnxparser::EnumMax< ErrorCode > ( )  [constexpr], [noexcept]
```

Specialization. See nvonnxparser::EnumMax() for details.

8.13.4.4 EnumMax< OnnxParserFlag >()

```
template<>
constexpr int32_t nvonnxparser::EnumMax< OnnxParserFlag > ( )  [constexpr], [noexcept]
```

Specialization. See nvonnxparser::EnumMax() for details.

8.14 nvonnxparser::anonymous_namespace{NvOnnxParser.h} Namespace Reference

Functions

- IParser * createParser (nvinfer1::INetworkDefinition &network, nvinfer1::ILogger &logger) noexcept
  Create a new parser object.
- IParserRefitter * createParserRefitter (nvinfer1::IRefitter &refitter, nvinfer1::ILogger &logger) noexcept
  Create a new ONNX refitter object.

8.14.1 Function Documentation

8.14.1.1 createParser()

```
IParser * nvonnxparser::anonymous_namespace{NvOnnxParser.h}::createParser (
            nvinfer1::INetworkDefinition & network,
            nvinfer1::ILogger & logger )  [inline], [noexcept]
```

Create a new parser object.

Parameters

network	The network definition that the parser will write to
logger	The logger to use

Returns

a new parser object or NULL if an error occurred

Any input dimensions that are constant should not be changed after parsing, because correctness of the translation may rely on those constants. Changing a dynamic input dimension, i.e. one that translates to -1 in TensorRT, to a constant is okay if the constant is consistent with the model. Each instance of the parser is designed to only parse one ONNX model once.

See also

IParser

8.14.1.2 createParserRefitter()

```
IParserRefitter * nvonnxparser::anonymous_namespace{NvOnnxParser.h}::createParserRefitter (
            nvinfer1::IRefitter & refitter,
            nvinfer1::ILogger & logger )  [inline], [noexcept]
```

Create a new ONNX refitter object.

Parameters

refitter	The Refitter object used to refit the model
logger	The logger to use

Returns

a new ParserRefitter object or NULL if an error occurred

See also

IParserRefitter

Chapter 9

Class Documentation

9.1 Dims Class Reference

Structure to define the dimensions of a tensor.

`#include <NvInferRuntimeBase.h>`

9.1.1 Detailed Description

Structure to define the dimensions of a tensor.

TensorRT can also return an ”invalid dims” structure. This structure is represented by nbDims == -1 and d[i] == 0 for all i.

TensorRT can also return an ”unknown rank” dims structure. This structure is represented by nbDims == -1 and d[i] == -1 for all i.

The documentation for this class was generated from the following file:

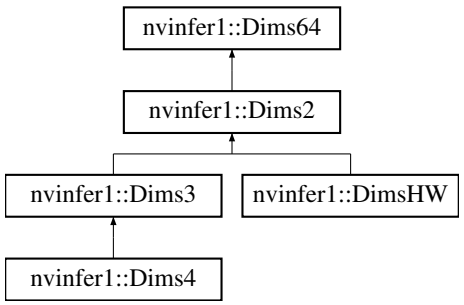
- NvInferRuntimeBase.h

9.2 nvinfer1::Dims2 Class Reference

Descriptor for two-dimensional data.

`#include <NvInferLegacyDims.h>`

Inheritance diagram for nvinfer1::Dims2:



Public Member Functions

- Dims2 ()

  Construct an empty Dims2 object.
- Dims2 (int64_t d0, int64_t d1)

  Construct a Dims2 from 2 elements.

Additional Inherited Members

9.2.1 Detailed Description

Descriptor for two-dimensional data.

9.2.2 Constructor & Destructor Documentation

9.2.2.1 Dims2() [1/2]

```
nvinfer1::Dims2::Dims2 ( )  [inline]
```

Construct an empty Dims2 object.

9.2.2.2 Dims2() [2/2]

```
nvinfer1::Dims2::Dims2 (
            int64_t d0,
            int64_t d1 )  [inline]
```

Construct a Dims2 from 2 elements.

Parameters

d0	The first element.
d1	The second element.

The documentation for this class was generated from the following file:

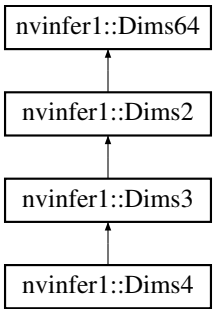
- NvInferLegacyDims.h

9.3 nvinfer1::Dims3 Class Reference

Descriptor for three-dimensional data.

`#include <NvInferLegacyDims.h>`

Inheritance diagram for nvinfer1::Dims3:



Public Member Functions

- Dims3 ()

  Construct an empty Dims3 object.
- Dims3 (int64_t d0, int64_t d1, int64_t d2)

  Construct a Dims3 from 3 elements.

Additional Inherited Members

9.3.1 Detailed Description

Descriptor for three-dimensional data.

9.3.2 Constructor & Destructor Documentation

9.3.2.1 Dims3() [1/2]

```
nvinfer1::Dims3::Dims3 ( )  [inline]
```

Construct an empty Dims3 object.

9.3.2.2 Dims3() [2/2]

```
nvinfer1::Dims3::Dims3 (
            int64_t d0,
            int64_t d1,
            int64_t d2 )  [inline]
```

Construct a Dims3 from 3 elements.

Parameters

d0	The first element.
d1	The second element.
d2	The third element.

The documentation for this class was generated from the following file:

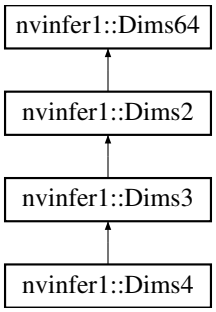
- NvInferLegacyDims.h

9.4 nvinfer1::Dims4 Class Reference

Descriptor for four-dimensional data.

`#include <NvInferLegacyDims.h>`

Inheritance diagram for nvinfer1::Dims4:



Public Member Functions

- Dims4 ()

  Construct an empty Dims4 object.
- Dims4 (int64_t d0, int64_t d1, int64_t d2, int64_t d3)

  Construct a Dims4 from 4 elements.

Additional Inherited Members

9.4.1 Detailed Description

Descriptor for four-dimensional data.

9.4.2 Constructor & Destructor Documentation

9.4.2.1 Dims4() [1/2]

```
nvinfer1::Dims4::Dims4 ( )  [inline]
```

Construct an empty Dims4 object.

9.4.2.2 Dims4() [2/2]

```
nvinfer1::Dims4::Dims4 (
            int64_t d0,
            int64_t d1,
            int64_t d2,
            int64_t d3 )  [inline]
```

Construct a Dims4 from 4 elements.

Parameters

d0	The first element.
d1	The second element.
d2	The third element.
d3	The fourth element.

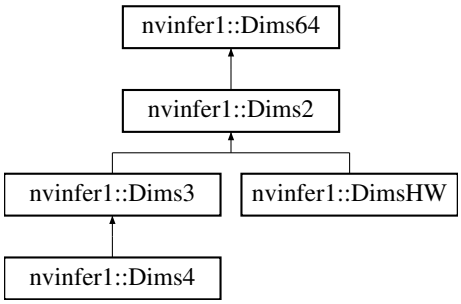
The documentation for this class was generated from the following file:

- NvInferLegacyDims.h

9.5 nvinfer1::Dims64 Class Reference

```
#include <NvInferRuntimeBase.h>
```

Inheritance diagram for nvinfer1::Dims64:



Public Attributes

- int32_t nbDims

  The rank (number of dimensions).
- int64_t d [MAX_DIMS]

  The extent of each dimension.

Static Public Attributes

- static constexpr int32_t MAX_DIMS {8}

  The maximum rank (number of dimensions) supported for a tensor.

9.5.1 Member Data Documentation

9.5.1.1 d

```
int64_t nvinfer1::Dims64::d[MAX_DIMS]
```

The extent of each dimension.

9.5.1.2 MAX_DIMS

```
constexpr int32_t nvinfer1::Dims64::MAX_DIMS {8}  [static], [constexpr]
```

The maximum rank (number of dimensions) supported for a tensor.

9.5.1.3 nbDims

```
int32_t nvinfer1::Dims64::nbDims
```

The rank (number of dimensions).

The documentation for this class was generated from the following file:

- NvInferRuntimeBase.h

9.6 nvinfer1::DimsExprs Class Reference

Analog of class Dims with expressions instead of constants for the dimensions.

```
#include <NvInferRuntime.h>
```

Public Attributes

- int32_t nbDims

  The number of dimensions.
- IDimensionExpr const * d [Dims::MAX_DIMS]

  The extent of each dimension.

9.6.1 Detailed Description

Analog of class Dims with expressions instead of constants for the dimensions.

9.6.2 Member Data Documentation

9.6.2.1 d

```
IDimensionExpr const* nvinfer1::DimsExprs::d[Dims::MAX_DIMS]
```

The extent of each dimension.

9.6.2.2 nbDims

`int32_t nvinfer1::DimsExprs::nbDims`

The number of dimensions.

The documentation for this class was generated from the following file:

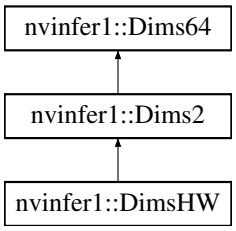
- NvInferRuntime.h

9.7 nvinfer1::DimsHW Class Reference

Descriptor for two-dimensional spatial data.

`#include <NvInferLegacyDims.h>`

Inheritance diagram for nvinfer1::DimsHW:



Public Member Functions

- DimsHW ()
  Construct an empty DimsHW object.
- DimsHW (int64_t height, int64_t width)
  Construct a DimsHW given height and width.
- int64_t & h ()
  Get the height.
- int64_t h () const
  Get the height.
- int64_t & w ()
  Get the width.
- int64_t w () const
  Get the width.

Additional Inherited Members

9.7.1 Detailed Description

Descriptor for two-dimensional spatial data.

9.7.2 Constructor & Destructor Documentation

9.7.2.1 DimsHW() [1/2]

```
nvinfer1::DimsHW::DimsHW ( )  [inline]
```

Construct an empty DimsHW object.

9.7.2.2 DimsHW() [2/2]

```
nvinfer1::DimsHW::DimsHW (
            int64_t height,
            int64_t width )  [inline]
```

Construct a DimsHW given height and width.

Parameters

height	the height of the data
width	the width of the data

9.7.3 Member Function Documentation

9.7.3.1 h() [1/2]

```
int64_t & nvinfer1::DimsHW::h ( )  [inline]
```

Get the height.

Returns

The height.

9.7.3.2 h() [2/2]

```
int64_t nvinfer1::DimsHW::h ( ) const  [inline]
```

Get the height.

Returns

The height.

9.7.3.3 w() [1/2]

```
int64_t & nvinfer1::DimsHW::w ( )  [inline]
```

Get the width.

Returns

The width.

9.7.3.4 w() [2/2]

```
int64_t nvinfer1::DimsHW::w ( ) const  [inline]
```

Get the width.

Returns

The width.

The documentation for this class was generated from the following file:

- NvInferLegacyDims.h

9.8 nvinfer1::DynamicPluginTensorDesc Struct Reference

Summarizes tensors that a plugin might see for an input or output.

```
#include <NvInferRuntime.h>
```

Public Attributes

- PluginTensorDesc desc

  Information required to interpret a pointer to tensor data, except that desc.dims has -1 in place of any runtime dimension.
- Dims min

  Lower bounds on tensor's dimensions.
- Dims max

  Upper bounds on tensor's dimensions.
- Dims opt

  Optimum value of tensor's dimensions specified for auto-tuning.

9.8.1 Detailed Description

Summarizes tensors that a plugin might see for an input or output.

9.8.2 Member Data Documentation

9.8.2.1 desc

```
PluginTensorDesc nvinfer1::DynamicPluginTensorDesc::desc
```

Information required to interpret a pointer to tensor data, except that desc.dims has -1 in place of any runtime dimension.

9.8.2.2 max

```
Dims nvinfer1::DynamicPluginTensorDesc::max
```

Upper bounds on tensor's dimensions.

9.8.2.3 min

```
Dims nvinfer1::DynamicPluginTensorDesc::min
```

Lower bounds on tensor's dimensions.

9.8.2.4 opt

```
Dims nvinfer1::DynamicPluginTensorDesc::opt
```

Optimum value of tensor's dimensions specified for auto-tuning.

The documentation for this struct was generated from the following file:

- NvInferRuntime.h

9.9 nvinfer1::impl::EnumMaxImpl< T > Struct Template Reference

Declaration of EnumMaxImpl struct to store the exclusive upper bound of an enumeration type.

9.9.1 Detailed Description

template<typename T>
struct nvinfer1::impl::EnumMaxImpl< T >

Declaration of EnumMaxImpl struct to store the exclusive upper bound of an enumeration type.

The documentation for this struct was generated from the following file:

- NvInferRuntimeBase.h

9.10 nvinfer2::safe::impl::EnumMaxImpl< T > Struct Template Reference

Declaration of EnumMaxImpl struct to store maximum number of elements in an enumeration type.

9.10.1 Detailed Description

template<typename T>
struct nvinfer2::safe::impl::EnumMaxImpl< T >

Declaration of EnumMaxImpl struct to store maximum number of elements in an enumeration type.

The documentation for this struct was generated from the following file:

- NvInferForwardDecl.h

9.11 nvinfer1::impl::EnumMaxImpl< ActivationType > Struct Reference

`#include <NvInfer.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 14

9.11.1 Detailed Description

Maximum number of elements in ActivationType enum.

See also

ActivationType

9.11.2 Member Data Documentation

9.11.2.1 kVALUE

`constexpr int32_t nvinfer1::impl::EnumMaxImpl< ActivationType >::kVALUE = 14 [static], [constexpr]`

The documentation for this struct was generated from the following file:

- NvInfer.h

9.12 nvinfer1::impl::EnumMaxImpl< AllocatorFlag > Struct Reference

Maximum number of elements in AllocatorFlag enum.

`#include <NvInferRuntime.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 1

  Declaration of kVALUE that represents the maximum number of elements in the AllocatorFlag enum.

9.12.1 Detailed Description

Maximum number of elements in AllocatorFlag enum.

See also

AllocatorFlag

9.12.2 Member Data Documentation

9.12.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< AllocatorFlag >::kVALUE = 1  [static], [constexpr]
```

Declaration of kVALUE that represents the maximum number of elements in the AllocatorFlag enum.

The documentation for this struct was generated from the following file:

- NvInferRuntime.h

9.13 nvinfer1::impl::EnumMaxImpl< APILanguage > Struct Reference

One greater than the maximum value of APILanguage enum.

`#include <NvInferRuntimeBase.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 2

  One greater than the maximum value of APILanguage enum.

9.13.1 Detailed Description

One greater than the maximum value of APILanguage enum.

See also

APILanguage

9.13.2 Member Data Documentation

9.13.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< APILanguage >::kVALUE = 2  [static], [constexpr]
```

One greater than the maximum value of APILanguage enum.

The documentation for this struct was generated from the following file:

- NvInferRuntimeBase.h

9.14 nvinfer1::impl::EnumMaxImpl< AttentionIOForm > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 2

9.14.1 Detailed Description

Maximum number of elements in AttentionIOForm enum.

See also

AttentionIOForm

9.14.2 Member Data Documentation

9.14.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< AttentionIOForm >::kVALUE = 2  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.15 nvinfer1::impl::EnumMaxImpl< AttentionNormalizationOp > Struct Reference

`#include <NvInfer.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 2

9.15.1 Detailed Description

Maximum number of elements in AttentionNormalizationOp enum.

See also

AttentionNormalizationOp

9.15.2 Member Data Documentation

9.15.2.1 kVALUE

`constexpr int32_t nvinfer1::impl::EnumMaxImpl< AttentionNormalizationOp >::kVALUE = 2 [static], [constexpr]`

The documentation for this struct was generated from the following file:

- NvInfer.h

9.16 nvinfer1::impl::EnumMaxImpl< BoundingBoxFormat > Struct Reference

`#include <NvInfer.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 2

9.16.1 Detailed Description

Maximum number of elements in BoundingBoxFormat enum.

See also

BoundingBoxFormat

9.16.2 Member Data Documentation

9.16.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< BoundingBoxFormat >::kVALUE = 2  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.17 nvinfer1::impl::EnumMaxImpl< BuilderFlag > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 28

9.17.1 Detailed Description

Maximum number of builder flags in BuilderFlag enum.

See also

BuilderFlag

9.17.2 Member Data Documentation

9.17.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< BuilderFlag >::kVALUE = 28  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.18 nvinfer1::impl::EnumMaxImpl< CausalMaskKind > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.18.1 Detailed Description

Maximum number of elements in CausalMaskKind enum.

See also

CausalMaskKind

9.18.2 Member Data Documentation

9.18.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< CausalMaskKind >::kVALUE = 3  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.19 nvinfer1::impl::EnumMaxImpl< CollectiveOperation > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 8

9.19.1 Detailed Description

Maximum number of elements in CollectiveOperation enum.

See also

CollectiveOperation

9.19.2 Member Data Documentation

9.19.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< CollectiveOperation >::kVALUE = 8  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.20 nvinfer1::impl::EnumMaxImpl< CumulativeOperation > Struct Reference

Maximum number of elements in CumulativeOperation enum.

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 1

9.20.1 Detailed Description

Maximum number of elements in CumulativeOperation enum.

See also

CumulativeOperation

9.20.2 Member Data Documentation

9.20.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< CumulativeOperation >::kVALUE = 1  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.21 nvinfer1::impl::EnumMaxImpl< DataType > Struct Reference

One greater than the maximum value of DataType enum.

```
#include <NvInferRuntimeBase.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 12
  One greater than the maximum value of DataType enum.

9.21.1 Detailed Description

One greater than the maximum value of DataType enum.

See also

DataType

9.21.2 Member Data Documentation

9.21.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< DataType >::kVALUE = 12  [static], [constexpr]
```

One greater than the maximum value of DataType enum.

The documentation for this struct was generated from the following file:

- NvInferRuntimeBase.h

9.22 nvinfer1::impl::EnumMaxImpl< DeviceType > Struct Reference

Maximum number of elements in DeviceType enum.

`#include <NvInferRuntime.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 2

9.22.1 Detailed Description

Maximum number of elements in DeviceType enum.

See also

DeviceType

9.22.2 Member Data Documentation

9.22.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< DeviceType >::kVALUE = 2  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInferRuntime.h

9.23 nvinfer1::impl::EnumMaxImpl< DimensionOperation > Struct Reference

Maximum number of elements in DimensionOperation enum.

`#include <NvInferRuntime.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 9

9.23.1 Detailed Description

Maximum number of elements in DimensionOperation enum.

See also

DimensionOperation

9.23.2 Member Data Documentation

9.23.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< DimensionOperation >::kVALUE = 9  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInferRuntime.h

9.24 nvinfer1::impl::EnumMaxImpl< ElementWiseOperation > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 14

9.24.1 Detailed Description

Maximum number of elements in ElementWiseOperation enum.

See also

ElementWiseOperation

9.24.2 Member Data Documentation

9.24.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< ElementWiseOperation >::kVALUE = 14  [static],
[constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.25 nvinfer1::impl::EnumMaxImpl< EngineCapability > Struct Reference

Maximum number of elements in EngineCapability enum.

```
#include <NvInferRuntime.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.25.1 Detailed Description

Maximum number of elements in EngineCapability enum.

See also

EngineCapability

9.25.2 Member Data Documentation

9.25.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< EngineCapability >::kVALUE = 3  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInferRuntime.h

9.26 nvinfer1::impl::EnumMaxImpl< EngineStat > Struct Reference

Maximum number of engine statistic kinds in EngineStat enum.

`#include <NvInferRuntime.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 2

9.26.1 Detailed Description

Maximum number of engine statistic kinds in EngineStat enum.

See also

EngineStat

9.26.2 Member Data Documentation

9.26.2.1 kVALUE

`constexpr int32_t nvinfer1::impl::EnumMaxImpl< EngineStat >::kVALUE = 2 [static], [constexpr]`

The documentation for this struct was generated from the following file:

- NvInferRuntime.h

9.27 nvinfer1::impl::EnumMaxImpl< ErrorCode > Struct Reference

One greater than the maximum value of ErrorCode enum.

`#include <NvInferRuntimeBase.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 11

  Declaration of kVALUE.

9.27.1 Detailed Description

One greater than the maximum value of ErrorCode enum.

See also

ErrorCode

9.27.2 Member Data Documentation

9.27.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< ErrorCode >::kVALUE = 11  [static], [constexpr]
```

Declaration of kVALUE.

The documentation for this struct was generated from the following file:

- NvInferRuntimeBase.h

9.28 nvinfer1::impl::EnumMaxImpl< ExecutionContextAllocationStrategy > Struct Reference

```
#include <NvInferRuntime.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.28.1 Detailed Description

Maximum number of memory allocation strategies in ExecutionContextAllocationStrategy enum.

See also

ExecutionContextAllocationStrategy

9.28.2 Member Data Documentation

9.28.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< ExecutionContextAllocationStrategy >::kVALUE = 3
[static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInferRuntime.h

9.29 nvinfer1::impl::EnumMaxImpl< FillOperation > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.29.1 Detailed Description

Maximum number of elements in FillOperation enum.

See also

FillOperation

9.29.2 Member Data Documentation

9.29.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< FillOperation >::kVALUE = 3  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.30 nvinfer1::impl::EnumMaxImpl< GatherMode > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.30.1 Detailed Description

Maximum number of elements in GatherMode enum.

See also

GatherMode

9.30.2 Member Data Documentation

9.30.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< GatherMode >::kVALUE = 3  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.31 nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.31.1 Detailed Description

Maximum number of elements in HardwareCompatibilityLevel enum.

See also

HardwareCompatibilityLevel

9.31.2 Member Data Documentation

9.31.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel >::kVALUE = 3  [static],
[constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.32 nvinfer1::impl::EnumMaxImpl< ILogger::Severity > Struct Reference

Maximum number of elements in ILogger::Severity enum.

```
#include <NvInferRuntime.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 5

  Declaration of kVALUE that represents the maximum number of elements in the ILogger::Severity enum.

9.32.1 Detailed Description

Maximum number of elements in ILogger::Severity enum.

See also

ILogger::Severity

9.32.2 Member Data Documentation

9.32.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< ILogger::Severity >::kVALUE = 5  [static], [constexpr]
```

Declaration of kVALUE that represents the maximum number of elements in the ILogger::Severity enum.

The documentation for this struct was generated from the following file:

- NvInferRuntime.h

9.33 nvinfer1::impl::EnumMaxImpl< InterpolationMode > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.33.1 Detailed Description

Maximum number of elements in InterpolationMode enum.

See also

InterpolationMode

9.33.2 Member Data Documentation

9.33.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< InterpolationMode >::kVALUE = 3  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.34 nvinfer1::impl::EnumMaxImpl< KVCacheMode > Struct Reference

`#include <NvInfer.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 1

9.34.1 Detailed Description

Maximum number of elements in KVCacheMode enum.

See also

KVCacheMode

9.34.2 Member Data Documentation

9.34.2.1 kVALUE

`constexpr int32_t nvinfer1::impl::EnumMaxImpl< KVCacheMode >::kVALUE = 1 [static], [constexpr]`

The documentation for this struct was generated from the following file:

- NvInfer.h

9.35 nvinfer1::impl::EnumMaxImpl< LayerInformationFormat > Struct Reference

Maximum number of layer information formats in LayerInformationFormat enum.

`#include <NvInferRuntime.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 2

9.35.1 Detailed Description

Maximum number of layer information formats in LayerInformationFormat enum.

See also

LayerInformationFormat

9.35.2 Member Data Documentation

9.35.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< LayerInformationFormat >::kVALUE = 2  [static],
[constexpr]
```

The documentation for this struct was generated from the following file:

- NvInferRuntime.h

9.36 nvinfer1::impl::EnumMaxImpl< LayerType > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 57

9.36.1 Detailed Description

Maximum number of elements in LayerType enum.

See also

LayerType

9.36.2 Member Data Documentation

9.36.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< LayerType >::kVALUE = 57  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.37 nvinfer1::impl::EnumMaxImpl< LoopOutput > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.37.1 Detailed Description

Maximum number of elements in LoopOutput enum.

See also

DataType

9.37.2 Member Data Documentation

9.37.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< LoopOutput >::kVALUE = 3  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.38 nvinfer1::impl::EnumMaxImpl< MatrixOperation > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.38.1 Detailed Description

Maximum number of elements in MatrixOperation enum.

See also

DataType

9.38.2 Member Data Documentation

9.38.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< MatrixOperation >::kVALUE = 3  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.39 nvinfer1::impl::EnumMaxImpl< MemoryPoolType > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 6

9.39.1 Detailed Description

Maximum number of memory pool types in the MemoryPoolType enum.

See also

MemoryPoolType

9.39.2 Member Data Documentation

9.39.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< MemoryPoolType >::kVALUE = 6  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.40 nvinfer1::impl::EnumMaxImpl< MoEActType > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 2

9.40.1 Detailed Description

Maximum number of elements in MoEActType enum.

See also

MoEActType

9.40.2 Member Data Documentation

9.40.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< MoEActType >::kVALUE = 2  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.41 nvinfer1::impl::EnumMaxImpl< NetworkDefinitionCreationFlag > Struct Reference

`#include <NvInfer.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.41.1 Detailed Description

Maximum number of elements in NetworkDefinitionCreationFlag enum.

See also

NetworkDefinitionCreationFlag

9.41.2 Member Data Documentation

9.41.2.1 kVALUE

`constexpr int32_t nvinfer1::impl::EnumMaxImpl< NetworkDefinitionCreationFlag >::kVALUE = 3 [static], [constexpr]`

The documentation for this struct was generated from the following file:

- NvInfer.h

9.42 nvinfer1::impl::EnumMaxImpl< OptProfileSelector > Struct Reference

Number of different values of OptProfileSelector enum.

`#include <NvInferRuntime.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.42.1 Detailed Description

Number of different values of OptProfileSelector enum.

See also

OptProfileSelector

9.42.2 Member Data Documentation

9.42.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< OptProfileSelector >::kVALUE = 3  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInferRuntime.h

9.43 nvinfer1::impl::EnumMaxImpl< PaddingMode > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 4

9.43.1 Detailed Description

Maximum number of elements in PaddingMode enum.

See also

PaddingMode

9.43.2 Member Data Documentation

9.43.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< PaddingMode >::kVALUE = 4  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.44 nvinfer1::impl::EnumMaxImpl< PoolingType > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.44.1 Detailed Description

Maximum number of elements in PoolingType enum.

See also

PoolingType

9.44.2 Member Data Documentation

9.44.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< PoolingType >::kVALUE = 3  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.45 nvinfer1::impl::EnumMaxImpl< PreviewFeature > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 2

9.45.1 Detailed Description

Maximum number of elements in PreviewFeature enum.

See also

PreviewFeature

9.45.2 Member Data Documentation

9.45.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< PreviewFeature >::kVALUE = 2  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.46 nvinfer1::impl::EnumMaxImpl< ProfilingVerbosity > Struct Reference

Maximum number of profile verbosity levels in ProfilingVerbosity enum.

```
#include <NvInferRuntime.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.46.1 Detailed Description

Maximum number of profile verbosity levels in ProfilingVerbosity enum.

See also

ProfilingVerbosity

9.46.2 Member Data Documentation

9.46.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< ProfilingVerbosity >::kVALUE = 3  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInferRuntime.h

9.47 nvinfer1::impl::EnumMaxImpl< ReduceOperation > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 6

9.47.1 Detailed Description

Maximum number of elements in ReduceOperation enum.

See also

ReduceOperation

9.47.2 Member Data Documentation

9.47.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< ReduceOperation >::kVALUE = 6  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.48 nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation > Struct Reference

`#include <NvInfer.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.48.1 Detailed Description

Maximum number of elements in ResizeCoordinateTransformation enum.

See also

ResizeCoordinateTransformation

9.48.2 Member Data Documentation

9.48.2.1 kVALUE

`constexpr int32_t nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation >::kVALUE = 3 [static], [constexpr]`

The documentation for this struct was generated from the following file:

- NvInfer.h

9.49 nvinfer1::impl::EnumMaxImpl< ResizeRoundMode > Struct Reference

`#include <NvInfer.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 4

9.49.1 Detailed Description

Maximum number of elements in ResizeRoundMode enum.

See also

ResizeRoundMode

9.49.2 Member Data Documentation

9.49.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< ResizeRoundMode >::kVALUE = 4  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.50 nvinfer1::impl::EnumMaxImpl< ResizeSelector > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 2

9.50.1 Detailed Description

Maximum number of elements in ResizeSelector enum.

See also

ResizeSelector

9.50.2 Member Data Documentation

9.50.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< ResizeSelector >::kVALUE = 2  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.51 nvinfer1::impl::EnumMaxImpl< RuntimePlatform > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 2

9.51.1 Detailed Description

Maximum number of elements in RuntimePlatform enum.

See also

RuntimePlatform

9.51.2 Member Data Documentation

9.51.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< RuntimePlatform >::kVALUE = 2  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.52 nvinfer1::impl::EnumMaxImpl< SampleMode > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 5

9.52.1 Detailed Description

Maximum number of elements in SampleMode enum.

See also

SampleMode

9.52.2 Member Data Documentation

9.52.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< SampleMode >::kVALUE = 5  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.53 nvinfer1::impl::EnumMaxImpl< ScaleMode > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.53.1 Detailed Description

Maximum number of elements in ScaleMode enum.

See also

ScaleMode

9.53.2 Member Data Documentation

9.53.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< ScaleMode >::kVALUE = 3  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.54 nvinfer1::impl::EnumMaxImpl< ScatterMode > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 2

9.54.1 Detailed Description

Maximum number of elements in ScatterMode enum.

See also

ScatterMode

9.54.2 Member Data Documentation

9.54.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< ScatterMode >::kVALUE = 2  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.55 nvinfer1::impl::EnumMaxImpl< SerializationFlag > Struct Reference

Maximum number of serialization flags in SerializationFlag enum.

`#include <NvInferRuntime.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.55.1 Detailed Description

Maximum number of serialization flags in SerializationFlag enum.

See also

SerializationFlag

9.55.2 Member Data Documentation

9.55.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< SerializationFlag >::kVALUE = 3  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInferRuntime.h

9.56 nvinfer2::safe::impl::EnumMaxImpl< Severity > Struct Reference

Provides a static constant for the maximum value of the Severity enum.

`#include <NvInferSafeRecorder.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 6

  Maximum number of elements in the Severity enum.

9.56.1 Detailed Description

Provides a static constant for the maximum value of the Severity enum.

9.56.2 Member Data Documentation

9.56.2.1 kVALUE

```
constexpr int32_t nvinfer2::safe::impl::EnumMaxImpl< Severity >::kVALUE = 6  [static], [constexpr]
```

Maximum number of elements in the Severity enum.

The documentation for this struct was generated from the following file:

- NvInferSafeRecorder.h

9.57 nvinfer1::impl::EnumMaxImpl< TacticSource > Struct Reference

Maximum number of tactic sources in TacticSource enum.

```
#include <NvInferRuntime.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 2

9.57.1 Detailed Description

Maximum number of tactic sources in TacticSource enum.

See also

TacticSource

9.57.2 Member Data Documentation

9.57.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< TacticSource >::kVALUE = 2  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInferRuntime.h

9.58 nvinfer1::impl::EnumMaxImpl< TempfileControlFlag > Struct Reference

Maximum number of elements in TempfileControlFlag enum.

```
#include <NvInferRuntime.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 2

9.58.1 Detailed Description

Maximum number of elements in TempfileControlFlag enum.

See also

TempfileControlFlag

9.58.2 Member Data Documentation

9.58.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< TempfileControlFlag >::kVALUE = 2  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInferRuntime.h

9.59 nvinfer1::impl::EnumMaxImpl< TensorFormat > Struct Reference

Maximum number of elements in TensorFormat enum.

`#include <NvInferRuntime.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 13

  Declaration of kVALUE that represents the maximum number of elements in the TensorFormat enum.

9.59.1 Detailed Description

Maximum number of elements in TensorFormat enum.

See also

TensorFormat

9.59.2 Member Data Documentation

9.59.2.1 kVALUE

`constexpr int32_t nvinfer1::impl::EnumMaxImpl< TensorFormat >::kVALUE = 13 [static], [constexpr]`

Declaration of kVALUE that represents the maximum number of elements in the TensorFormat enum.

The documentation for this struct was generated from the following file:

- NvInferRuntime.h

9.60 nvinfer1::impl::EnumMaxImpl< TensorIOMode > Struct Reference

One greater than the maximum value of TensorIOMode enum.

`#include <NvInferRuntimeBase.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.60.1 Detailed Description

One greater than the maximum value of TensorIOMode enum.

See also

TensorIOMode

9.60.2 Member Data Documentation

9.60.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< TensorIOMode >::kVALUE = 3  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInferRuntimeBase.h

9.61 nvinfer1::impl::EnumMaxImpl< TensorLocation > Struct Reference

Maximum number of elements in TensorLocation enum.

```
#include <NvInferRuntime.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 2

9.61.1 Detailed Description

Maximum number of elements in TensorLocation enum.

See also

TensorLocation

9.61.2 Member Data Documentation

9.61.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< TensorLocation >::kVALUE = 2  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInferRuntime.h

9.62 nvinfer1::impl::EnumMaxImpl< TilingOptimizationLevel > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 4

9.62.1 Detailed Description

Maximum number of elements in TilingOptimizationLevel enum.

See also

TilingOptimizationLevel

9.62.2 Member Data Documentation

9.62.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< TilingOptimizationLevel >::kVALUE = 4  [static],
[constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.63 nvinfer1::impl::EnumMaxImpl< TopKOperation > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 2

9.63.1 Detailed Description

Maximum number of elements in TopKOperation enum.

See also

TopKOperation

9.63.2 Member Data Documentation

9.63.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< TopKOperation >::kVALUE = 2  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.64 nvinfer1::impl::EnumMaxImpl< TripLimit > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 2

9.64.1 Detailed Description

Maximum number of elements in TripLimit enum.

See also

DataType

9.64.2 Member Data Documentation

9.64.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< TripLimit >::kVALUE = 2  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.65 nvinfer1::impl::EnumMaxImpl< UnaryOperation > Struct Reference

```
#include <NvInfer.h>
```

Static Public Attributes

- static constexpr int32_t kVALUE = 25

9.65.1 Detailed Description

Maximum number of elements in UnaryOperation enum.

See also

UnaryOperation

9.65.2 Member Data Documentation

9.65.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< UnaryOperation >::kVALUE = 25  [static], [constexpr]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.66 nvinfer1::impl::EnumMaxImpl< WeightsRole > Struct Reference

Maximum number of elements in WeightsRole enum.

`#include <NvInferRuntime.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 6

9.66.1 Detailed Description

Maximum number of elements in WeightsRole enum.

See also

WeightsRole

9.66.2 Member Data Documentation

9.66.2.1 kVALUE

`constexpr int32_t nvinfer1::impl::EnumMaxImpl< WeightsRole >::kVALUE = 6 [static], [constexpr]`

The documentation for this struct was generated from the following file:

- NvInferRuntime.h

9.67 nvinfer1::plugin::GridAnchorParameters Struct Reference

The Anchor Generator plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions (H x W). GridAnchorParameters defines a set of parameters for creating the plugin layer for all feature maps.

`#include <NvInferPluginUtils.h>`

Public Attributes

- float minSize

  Scale of anchors corresponding to finest resolution.
- float maxSize

  Scale of anchors corresponding to coarsest resolution.
- float ∗ aspectRatios

  List of aspect ratios to place on each grid point.
- int32_t numAspectRatios

  Number of elements in aspectRatios.
- int32_t H

  Height of feature map to generate anchors for.
- int32_t W

  Width of feature map to generate anchors for.
- float variance [4]

  Variance for adjusting the prior boxes.

9.67.1 Detailed Description

The Anchor Generator plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions (H x W). GridAnchorParameters defines a set of parameters for creating the plugin layer for all feature maps.

9.67.2 Member Data Documentation

9.67.2.1 aspectRatios

```
float* nvinfer1::plugin::GridAnchorParameters::aspectRatios
```

List of aspect ratios to place on each grid point.

9.67.2.2 H

```
int32_t nvinfer1::plugin::GridAnchorParameters::H
```

Height of feature map to generate anchors for.

9.67.2.3 maxSize

```
float nvinfer1::plugin::GridAnchorParameters::maxSize
```

Scale of anchors corresponding to coarsest resolution.

9.67.2.4 minSize

```
float nvinfer1::plugin::GridAnchorParameters::minSize
```

Scale of anchors corresponding to finest resolution.

9.67.2.5 numAspectRatios

```
int32_t nvinfer1::plugin::GridAnchorParameters::numAspectRatios
```

Number of elements in aspectRatios.

9.67.2.6 variance

```
float nvinfer1::plugin::GridAnchorParameters::variance[4]
```

Variance for adjusting the prior boxes.

9.67.2.7 W

```
int32_t nvinfer1::plugin::GridAnchorParameters::W
```

Width of feature map to generate anchors for.

The documentation for this struct was generated from the following file:

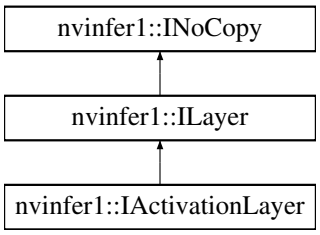
- NvInferPluginUtils.h

9.68 nvinfer1::IActivationLayer Class Reference

An Activation layer in a network definition.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IActivationLayer:



Public Member Functions

- void setActivationType (ActivationType type) noexcept
  Set the type of activation to be performed.
- ActivationType getActivationType () const noexcept
  Get the type of activation to be performed.
- void setAlpha (float alpha) noexcept
  Set the alpha parameter (must be finite).
- void setBeta (float beta) noexcept
  Set the beta parameter (must be finite).
- float getAlpha () const noexcept
  Get the alpha parameter.
- float getBeta () const noexcept
  Get the beta parameter.

Protected Member Functions

- virtual ~IActivationLayer () noexcept=0

Protected Attributes

- apiv::VActivationLayer * mImpl

9.68.1 Detailed Description

An Activation layer in a network definition.

This layer applies a per-element activation function to its input.

The output has the same shape as the input.

The input is a shape tensor if the output is a shape tensor.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.68.2 Constructor & Destructor Documentation

9.68.2.1 ~IActivationLayer()

```
nvinfer1::IActivationLayer::~IActivationLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.68.3 Member Function Documentation

9.68.3.1 getActivationType()

```
ActivationType nvinfer1::IActivationLayer::getActivationType ( ) const  [inline], [noexcept]
```

Get the type of activation to be performed.

See also

setActivationType(), ActivationType

9.68.3.2 getAlpha()

```
float nvinfer1::IActivationLayer::getAlpha ( ) const  [inline], [noexcept]
```

Get the alpha parameter.

See also

getBeta(), setAlpha()

9.68.3.3 getBeta()

```
float nvinfer1::IActivationLayer::getBeta ( ) const  [inline], [noexcept]
```

Get the beta parameter.

See also

getAlpha(), setBeta()

9.68.3.4 setActivationType()

```
void nvinfer1::IActivationLayer::setActivationType (
             ActivationType type )  [inline], [noexcept]
```

Set the type of activation to be performed.

On the DLA, the valid activation types are kRELU, kSIGMOID, kTANH, and kCLIP.

See also

getActivationType(), ActivationType

9.68.3.5 setAlpha()

```
void nvinfer1::IActivationLayer::setAlpha (
             float alpha )  [inline], [noexcept]
```

Set the alpha parameter (must be finite).

This parameter is used by the following activations: LeakyRelu, Elu, Selu, Softplus, Clip, HardSigmoid, ScaledTanh, ThresholdedRelu.

It is ignored by the other activations.

See also

getAlpha(), setBeta()

9.68.3.6 setBeta()

```
void nvinfer1::IActivationLayer::setBeta (
             float beta )  [inline], [noexcept]
```

Set the beta parameter (must be finite).

This parameter is used by the following activations: Selu, Softplus, Clip, HardSigmoid, ScaledTanh.

It is ignored by the other activations.

See also

getBeta(), setAlpha()

9.68.4 Member Data Documentation

9.68.4.1 mImpl

```
apiv::VActivationLayer* nvinfer1::IActivationLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

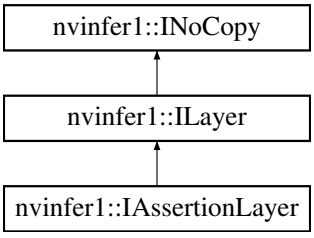
- NvInfer.h

9.69 nvinfer1::IAssertionLayer Class Reference

An assertion layer in a network.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IAssertionLayer:



Public Member Functions

- void setMessage (char const *message) noexcept

  Set the message to print if the assertion fails.
- char const * getMessage () const noexcept

  Return the assertion message.

Protected Member Functions

- virtual ~IAssertionLayer () noexcept=0

Protected Attributes

- apiv::VAssertionLayer * mImpl

9.69.1 Detailed Description

An assertion layer in a network.

The layer has a single input and no output. The input must be a boolean shape tensor. If any element of the input is provably false at build time, the network is rejected. If any element of the input is false at runtime for the supplied runtime dimensions, an error occurs, much the same as if any other runtime error (e.g. using IShuffleLayer to change the volume of a tensor) is handled.

Asserting equality of input dimensions may help the optimizer.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.69.2 Constructor & Destructor Documentation

9.69.2.1 ~IAssertionLayer()

```
nvinfer1::IAssertionLayer::~IAssertionLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.69.3 Member Function Documentation

9.69.3.1 getMessage()

```
char const * nvinfer1::IAssertionLayer::getMessage ( ) const  [inline], [noexcept]
```

Return the assertion message.

See also

setMessage()

9.69.3.2 setMessage()

```
void nvinfer1::IAssertionLayer::setMessage (
            char const * message )  [inline], [noexcept]
```

Set the message to print if the assertion fails.

The name is used in error diagnostics. This method copies the message string.

See also

getMessage()

9.69.4 Member Data Documentation

9.69.4.1 mImpl

```
apiv::VAssertionLayer* nvinfer1::IAssertionLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

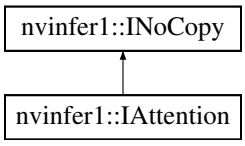
- NvInfer.h

9.70 nvinfer1::IAttention Class Reference

Helper for constructing an attention that consumes query, key and value tensors.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IAttention:



Public Member Functions

- bool setNormalizationOperation (AttentionNormalizationOp op) noexcept

  Set the normalization operation for the attention.
- AttentionNormalizationOp getNormalizationOperation () const noexcept

  Get the normalization operation for the attention.
- bool setMask (ITensor &mask) noexcept

  Set whether a mask will be used for the normalization operation.
- ITensor * getMask () noexcept

  Get the optional mask in attention.
- TRT_DEPRECATED bool setCausal (bool isCausal) noexcept

  Set whether the attention will run a causal inference. Cannot be used together with setMask().
- TRT_DEPRECATED bool getCausal () const noexcept

  Get whether the attention will run a causal inference.
- bool setCausalKind (CausalMaskKind kind) noexcept

  Set the causal mask alignment orientation for the attention.
- CausalMaskKind getCausalKind () const noexcept

  Get the causal mask alignment orientation for the attention.
- bool setDecomposable (bool decomposable) noexcept

  Set whether the attention can be decomposed to use multiple kernels if no fused kernel support found.
- bool getDecomposable () const noexcept

  Get whether the attention can be decomposed to use multiple kernels if no fused kernel support found.
- bool setInput (int32_t index, ITensor &input) noexcept

  Append or replace an input of this layer with a specific tensor.
- int32_t getNbInputs () const noexcept

  Get the number of inputs of IAttention. IAttention has three inputs.
- ITensor * getInput (int32_t index) const noexcept

  Get the IAttention input corresponding to the given index.
- int32_t getNbOutputs () const noexcept

  Get the number of outputs of a layer. IAttention has one output.
- ITensor * getOutput (int32_t index) const noexcept

  Get the IAttention output corresponding to the given index. IAttention has only one output.
- bool setName (char const *name) noexcept

  Set the name of the attention.
- char const * getName () const noexcept

  Return the name of the attention.
- bool setNormalizationQuantizeScale (ITensor &tensor) noexcept

  Set the quantization scale for the attention normalization output.
- ITensor * getNormalizationQuantizeScale () const noexcept

  Get the quantization scale for the attention normalization output.
- bool setNormalizationQuantizeToType (DataType type) noexcept

  Set the datatype the attention normalization is quantized to.
- DataType getNormalizationQuantizeToType () const noexcept

  Get the datatype the attention normalization is quantized to.
- bool setMetadata (char const *metadata) noexcept

  Set the metadata for IAttention.
- char const * getMetadata () const noexcept

Get the metadata of IAttention.

- bool setNbRanks (int32_t nbRanks) noexcept

  Set the number of ranks for multi-device attention execution.
- int32_t getNbRanks () const noexcept

  Get the number of ranks for multi-device execution.
- TRT_NODISCARD bool setQueryForm (AttentionIOForm form) noexcept

  Set the query form.
- TRT_NODISCARD AttentionIOForm getQueryForm () const noexcept

  Get the query form.
- TRT_NODISCARD bool setKeyValueForm (AttentionIOForm form) noexcept

  Set the key-value form.
- TRT_NODISCARD AttentionIOForm getKeyValueForm () const noexcept

  Get the key-value form.
- TRT_NODISCARD bool setQueryLengths (ITensor *lengths) noexcept

  Set the query lengths tensor.
- TRT_NODISCARD ITensor * getQueryLengths () const noexcept

  Get the query lengths tensor.
- TRT_NODISCARD bool setKeyValueLengths (ITensor *lengths) noexcept

  Set the key-value lengths tensor.
- TRT_NODISCARD ITensor * getKeyValueLengths () const noexcept

  Get the key-value lengths tensor.

Protected Member Functions

- virtual ~IAttention () noexcept=0

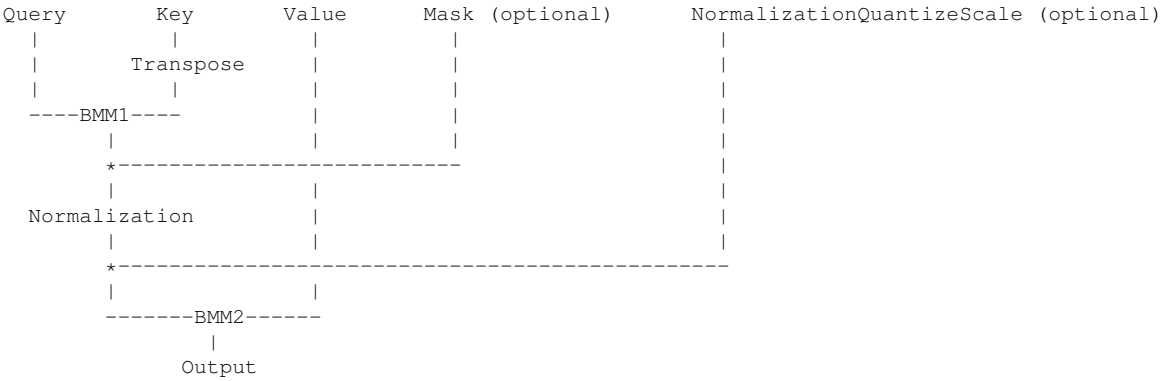
Protected Attributes

- apiv::VAttention * mImpl

9.70.1 Detailed Description

Helper for constructing an attention that consumes query, key and value tensors.

An attention subgraph implicitly includes three main components, two MatrixMultiply layers known as BMM1 and BMM2, and one normalization operation which defaults to be a Softmax. By default, IAttention is not decomposable and TensorRT will try to use a single fused kernel, which may be more efficient than if the subgraph is expressed without IAttention. Setting the IAttention to decomposable=True can allow IAttention to be decomposed to use multiple kernels if no fused kernel support found.



The attention has the following inputs, in order of input index:

- Query contains the input query. It is a tensor of type kFLOAT, kHALF or kBF16, its shape depends on the query form.
  - For query form kPADDED_BHND, shape is [batchSize, numHeadsQuery, numTokens, dimHead]
  - For query form kPACKED_NHD, shape is [totalTokens, numHeadsQuery, dimHead]
- Key contains the input key. It is a tensor of type kFLOAT, kHALF or kBF16, its shape depends on the key value form.
  - For key value form kPADDED_BHND, shape is [batchSize, numHeadsKeyValue, numTokens, dimHead]
  - For key value form kPACKED_NHD, shape is [totalTokens, numHeadsKeyValue, dimHead]
- Value contains the input value. It is a tensor of type kFLOAT, kHALF or kBF16, its shape depends on the key value form.
  - For key value form kPADDED_BHND, shape is [batchSize, numHeadsKeyValue, numTokens, dimHead]
  - For key value form kPACKED_NHD, shape is [totalTokens, numHeadsKeyValue, dimHead]
- Mask (optional) contains the mask value. It is a tensor of type kBOOL or the same data type of BMM1 output. Shape is [batchSize, numHeadsQuery, numTokensQuery, numTokensKeyValue] with batchSize and numHeadsQuery broadcastable. TensorRT uses stride-based indexing to load the mask data.
  - For a kBOOL mask, a True value indicates that the corresponding position is allowed to attend.
  - For other data types, the mask values will be added to the BMM1 output, known as an add mask.
- NormalizationQuantizeScale (optional) contains the quantization scale for the attention normalization output. It is a tensor of type kFLOAT, kHALF or kBF16 with dimension 0 or 1.

The attention has one output:

- Output has the same shape, form, and data type as the query input.
  - For query form kPADDED_BHND, output shape is [batchSize, numHeadsQuery, numTokens, dimHead]
  - For query form kPACKED_NHD, output shape is [totalTokens, numHeadsQuery, dimHead]

See also

https://docs.nvidia.com/deeplearning/tensorrt/latest/performance/best-practices.html#multi-head-attention-fusion for the complete matrix of fused kernel support.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.70.2 Constructor & Destructor Documentation

9.70.2.1 ~IAttention()

```
nvinfer1::IAttention::~IAttention ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.70.3 Member Function Documentation

9.70.3.1 getCausal()

```
TRT_DEPRECATED bool nvinfer1::IAttention::getCausal ( ) const  [inline], [noexcept]
```

Get whether the attention will run a causal inference.

See also

setCausal(), getCausalKind()

Deprecated Deprecated in TensorRT 10.16. Superseded by getCausalKind.

Returns

True if the attention will run a causal inference, false otherwise. Default is false.

9.70.3.2 getCausalKind()

```
CausalMaskKind nvinfer1::IAttention::getCausalKind ( ) const  [inline], [noexcept]
```

Get the causal mask alignment orientation for the attention.

See also

setCausalKind(), CausalMaskKind

Returns

The causal mask alignment orientation. Default is kNONE.

9.70.3.3 getDecomposable()

```
bool nvinfer1::IAttention::getDecomposable ( ) const  [inline], [noexcept]
```

Get whether the attention can be decomposed to use multiple kernels if no fused kernel support found.

Returns

True if the attention can be decomposed to use multiple kernels by the compiler, false otherwise. Default is false.

See also

setDecomposable

9.70.3.4 getInput()

```
ITensor * nvinfer1::IAttention::getInput (
            int32_t index ) const  [inline], [noexcept]
```

Get the IAttention input corresponding to the given index.

Parameters

index	The index of the input tensor.

Returns

The input tensor, or nullptr if the index is out of range.

9.70.3.5 getKeyValueForm()

```
TRT_NODISCARD AttentionIOForm nvinfer1::IAttention::getKeyValueForm ( ) const  [inline], [noexcept]
```

Get the key-value form.

Returns

The key-value form. Default is kPADDED_BHND.

See also

setKeyValueForm()

AttentionIOForm

9.70.3.6 getKeyValueLengths()

```
TRT_NODISCARD ITensor * nvinfer1::IAttention::getKeyValueLengths ( ) const  [inline], [noexcept]
```

Get the key-value lengths tensor.

Returns

The key-value lengths tensor, or nullptr if not set.

See also

setKeyValueLengths()

9.70.3.7 getMask()

```
ITensor * nvinfer1::IAttention::getMask ( )  [inline], [noexcept]
```

Get the optional mask in attention.

See also

setMask

Returns

The optional mask in attention, nullptr if no mask is set.

9.70.3.8 getMetadata()

```
char const * nvinfer1::IAttention::getMetadata ( ) const  [inline], [noexcept]
```

Get the metadata of IAttention.

Returns

The metadata as a null-terminated C-style string. If setMetadata() has not been called, an empty string "" will be returned as a default value.

See also

setMetadata()

9.70.3.9 getName()

```
char const * nvinfer1::IAttention::getName ( ) const  [inline], [noexcept]
```

Return the name of the attention.

See also

setName()

Returns

The name of the attention.

9.70.3.10 getNbInputs()

```
int32_t nvinfer1::IAttention::getNbInputs ( ) const  [inline], [noexcept]
```

Get the number of inputs of IAttention. IAttention has three inputs.

Returns

The number of inputs of IAttention.

9.70.3.11 getNbOutputs()

```
int32_t nvinfer1::IAttention::getNbOutputs ( ) const  [inline], [noexcept]
```

Get the number of outputs of a layer. IAttention has one output.

9.70.3.12 getNbRanks()

```
int32_t nvinfer1::IAttention::getNbRanks ( ) const  [inline], [noexcept]
```

Get the number of ranks for multi-device execution.

Returns

The number of ranks configured for multi-device attention. Default is 1.

See also

setNbRanks()

9.70.3.13 getNormalizationOperation()

```
AttentionNormalizationOp nvinfer1::IAttention::getNormalizationOperation ( ) const  [inline],
[noexcept]
```

Get the normalization operation for the attention.

See also

setNormalizationOperation(), AttentionNormalizationOp

Returns

The normalization operation for the attention. Default is kSOFTMAX.

9.70.3.14 getNormalizationQuantizeScale()

```
ITensor * nvinfer1::IAttention::getNormalizationQuantizeScale ( ) const  [inline], [noexcept]
```

Get the quantization scale for the attention normalization output.

Returns

The quantization scale for the attention normalization output or nullptr if no quantization scale is set.

9.70.3.15 getNormalizationQuantizeToType()

```
DataType nvinfer1::IAttention::getNormalizationQuantizeToType ( ) const  [inline], [noexcept]
```

Get the datatype the attention normalization is quantized to.

Returns

The datatype the attention normalization is quantized to. The default value is DataType::kFLOAT.

Warning

Must be used after normalization quantization to type is set by setNormalizationQuantizeToType.

9.70.3.16 getOutput()

```
ITensor * nvinfer1::IAttention::getOutput (
            int32_t index ) const  [inline], [noexcept]
```

Get the IAttention output corresponding to the given index. IAttention has only one output.

Parameters

index	The index of the output tensor.

Returns

The indexed output tensor, or nullptr if the index is out of range.

9.70.3.17 getQueryForm()

```
TRT_NODISCARD AttentionIOForm nvinfer1::IAttention::getQueryForm ( ) const  [inline], [noexcept]
```

Get the query form.

Returns

The query form. Default is kPADDED_BHND.

See also

setQueryForm()

AttentionIOForm

9.70.3.18 getQueryLengths()

```
TRT_NODISCARD ITensor * nvinfer1::IAttention::getQueryLengths ( ) const  [inline], [noexcept]
```

Get the query lengths tensor.

Returns

The query lengths tensor, or nullptr if not set.

See also

setQueryLengths()

9.70.3.19 setCausal()

```
TRT_DEPRECATED bool nvinfer1::IAttention::setCausal (
            bool isCausal )  [inline], [noexcept]
```

Set whether the attention will run a causal inference. Cannot be used together with setMask().

Parameters

isCausal	True to enable causal masking with kUPPER_LEFT alignment, false to disable causal masking.

See also

getCausal(), setCausalKind()

Deprecated Deprecated in TensorRT 10.16. Superseded by setCausalKind.

Returns

True if the causal inference is set successfully, false otherwise.

9.70.3.20 setCausalKind()

```
bool nvinfer1::IAttention::setCausalKind (
            CausalMaskKind kind )  [inline], [noexcept]
```

Set the causal mask alignment orientation for the attention.

When set to kUPPER_LEFT or kLOWER_RIGHT, an implicit causal mask is applied. When set to kNONE, no causal masking is applied.

Cannot be used together with setMask(). Building with both a mask tensor and a causal orientation other than kNONE will fail validation.

Parameters

kind	The causal mask alignment to apply.

See also

getCausalKind(), CausalMaskKind

Returns

True if the causal mask kind is set successfully, false otherwise.

9.70.3.21 setDecomposable()

```
bool nvinfer1::IAttention::setDecomposable (
            bool decomposable )  [inline], [noexcept]
```

Set whether the attention can be decomposed to use multiple kernels if no fused kernel support found.

See also

getDecomposable

Returns

True if the decomposable attention is set successfully, false otherwise.

9.70.3.22 setInput()

```
bool nvinfer1::IAttention::setInput (
            int32_t index,
            ITensor & input )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
input	the new input tensor.

The indices are as follows:

Input 0 is the input query tensor. Input 1 is the input key tensor. Input 2 is the input value tensor.

Returns

True if the input tensor is set successfully, false otherwise.

9.70.3.23 setKeyValueForm()

```
TRT_NODISCARD bool nvinfer1::IAttention::setKeyValueForm (
            AttentionIOForm form )  [inline], [noexcept]
```

Set the key-value form.

Default is kPADDED_BHND.

Parameters

form	The key-value form.

Returns

True if the key-value form is set successfully, false otherwise.

See also

getKeyValueForm()

AttentionIOForm

9.70.3.24 setKeyValueLengths()

```
TRT_NODISCARD bool nvinfer1::IAttention::setKeyValueLengths (
            ITensor * lengths )  [inline], [noexcept]
```

Set the key-value lengths tensor.

An optional tensor to specify per-batch key-value lengths. The semantics depend on the key-value form:

- When key-value form is kPADDED_BHND: contains per-batch lengths with shape [batchSize]. Each element must be <= the sequence length dimension of the KV tensor. If not set, the sequence length dimension of the KV tensor is used for all batches.
- When key-value form is kPACKED_NHD: contains cumulative token counts with shape [batchSize + 1]. The first element should be 0 and the last element equals totalTokens. The total_tokens dimension of the KV tensor must be >= the last element of this tensor. Must be set when key-value form is kPACKED_NHD.

Warning

When key-value form is kPACKED_NHD, providing a first element that is not 0 results in undefined behavior.

Parameters

lengths	A 1D tensor of type kINT32. If nullptr, clears a previously set key-value lengths tensor.

Returns

True if the key-value lengths tensor is set or cleared successfully, false otherwise.

See also

getKeyValueLengths()

9.70.3.25 setMask()

```
bool nvinfer1::IAttention::setMask (
            ITensor & mask )  [inline], [noexcept]
```

Set whether a mask will be used for the normalization operation.

Parameters

mask	the mask tensor of type kBOOL or the same data type of BMM1 output with 4d shape broadcastable to [batchSize, numHeadsQuery, sequenceLengthQuery, sequenceLengthKeyValue]. For a kBOOL mask, a True value indicates that the corresponding position is allowed to attend. For other data types, the mask values will be added to the BMM1 output, known as an add mask.

See also

getMask

Returns

True if the mask is set successfully, false otherwise.

9.70.3.26 setMetadata()

```
bool nvinfer1::IAttention::setMetadata (
            char const * metadata )  [inline], [noexcept]
```

Set the metadata for IAttention.

The metadata is emitted in the JSON returned by IEngineInspector with ProfilingVerbosity set to kDETAILED.

Parameters

metadata	The per-layer metadata.

Warning

The string name must be null-terminated and be at most 4096 bytes including the terminator.

See also

getMetadata()

getLayerInformation()

Returns

True if the metadata is set successfully, false otherwise.

9.70.3.27 setName()

```
bool nvinfer1::IAttention::setName (
            char const * name )  [inline], [noexcept]
```

Set the name of the attention.

The name is used in error diagnostics. This method copies the name string.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

See also

getName()

Returns

True if the name is set successfully, false otherwise.

9.70.3.28 setNbRanks()

```
bool nvinfer1::IAttention::setNbRanks (
            int32_t nbRanks )  [inline], [noexcept]
```

Set the number of ranks for multi-device attention execution.

When nbRanks > 1, this hints attention to perform multi-device attention.

Parameters

nbRanks	The number of ranks. Must be >= 1.

Returns

True if successful, false otherwise.

See also

getNbRanks()

9.70.3.29 setNormalizationOperation()

```
bool nvinfer1::IAttention::setNormalizationOperation (
            AttentionNormalizationOp op )  [inline], [noexcept]
```

Set the normalization operation for the attention.

See also

getNormalizationOperation(), AttentionNormalizationOp

Returns

True if the normalization operation is set successfully, false otherwise.

9.70.3.30 setNormalizationQuantizeScale()

```
bool nvinfer1::IAttention::setNormalizationQuantizeScale (
            ITensor & tensor )  [inline], [noexcept]
```

Set the quantization scale for the attention normalization output.

Parameters

tensor	for quantization scale. Data type must be DataType::kFLOAT, DataType::kHALF or DataType::kBF16. Must be a 0-d or 1-d.

Returns

True if the quantization scale is set successfully, false otherwise.

Warning

Must be used together with setNormalizationQuantizeToType to set normalization output datatype to DataType::kFP8 or DataType::kINT8.

9.70.3.31 setNormalizationQuantizeToType()

```
bool nvinfer1::IAttention::setNormalizationQuantizeToType (
            DataType type ) [inline], [noexcept]
```

Set the datatype the attention normalization is quantized to.

Parameters

type	the datatype the attention normalization is quantized to. Must be one of DataType::kFP8, DataType::kINT8.

Returns

True if the quantization to type is set successfully, false otherwise.

9.70.3.32 setQueryForm()

```
TRT_NODISCARD bool nvinfer1::IAttention::setQueryForm (
            AttentionIOForm form ) [inline], [noexcept]
```

Set the query form.

Default is kPADDED_BHND.

Parameters

form	The query form.

Returns

True if the query form is set successfully, false otherwise.

See also

getQueryForm()

AttentionIOForm

9.70.3.33 setQueryLengths()

```
TRT_NODISCARD bool nvinfer1::IAttention::setQueryLengths (
            ITensor * lengths )  [inline], [noexcept]
```

Set the query lengths tensor.

An optional tensor to specify the cumulative number of tokens per batch element. Must be set when query form is kPACKED_NHD. Ignored when query form is kPADDED_BHND. When set, contains cumulative token counts with shape [batchSize + 1]. The first element should be 0 and the last element equals totalTokens. The number of tokens for batch i is lengths[i + 1] - lengths[i]. The total_tokens dimension of the query tensor must be >= the last element of this tensor.

Warning

Providing a first element that is not 0 results in undefined behavior.

Parameters

lengths	A 1D tensor of type kINT32 with shape [batchSize + 1]. If nullptr, clears a previously set query lengths tensor.

Returns

True if the query lengths tensor is set or cleared successfully, false otherwise.

See also

getQueryLengths()

9.70.4 Member Data Documentation

9.70.4.1 mImpl

```
apiv::VAttention* nvinfer1::IAttention::mImpl  [protected]
```

The documentation for this class was generated from the following file:

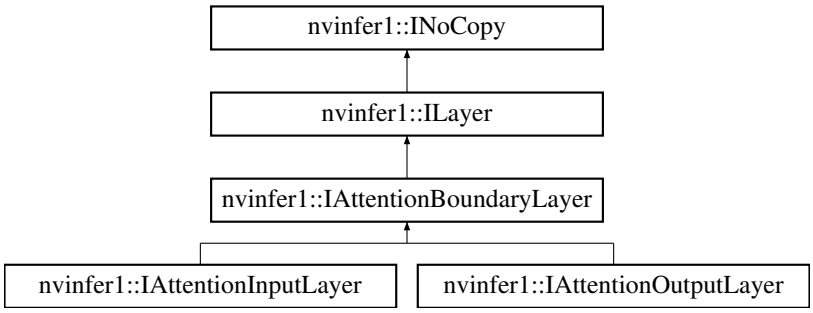
- NvInfer.h

9.71 nvinfer1::IAttentionBoundaryLayer Class Reference

This is a base class for Attention boundary layers.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IAttentionBoundaryLayer:



Public Member Functions

- IAttention * getAttention () const noexcept

  Get a pointer to the IAttention associated with this boundary layer.

Protected Member Functions

- virtual ~IAttentionBoundaryLayer () noexcept=0

Protected Attributes

- apiv::VAttentionBoundaryLayer * mBoundary

9.71.1 Detailed Description

This is a base class for Attention boundary layers.

Boundary layers are used to demarcate the boundaries of IAttention. Typically client code does not deal directly with the boundary layers. However, they are indirectly visible via method `INetworkDefinition::get↩Layer(int32_t index)`.

9.71.2 Constructor & Destructor Documentation

9.71.2.1 ~IAttentionBoundaryLayer()

```
nvinfer1::IAttentionBoundaryLayer::~IAttentionBoundaryLayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.71.3 Member Function Documentation

9.71.3.1 getAttention()

```
IAttention * nvinfer1::IAttentionBoundaryLayer::getAttention ( ) const  [inline], [noexcept]
```

Get a pointer to the IAttention associated with this boundary layer.

9.71.4 Member Data Documentation

9.71.4.1 mBoundary

```
apiv::VAttentionBoundaryLayer* nvinfer1::IAttentionBoundaryLayer::mBoundary  [protected]
```

The documentation for this class was generated from the following file:

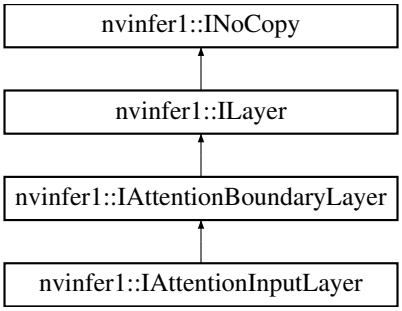
- NvInfer.h

9.72 nvinfer1::IAttentionInputLayer Class Reference

This layer represents an input to an attention subgraph.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IAttentionInputLayer:



Public Member Functions

- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~IAttentionInputLayer () noexcept=0

Protected Attributes

- apiv::VAttentionInputLayer * mImpl

9.72.1 Detailed Description

This layer represents an input to an attention subgraph.

This layer is automatically created when an `IAttention` is created. Clients typically do not deal with the layer directly, but instead specify its input via `addAttention` or `IAttention::setInput`.

An IAttentionInputLayer has three to four inputs and one output.

9.72.2 Constructor & Destructor Documentation

9.72.2.1 ~IAttentionInputLayer()

```
nvinfer1::IAttentionInputLayer::~IAttentionInputLayer ( )  [inline], [protected], [pure virtual],
[default], [noexcept]
```

9.72.3 Member Function Documentation

9.72.3.1 setInput()

```
void nvinfer1::ILayer::setInput (
            int32_t index,
            ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor

The indices are as follows:

Input 0 is the input query tensor. Input 1 is the input key tensor. Input 2 is the input value tensor. Input 3 is the optional mask tensor. setMask should be used instead of setInput Input 4 is the optional normalizationQuantizeScale tensor. setNormalizationQuantizeScale should be used instead of setInput Input 5 is the optional queryLengths tensor. setQueryLengths should be used instead of setInput Input 6 is the optional keyValueLengths tensor. setKeyValueLengths should be used instead of setInput

9.72.4 Member Data Documentation

9.72.4.1 mImpl

```
apiv::VAttentionInputLayer* nvinfer1::IAttentionInputLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

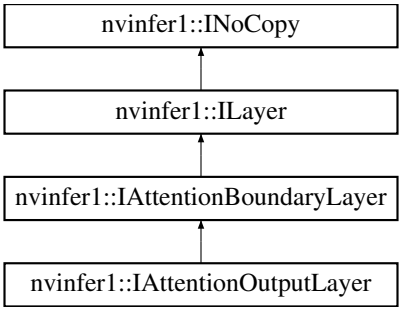
- NvInfer.h

9.73 nvinfer1::IAttentionOutputLayer Class Reference

This layer represents an output of an IAttention.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IAttentionOutputLayer:



Protected Member Functions

- virtual ~IAttentionOutputLayer () noexcept=0

Protected Attributes

- apiv::VAttentionOutputLayer * mImpl

Additional Inherited Members

9.73.1 Detailed Description

This layer represents an output of an IAttention.

This layer is automatically created when an `IAttention` is created. Clients typically do not deal with the layer directly, but instead getting its output via `IAttention::getOutput`.

An IAttentionOutputLayer has one input and one output.

9.73.2 Constructor & Destructor Documentation

9.73.2.1 ~IAttentionOutputLayer()

```
nvinfer1::IAttentionOutputLayer::~IAttentionOutputLayer ( )  [inline], [protected], [pure virtual],
[default], [noexcept]
```

9.73.3 Member Data Documentation

9.73.3.1 mImpl

```
apiv::VAttentionOutputLayer* nvinfer1::IAttentionOutputLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

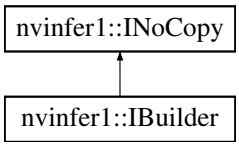
- NvInfer.h

9.74 nvinfer1::IBuilder Class Reference

Builds an engine from a network definition.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IBuilder:



Public Member Functions

- virtual ~IBuilder () noexcept=0
- int32_t getMaxDLABatchSize () const noexcept

  Get the maximum batch size DLA can support. For any tensor the total volume of index dimensions combined(dimensions other than CHW) with the requested batch size should not exceed the value returned by this function.
- int32_t getNbDLACores () const noexcept

  Return the number of DLA engines available to this builder.
- void setGpuAllocator (IGpuAllocator *allocator) noexcept

  Set the GPU allocator.
- nvinfer1::IBuilderConfig * createBuilderConfig () noexcept

  Create a builder configuration object.
- nvinfer1::INetworkDefinition * createNetworkV2 (NetworkDefinitionCreationFlags flags) noexcept

  Create a network definition object.
- nvinfer1::IOptimizationProfile * createOptimizationProfile () noexcept

  Create a new optimization profile.
- void setErrorRecorder (IErrorRecorder *recorder) noexcept

  Set the ErrorRecorder for this interface.
- IErrorRecorder * getErrorRecorder () const noexcept

  get the ErrorRecorder assigned to this interface.
- void reset () noexcept

  Resets the builder state to default values.
- nvinfer1::IHostMemory * buildSerializedNetwork (INetworkDefinition &network, IBuilderConfig &config) noexcept

  Builds and serializes a network for the given INetworkDefinition and IBuilderConfig.
- bool buildSerializedNetworkToStream (INetworkDefinition &network, IBuilderConfig &config, IStreamWriter &writer) noexcept

  Builds and serializes a network into stream for the given INetworkDefinition and IBuilderConfig.
- nvinfer1::IHostMemory * buildSerializedNetwork (INetworkDefinition &network, IBuilderConfig &config, IHostMemory *&kernelText) noexcept

  Extended form of buildSerializedNetwork that optionally permits getting the kernelText.
- nvinfer1::ICudaEngine * buildEngineWithConfig (INetworkDefinition &network, IBuilderConfig &config) noexcept

Builds a network for the given INetworkDefinition and IBuilderConfig.

- bool isNetworkSupported (INetworkDefinition const &network, IBuilderConfig const &config) const noexcept

  Checks that a network is within the scope of the IBuilderConfig settings.
- ILogger * getLogger () const noexcept

  get the logger with which the builder was created
- bool setMaxThreads (int32_t maxThreads) noexcept

  Set the maximum number of threads.
- int32_t getMaxThreads () const noexcept

  get the maximum number of threads that can be used by the builder.
- IPluginRegistry & getPluginRegistry () noexcept

  get the local plugin registry that can be used by the builder.

Protected Attributes

- apiv::VBuilder * mImpl

Additional Inherited Members

9.74.1 Detailed Description

Builds an engine from a network definition.

> **Warning**
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.74.2 Constructor & Destructor Documentation

9.74.2.1 ~IBuilder()

```
nvinfer1::IBuilder::~IBuilder ( )  [inline], [pure virtual], [default], [noexcept]
```

9.74.3 Member Function Documentation

9.74.3.1 buildEngineWithConfig()

```
nvinfer1::ICudaEngine * nvinfer1::IBuilder::buildEngineWithConfig (
            INetworkDefinition & network,
            IBuilderConfig & config )  [inline], [noexcept]
```

Builds a network for the given INetworkDefinition and IBuilderConfig.

Parameters

network	Network definition.
config	Builder configuration.

Returns

A pointer to a ICudaEngine object that contains an engine.

Note

This function will synchronize the CUDA stream returned by `config.getProfileStream()` before returning.

This function does not support `BuilderFlag::kVERSION_COMPATIBLE`. Please use `buildSerializedNetwork` to get a version compatible engine.

See also

INetworkDefinition, IBuilderConfig, ICudaEngine

9.74.3.2 buildSerializedNetwork() [1/2]

```
nvinfer1::IHostMemory * nvinfer1::IBuilder::buildSerializedNetwork (
            INetworkDefinition & network,
            IBuilderConfig & config )  [inline], [noexcept]
```

Builds and serializes a network for the given INetworkDefinition and IBuilderConfig.

This function allows building and serialization of a network without creating an engine.

Parameters

network	Network definition.
config	Builder configuration.

Returns

A pointer to a IHostMemory object that contains a serialized network.

Note

This function will synchronize the CUDA stream returned by `config.getProfileStream()` before returning.

See also

INetworkDefinition, IBuilderConfig, IHostMemory

9.74.3.3 buildSerializedNetwork() [2/2]

```
nvinfer1::IHostMemory * nvinfer1::IBuilder::buildSerializedNetwork (
            INetworkDefinition & network,
            IBuilderConfig & config,
            IHostMemory *& kernelText )  [inline], [noexcept]
```

Extended form of buildSerializedNetwork that optionally permits getting the kernelText.

Similar to two-argument form, except that if an engine with safe capability is successfully built and there are kernels, sets kernelText to ..... Otherwise sets kernelText=nullptr.

This function allows building and serialization of a network without creating an engine.

Parameters

network	Network definition.
config	Builder configuration.
kernelText	A reference to a pointer to a IHostMemory object that will be set to the kernel CPP code text

Returns

A pointer to a IHostMemory object that contains a serialized network.

Note

This function will synchronize the CUDA stream returned by `config.getProfileStream()` before returning.

See also

INetworkDefinition, IBuilderConfig, IHostMemory

9.74.3.4 buildSerializedNetworkToStream()

```
bool nvinfer1::IBuilder::buildSerializedNetworkToStream (
            INetworkDefinition & network,
            IBuilderConfig & config,
            IStreamWriter & writer )  [inline], [noexcept]
```

Builds and serializes a network into stream for the given INetworkDefinition and IBuilderConfig.

This function allows building and serialization of a network without creating an engine. The engine is finally serialized into the writer stream.

Parameters

network	Network definition.
config	Builder configuration.
writer	Output writer stream.

Returns

true if build succeed, otherwise false.

Note

This function will synchronize the CUDA stream returned by `config.getProfileStream()` before returning.

See also

INetworkDefinition, IBuilderConfig, IStreamWriter

9.74.3.5 createBuilderConfig()

```
nvinfer1::IBuilderConfig * nvinfer1::IBuilder::createBuilderConfig ( )  [inline], [noexcept]
```

Create a builder configuration object.

The caller owns the new IBuilderConfig, which must be destroyed with operator delete before this IBuilder is destroyed. Destroying this IBuilder before destroying the IBuilderConfig causes undefined behavior.

See also

IBuilderConfig

9.74.3.6 createNetworkV2()

```
nvinfer1::INetworkDefinition * nvinfer1::IBuilder::createNetworkV2 (
            NetworkDefinitionCreationFlags flags )  [inline], [noexcept]
```

Create a network definition object.

Creates a network definition object with immutable properties specified using the flags parameter.

createNetworkV2 supports creating network with properties from NetworkDefinitionCreationFlags.

CreateNetworkV2 supports dynamic shapes and explicit batch dimensions by default.

The network is always strongly typed: tensor data types are inferred from network input types and operator type specification. The kSTRONGLY_TYPED flag is deprecated and ignored.

The caller owns the new INetworkDefinition, which must be destroyed with operator delete before this IBuilder is destroyed. Destroying this IBuilder before destroying the INetworkDefinition causes undefined behavior.

Parameters

flags	Bitset of NetworkDefinitionCreationFlags specifying network properties combined with bitwise OR, e.g., 1U << NetworkDefinitionCreationFlag::kPREFER_JIT_PYTHON_PLUGINS.

See also

INetworkDefinition, NetworkDefinitionCreationFlags

9.74.3.7 createOptimizationProfile()

```
nvinfer1::IOptimizationProfile * nvinfer1::IBuilder::createOptimizationProfile ( )  [inline], [noexcept]
```

Create a new optimization profile.

If the network has any dynamic input tensors, the appropriate calls to setDimensions() must be made. Likewise, if there are any shape input tensors, the appropriate calls to IOptimizationProfile::setShapeValuesV2() are required. The builder retains ownership of the created optimization profile and returns a raw pointer, i.e. the users must not attempt to delete the returned pointer.

See also

IOptimizationProfile

9.74.3.8 getErrorRecorder()

```
IErrorRecorder * nvinfer1::IBuilder::getErrorRecorder ( ) const  [inline], [noexcept]
```

get the ErrorRecorder assigned to this interface.

Retrieves the assigned error recorder object for the given class. A nullptr will be returned if setErrorRecorder has not been called.

Returns

A pointer to the IErrorRecorder object that has been registered.

See also

setErrorRecorder()

9.74.3.9 getLogger()

```
ILogger * nvinfer1::IBuilder::getLogger ( ) const  [inline], [noexcept]
```

get the logger with which the builder was created

Returns

the logger

9.74.3.10 getMaxDLABatchSize()

```
int32_t nvinfer1::IBuilder::getMaxDLABatchSize ( ) const  [inline], [noexcept]
```

Get the maximum batch size DLA can support. For any tensor the total volume of index dimensions combined(dimensions other than CHW) with the requested batch size should not exceed the value returned by this function.

Warning

getMaxDLABatchSize does not work with dynamic shapes.

9.74.3.11 getMaxThreads()

```
int32_t nvinfer1::IBuilder::getMaxThreads ( ) const  [inline], [noexcept]
```

get the maximum number of threads that can be used by the builder.

Retrieves the maximum number of threads that can be used by the builder.

Returns

The maximum number of threads that can be used by the builder.

See also

setMaxThreads()

9.74.3.12 getNbDLACores()

```
int32_t nvinfer1::IBuilder::getNbDLACores ( ) const  [inline], [noexcept]
```

Return the number of DLA engines available to this builder.

9.74.3.13 getPluginRegistry()

```
IPluginRegistry & nvinfer1::IBuilder::getPluginRegistry ( )  [inline], [noexcept]
```

get the local plugin registry that can be used by the builder.

Returns

The local plugin registry that can be used by the builder.

9.74.3.14 isNetworkSupported()

```
bool nvinfer1::IBuilder::isNetworkSupported (
            INetworkDefinition const & network,
            IBuilderConfig const & config ) const  [inline], [noexcept]
```

Checks that a network is within the scope of the IBuilderConfig settings.

Parameters

network	The network definition to check for configuration compliance.
config	The configuration of the builder to use when checking `network`.

Given an INetworkDefinition, `network`, and an IBuilderConfig, `config`, check if the network falls within the constraints of the builder configuration based on the EngineCapability, BuilderFlag, and DeviceType. If the network is within the constraints, then the function returns true, and false if a violation occurs. This function reports the conditions that are violated to the registered ErrorRecorder.

Returns

True if network is within the scope of the restrictions specified by the builder config, false otherwise.

Note

A `true` return value does not guarantee that engine building will succeed, as backends may reject it for reasons not detectable with this fast validation. To definitively check whether a network can be built with a given config, use `buildEngineWithConfig` or `buildSerializedNetwork` (depending on the engine capability).

This function will synchronize the CUDA stream returned by `config.getProfileStream()` before returning.

9.74.3.15 reset()

```
void nvinfer1::IBuilder::reset ( )  [inline], [noexcept]
```

Resets the builder state to default values.

9.74.3.16 setErrorRecorder()

```
void nvinfer1::IBuilder::setErrorRecorder (
            IErrorRecorder * recorder )  [inline], [noexcept]
```

Set the ErrorRecorder for this interface.

Assigns the ErrorRecorder to this interface. The ErrorRecorder will track all errors during execution. This function will call incRefCount of the registered ErrorRecorder at least once. Setting recorder to nullptr unregisters the recorder with the interface, resulting in a call to decRefCount if a recorder has been registered.

If an error recorder is not set, messages will be sent to the global log stream.

Parameters

recorder	The error recorder to register with this interface.

See also

getErrorRecorder()

9.74.3.17 setGpuAllocator()

```
void nvinfer1::IBuilder::setGpuAllocator (
            IGpuAllocator * allocator )  [inline], [noexcept]
```

Set the GPU allocator.

Parameters

allocator	Set the GPU allocator to be used by the builder. All GPU memory acquired will use this allocator. If NULL is passed, the default allocator will be used.

Default: allocateAsync uses cudaMallocAsync if cudaDevAttrMemoryPoolsSupported returns true, otherwise falls back to cudaMalloc. allocate always uses cudaMalloc.

Note

This allocator will be passed to any engines created via the builder; thus the lifetime of the allocator must span the lifetime of those engines as well as that of the builder. If nullptr is passed, the default allocator will be used.

9.74.3.18 setMaxThreads()

```
bool nvinfer1::IBuilder::setMaxThreads (
             int32_t maxThreads ) [inline], [noexcept]
```

Set the maximum number of threads.

Parameters

maxThreads	The maximum number of threads that can be used by the builder.

Returns

True if successful, false otherwise.

The default value is 1 and includes the current thread. A value greater than 1 permits TensorRT to use multi-threaded algorithms. A value less than 1 triggers a kINVALID_ARGUMENT error.

9.74.4 Member Data Documentation

9.74.4.1 mImpl

```
apiv::VBuilder* nvinfer1::IBuilder::mImpl [protected]
```

The documentation for this class was generated from the following file:

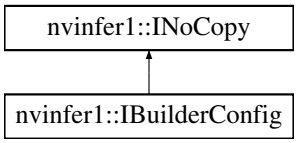
- NvInfer.h

9.75 nvinfer1::IBuilderConfig Class Reference

Holds properties for configuring a builder to produce an engine.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IBuilderConfig:



Public Member Functions

- virtual ~IBuilderConfig () noexcept=0
- virtual void setAvgTimingIterations (int32_t avgTiming) noexcept

  Set the number of averaging iterations used when timing layers.
- int32_t getAvgTimingIterations () const noexcept

  Query the number of averaging iterations.
- void setEngineCapability (EngineCapability capability) noexcept

  Configure the builder to target specified EngineCapability flow.
- EngineCapability getEngineCapability () const noexcept

  Query EngineCapability flow configured for the builder.
- void setFlags (BuilderFlags builderFlags) noexcept

  Set the build mode flags to turn on builder options for this network.
- BuilderFlags getFlags () const noexcept

  Get the build mode flags for this builder config. Defaults to 0.
- void clearFlag (BuilderFlag builderFlag) noexcept

  clear a single build mode flag.
- void setFlag (BuilderFlag builderFlag) noexcept

  Set a single build mode flag.
- bool getFlag (BuilderFlag builderFlag) const noexcept

  Returns true if the build mode flag is set.
- void setDeviceType (ILayer const *layer, DeviceType deviceType) noexcept

  Set the device that this layer must execute on.
- DeviceType getDeviceType (ILayer const *layer) const noexcept

  Get the device that this layer executes on.
- bool isDeviceTypeSet (ILayer const *layer) const noexcept

  whether the DeviceType has been explicitly set for this layer
- void resetDeviceType (ILayer const *layer) noexcept

  reset the DeviceType for this layer
- bool canRunOnDLA (ILayer const *layer) const noexcept

  Checks if a layer can run on DLA.
- void setDLACore (int32_t dlaCore) noexcept

  Sets the DLA core used by the network. Defaults to -1.

- int32_t getDLACore () const noexcept

  Get the DLA core that the engine executes on.
- void setDefaultDeviceType (DeviceType deviceType) noexcept

  Sets the default DeviceType to be used by the builder. It ensures that all the layers that can run on this device will run on it, unless setDeviceType is used to override the default DeviceType for a layer.
- DeviceType getDefaultDeviceType () const noexcept

  Get the default DeviceType which was set by setDefaultDeviceType.
- void reset () noexcept

  Resets the builder configuration to defaults.
- void setProfileStream (const cudaStream_t stream) noexcept

  Set the CUDA stream that is used to profile this network.
- cudaStream_t getProfileStream () const noexcept

  Get the CUDA stream that is used to profile this network.
- int32_t addOptimizationProfile (IOptimizationProfile const *profile) noexcept

  Add an optimization profile.
- int32_t getNbOptimizationProfiles () const noexcept

  Get number of optimization profiles.
- void setProfilingVerbosity (ProfilingVerbosity verbosity) noexcept

  Set verbosity level of layer information exposed in NVTX annotations and IEngineInspector.
- ProfilingVerbosity getProfilingVerbosity () const noexcept

  Get verbosity level of layer information exposed in NVTX annotations and IEngineInspector.
- bool setTacticSources (TacticSources tacticSources) noexcept

  Set tactic sources.
- TacticSources getTacticSources () const noexcept

  Get tactic sources.
- nvinfer1::ITimingCache * createTimingCache (void const *blob, std::size_t size) const noexcept

  Create timing cache.
- bool setTimingCache (ITimingCache const &cache, bool ignoreMismatch) noexcept

  Attach a timing cache to IBuilderConfig.
- nvinfer1::ITimingCache const * getTimingCache () const noexcept

  Get the pointer to the timing cache from current IBuilderConfig.
- void setMemoryPoolLimit (MemoryPoolType pool, std::size_t poolSize) noexcept

  Set the memory size for the memory pool.
- std::size_t getMemoryPoolLimit (MemoryPoolType pool) const noexcept

  Get the memory size limit of the memory pool.
- void setPreviewFeature (PreviewFeature feature, bool enable) noexcept

  Enable or disable a specific preview feature.
- bool getPreviewFeature (PreviewFeature feature) const noexcept

  Get status of preview feature.
- void setBuilderOptimizationLevel (int32_t level) noexcept

  Set builder optimization level.
- int32_t getBuilderOptimizationLevel () noexcept

  Get builder optimization level.
- void setHardwareCompatibilityLevel (HardwareCompatibilityLevel hardwareCompatibilityLevel) noexcept

  Set the hardware compatibility level.
- HardwareCompatibilityLevel getHardwareCompatibilityLevel () const noexcept

  Get the hardware compatibility level.

- void setPluginsToSerialize (char const *const *paths, int32_t nbPaths) noexcept

  Set the plugin libraries to be serialized with version-compatible engines.
- char const * getPluginToSerialize (int32_t index) const noexcept

  Get the plugin library path to be serialized with version-compatible engines.
- int32_t getNbPluginsToSerialize () const noexcept

  Get the number of plugin library paths to be serialized with version-compatible engines.
- bool setMaxAuxStreams (int32_t nbStreams) noexcept

  Set the maximum number of auxiliary streams that TRT is allowed to use.
- int32_t getMaxAuxStreams () const noexcept

  Get the maximum number of auxiliary streams that TRT is allowed to use.
- void setProgressMonitor (IProgressMonitor *monitor) noexcept

  Sets the progress monitor for building a network.
- IProgressMonitor * getProgressMonitor () const noexcept
- void setRuntimePlatform (RuntimePlatform runtimePlatform) noexcept

  Set the target platform for runtime execution.
- RuntimePlatform getRuntimePlatform () const noexcept

  Get the target platform for runtime execution.
- void setMaxNbTactics (int32_t maxNbTactics) noexcept

  Set the maximum number of tactics to time when there is a choice of tactics.
- int32_t getMaxNbTactics () const noexcept

  Query the maximum number of tactics timed when there is a choice.
- bool setTilingOptimizationLevel (TilingOptimizationLevel level) noexcept

  Set the Tiling optimization level.
- TilingOptimizationLevel getTilingOptimizationLevel () const noexcept

  Get the Tiling optimization level.
- bool setL2LimitForTiling (int64_t size) noexcept

  Set the L2 cache usage limit for Tiling optimization.
- int64_t getL2LimitForTiling () const noexcept

  Get the L2 cache usage limit for tiling optimization.
- bool setRemoteAutoTuningConfig (char const *config) noexcept

  Set a config string for remote auto tuning.
- char const * getRemoteAutoTuningConfig () const noexcept

  Get a config string for remote auto tuning.

Protected Attributes

- apiv::VBuilderConfig * mImpl

Additional Inherited Members

9.75.1 Detailed Description

Holds properties for configuring a builder to produce an engine.

See also

BuilderFlags

9.75.2 Constructor & Destructor Documentation

9.75.2.1 ~IBuilderConfig()

```
nvinfer1::IBuilderConfig::~IBuilderConfig ( )  [inline], [pure virtual], [default], [noexcept]
```

9.75.3 Member Function Documentation

9.75.3.1 addOptimizationProfile()

```
int32_t nvinfer1::IBuilderConfig::addOptimizationProfile (
            IOptimizationProfile const * profile )  [inline], [noexcept]
```

Add an optimization profile.

This function must be called at least once if the network has dynamic or shape input tensors. This function may be called at most once when building a refittable engine, as more than a single optimization profile are not supported for refittable engines.

Parameters

profile	The new optimization profile, which must satisfy profile->isValid() == true

Returns

The index of the optimization profile (starting from 0) if the input is valid, or -1 if the input is not valid.

9.75.3.2 canRunOnDLA()

```
bool nvinfer1::IBuilderConfig::canRunOnDLA (
            ILayer const * layer ) const  [inline], [noexcept]
```

Checks if a layer can run on DLA.

Returns

status true if the layer can on DLA else returns false.

9.75.3.3 clearFlag()

```
void nvinfer1::IBuilderConfig::clearFlag (
            BuilderFlag builderFlag )  [inline], [noexcept]
```

clear a single build mode flag.

clears the builder mode flag from the enabled flags.

See also

setFlags()

9.75.3.4 createTimingCache()

```
nvinfer1::ITimingCache * nvinfer1::IBuilderConfig::createTimingCache (
            void const * blob,
            std::size_t size ) const  [inline], [noexcept]
```

Create timing cache.

Create ITimingCache instance from serialized raw data. The created timing cache doesn't belong to a specific IBuilderConfig. It can be shared by multiple builder instances. Call setTimingCache() before launching a builder to attach cache to builder instance. The lifetime of the ITimingCache must exceed the lifetime of all builders that use it.

Parameters

blob	A pointer to the raw data that contains serialized timing cache
size	The size in bytes of the serialized timing cache. Size 0 means create a new cache from scratch

See also

setTimingCache

Returns

the pointer to ITimingCache created

9.75.3.5 getAvgTimingIterations()

```
int32_t nvinfer1::IBuilderConfig::getAvgTimingIterations ( ) const  [inline], [noexcept]
```

Query the number of averaging iterations.

By default the number of averaging iterations is 1.

See also

setAvgTimingIterations()

9.75.3.6 getBuilderOptimizationLevel()

```
int32_t nvinfer1::IBuilderConfig::getBuilderOptimizationLevel ( )  [inline], [noexcept]
```

Get builder optimization level.

Returns

the current builder optimization level

See also

setBuilderOptimizationLevel

9.75.3.7 getDefaultDeviceType()

```
DeviceType nvinfer1::IBuilderConfig::getDefaultDeviceType ( ) const  [inline], [noexcept]
```

Get the default DeviceType which was set by setDefaultDeviceType.

By default it returns DeviceType::kGPU.

9.75.3.8 getDeviceType()

```
DeviceType nvinfer1::IBuilderConfig::getDeviceType (
            ILayer const * layer ) const  [inline], [noexcept]
```

Get the device that this layer executes on.

Returns

Returns DeviceType of the layer.

9.75.3.9 getDLACore()

```
int32_t nvinfer1::IBuilderConfig::getDLACore ( ) const  [inline], [noexcept]
```

Get the DLA core that the engine executes on.

Returns

assigned DLA core or -1 for DLA not present or unset.

9.75.3.10 getEngineCapability()

```
EngineCapability nvinfer1::IBuilderConfig::getEngineCapability ( ) const  [inline], [noexcept]
```

Query EngineCapability flow configured for the builder.

By default it returns EngineCapability::kSTANDARD.

See also

setEngineCapability()

9.75.3.11 getFlag()

```
bool nvinfer1::IBuilderConfig::getFlag (
            BuilderFlag builderFlag ) const  [inline], [noexcept]
```

Returns true if the build mode flag is set.

See also

getFlags()

Returns

True if flag is set, false if unset.

9.75.3.12 getFlags()

```
BuilderFlags nvinfer1::IBuilderConfig::getFlags ( ) const  [inline], [noexcept]
```

Get the build mode flags for this builder config. Defaults to 0.

Returns

The build options as a bitmask.

See also

setFlags()

9.75.3.13 getHardwareCompatibilityLevel()

```
HardwareCompatibilityLevel nvinfer1::IBuilderConfig::getHardwareCompatibilityLevel ( ) const  [inline],
[noexcept]
```

Get the hardware compatibility level.

Returns

hardwareCompatibilityLevel The level of hardware compatibility.

See also

setHardwareCompatibilityLevel()

9.75.3.14 getL2LimitForTiling()

```
int64_t nvinfer1::IBuilderConfig::getL2LimitForTiling ( ) const  [inline], [noexcept]
```

Get the L2 cache usage limit for tiling optimization.

Returns

L2 cache usage limit for tiling optimization.

See also

setL2LimitForTiling()

9.75.3.15 getMaxAuxStreams()

```
int32_t nvinfer1::IBuilderConfig::getMaxAuxStreams ( ) const  [inline], [noexcept]
```

Get the maximum number of auxiliary streams that TRT is allowed to use.

See also

setMaxAuxStreams()

9.75.3.16 getMaxNbTactics()

```
int32_t nvinfer1::IBuilderConfig::getMaxNbTactics ( ) const  [inline], [noexcept]
```

Query the maximum number of tactics timed when there is a choice.

By default the value is -1, indicating TensorRT can determine the number of tactics based on its own heuristic.

See also

setMaxNbTactics()

9.75.3.17 getMemoryPoolLimit()

```
std::size_t nvinfer1::IBuilderConfig::getMemoryPoolLimit (
            MemoryPoolType pool ) const  [inline], [noexcept]
```

Get the memory size limit of the memory pool.

Retrieve the memory size limit of the corresponding pool in bytes. If setMemoryPoolLimit for the pool has not been called, this returns the default value used by TensorRT. This default value is not necessarily the maximum possible value for that configuration.

Parameters

pool	The memory pool to get the limit for.

Returns

The size of the memory limit, in bytes, for the corresponding pool.

See also

setMemoryPoolLimit

9.75.3.18 getNbOptimizationProfiles()

```
int32_t nvinfer1::IBuilderConfig::getNbOptimizationProfiles ( ) const  [inline], [noexcept]
```

Get number of optimization profiles.

This is one higher than the index of the last optimization profile that has be defined (or zero, if none has been defined yet).

Returns

The number of the optimization profiles.

9.75.3.19 getNbPluginsToSerialize()

```
int32_t nvinfer1::IBuilderConfig::getNbPluginsToSerialize ( ) const  [inline], [noexcept]
```

Get the number of plugin library paths to be serialized with version-compatible engines.

Returns

The number of paths.

9.75.3.20 getPluginToSerialize()

```
char const * nvinfer1::IBuilderConfig::getPluginToSerialize (
            int32_t index ) const  [inline], [noexcept]
```

Get the plugin library path to be serialized with version-compatible engines.

Parameters

index	Index of the plugin library path in the list. Should be in the range [0, getNbPluginsToSerialize()).

Returns

The path to the plugin library.

9.75.3.21 getPreviewFeature()

```
bool nvinfer1::IBuilderConfig::getPreviewFeature (
            PreviewFeature feature ) const  [inline], [noexcept]
```

Get status of preview feature.

Parameters

feature	the feature to query

Returns

true if the `feature` is enabled, false otherwise

See also

PreviewFeature, setPreviewFeature

9.75.3.22 getProfileStream()

```
cudaStream_t nvinfer1::IBuilderConfig::getProfileStream ( ) const  [inline], [noexcept]
```

Get the CUDA stream that is used to profile this network.

Returns

The CUDA stream set by setProfileStream, nullptr if setProfileStream has not been called.

See also

setProfileStream()

9.75.3.23 getProfilingVerbosity()

```
ProfilingVerbosity nvinfer1::IBuilderConfig::getProfilingVerbosity ( ) const  [inline], [noexcept]
```

Get verbosity level of layer information exposed in NVTX annotations and IEngineInspector.

Get the current setting of verbosity level of layer information exposed in NVTX annotations and IEngineInspector. Default value is ProfilingVerbosity::kLAYER_NAMES_ONLY.

See also

ProfilingVerbosity, setProfilingVerbosity(), IEngineInspector

9.75.3.24 getProgressMonitor()

```
IProgressMonitor * nvinfer1::IBuilderConfig::getProgressMonitor ( ) const  [inline], [noexcept]
```

Returns

The progress monitor set by the application or nullptr.

See also

IBuilderConfig::setProgressMonitor

9.75.3.25 getRemoteAutoTuningConfig()

```
char const * nvinfer1::IBuilderConfig::getRemoteAutoTuningConfig ( ) const  [inline], [noexcept]
```

Get a config string for remote auto tuning.

Returns

The current string for remote auto tuning, or nullptr if not set.

9.75.3.26 getRuntimePlatform()

```
RuntimePlatform nvinfer1::IBuilderConfig::getRuntimePlatform ( ) const  [inline], [noexcept]
```

Get the target platform for runtime execution.

Returns

The target platform for runtime execution.

See also

IBuilderConfig::setRuntimePlatform()

9.75.3.27 getTacticSources()

```
TacticSources nvinfer1::IBuilderConfig::getTacticSources ( ) const  [inline], [noexcept]
```

Get tactic sources.

Get the tactic sources currently set in the engine build configuration.

See also

setTacticSources()

Returns

tactic sources

9.75.3.28 getTilingOptimizationLevel()

```
TilingOptimizationLevel nvinfer1::IBuilderConfig::getTilingOptimizationLevel ( ) const  [inline],
[noexcept]
```

Get the Tiling optimization level.

Returns

TilingOptimizationLevel The level of Tiling optimization.

See also

setTilingOptimizationLevel()

9.75.3.29 getTimingCache()

```
nvinfer1::ITimingCache const * nvinfer1::IBuilderConfig::getTimingCache ( ) const  [inline], [noexcept]
```

Get the pointer to the timing cache from current IBuilderConfig.

Returns

pointer to the timing cache used in current IBuilderConfig

9.75.3.30 isDeviceTypeSet()

```
bool nvinfer1::IBuilderConfig::isDeviceTypeSet (
            ILayer const * layer ) const  [inline], [noexcept]
```

whether the DeviceType has been explicitly set for this layer

Returns

true if device type is not default

See also

setDeviceType() getDeviceType() resetDeviceType()

9.75.3.31 reset()

```
void nvinfer1::IBuilderConfig::reset ( )  [inline], [noexcept]
```

Resets the builder configuration to defaults.

Useful for initializing a builder config object to its original state.

9.75.3.32 resetDeviceType()

```
void nvinfer1::IBuilderConfig::resetDeviceType (
            ILayer const * layer )  [inline], [noexcept]
```

reset the DeviceType for this layer

See also

setDeviceType() getDeviceType() isDeviceTypeSet()

9.75.3.33 setAvgTimingIterations()

```
virtual void nvinfer1::IBuilderConfig::setAvgTimingIterations (
            int32_t avgTiming )  [inline], [virtual], [noexcept]
```

Set the number of averaging iterations used when timing layers.

When timing layers, the builder minimizes over a set of average times for layer execution. This parameter controls the number of iterations used in averaging.

See also

getAvgTimingIterations()

9.75.3.34 setBuilderOptimizationLevel()

```
void nvinfer1::IBuilderConfig::setBuilderOptimizationLevel (
            int32_t level )  [inline], [noexcept]
```

Set builder optimization level.

Set the builder optimization level. Setting a higher optimization level allows the optimizer to spend more time searching for optimization opportunities. The resulting engine may have better performance compared to an engine built with a lower optimization level.

The default optimization level is 3. Valid values include integers from 0 to the maximum optimization level, which is currently 5. Setting it to greater than the maximum level results in behavior identical to the maximum level.

Below are the descriptions about each builder optimization level:

- Level 0: This enables the fastest compilation by disabling dynamic kernel generation and selecting the first tactic that succeeds in execution. This will also not respect a timing cache.
- Level 1: Available tactics are sorted by heuristics, but only the top are tested to select the best. If a dynamic kernel is generated its compile optimization is low.
- Level 2: Available tactics are sorted by heuristics, but only the fastest tactics are tested to select the best.
- Level 3: Apply heuristics to see if a static precompiled kernel is applicable or if a new one has to be compiled dynamically.
- Level 4: Always compiles a dynamic kernel.
- Level 5: Always compiles a dynamic kernel and compares it to static kernels.

Parameters

level	The optimization level to set to. Must be non-negative.

See also

getBuilderOptimizationLevel

9.75.3.35 setDefaultDeviceType()

```
void nvinfer1::IBuilderConfig::setDefaultDeviceType (
            DeviceType deviceType )  [inline], [noexcept]
```

Sets the default DeviceType to be used by the builder. It ensures that all the layers that can run on this device will run on it, unless setDeviceType is used to override the default DeviceType for a layer.

See also

getDefaultDeviceType()

9.75.3.36 setDeviceType()

```
void nvinfer1::IBuilderConfig::setDeviceType (
            ILayer const * layer,
            DeviceType deviceType )  [inline], [noexcept]
```

Set the device that this layer must execute on.

Parameters

layer	which layer to execute.
deviceType	that this layer must execute on. If DeviceType is not set or is reset, TensorRT will use the default DeviceType set in the builder.

Note

The device type for a layer must be compatible with the safety flow (if specified). For example a layer cannot be marked for DLA execution while the builder is configured for kSAFETY.

See also

getDeviceType()

9.75.3.37 setDLACore()

```
void nvinfer1::IBuilderConfig::setDLACore (
            int32_t dlaCore )  [inline], [noexcept]
```

Sets the DLA core used by the network. Defaults to -1.

Parameters

dlaCore	The DLA core to execute the engine on, in the range [0,getNbDlaCores()).

This function is used to specify which DLA core to use via indexing, if multiple DLA cores are available.

Warning

if getNbDLACores() returns 0, then this function does nothing.

See also

IRuntime::setDLACore() getDLACore()

9.75.3.38 setEngineCapability()

```
void nvinfer1::IBuilderConfig::setEngineCapability (
            EngineCapability capability )  [inline], [noexcept]
```

Configure the builder to target specified EngineCapability flow.

The flow means a sequence of API calls that allow an application to set up a runtime, engine, and execution context in order to run inference.

The supported flows are specified in the EngineCapability enum.

9.75.3.39 setFlag()

```
void nvinfer1::IBuilderConfig::setFlag (
            BuilderFlag builderFlag )  [inline], [noexcept]
```

Set a single build mode flag.

Add the input builder mode flag to the already enabled flags.

See also

setFlags()

9.75.3.40 setFlags()

```
void nvinfer1::IBuilderConfig::setFlags (
            BuilderFlags builderFlags )  [inline], [noexcept]
```

Set the build mode flags to turn on builder options for this network.

The flags are listed in the BuilderFlags enum. The flags set configuration options to build the network.

Parameters

builderFlags	The build option for an engine.

Note

This function will override the previous set flags, rather than bitwise ORing the new flag.

See also

getFlags()

9.75.3.41 setHardwareCompatibilityLevel()

```
void nvinfer1::IBuilderConfig::setHardwareCompatibilityLevel (
            HardwareCompatibilityLevel hardwareCompatibilityLevel )  [inline], [noexcept]
```

Set the hardware compatibility level.

Hardware compatibility allows an engine to run on GPU architectures other than that of the GPU where the engine was built.

The default hardware compatibility level is HardwareCompatibilityLevel::kNONE.

Parameters

hardwareCompatibilityLevel	The level of hardware compatibility.

9.75.3.42 setL2LimitForTiling()

```
bool nvinfer1::IBuilderConfig::setL2LimitForTiling (
            int64_t size )  [inline], [noexcept]
```

Set the L2 cache usage limit for Tiling optimization.

Parameter for tiling optimization. This API only takes effect when TilingOptimizationLevel is not kNONE.

Note

If setL2LimitForTiling() has not been called, TensorRT would choose a default value between 0 and L2 capacity size.

Parameters

size	The size of the L2 cache usage limit for Tiling optimization.

Returns

True if successful, false otherwise

9.75.3.43 setMaxAuxStreams()

```
bool nvinfer1::IBuilderConfig::setMaxAuxStreams (
            int32_t nbStreams ) [inline], [noexcept]
```

Set the maximum number of auxiliary streams that TRT is allowed to use.

If the network contains operators that can run in parallel, TRT can execute them using auxiliary streams in addition to the one provided to the IExecutionContext::enqueueV3() call.

The default maximum number of auxiliary streams is determined by the heuristics in TensorRT on whether enabling multi-stream would improve the performance. This behavior can be overridden by calling this API to set the maximum number of auxiliary streams explicitly. Set this to 0 to enforce single-stream inference.

The resulting engine may use fewer auxiliary streams than the maximum if the network does not contain enough parallelism or if TensorRT determines that using more auxiliary streams does not help improve the performance.

Note

Allowing more auxiliary streams does not always give better performance since there will be synchronizations overhead between streams. Using CUDA graphs at runtime can help reduce the overhead caused by cross-stream synchronizations.

Using more auxiliary leads to more memory usage at runtime since some activation memory blocks will not be able to be reused.

Parameters

nbStreams	The maximum number of auxiliary streams that TRT is allowed to use. Must be non-negative.

Returns

true if the value was set successfully, false if nbStreams is negative.

See also

getMaxAuxStreams(), ICudaEngine::getNbAuxStreams(), IExecutionContext::setAuxStreams()

9.75.3.44 setMaxNbTactics()

```
void nvinfer1::IBuilderConfig::setMaxNbTactics (
            int32_t maxNbTactics )  [inline], [noexcept]
```

Set the maximum number of tactics to time when there is a choice of tactics.

This function controls the number of tactics timed when there are multiple tactics to choose from.

See also

getMaxNbTactics()

9.75.3.45 setMemoryPoolLimit()

```
void nvinfer1::IBuilderConfig::setMemoryPoolLimit (
            MemoryPoolType pool,
            std::size_t poolSize )  [inline], [noexcept]
```

Set the memory size for the memory pool.

TensorRT layers access different memory pools depending on the operation. This function sets in the IBuilderConfig the size limit, specified by `poolSize`, for the corresponding memory pool, specified by `pool`. TensorRT will build a plan file that is constrained by these limits or report which constraint caused the failure.

If the size of the pool, specified by `poolSize`, fails to meet the size requirements for the pool, this function does nothing and emits the recoverable error, ErrorCode::kINVALID_ARGUMENT, to the registered IErrorRecorder.

If the size of the pool is larger than the maximum possible value for the configuration, this function does nothing and emits ErrorCode::kUNSUPPORTED_STATE.

If the pool does not exist on the requested device type when building the network, a warning is emitted to the logger, and the memory pool value is ignored.

Refer to MemoryPoolType to see the size requirements for each pool.

Parameters

pool	The memory pool to limit the available memory for.
poolSize	The size of the pool in bytes.

See also

getMemoryPoolLimit, MemoryPoolType

9.75.3.46 setPluginsToSerialize()

```
void nvinfer1::IBuilderConfig::setPluginsToSerialize (
            char const *const * paths,
            int32_t nbPaths )  [inline], [noexcept]
```

Set the plugin libraries to be serialized with version-compatible engines.

Each entry in the list of libraries must be unique.

Parameters

paths	The paths of plugin libraries.
nbPaths	The number of paths.

9.75.3.47 setPreviewFeature()

```
void nvinfer1::IBuilderConfig::setPreviewFeature (
            PreviewFeature feature,
            bool enable )  [inline], [noexcept]
```

Enable or disable a specific preview feature.

Allows enabling or disabling experimental features, which are not enabled by default in the current release.

Refer to PreviewFeature for additional information, and a list of the available features.

Parameters

feature	the feature to enable / disable
enable	true for enable, false for disable

See also

PreviewFeature, getPreviewFeature

9.75.3.48 setProfileStream()

```
void nvinfer1::IBuilderConfig::setProfileStream (
            const cudaStream_t stream )  [inline], [noexcept]
```

Set the CUDA stream that is used to profile this network.

Parameters

stream	The CUDA stream used for profiling by the builder.

See also

getProfileStream()

9.75.3.49 setProfilingVerbosity()

```
void nvinfer1::IBuilderConfig::setProfilingVerbosity (
            ProfilingVerbosity verbosity )  [inline], [noexcept]
```

Set verbosity level of layer information exposed in NVTX annotations and IEngineInspector.

Control how much layer information will be exposed in NVTX annotations and IEngineInspector.

See also

ProfilingVerbosity, getProfilingVerbosity(), IEngineInspector

9.75.3.50 setProgressMonitor()

```
void nvinfer1::IBuilderConfig::setProgressMonitor (
            IProgressMonitor * monitor )  [inline], [noexcept]
```

Sets the progress monitor for building a network.

Parameters

monitor	The progress monitor to assign to the IBuilderConfig.

The progress monitor signals to the application when different phases of the compiler are being executed. Setting to nullptr unsets the monitor so that the application is not signaled.

See also

IBuilderConfig::getProgressMonitor

9.75.3.51 setRemoteAutoTuningConfig()

```
bool nvinfer1::IBuilderConfig::setRemoteAutoTuningConfig (
            char const * config )  [inline], [noexcept]
```

Set a config string for remote auto tuning.

Remote auto-tuning is supported only for engines built with EngineCapability::kSAFETY.

Parameters

config	The config string to be used during remote auto tuning.

Returns

True if successful, false otherwise

9.75.3.52 setRuntimePlatform()

```
void nvinfer1::IBuilderConfig::setRuntimePlatform (
            RuntimePlatform runtimePlatform )  [inline], [noexcept]
```

Set the target platform for runtime execution.

Cross-platform compatibility allows an engine to be built and executed on different platforms.

The default cross-platform target is RuntimePlatform::kSAME_AS_BUILD.

Parameters

runtimePlatform	The target platform for runtime execution.

See also

IBuilderConfig::getRuntimePlatform()

9.75.3.53 setTacticSources()

```
bool nvinfer1::IBuilderConfig::setTacticSources (
            TacticSources tacticSources )  [inline], [noexcept]
```

Set tactic sources.

This bitset controls which tactic sources TensorRT is allowed to use for tactic selection.

Multiple tactic sources may be combined with a bitwise OR operation. For example, to enable edge mask convolutions and JIT convolutions as tactic sources, use a value of:

1U << static_cast<uint32_t>(TacticSource::kEDGE_MASK_CONVOLUTIONS) | 1U << static_cast<uint32_t>(↩ TacticSource::kJIT_CONVOLUTIONS)

See also

getTacticSources

Returns

true if the tactic sources in the build configuration were updated. The tactic sources in the build configuration will not be updated if the provided value is invalid.

9.75.3.54 setTilingOptimizationLevel()

```
bool nvinfer1::IBuilderConfig::setTilingOptimizationLevel (
            TilingOptimizationLevel level )  [inline], [noexcept]
```

Set the Tiling optimization level.

Tiling allows TensorRT to try an on-chip caching strategy.

The default getTilingOptimizationLevel is TilingOptimizationLevel::kNONE.

Parameters

level	The level of Tiling optimization.

Returns

True if successful, false otherwise

9.75.3.55 setTimingCache()

```
bool nvinfer1::IBuilderConfig::setTimingCache (
            ITimingCache const & cache,
            bool ignoreMismatch )  [inline], [noexcept]
```

Attach a timing cache to IBuilderConfig.

The timing cache has verification header to make sure the provided cache can be used in current environment. A failure will be reported if the CUDA device property in the provided cache is different from current environment. ignoreMismatch = true skips strict verification and allows loading cache created from a different device.

The cache must not be destroyed until after the engine is built.

Parameters

cache	the timing cache to be used
ignoreMismatch	whether or not allow using a cache that contains different CUDA device property

Returns

true if set successfully, false otherwise

Warning

Using cache generated from devices with different CUDA device properties may lead to functional/performance bugs.

9.75.4 Member Data Documentation

9.75.4.1 mImpl

```
apiv::VBuilderConfig* nvinfer1::IBuilderConfig::mImpl  [protected]
```

The documentation for this class was generated from the following file:

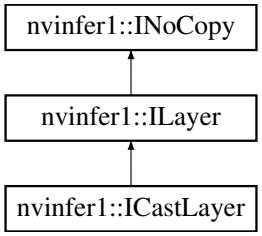
- NvInfer.h

9.76 nvinfer1::ICastLayer Class Reference

A cast layer in a network.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::ICastLayer:



Public Member Functions

- void setToType (DataType toType) noexcept
  Set cast layer output type.
- DataType getToType () const noexcept
  Return cast layer output type.

Protected Member Functions

- virtual ~ICastLayer () noexcept=0

Protected Attributes

- apiv::VCastLayer * mImpl

9.76.1 Detailed Description

A cast layer in a network.

This layer casts a given tensor to the datatype specified by `toType`.

9.76.2 Constructor & Destructor Documentation

9.76.2.1 ~ICastLayer()

```
nvinfer1::ICastLayer::~ICastLayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.76.3 Member Function Documentation

9.76.3.1 getToType()

```
DataType nvinfer1::ICastLayer::getToType ( ) const  [inline], [noexcept]
```

Return cast layer output type.

Returns

toType parameter set during layer creation or by setToType(). The return value is the output type of the cast layer.

9.76.3.2 setToType()

```
void nvinfer1::ICastLayer::setToType (
            DataType toType )  [inline], [noexcept]
```

Set cast layer output type.

Parameters

toType	The DataType of the output tensor.

Set the output type of the cast layer.

9.76.4 Member Data Documentation

9.76.4.1 mImpl

```
apiv::VCastLayer* nvinfer1::ICastLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

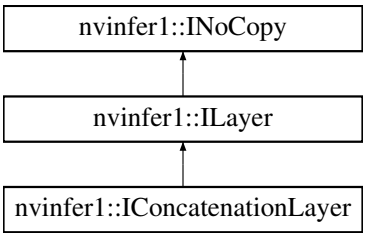
- NvInfer.h

9.77 nvinfer1::IConcatenationLayer Class Reference

A concatenation layer in a network definition.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IConcatenationLayer:



Public Member Functions

- void setAxis (int32_t axis) noexcept

  Set the axis along which concatenation occurs.
- int32_t getAxis () const noexcept

  Get the axis along which concatenation occurs.

Protected Member Functions

- virtual ~IConcatenationLayer () noexcept=0

Protected Attributes

- apiv::VConcatenationLayer * mImpl

9.77.1 Detailed Description

A concatenation layer in a network definition.

The output dimension along the concatenation axis is the sum of the corresponding input dimensions. Every other output dimension is the same as the corresponding dimension of the inputs.

Warning

All tensors must have the same dimensions except along the concatenation axis.

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.77.2 Constructor & Destructor Documentation

9.77.2.1 ~IConcatenationLayer()

```
nvinfer1::IConcatenationLayer::~IConcatenationLayer ( )  [inline], [protected], [pure virtual],
[default], [noexcept]
```

9.77.3 Member Function Documentation

9.77.3.1 getAxis()

```
int32_t nvinfer1::IConcatenationLayer::getAxis ( ) const  [inline], [noexcept]
```

Get the axis along which concatenation occurs.

See also

setAxis()

9.77.3.2 setAxis()

```
void nvinfer1::IConcatenationLayer::setAxis (
            int32_t axis )  [inline], [noexcept]
```

Set the axis along which concatenation occurs.

The default axis is the number of tensor dimensions minus three, or zero if the tensor has fewer than three dimensions. For example, for a tensor with dimensions NCHW, it is C.

When running this layer on the DLA, the concatenation axis must be the third to last axis, e.g. C if tensor dimensions are NCHW.

Parameters

axis	The axis along which concatenation occurs.

9.77.4 Member Data Documentation

9.77.4.1 mImpl

```
apiv::VConcatenationLayer* nvinfer1::IConcatenationLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

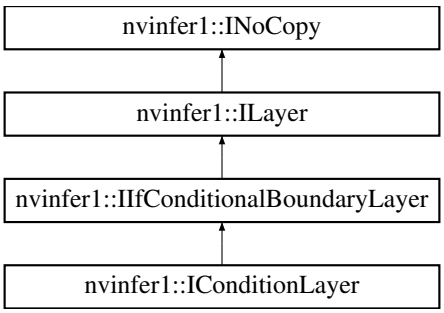
- NvInfer.h

9.78 nvinfer1::IConditionLayer Class Reference

This layer represents a condition input to an IIfConditional.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IConditionLayer:



Protected Member Functions

- virtual ~IConditionLayer () noexcept=0

Protected Attributes

- apiv::VConditionLayer * mImpl

Additional Inherited Members

9.78.1 Detailed Description

This layer represents a condition input to an IIfConditional.

9.78.2 Constructor & Destructor Documentation

9.78.2.1 ~IConditionLayer()

```
nvinfer1::IConditionLayer::~IConditionLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.78.3 Member Data Documentation

9.78.3.1 mImpl

```
apiv::VConditionLayer* nvinfer1::IConditionLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.79 nvinfer2::safe::consistency::IConsistencyChecker Class Reference

Validates a TRT engine (Serialized binary blob)

```
#include <NvInferConsistency.h>
```

Public Member Functions

- virtual bool validate () noexcept=0

  Check that a blob that was input to createConsistencyChecker method represents a valid TRT engine.
- virtual ~IConsistencyChecker ()=default

  De-allocates any internally allocated memory.

Protected Member Functions

- IConsistencyChecker ()=default

  Default constructor for IConsistencyChecker.
- IConsistencyChecker (IConsistencyChecker const &other)=delete

  Deleted copy constructor to maintain non-copyability.
- IConsistencyChecker & operator= (IConsistencyChecker const &other) &=delete

  Deleted copy assignment operator (lvalue) to maintain non-copyability.
- IConsistencyChecker (IConsistencyChecker &&other)=delete

  Deleted move constructor to maintain non-movability.
- IConsistencyChecker & operator= (IConsistencyChecker &&other) &=delete

  Deleted move assignment operator (lvalue) to maintain non-movability.

9.79.1 Detailed Description

Validates a TRT engine (Serialized binary blob)

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.79.2 Constructor & Destructor Documentation

9.79.2.1 ~IConsistencyChecker()

```
virtual nvinfer2::safe::consistency::IConsistencyChecker::~IConsistencyChecker ( )  [virtual],
[default]
```

De-allocates any internally allocated memory.

9.79.2.2 IConsistencyChecker() [1/3]

```
nvinfer2::safe::consistency::IConsistencyChecker::IConsistencyChecker ( )  [protected], [default]
```

Default constructor for IConsistencyChecker.

Constructs an object in a default state. Access is protected; only derived types or the factory createConsistencyChecker() may create instances. The object does not yet hold or validate any blob until initialized by the factory.

Precondition

none

Postcondition

Object is in default-constructed state and is not yet bound to any engine blob.

9.79.2.3 IConsistencyChecker() [2/3]

```
nvinfer2::safe::consistency::IConsistencyChecker::IConsistencyChecker (
            IConsistencyChecker const & other )  [protected], [delete]
```

Deleted copy constructor to maintain non-copyability.

Precondition

none

Postcondition

none

9.79.2.4 IConsistencyChecker() [3/3]

```
nvinfer2::safe::consistency::IConsistencyChecker::IConsistencyChecker (
            IConsistencyChecker && other )  [protected], [delete]
```

Deleted move constructor to maintain non-movability.

Precondition

none

Postcondition

none

9.79.3 Member Function Documentation

9.79.3.1 operator=() [1/2]

```
IConsistencyChecker & nvinfer2::safe::consistency::IConsistencyChecker::operator= (
            IConsistencyChecker && other ) &  [protected], [delete]
```

Deleted move assignment operator (lvalue) to maintain non-movability.

Precondition

none

Postcondition

none

9.79.3.2 operator=() [2/2]

```
IConsistencyChecker & nvinfer2::safe::consistency::IConsistencyChecker::operator= (
            IConsistencyChecker const & other ) &  [protected], [delete]
```

Deleted copy assignment operator (lvalue) to maintain non-copyability.

Precondition

none

Postcondition

none

9.79.3.3 validate()

```
virtual bool nvinfer2::safe::consistency::IConsistencyChecker::validate ( )  [pure virtual], [noexcept]
```

Check that a blob that was input to createConsistencyChecker method represents a valid TRT engine.

Returns

true if the original blob encoded an engine that belongs to valid TRT engine, false otherwise.

The documentation for this class was generated from the following file:

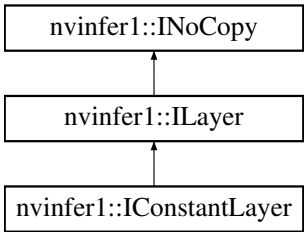
- NvInferConsistency.h

9.80 nvinfer1::IConstantLayer Class Reference

Layer that represents a constant value.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IConstantLayer:



Public Member Functions

- void setWeights (Weights weights) noexcept

  Set the weights for the layer.
- Weights getWeights () const noexcept

  Get the weights for the layer.
- void setDimensions (Dims const &dimensions) noexcept

  Set the dimensions for the layer.
- Dims getDimensions () const noexcept

  Get the dimensions for the layer.

Protected Member Functions

- virtual ~IConstantLayer () noexcept=0

Protected Attributes

- apiv::VConstantLayer * mImpl

9.80.1 Detailed Description

Layer that represents a constant value.

Note

This layer does not support boolean types.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.80.2 Constructor & Destructor Documentation

9.80.2.1 ~IConstantLayer()

```
nvinfer1::IConstantLayer::~IConstantLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.80.3 Member Function Documentation

9.80.3.1 getDimensions()

```
Dims nvinfer1::IConstantLayer::getDimensions ( ) const  [inline], [noexcept]
```

Get the dimensions for the layer.

Returns

the dimensions for the layer

See also

getDimensions

9.80.3.2 getWeights()

```
Weights nvinfer1::IConstantLayer::getWeights ( ) const  [inline], [noexcept]
```

Get the weights for the layer.

See also

setWeights

9.80.3.3 setDimensions()

```
void nvinfer1::IConstantLayer::setDimensions (
            Dims const & dimensions )  [inline], [noexcept]
```

Set the dimensions for the layer.

Parameters

dimensions	The dimensions of the layer

See also

setDimensions

9.80.3.4 setWeights()

```
void nvinfer1::IConstantLayer::setWeights (
            Weights weights )  [inline], [noexcept]
```

Set the weights for the layer.

The output type is weights.type. If the network is weakly typed and the weights have a real type, the output type might be different per TensorRT's type conversion rules.

See also

getWeights()

9.80.4 Member Data Documentation

9.80.4.1 mImpl

```
apiv::VConstantLayer* nvinfer1::IConstantLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

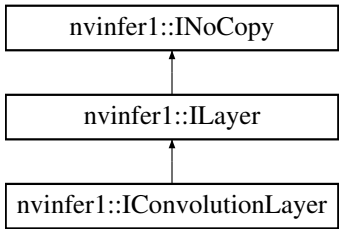
- NvInfer.h

9.81 nvinfer1::IConvolutionLayer Class Reference

A convolution layer in a network definition.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IConvolutionLayer:



Public Member Functions

- void setNbOutputMaps (int64_t nbOutputMaps) noexcept

  Set the number of output maps for the convolution.
- int64_t getNbOutputMaps () const noexcept

  Get the number of output maps for the convolution.
- void setNbGroups (int64_t nbGroups) noexcept

  Set the number of groups for a convolution.
- int64_t getNbGroups () const noexcept

  Get the number of groups of the convolution.
- void setKernelWeights (Weights weights) noexcept

  Set the kernel weights for the convolution.
- Weights getKernelWeights () const noexcept

  Get the kernel weights of the convolution.
- void setBiasWeights (Weights weights) noexcept

  Set the bias weights for the convolution.
- Weights getBiasWeights () const noexcept

  Get the bias weights for the convolution.
- void setPrePadding (Dims const &padding) noexcept

  Set the multi-dimension pre-padding of the convolution.
- Dims getPrePadding () const noexcept

  Get the pre-padding.
- void setPostPadding (Dims const &padding) noexcept

  Set the multi-dimension post-padding of the convolution.
- Dims getPostPadding () const noexcept

  Get the post-padding.
- void setPaddingMode (PaddingMode paddingMode) noexcept

  Set the padding mode.
- PaddingMode getPaddingMode () const noexcept

  Get the padding mode.
- void setKernelSizeNd (Dims const &kernelSize) noexcept

  Set the multi-dimension kernel size of the convolution.
- Dims getKernelSizeNd () const noexcept

  Get the multi-dimension kernel size of the convolution.
- void setStrideNd (Dims const &stride) noexcept

  Set the multi-dimension stride of the convolution.
- Dims getStrideNd () const noexcept

  Get the multi-dimension stride of the convolution.
- void setPaddingNd (Dims const &padding) noexcept

  Set the multi-dimension padding of the convolution.
- Dims getPaddingNd () const noexcept

  Get the multi-dimension padding of the convolution.
- void setDilationNd (Dims const &dilation) noexcept

  Set the multi-dimension dilation of the convolution.
- Dims getDilationNd () const noexcept

  Get the multi-dimension dilation of the convolution.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~IConvolutionLayer () noexcept=0

Protected Attributes

- apiv::VConvolutionLayer * mImpl

9.81.1 Detailed Description

A convolution layer in a network definition.

This layer performs a correlation operation between 3 or 4 dimensional filter with a 4 or 5 dimensional tensor to produce another 4 or 5 dimensional tensor.

An optional bias argument is supported, which adds a per-channel constant to each value in the output.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.81.2 Constructor & Destructor Documentation

9.81.2.1 ~IConvolutionLayer()

```
nvinfer1::IConvolutionLayer::~IConvolutionLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.81.3 Member Function Documentation

9.81.3.1 getBiasWeights()

```
Weights nvinfer1::IConvolutionLayer::getBiasWeights ( ) const  [inline], [noexcept]
```

Get the bias weights for the convolution.

See also

setBiasWeights()

9.81.3.2 getDilationNd()

```
Dims nvinfer1::IConvolutionLayer::getDilationNd ( ) const  [inline], [noexcept]
```

Get the multi-dimension dilation of the convolution.

See also

setDilationNd()

9.81.3.3 getKernelSizeNd()

```
Dims nvinfer1::IConvolutionLayer::getKernelSizeNd ( ) const  [inline], [noexcept]
```

Get the multi-dimension kernel size of the convolution.

See also

setKernelSizeNd()

9.81.3.4 getKernelWeights()

```
Weights nvinfer1::IConvolutionLayer::getKernelWeights ( ) const  [inline], [noexcept]
```

Get the kernel weights of the convolution.

See also

setKernelWeights()

9.81.3.5 getNbGroups()

```
int64_t nvinfer1::IConvolutionLayer::getNbGroups ( ) const  [inline], [noexcept]
```

Get the number of groups of the convolution.

See also

setNbGroups()

9.81.3.6 getNbOutputMaps()

```
int64_t nvinfer1::IConvolutionLayer::getNbOutputMaps ( ) const  [inline], [noexcept]
```

Get the number of output maps for the convolution.

See also

setNbOutputMaps()

9.81.3.7 getPaddingMode()

```
PaddingMode nvinfer1::IConvolutionLayer::getPaddingMode ( ) const  [inline], [noexcept]
```

Get the padding mode.

Default: kEXPLICIT_ROUND_DOWN

See also

setPaddingMode()

9.81.3.8 getPaddingNd()

```
Dims nvinfer1::IConvolutionLayer::getPaddingNd ( ) const  [inline], [noexcept]
```

Get the multi-dimension padding of the convolution.

If the padding is asymmetric, the pre-padding is returned.

See also

setPaddingNd()

9.81.3.9 getPostPadding()

```
Dims nvinfer1::IConvolutionLayer::getPostPadding ( ) const  [inline], [noexcept]
```

Get the post-padding.

See also

setPostPadding()

9.81.3.10 getPrePadding()

```
Dims nvinfer1::IConvolutionLayer::getPrePadding ( ) const  [inline], [noexcept]
```

Get the pre-padding.

See also

setPrePadding()

9.81.3.11 getStrideNd()

```
Dims nvinfer1::IConvolutionLayer::getStrideNd ( ) const  [inline], [noexcept]
```

Get the multi-dimension stride of the convolution.

See also

setStrideNd()

9.81.3.12 setBiasWeights()

```
void nvinfer1::IConvolutionLayer::setBiasWeights (
            Weights weights )  [inline], [noexcept]
```

Set the bias weights for the convolution.

Bias is optional. To omit bias, set the count value of the weights structure to zero.

The bias is applied per-channel, so the number of weights (if non-zero) must be equal to the number of output feature maps.

See also

getBiasWeights()

9.81.3.13 setDilationNd()

```
void nvinfer1::IConvolutionLayer::setDilationNd (
            Dims const & dilation )  [inline], [noexcept]
```

Set the multi-dimension dilation of the convolution.

Default: (1, 1, ..., 1)

If executing this layer on DLA, only support 2D padding, both height and width must be in the range [1,32].

See also

getDilationNd()

9.81.3.14 setInput()

```
void nvinfer1::ILayer::setInput (
            int32_t index,
            ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor

The indices are as follows:

Input 0 is the input activation tensor. Input 1 is the kernel tensor. If used, the kernel weights parameter must be set to empty weights. Input 2 is the bias tensor. If used, the bias parameter must be set to empty weights.

See also

getKernelWeights(), setKernelWeights(), getBiasWeights(), setBiasWeights()

9.81.3.15 setKernelSizeNd()

```
void nvinfer1::IConvolutionLayer::setKernelSizeNd (
            Dims const & kernelSize )  [inline], [noexcept]
```

Set the multi-dimension kernel size of the convolution.

If executing this layer on DLA, only support 2D kernel size, both height and width of kernel size must be in the range [1,32].

See also

getKernelSizeNd()

9.81.3.16 setKernelWeights()

```
void nvinfer1::IConvolutionLayer::setKernelWeights (
            Weights weights )  [inline], [noexcept]
```

Set the kernel weights for the convolution.

The weights are specified as a contiguous array in `GKCRS` order, where `G` is the number of groups, `K` the number of output feature maps, `C` the number of input channels, and `R` and `S` are the height and width of the filter.

See also

getKernelWeights()

9.81.3.17 setNbGroups()

```
void nvinfer1::IConvolutionLayer::setNbGroups (
            int64_t nbGroups )  [inline], [noexcept]
```

Set the number of groups for a convolution.

The input tensor channels are divided into `nbGroups` groups, and a convolution is executed for each group, using a filter per group. The results of the group convolutions are concatenated to form the output.

Note

When using groups in int8 mode, the size of the groups (i.e. the channel count divided by the group count) must be a multiple of 4 for both input and output.

Default: 1

If executing this layer on DLA, the max number of groups is 8192.

See also

getNbGroups()

9.81.3.18 setNbOutputMaps()

```
void nvinfer1::IConvolutionLayer::setNbOutputMaps (
            int64_t nbOutputMaps )  [inline], [noexcept]
```

Set the number of output maps for the convolution.

If executing this layer on DLA, the number of output maps must be in the range [1,8192].

See also

getNbOutputMaps()

9.81.3.19 setPaddingMode()

```
void nvinfer1::IConvolutionLayer::setPaddingMode (
            PaddingMode paddingMode )  [inline], [noexcept]
```

Set the padding mode.

Padding mode takes precedence if both setPaddingMode and setPre/PostPadding are used.

Default: kEXPLICIT_ROUND_DOWN

See also

getPaddingMode()

9.81.3.20 setPaddingNd()

```
void nvinfer1::IConvolutionLayer::setPaddingNd (
            Dims const & padding )  [inline], [noexcept]
```

Set the multi-dimension padding of the convolution.

The input will be zero-padded by this number of elements in each dimension. Padding is symmetric.

Default: (0, 0, ..., 0)

If executing this layer on DLA, only support 2D padding, both height and width of padding must be in the range [0,31], and the padding must be less than the kernel size.

See also

getPaddingNd() setPadding() getPadding()

9.81.3.21 setPostPadding()

```
void nvinfer1::IConvolutionLayer::setPostPadding (
            Dims const & padding )  [inline], [noexcept]
```

Set the multi-dimension post-padding of the convolution.

The end of the input will be zero-padded by this number of elements in each dimension.

Default: (0, 0, ..., 0)

If executing this layer on DLA, only support 2D padding, both height and width of padding must be in the range [0,31], and the padding must be less than the kernel size.

See also

getPostPadding()

9.81.3.22 setPrePadding()

```
void nvinfer1::IConvolutionLayer::setPrePadding (
            Dims const & padding )  [inline], [noexcept]
```

Set the multi-dimension pre-padding of the convolution.

The start of the input will be zero-padded by this number of elements in each dimension.

Default: (0, 0, ..., 0)

If executing this layer on DLA, only support 2D padding, both height and width of padding must be in the range [0,31], and the padding must be less than the kernel size.

See also

getPrePadding()

9.81.3.23 setStrideNd()

```
void nvinfer1::IConvolutionLayer::setStrideNd (
            Dims const & stride )  [inline], [noexcept]
```

Set the multi-dimension stride of the convolution.

Default: (1, 1, ..., 1)

If executing this layer on DLA, only support 2D stride, both height and width of stride must be in the range [1,8].

See also

getStrideNd()

9.81.4 Member Data Documentation

9.81.4.1 mImpl

```
apiv::VConvolutionLayer* nvinfer1::IConvolutionLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

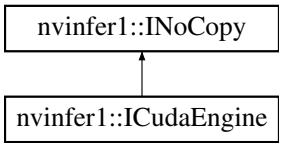
- NvInfer.h

9.82 nvinfer1::ICudaEngine Class Reference

An engine for executing inference on a built network, with functionally unsafe features.

```
#include <NvInferRuntime.h>
```

Inheritance diagram for nvinfer1::ICudaEngine:



Public Member Functions

- virtual ~ICudaEngine () noexcept=0
- Dims getTensorShape (char const *tensorName) const noexcept

  Get shape of an input or output tensor.
- DataType getTensorDataType (char const *tensorName) const noexcept

  Determine the required data type for a buffer from its tensor name.
- int32_t getNbLayers () const noexcept

  Get the number of layers in the network.
- IHostMemory * serialize () const noexcept

  Serialize the network to a stream.
- IExecutionContext * createExecutionContext (ExecutionContextAllocationStrategy strategy=ExecutionContextAllocationStrategy noexcept

  Create an execution context and specify the strategy for allocating internal activation memory.
- TensorLocation getTensorLocation (char const *tensorName) const noexcept

  Get whether an input or output tensor must be on GPU or CPU.
- bool isShapeInferenceIO (char const *tensorName) const noexcept

  True if tensor is required as input for shape calculations or is output from shape calculations.
- TensorIOMode getTensorIOMode (char const *tensorName) const noexcept

Determine whether a tensor is an input or output tensor.

- TRT_NODISCARD char const * getAliasedInputTensor (char const *tensorName) const noexcept

  Get the input tensor name that an output tensor should alias with.
- IExecutionContext * createExecutionContext (IRuntimeConfig *runtimeConfig) noexcept

  Create an execution context with TensorRT JIT runtime config.
- IRuntimeConfig * createRuntimeConfig () noexcept

  Create a runtime config for TensorRT JIT. The caller is responsible for ownership of the returned IRuntimeConfig object.
- int64_t getDeviceMemorySizeV2 () const noexcept

  Return the maximum device memory required by the context over all profiles.
- int64_t getDeviceMemorySizeForProfileV2 (int32_t profileIndex) const noexcept

  Return the maximum device memory required by the context for a profile.
- bool isRefittable () const noexcept

  Return true if an engine can be refit.
- int32_t getTensorBytesPerComponent (char const *tensorName) const noexcept

  Return the number of bytes per component of an element, or -1 if the tensor is not vectorized or provided name does not map to an input or output tensor.
- int32_t getTensorBytesPerComponent (char const *tensorName, int32_t profileIndex) const noexcept

  Return the number of bytes per component of an element given of given profile, or -1 if the tensor is not vectorized or provided name does not map to an input or output tensor.
- int32_t getTensorComponentsPerElement (char const *tensorName) const noexcept

  Return the number of components included in one element, or -1 if tensor is not vectorized or if the provided name does not map to an input or output tensor.
- int32_t getTensorComponentsPerElement (char const *tensorName, int32_t profileIndex) const noexcept

  Return the number of components included in one element of given profile, or -1 if tensor is not vectorized or the provided name does not map to an input or output tensor.
- TensorFormat getTensorFormat (char const *tensorName) const noexcept

  Return the tensor format, or TensorFormat::kLINEAR if the provided name does not map to an input or output tensor.
- TensorFormat getTensorFormat (char const *tensorName, int32_t profileIndex) const noexcept

  Return the tensor format of given profile, or TensorFormat::kLINEAR if the provided name does not map to an input or output tensor.
- char const * getTensorFormatDesc (char const *tensorName) const noexcept

  Return the human readable description of the tensor format, or empty string if the provided name does not map to an input or output tensor.
- char const * getTensorFormatDesc (char const *tensorName, int32_t profileIndex) const noexcept

  Return the human readable description of the tensor format of given profile, or empty string if the provided name does not map to an input or output tensor.
- int32_t getTensorVectorizedDim (char const *tensorName) const noexcept

  Return the dimension index that the buffer is vectorized, or -1 if the provided name does not map to an input or output tensor.
- int32_t getTensorVectorizedDim (char const *tensorName, int32_t profileIndex) const noexcept

  Return the dimension index that the buffer is vectorized of given profile, or -1 if the provided name does not map to an input or output tensor.
- char const * getName () const noexcept

  Returns the name of the network associated with the engine.
- int32_t getNbOptimizationProfiles () const noexcept

  Get the number of optimization profiles defined for this engine.
- Dims getProfileShape (char const *tensorName, int32_t profileIndex, OptProfileSelector select) const noexcept

  Get the minimum / optimum / maximum dimensions for an input tensor given its name under an optimization profile.
- EngineCapability getEngineCapability () const noexcept

Determine what execution capability this engine has.

- void setErrorRecorder (IErrorRecorder *recorder) noexcept

  Set the ErrorRecorder for this interface.

- IErrorRecorder * getErrorRecorder () const noexcept

  Get the ErrorRecorder assigned to this interface.

- TacticSources getTacticSources () const noexcept

  return the tactic sources required by this engine.

- ProfilingVerbosity getProfilingVerbosity () const noexcept

  Return the ProfilingVerbosity the builder config was set to when the engine was built.

- IEngineInspector * createEngineInspector () const noexcept

  Create a new engine inspector which prints the layer information in an engine or an execution context.

- int32_t getNbIOTensors () const noexcept

  Return number of IO tensors.

- char const * getIOTensorName (int32_t index) const noexcept

  Return name of an IO tensor.

- HardwareCompatibilityLevel getHardwareCompatibilityLevel () const noexcept

  Return the hardware compatibility level of this engine.

- int32_t getNbAuxStreams () const noexcept

  Return the number of auxiliary streams used by this engine.

- ISerializationConfig * createSerializationConfig () noexcept

  Create a serialization configuration object.

- IHostMemory * serializeWithConfig (ISerializationConfig &config) const noexcept

  Serialize the network to a stream with the provided SerializationConfig.

- int64_t getStreamableWeightsSize () const noexcept

  Get the total size in bytes of all streamable weights.

- bool setWeightStreamingBudgetV2 (int64_t gpuMemoryBudget) noexcept

  Limit the maximum amount of GPU memory usable for network weights in bytes.

- int64_t getWeightStreamingBudgetV2 () const noexcept

  Returns the current weight streaming device memory budget in bytes.

- int64_t getWeightStreamingAutomaticBudget () const noexcept

  TensorRT automatically determines a device memory budget for the model to run. The budget is close to the current free memory size, leaving some space for other memory needs in the user's application. If the budget exceeds the size obtained from getStreamableWeightsSize(), it is capped to that size, effectively disabling weight streaming. Since TensorRT lacks information about the user's allocations, the remaining memory size might be larger than required, leading to wasted memory, or smaller than required, causing an out-of-memory error. For optimal memory allocation, it is recommended to manually calculate and set the budget.

- int64_t getWeightStreamingScratchMemorySize () const noexcept

  Returns the size of the scratch memory required by the current weight streaming budget.

- bool isDebugTensor (char const *name) const noexcept

  Check if a tensor is marked as a debug tensor.

- int64_t const * getProfileTensorValuesV2 (char const *tensorName, int32_t profileIndex, OptProfileSelector select) const noexcept

  Get the minimum / optimum / maximum values (not dimensions) for an input tensor given its name under an optimization profile. These correspond to the values set using IOptimizationProfile::setShapeValuesV2 when the engine was built.

- int64_t getEngineStat (EngineStat stat) const noexcept

  Get engine statistics according to the given enum value.

Protected Attributes

- apiv::VCudaEngine * mImpl

Additional Inherited Members

9.82.1 Detailed Description

An engine for executing inference on a built network, with functionally unsafe features.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.82.2 Constructor & Destructor Documentation

9.82.2.1 ~ICudaEngine()

```
nvinfer1::ICudaEngine::~ICudaEngine ( )  [inline], [pure virtual], [default], [noexcept]
```

9.82.3 Member Function Documentation

9.82.3.1 createEngineInspector()

```
IEngineInspector * nvinfer1::ICudaEngine::createEngineInspector ( ) const  [inline], [noexcept]
```

Create a new engine inspector which prints the layer information in an engine or an execution context.

See also

IEngineInspector.

9.82.3.2 createExecutionContext() [1/2]

```
IExecutionContext * nvinfer1::ICudaEngine::createExecutionContext (
            ExecutionContextAllocationStrategy strategy = ExecutionContextAllocationStrategy::kSTATIC
)  [inline], [noexcept]
```

Create an execution context and specify the strategy for allocating internal activation memory.

The default value for the allocation strategy is ExecutionContextAllocationStrategy::kSTATIC, which means the context will pre-allocate a block of device memory that is sufficient for all profiles. The newly created execution context will be assigned optimization profile 0. If an error recorder has been set for the engine, it will also be passed to the execution context.

See also

IExecutionContext

IExecutionContext::setOptimizationProfileAsync()

ExecutionContextAllocationStrategy

9.82.3.3 createExecutionContext() [2/2]

```
IExecutionContext * nvinfer1::ICudaEngine::createExecutionContext (
            IRuntimeConfig * runtimeConfig )  [inline], [noexcept]
```

Create an execution context with TensorRT JIT runtime config.

Parameters

runtimeConfig	The runtime config for TensorRT JIT.

See also

IRuntimeConfig

9.82.3.4 createRuntimeConfig()

```
IRuntimeConfig * nvinfer1::ICudaEngine::createRuntimeConfig ( )  [inline], [noexcept]
```

Create a runtime config for TensorRT JIT. The caller is responsible for ownership of the returned IRuntimeConfig object.

Returns

A IRuntimeConfig object.

See also

IRuntimeConfig

9.82.3.5 createSerializationConfig()

```
ISerializationConfig * nvinfer1::ICudaEngine::createSerializationConfig ( )  [inline], [noexcept]
```

Create a serialization configuration object.

See also

ISerializationConfig

9.82.3.6 getAliasedInputTensor()

```
TRT_NODISCARD char const * nvinfer1::ICudaEngine::getAliasedInputTensor (
            char const * tensorName ) const  [inline], [noexcept]
```

Get the input tensor name that an output tensor should alias with.

Some operations (e.g., KVCacheUpdate) require that certain output tensors share memory with input tensors. This method returns the name of the input tensor that a given output tensor should alias with.

Parameters

tensorName	The name of an output tensor.

Returns

The name of the input tensor to alias with, or nullptr if tensorName is not an output tensor or the output does not alias with any input.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

9.82.3.7 getDeviceMemorySizeForProfileV2()

```
int64_t nvinfer1::ICudaEngine::getDeviceMemorySizeForProfileV2 (
            int32_t profileIndex ) const  [inline], [noexcept]
```

Return the maximum device memory required by the context for a profile.

This API is stateful, so its call returns different values based on the following calls:

- setWeightStreamingBudgetV2()

See also

IExecutionContext::setDeviceMemoryV2()
setWeightStreamingBudgetV2()

9.82.3.8 getDeviceMemorySizeV2()

```
int64_t nvinfer1::ICudaEngine::getDeviceMemorySizeV2 ( ) const  [inline], [noexcept]
```

Return the maximum device memory required by the context over all profiles.

This API is stateful, so its call returns different values based on the following calls:

- setWeightStreamingBudgetV2()

See also

IExecutionContext::setDeviceMemoryV2()
setWeightStreamingBudgetV2()

9.82.3.9 getEngineCapability()

```
EngineCapability nvinfer1::ICudaEngine::getEngineCapability ( ) const  [inline], [noexcept]
```

Determine what execution capability this engine has.

If the engine has EngineCapability::kSTANDARD, then all engine functionality is valid. If the engine has EngineCapability::kSAFETY, then only the functionality in safe engine is valid. If the engine has EngineCapability::kDLA_STANDALONE, then only serialize, destroy, and const-accessor functions are valid.

Returns

The EngineCapability flag that the engine was built for.

9.82.3.10 getEngineStat()

```
int64_t nvinfer1::ICudaEngine::getEngineStat (
            EngineStat stat ) const  [inline], [noexcept]
```

Get engine statistics according to the given enum value.

Parameters

stat	The kind of statistics to query.

If stat is kTOTAL_WEIGHTS_SIZE, the return value is the total weights size in bytes in the engine. If stat is kSTRIPPED_WEIGHTS_SIZE, the return value is the stripped weight size in bytes for engines built with BuilderFlag::kSTRIP_PLAN.

When the BuilderFlag::kWEIGHT_STREAMING flag is enabled, engine weights may not be fully copied to the device. The reported total weight size reflects the sum of all weights utilized by the engine, which does not necessarily correspond to the actual GPU memory allocated.

Returns

The kind of statistics specified by EngineStat.

Warning

if kSTRIPPED_WEIGHTS_SIZE is passed to query a normal engine, this function will return -1 to indicate invalid enum value.

See also

EngineStat

BuilderFlag::kWEIGHT_STREAMING

getStreamableWeightsSize()

9.82.3.11 getErrorRecorder()

```
IErrorRecorder * nvinfer1::ICudaEngine::getErrorRecorder ( ) const  [inline], [noexcept]
```

Get the ErrorRecorder assigned to this interface.

Retrieves the assigned error recorder object for the given class. A nullptr will be returned if an error handler has not been set.

Returns

A pointer to the IErrorRecorder object that has been registered.

See also

setErrorRecorder()

9.82.3.12 getHardwareCompatibilityLevel()

```
HardwareCompatibilityLevel nvinfer1::ICudaEngine::getHardwareCompatibilityLevel ( ) const  [inline],
[noexcept]
```

Return the hardware compatibility level of this engine.

Returns

hardwareCompatibilityLevel The level of hardware compatibility.

9.82.3.13 getIOTensorName()

```
char const * nvinfer1::ICudaEngine::getIOTensorName (
            int32_t index ) const  [inline], [noexcept]
```

Return name of an IO tensor.

Parameters

index	value between 0 and getNbIOTensors()-1

See also

getNbIOTensors()

9.82.3.14 getName()

```
char const * nvinfer1::ICudaEngine::getName ( ) const  [inline], [noexcept]
```

Returns the name of the network associated with the engine.

The name is set during network creation and is retrieved after building or deserialization.

See also

INetworkDefinition::setName(), INetworkDefinition::getName()

Returns

A null-terminated C-style string representing the name of the network.

9.82.3.15 getNbAuxStreams()

```
int32_t nvinfer1::ICudaEngine::getNbAuxStreams ( ) const  [inline], [noexcept]
```

Return the number of auxiliary streams used by this engine.

This number will be less than or equal to the maximum allowed number of auxiliary streams set by IBuilderConfig::setMaxAuxStreams() API call when the engine was built.

Returns

the number of auxiliary streams used by this engine.

See also

IBuilderConfig::setMaxAuxStreams(), IExecutionContext::setAuxStreams()

9.82.3.16 getNbIOTensors()

```
int32_t nvinfer1::ICudaEngine::getNbIOTensors ( ) const  [inline], [noexcept]
```

Return number of IO tensors.

It is the number of input and output tensors for the network from which the engine was built. The names of the IO tensors can be discovered by calling getIOTensorName(i) for i in 0 to getNbIOTensors()-1.

See also

getIOTensorName()

9.82.3.17 getNbLayers()

```
int32_t nvinfer1::ICudaEngine::getNbLayers ( ) const  [inline], [noexcept]
```

Get the number of layers in the network.

The number of layers in the network is not necessarily the number in the original network definition, as layers may be combined or eliminated as the engine is optimized. This value can be useful when building per-layer tables, such as when aggregating profiling data over a number of executions.

Returns

The number of layers in the network.

9.82.3.18 getNbOptimizationProfiles()

```
int32_t nvinfer1::ICudaEngine::getNbOptimizationProfiles ( ) const  [inline], [noexcept]
```

Get the number of optimization profiles defined for this engine.

Returns

Number of optimization profiles. It is always at least 1.

See also

IExecutionContext::setOptimizationProfileAsync()

9.82.3.19 getProfileShape()

```
Dims nvinfer1::ICudaEngine::getProfileShape (
            char const * tensorName,
            int32_t profileIndex,
            OptProfileSelector select ) const  [inline], [noexcept]
```

Get the minimum / optimum / maximum dimensions for an input tensor given its name under an optimization profile.

Parameters

tensorName	The name of an input tensor.
profileIndex	The profile index, which must be between 0 and getNbOptimizationProfiles()-1.
select	Whether to query the minimum, optimum, or maximum dimensions for this input tensor.

Returns

The minimum / optimum / maximum dimensions for an input tensor in this profile. If the profileIndex is invalid or provided name does not map to an input tensor, return Dims{-1, {}}

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

9.82.3.20 getProfileTensorValuesV2()

```
int64_t const * nvinfer1::ICudaEngine::getProfileTensorValuesV2 (
            char const * tensorName,
```

```
int32_t profileIndex,
OptProfileSelector select ) const  [inline], [noexcept]
```

Get the minimum / optimum / maximum values (not dimensions) for an input tensor given its name under an optimization profile. These correspond to the values set using IOptimizationProfile::setShapeValuesV2 when the engine was built.

Parameters

tensorName	The name of an input tensor.
profileIndex	The profile index, which must be between 0 and getNbOptimizationProfiles()-1.
select	Whether to query the minimum, optimum, or maximum values for this input tensor.

Returns

The minimum / optimum / maximum values for an input tensor in this profile. If the profileIndex is invalid or the provided name does not map to an input tensor, or the tensor is not a shape binding, return nullptr.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

9.82.3.21 getProfilingVerbosity()

```
ProfilingVerbosity nvinfer1::ICudaEngine::getProfilingVerbosity ( ) const  [inline], [noexcept]
```

Return the ProfilingVerbosity the builder config was set to when the engine was built.

Returns

the profiling verbosity the builder config was set to when the engine was built.

See also

IBuilderConfig::setProfilingVerbosity()

9.82.3.22 getStreamableWeightsSize()

```
int64_t nvinfer1::ICudaEngine::getStreamableWeightsSize ( ) const  [inline], [noexcept]
```

Get the total size in bytes of all streamable weights.

The set of streamable weights is a subset of all network weights. The total size may exceed free GPU memory.

Returns

The total size in bytes of all streamable weights. Returns 0 if BuilderFlag::kWEIGHT_STREAMING is unset during engine building.

See also

setWeightStreamingBudgetV2()

9.82.3.23 getTacticSources()

```
TacticSources nvinfer1::ICudaEngine::getTacticSources ( ) const  [inline], [noexcept]
```

return the tactic sources required by this engine.

The value returned is equal to zero or more tactics sources set at build time via setTacticSources() in IBuilderConfig. Sources set by the latter but not returned by ICudaEngine::getTacticSources do not reduce overall engine execution time, and can be removed from future builds to reduce build time.

See also

IBuilderConfig::setTacticSources()

9.82.3.24 getTensorBytesPerComponent() [1/2]

```
int32_t nvinfer1::ICudaEngine::getTensorBytesPerComponent (
            char const * tensorName ) const  [inline], [noexcept]
```

Return the number of bytes per component of an element, or -1 if the tensor is not vectorized or provided name does not map to an input or output tensor.

The vector component size is returned if getTensorVectorizedDim(tensorName) != -1.

Parameters

tensorName	The name of an input or output tensor.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

The function can only return the result of profile 0, and issues a warning message when there are multiple profiles in the engine, use getTensorBytesPerComponent with profileIndex when there are multiple profiles.

See also

getTensorVectorizedDim()

getTensorBytesPerComponent(tensorName, profileIndex)

9.82.3.25 getTensorBytesPerComponent() [2/2]

```
int32_t nvinfer1::ICudaEngine::getTensorBytesPerComponent (
            char const * tensorName,
            int32_t profileIndex ) const  [inline], [noexcept]
```

Return the number of bytes per component of an element given of given profile, or -1 if the tensor is not vectorized or provided name does not map to an input or output tensor.

The vector component size is returned if getTensorVectorizedDim(tensorName, profileIndex) != -1.

Parameters

tensorName	The name of an input or output tensor.
profileIndex	The profile index to query

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

See also

getTensorVectorizedDim(tensorName, profileIndex)

9.82.3.26 getTensorComponentsPerElement() [1/2]

```
int32_t nvinfer1::ICudaEngine::getTensorComponentsPerElement (
            char const * tensorName ) const  [inline], [noexcept]
```

Return the number of components included in one element, or -1 if tensor is not vectorized or if the provided name does not map to an input or output tensor.

The number of elements in the vectors is returned if getTensorVectorizedDim(tensorName) != -1.

Parameters

tensorName	The name of an input or output tensor.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

The function can only return the result of profile 0, and issues a warning message when there are multiple profiles in the engine, use getTensorComponentsPerElement with profileIndex when there are multiple profiles.

See also

getTensorVectorizedDim()

getTensorComponentsPerElement(tensorName, profileIndex)

9.82.3.27 getTensorComponentsPerElement() [2/2]

```
int32_t nvinfer1::ICudaEngine::getTensorComponentsPerElement (
            char const * tensorName,
            int32_t profileIndex ) const  [inline], [noexcept]
```

Return the number of components included in one element of given profile, or -1 if tensor is not vectorized or the provided name does not map to an input or output tensor.

The number of elements in the vectors is returned if getTensorVectorizedDim(tensorName, profileIndex) != -1.

Parameters

tensorName	The name of an input or output tensor.
profileIndex	The profile index to query

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

See also

getTensorVectorizedDim(tensorName, profileIndex)

9.82.3.28 getTensorDataType()

```
DataType nvinfer1::ICudaEngine::getTensorDataType (
            char const * tensorName ) const  [inline], [noexcept]
```

Determine the required data type for a buffer from its tensor name.

Parameters

tensorName	The name of an input or output tensor.

Returns

The type of the data in the buffer, or DataType::kFLOAT if the provided name does not map to an input or output tensor.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

9.82.3.29 getTensorFormat() [1/2]

```
TensorFormat nvinfer1::ICudaEngine::getTensorFormat (
            char const * tensorName ) const  [inline], [noexcept]
```

Return the tensor format, or TensorFormat::kLINEAR if the provided name does not map to an input or output tensor.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

This API can only return the tensor format of profile 0, and issues a warning message when there are multiple profiles in the engine, use getTensorFormat with profileIndex when there are multiple profiles.

See also

getTensorFormat(tensorName, profileIndex)

9.82.3.30 getTensorFormat() [2/2]

```
TensorFormat nvinfer1::ICudaEngine::getTensorFormat (
            char const * tensorName,
            int32_t profileIndex ) const  [inline], [noexcept]
```

Return the tensor format of given profile, or TensorFormat::kLINEAR if the provided name does not map to an input or output tensor.

Parameters

tensorName	The name of an input or output tensor.
profileIndex	The profile index to query the format for.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

9.82.3.31 getTensorFormatDesc() [1/2]

```
char const * nvinfer1::ICudaEngine::getTensorFormatDesc (
            char const * tensorName ) const  [inline], [noexcept]
```

Return the human readable description of the tensor format, or empty string if the provided name does not map to an input or output tensor.

The description includes the order, vectorization, data type, and strides. Examples are shown as follows: Example 1: kCHW + FP32 ”Row-major linear FP32 format” Example 2: kCHW2 + FP16 ”Two-wide channel vectorized row-major FP16 format” Example 3: kHWC8 + FP16 + Line Stride = 32 ”Channel major FP16 format where C % 8 == 0 and H Stride % 32 == 0”

Parameters

tensorName	The name of an input or output tensor.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

The function can only return the result of profile 0, and issues a warning message when there are multiple profiles in the engine, use getTensorFormatDesc with profileIndex when there are multiple profiles.

9.82.3.32 getTensorFormatDesc() [2/2]

```
char const * nvinfer1::ICudaEngine::getTensorFormatDesc (
             char const * tensorName,
             int32_t profileIndex ) const  [inline], [noexcept]
```

Return the human readable description of the tensor format of given profile, or empty string if the provided name does not map to an input or output tensor.

The description includes the order, vectorization, data type, and strides. Examples are shown as follows: Example 1: kCHW + FP32 ”Row-major linear FP32 format” Example 2: kCHW2 + FP16 ”Two-wide channel vectorized row-major FP16 format” Example 3: kHWC8 + FP16 + Line Stride = 32 ”Channel major FP16 format where C % 8 == 0 and H Stride % 32 == 0”

Parameters

tensorName	The name of an input or output tensor.
profileIndex	The profile index to query the format for.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

9.82.3.33 getTensorIOMode()

```
TensorIOMode nvinfer1::ICudaEngine::getTensorIOMode (
             char const * tensorName ) const  [inline], [noexcept]
```

Determine whether a tensor is an input or output tensor.

Parameters

tensorName	The name of an input or output tensor.

Returns

kINPUT if tensorName is an input, kOUTPUT if tensorName is an output, or kNONE if neither.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

9.82.3.34 getTensorLocation()

```
TensorLocation nvinfer1::ICudaEngine::getTensorLocation (
            char const * tensorName ) const  [inline], [noexcept]
```

Get whether an input or output tensor must be on GPU or CPU.

Parameters

tensorName	The name of an input or output tensor.

Returns

TensorLocation::kDEVICE if tensorName must be on GPU, or TensorLocation::kHOST if on CPU, or TensorLocation::kDEVICE if the provided name does not map to an input or output tensor.

The location is established at build time. E.g. shape tensors inputs are typically required to be on the CPU.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

9.82.3.35 getTensorShape()

```
Dims nvinfer1::ICudaEngine::getTensorShape (
            char const * tensorName ) const  [inline], [noexcept]
```

Get shape of an input or output tensor.

Parameters

tensorName	The name of an input or output tensor.

Returns

shape of the tensor, with -1 in place of each dynamic runtime dimension, or Dims{-1, {}} if the provided name does not map to an input or output tensor.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

9.82.3.36 getTensorVectorizedDim() [1/2]

```
int32_t nvinfer1::ICudaEngine::getTensorVectorizedDim (
            char const * tensorName ) const  [inline], [noexcept]
```

Return the dimension index that the buffer is vectorized, or -1 if the provided name does not map to an input or output tensor.

Specifically -1 is returned if scalars per vector is 1.

Parameters

tensorName	The name of an input or output tensor.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

The function can only return the result of profile 0, and issues a warning message when there are multiple profiles in the engine, use getTensorVectorizedDim with profileIndex when there are multiple profiles.

9.82.3.37 getTensorVectorizedDim() [2/2]

```
int32_t nvinfer1::ICudaEngine::getTensorVectorizedDim (
            char const * tensorName,
            int32_t profileIndex ) const  [inline], [noexcept]
```

Return the dimension index that the buffer is vectorized of given profile, or -1 if the provided name does not map to an input or output tensor.

Specifically -1 is returned if scalars per vector is 1.

Parameters

tensorName	The name of an input.
profileIndex	The profile index to query the format for.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

9.82.3.38 getWeightStreamingAutomaticBudget()

```
int64_t nvinfer1::ICudaEngine::getWeightStreamingAutomaticBudget ( ) const  [inline], [noexcept]
```

TensorRT automatically determines a device memory budget for the model to run. The budget is close to the current free memory size, leaving some space for other memory needs in the user's application. If the budget exceeds the size obtained from getStreamableWeightsSize(), it is capped to that size, effectively disabling weight streaming. Since TensorRT lacks information about the user's allocations, the remaining memory size might be larger than required, leading to wasted memory, or smaller than required, causing an out-of-memory error. For optimal memory allocation, it is recommended to manually calculate and set the budget.

Warning

BuilderFlag::kWEIGHT_STREAMING must be set during engine building.

The return value may change between TensorRT minor versions.

Returns

The weight streaming budget in bytes. Please set with setWeightStreamingBudgetV2().

See also

BuilderFlag::kWEIGHT_STREAMING

setWeightStreamingBudgetV2()

9.82.3.39 getWeightStreamingBudgetV2()

```
int64_t nvinfer1::ICudaEngine::getWeightStreamingBudgetV2 ( ) const  [inline], [noexcept]
```

Returns the current weight streaming device memory budget in bytes.

Warning

BuilderFlag::kWEIGHT_STREAMING must be set during engine building.

Returns

The weight streaming budget in bytes. Please see setWeightStreamingBudgetV2() for the possible return values. Returns getStreamableWeightsSize() if weight streaming is disabled.

See also

BuilderFlag::kWEIGHT_STREAMING

getStreamableWeightsSize()

9.82.3.40 getWeightStreamingScratchMemorySize()

```
int64_t nvinfer1::ICudaEngine::getWeightStreamingScratchMemorySize ( ) const  [inline], [noexcept]
```

Returns the size of the scratch memory required by the current weight streaming budget.

Weight streaming requires small amounts of scratch memory on the GPU to stage CPU weights right before execution. This value is typically much smaller than the total streamable weights size. Each IExecutionContext will then allocate this additional memory or the user can provide the additional memory through getDeviceMemorySizeV2() and IExecutionContext::setDeviceMemoryV2().

The return value of this call depends on setWeightStreamingBudgetV2().

Warning

BuilderFlag::kWEIGHT_STREAMING must be set during engine building.

Returns

The weight streaming scratch memory in bytes. Returns 0 if weight streaming is disabled.

See also

BuilderFlag::kWEIGHT_STREAMING

setWeightStreamingBudgetV2()

getStreamableWeightsSize()

getDeviceMemorySizeV2()

getDeviceMemorySizeForProfileV2()

IExecutionContext::setDeviceMemoryV2()

9.82.3.41 isDebugTensor()

```
bool nvinfer1::ICudaEngine::isDebugTensor (
            char const * name ) const  [inline], [noexcept]
```

Check if a tensor is marked as a debug tensor.

Determine whether the given name corresponds to a debug tensor.

Returns

True if tensor is a debug tensor, false otherwise.

See also

INetworkDefinition::markDebug

9.82.3.42 isRefittable()

```
bool nvinfer1::ICudaEngine::isRefittable ( ) const  [inline], [noexcept]
```

Return true if an engine can be refit.

See also

nvinfer1::createInferRefitter()

9.82.3.43 isShapeInferenceIO()

```
bool nvinfer1::ICudaEngine::isShapeInferenceIO (
            char const * tensorName ) const  [inline], [noexcept]
```

True if tensor is required as input for shape calculations or is output from shape calculations.

Return true for either of the following conditions:

- The tensor is a network input, and its value is required for IExecutionContext::getTensorShape() to return the shape of a network output.
- The tensor is a network output, and inferShape() will compute its values.

For example, if a network uses an input tensor "foo" as an addend to an IElementWiseLayer that computes the "reshape dimensions" for IShuffleLayer, then isShapeInferenceIO("foo") == true. If the network copies said input tensor "foo" to an output "bar", then isShapeInferenceIO("bar") == true and IExecutionContext::inferShapes() will write to "bar".

9.82.3.44 serialize()

```
IHostMemory * nvinfer1::ICudaEngine::serialize ( ) const  [inline], [noexcept]
```

Serialize the network to a stream.

Returns

A IHostMemory object that contains the serialized engine.

The network may be deserialized with IRuntime::deserializeCudaEngine().

See also

IRuntime::deserializeCudaEngine()

9.82.3.45 serializeWithConfig()

```
IHostMemory * nvinfer1::ICudaEngine::serializeWithConfig (
            ISerializationConfig & config ) const  [inline], [noexcept]
```

Serialize the network to a stream with the provided SerializationConfig.

Returns

An IHostMemory object that contains the serialized engine.

The network may be deserialized with IRuntime::deserializeCudaEngine(). Serializing plan file with SerializationFlag::kEXCLUDE_WE requires building the engine with kREFIT, kREFIT_IDENTICAL or kREFIT_INDIVIDUAL.

The only applicable scenario for SerializationFlag::kINCLUDE_REFIT is when serializing weight-stripping engines without kEXCLUDE_WEIGHTS. By default, the resulting serialized engine is unrefittable. Setting SerializationFlag::kINCLUDE_REFIT ensures that the serialized engine remains refittable.

See also

IRuntime::deserializeCudaEngine()

9.82.3.46 setErrorRecorder()

```
void nvinfer1::ICudaEngine::setErrorRecorder (
            IErrorRecorder * recorder )  [inline], [noexcept]
```

Set the ErrorRecorder for this interface.

Assigns the ErrorRecorder to this interface. The ErrorRecorder will track all errors during execution. This function will call incRefCount of the registered ErrorRecorder at least once. Setting recorder to nullptr unregisters the recorder with the interface, resulting in a call to decRefCount if a recorder has been registered.

If an error recorder is not set, messages will be sent to the global log stream.

Parameters

recorder	The error recorder to register with this interface.

See also

getErrorRecorder()

9.82.3.47 setWeightStreamingBudgetV2()

```
bool nvinfer1::ICudaEngine::setWeightStreamingBudgetV2 (
            int64_t gpuMemoryBudget ) [inline], [noexcept]
```

Limit the maximum amount of GPU memory usable for network weights in bytes.

Parameters

gpuMemoryBudget	This parameter must be a non-negative value. 0: Only small amounts of scratch memory will required to run the model. >= getStreamableWeightsSize (default): Disables weight streaming. The execution may fail if the network is too large for GPU memory.

By setting a weight limit, users can expect a GPU memory usage reduction on the order of (total bytes for network weights) - gpuMemoryBudget bytes. Maximum memory savings occur when gpuMemoryBudget is set to 0. Each IExecutionContext will require getWeightStreamingScratchMemorySize() bytes of additional device memory if the engine is streaming its weights (budget < getStreamableWeightsSize()).

Streaming larger amounts of memory will likely result in lower performance except in some boundary cases where streaming weights allows the user to run larger batch sizes. The higher throughput offsets the increased latency in these cases. Tuning the value of the memory limit is recommended for best performance.

Warning

GPU memory for the weights is allocated in this call and will be deallocated by enabling weight streaming or destroying the ICudaEngine.

BuilderFlag::kWEIGHT_STREAMING must be set during engine building.

The weights streaming budget cannot be modified while there are active IExecutionContexts.

Returns

true if the memory limit is valid and the call was successful, false otherwise.

See also

BuilderFlag::kWEIGHT_STREAMING
getWeightStreamingBudgetV2()
getWeightStreamingScratchMemorySize()
getWeightStreamingAutomaticBudget()
getStreamableWeightsSize()

9.82.4 Member Data Documentation

9.82.4.1 mImpl

```
apiv::VCudaEngine* nvinfer1::ICudaEngine::mImpl  [protected]
```

The documentation for this class was generated from the following file:

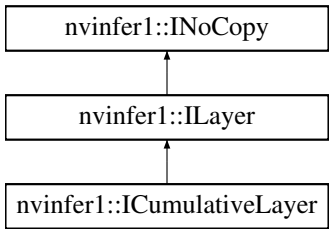
- NvInferRuntime.h

9.83 nvinfer1::ICumulativeLayer Class Reference

Layer that represents a cumulative operation across a tensor.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::ICumulativeLayer:



Public Member Functions

- bool setOperation (CumulativeOperation op) noexcept

  Set the cumulative operation for the layer.
- CumulativeOperation getOperation () const noexcept

  Get the cumulative operation for the layer.
- void setExclusive (bool exclusive) noexcept

  Set whether it is an exclusive accumulation or inclusive accumulation.
- bool getExclusive () const noexcept

  Get whether it is exclusive accumulation or inclusive accumulation.
- void setReverse (bool reverse) noexcept

  Specify whether the cumulative operation should be applied backward.
- bool getReverse () const noexcept

  Get the boolean that specifies whether the cumulative operation should be applied backward.

Protected Member Functions

- virtual ~ICumulativeLayer () noexcept=0

Protected Attributes

- apiv::VCumulativeLayer * mImpl

9.83.1 Detailed Description

Layer that represents a cumulative operation across a tensor.

It computes successive reductions across an axis of a tensor. The output always has the same shape as the input.

If the reduction operation is summation, then this is also known as prefix-sum or cumulative sum.

The operation has forward vs. reverse variants, and inclusive vs. exclusive variants.

For example, let the input be a vector x of length n and the output be vector y. Then y[j] = sum(x[...]) where ... denotes a sequence of indices from this table:

	forward	reverse
inclusive	0..j	j..n-1
exclusive	0..j-1	j+1..n-1

For multidimensional tensors, the reductions apply across a specified axis. For example, given a 2D input, a forward inclusive cumulative operation across axis 0 generates cumulative sums within each column.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.83.2 Constructor & Destructor Documentation

9.83.2.1 ~ICumulativeLayer()

```
nvinfer1::ICumulativeLayer::~ICumulativeLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.83.3 Member Function Documentation

9.83.3.1 getExclusive()

```
bool nvinfer1::ICumulativeLayer::getExclusive ( ) const  [inline], [noexcept]
```

Get whether it is exclusive accumulation or inclusive accumulation.

Returns

Whether the operation will exclude the element at the current index

See also

setExclusive

9.83.3.2 getOperation()

```
CumulativeOperation nvinfer1::ICumulativeLayer::getOperation ( ) const  [inline], [noexcept]
```

Get the cumulative operation for the layer.

Returns

The reduction operation to be performed

See also

setOperation(), CumulativeOperation

9.83.3.3 getReverse()

```
bool nvinfer1::ICumulativeLayer::getReverse ( ) const  [inline], [noexcept]
```

Get the boolean that specifies whether the cumulative operation should be applied backward.

Returns

Whether the cumulative will run in the reverse direction from the last element

See also

setReverse

9.83.3.4 setExclusive()

```
void nvinfer1::ICumulativeLayer::setExclusive (
             bool exclusive )  [inline], [noexcept]
```

Set whether it is an exclusive accumulation or inclusive accumulation.

Parameters

exclusive	Whether the operation will exclude the element at the current index

See also

getExclusive

9.83.3.5 setOperation()

```
bool nvinfer1::ICumulativeLayer::setOperation (
            CumulativeOperation op )  [inline], [noexcept]
```

Set the cumulative operation for the layer.

Parameters

op	The reduction operation to be performed

Returns

Whether `op` is valid and the operation successfully set

See also

getOperation(), CumulativeOperation

9.83.3.6 setReverse()

```
void nvinfer1::ICumulativeLayer::setReverse (
            bool reverse )  [inline], [noexcept]
```

Specify whether the cumulative operation should be applied backward.

Parameters

reverse	Whether the cumulative will run in the reverse direction from the last element

See also

getReverse

9.83.4 Member Data Documentation

9.83.4.1 mImpl

```
apiv::VCumulativeLayer* nvinfer1::ICumulativeLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.84 IDebugListener Class Reference

User-implemented callback for notification when value of a debug tensor is updated.

```
#include <NvInferRuntime.h>
```

9.84.1 Detailed Description

User-implemented callback for notification when value of a debug tensor is updated.

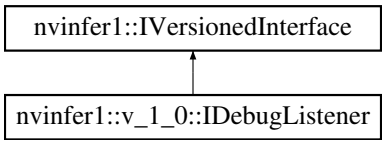
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.85 nvinfer1::v_1_0::IDebugListener Class Reference

```
#include <NvInferRuntime.h>
```

Inheritance diagram for nvinfer1::v_1_0::IDebugListener:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual bool processDebugTensor (void const ∗addr, TensorLocation location, DataType type, Dims const &shape, char const ∗name, cudaStream_t stream)=0

  Callback function that is called when a debug tensor's value is updated and the debug state of the tensor is set to true. Content in the given address is only guaranteed to be valid for the duration of the callback.
- ∼IDebugListener () override=default

Additional Inherited Members

9.85.1 Constructor & Destructor Documentation

9.85.1.1 ∼IDebugListener()

```
nvinfer1::v_1_0::IDebugListener::∼IDebugListener ( )  [override], [default]
```

9.85.2 Member Function Documentation

9.85.2.1 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IDebugListener::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.85.2.2 processDebugTensor()

```
virtual bool nvinfer1::v_1_0::IDebugListener::processDebugTensor (
            void const * addr,
            TensorLocation location,
            DataType type,
            Dims const & shape,
            char const * name,
            cudaStream_t stream )  [pure virtual]
```

Callback function that is called when a debug tensor's value is updated and the debug state of the tensor is set to true. Content in the given address is only guaranteed to be valid for the duration of the callback.

Parameters

location	TensorLocation of the tensor.
addr	pointer to buffer.
type	data Type of the tensor.
shape	shape of the tensor.
name	name of the tensor.
stream	CUDA stream object.

Returns

True on success, false otherwise.

The documentation for this class was generated from the following file:

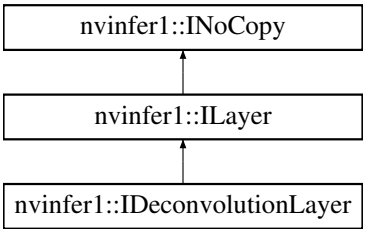
- NvInferRuntime.h

9.86 nvinfer1::IDeconvolutionLayer Class Reference

A deconvolution layer in a network definition.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IDeconvolutionLayer:



Public Member Functions

- void setNbOutputMaps (int64_t nbOutputMaps) noexcept

  Set the number of output feature maps for the deconvolution.
- int64_t getNbOutputMaps () const noexcept

  Get the number of output feature maps for the deconvolution.
- void setNbGroups (int64_t nbGroups) noexcept

  Set the number of groups for a deconvolution.
- int64_t getNbGroups () const noexcept

  Get the number of groups for a deconvolution.
- void setKernelWeights (Weights weights) noexcept

Set the kernel weights for the deconvolution.

- Weights getKernelWeights () const noexcept

  Get the kernel weights for the deconvolution.
- void setBiasWeights (Weights weights) noexcept

  Set the bias weights for the deconvolution.
- Weights getBiasWeights () const noexcept

  Get the bias weights for the deconvolution.
- void setPrePadding (Dims const &padding) noexcept

  Set the multi-dimension pre-padding of the deconvolution.
- Dims getPrePadding () const noexcept

  Get the pre-padding.
- void setPostPadding (Dims const &padding) noexcept

  Set the multi-dimension post-padding of the deconvolution.
- Dims getPostPadding () const noexcept

  Get the padding.
- void setPaddingMode (PaddingMode paddingMode) noexcept

  Set the padding mode.
- PaddingMode getPaddingMode () const noexcept

  Get the padding mode.
- void setKernelSizeNd (Dims const &kernelSize) noexcept

  Set the multi-dimension kernel size of the deconvolution.
- Dims getKernelSizeNd () const noexcept

  Get the multi-dimension kernel size of the deconvolution.
- void setStrideNd (Dims const &stride) noexcept

  Set the multi-dimension stride of the deconvolution.
- Dims getStrideNd () const noexcept

  Get the multi-dimension stride of the deconvolution.
- void setPaddingNd (Dims const &padding) noexcept

  Set the multi-dimension padding of the deconvolution.
- Dims getPaddingNd () const noexcept

  Get the multi-dimension padding of the deconvolution.
- void setDilationNd (Dims const &dilation) noexcept

  Set the multi-dimension dilation of the deconvolution.
- Dims getDilationNd () const noexcept

  Get the multi-dimension dilation of the deconvolution.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~IDeconvolutionLayer () noexcept=0

Protected Attributes

- apiv::VDeconvolutionLayer * mImpl

9.86.1 Detailed Description

A deconvolution layer in a network definition.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.86.2 Constructor & Destructor Documentation

9.86.2.1 ~IDeconvolutionLayer()

```
nvinfer1::IDeconvolutionLayer::~IDeconvolutionLayer ( )  [inline], [protected], [pure virtual],
[default], [noexcept]
```

9.86.3 Member Function Documentation

9.86.3.1 getBiasWeights()

```
Weights nvinfer1::IDeconvolutionLayer::getBiasWeights ( ) const  [inline], [noexcept]
```

Get the bias weights for the deconvolution.

See also

getBiasWeights()

9.86.3.2 getDilationNd()

```
Dims nvinfer1::IDeconvolutionLayer::getDilationNd ( ) const  [inline], [noexcept]
```

Get the multi-dimension dilation of the deconvolution.

See also

setDilationNd()

9.86.3.3 getKernelSizeNd()

```
Dims nvinfer1::IDeconvolutionLayer::getKernelSizeNd ( ) const  [inline], [noexcept]
```

Get the multi-dimension kernel size of the deconvolution.

See also

setKernelSizeNd()

9.86.3.4 getKernelWeights()

```
Weights nvinfer1::IDeconvolutionLayer::getKernelWeights ( ) const  [inline], [noexcept]
```

Get the kernel weights for the deconvolution.

See also

setNbGroups()

9.86.3.5 getNbGroups()

```
int64_t nvinfer1::IDeconvolutionLayer::getNbGroups ( ) const  [inline], [noexcept]
```

Get the number of groups for a deconvolution.

See also

setNbGroups()

9.86.3.6 getNbOutputMaps()

```
int64_t nvinfer1::IDeconvolutionLayer::getNbOutputMaps ( ) const  [inline], [noexcept]
```

Get the number of output feature maps for the deconvolution.

See also

setNbOutputMaps()

9.86.3.7 getPaddingMode()

```
PaddingMode nvinfer1::IDeconvolutionLayer::getPaddingMode ( ) const  [inline], [noexcept]
```

Get the padding mode.

Default: kEXPLICIT_ROUND_DOWN

See also

setPaddingMode()

9.86.3.8 getPaddingNd()

```
Dims nvinfer1::IDeconvolutionLayer::getPaddingNd ( ) const  [inline], [noexcept]
```

Get the multi-dimension padding of the deconvolution.

If the padding is asymmetric, the pre-padding is returned.

See also

setPaddingNd()

9.86.3.9 getPostPadding()

```
Dims nvinfer1::IDeconvolutionLayer::getPostPadding ( ) const  [inline], [noexcept]
```

Get the padding.

See also

setPostPadding()

9.86.3.10 getPrePadding()

```
Dims nvinfer1::IDeconvolutionLayer::getPrePadding ( ) const  [inline], [noexcept]
```

Get the pre-padding.

See also

setPrePadding()

9.86.3.11 getStrideNd()

```
Dims nvinfer1::IDeconvolutionLayer::getStrideNd ( ) const  [inline], [noexcept]
```

Get the multi-dimension stride of the deconvolution.

See also

setStrideNd()

9.86.3.12 setBiasWeights()

```
void nvinfer1::IDeconvolutionLayer::setBiasWeights (
            Weights weights )  [inline], [noexcept]
```

Set the bias weights for the deconvolution.

Bias is optional. To omit bias, set the count value of the weights structure to zero.

The bias is applied per-feature-map, so the number of weights (if non-zero) must be equal to the number of output feature maps.

See also

getBiasWeights()

9.86.3.13 setDilationNd()

```
void nvinfer1::IDeconvolutionLayer::setDilationNd (
            Dims const & dilation )  [inline], [noexcept]
```

Set the multi-dimension dilation of the deconvolution.

Default: (1, 1, ..., 1)

See also

getDilationNd()

9.86.3.14 setInput()

```
void nvinfer1::ILayer::setInput (
            int32_t index,
            ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor

Input 0 is the input activation tensor. Input 1 is the kernel tensor. If used, the kernel weights parameter must be set to empty weights. Input 2 is the bias tensor. If used, the bias parameter must be set to empty weights.

See also

getKernelWeights(), setKernelWeights(), getBiasWeights(), setBiasWeights()

9.86.3.15 setKernelSizeNd()

```
void nvinfer1::IDeconvolutionLayer::setKernelSizeNd (
            Dims const & kernelSize )  [inline], [noexcept]
```

Set the multi-dimension kernel size of the deconvolution.

If executing this layer on DLA, there are two restrictions: 1) Only 2D Kernel is supported. 2) Kernel height and width must be in the range [1,32] or the combinations of [64, 96, 128] in one dimension and 1 in the other dimensions, i.e. [1x64] or [64x1] are valid, but not [64x64].

See also

getKernelSizeNd()

9.86.3.16 setKernelWeights()

```
void nvinfer1::IDeconvolutionLayer::setKernelWeights (
            Weights weights )  [inline], [noexcept]
```

Set the kernel weights for the deconvolution.

The weights are specified as a contiguous array in `CKRS` order, where `C` the number of input channels, `K` the number of output feature maps, and `R` and `S` are the height and width of the filter.

See also

getWeights()

9.86.3.17 setNbGroups()

```
void nvinfer1::IDeconvolutionLayer::setNbGroups (
            int64_t nbGroups ) [inline], [noexcept]
```

Set the number of groups for a deconvolution.

The input tensor channels are divided into `nbGroups` groups, and a deconvolution is executed for each group, using a filter per group. The results of the group convolutions are concatenated to form the output.

If executing this layer on DLA, nbGroups must be one

Note

When using groups in int8 mode, the size of the groups (i.e. the channel count divided by the group count) must be a multiple of 4 for both input and output.

Default: 1

See also

getNbGroups()

9.86.3.18 setNbOutputMaps()

```
void nvinfer1::IDeconvolutionLayer::setNbOutputMaps (
            int64_t nbOutputMaps ) [inline], [noexcept]
```

Set the number of output feature maps for the deconvolution.

If executing this layer on DLA, the number of output maps must be in the range [1,8192].

See also

getNbOutputMaps()

9.86.3.19 setPaddingMode()

```
void nvinfer1::IDeconvolutionLayer::setPaddingMode (
            PaddingMode paddingMode ) [inline], [noexcept]
```

Set the padding mode.

Padding mode takes precedence if both setPaddingMode and setPre/PostPadding are used.

Default: kEXPLICIT_ROUND_DOWN

See also

getPaddingMode()

9.86.3.20 setPaddingNd()

```
void nvinfer1::IDeconvolutionLayer::setPaddingNd (
            Dims const & padding )  [inline], [noexcept]
```

Set the multi-dimension padding of the deconvolution.

The output will be trimmed by this number of elements on both sides of every dimension. In other words, it resembles the inverse of a convolution layer with this padding size. Padding is symmetric, and negative padding is not supported.

Default: (0, 0, ..., 0)

If executing this layer on DLA, padding must be 0.

See also

getPaddingNd() setPadding() getPadding()

9.86.3.21 setPostPadding()

```
void nvinfer1::IDeconvolutionLayer::setPostPadding (
            Dims const & padding )  [inline], [noexcept]
```

Set the multi-dimension post-padding of the deconvolution.

The output will be trimmed by this number of elements on the end of every dimension. In other words, it resembles the inverse of a convolution layer with this padding size. Negative padding is not supported.

Default: (0, 0, ..., 0)

See also

getPostPadding()

9.86.3.22 setPrePadding()

```
void nvinfer1::IDeconvolutionLayer::setPrePadding (
            Dims const & padding )  [inline], [noexcept]
```

Set the multi-dimension pre-padding of the deconvolution.

The output will be trimmed by this number of elements on the start of every dimension. In other words, it resembles the inverse of a convolution layer with this padding size. Negative padding is not supported.

Default: (0, 0, ..., 0)

See also

getPrePadding()

9.86.3.23 setStrideNd()

```
void nvinfer1::IDeconvolutionLayer::setStrideNd (
            Dims const & stride )  [inline], [noexcept]
```

Set the multi-dimension stride of the deconvolution.

Default: (1, 1, ..., 1)

If executing this layer on DLA, there are two restrictions: 1) Only 2D Stride is supported. 2) Stride height and width must be in the range [1,32] or the combinations of [64, 96, 128] in one dimension and 1 in the other dimensions, i.e. [1x64] or [64x1] are valid, but not [64x64].

See also

getStrideNd()

9.86.4 Member Data Documentation

9.86.4.1 mImpl

```
apiv::VDeconvolutionLayer* nvinfer1::IDeconvolutionLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

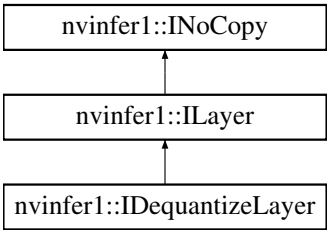
- NvInfer.h

9.87 nvinfer1::IDequantizeLayer Class Reference

A Dequantize layer in a network definition.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IDequantizeLayer:



Public Member Functions

- int32_t getAxis () const noexcept

  Get the quantization axis.
- void setAxis (int32_t axis) noexcept

  Set the quantization axis.
- bool setBlockShape (Dims const &blockShape) noexcept

  Set the shape of the quantization block.
- TRT_NODISCARD Dims getBlockShape () const noexcept

  Get the shape of the quantization block.
- void setToType (DataType toType) noexcept

  Set the Dequantize layer output type.
- DataType getToType () const noexcept

  Return the Dequantize layer output type.

Protected Member Functions

- virtual ~IDequantizeLayer () noexcept=0

Protected Attributes

- apiv::VDequantizeLayer * mImpl

9.87.1 Detailed Description

A Dequantize layer in a network definition.

This layer accepts a quantized type input tensor, and uses the configured scale and zeroPt inputs to dequantize the input according to: `output = (input - zeroPt) * scale`

The first input (index 0) is the tensor to be dequantized. The second (index 1) and third (index 2) are the scale and zero point respectively. `scale` and `zeroPt` should have identical dimensions, and a rank that is lower or equal to 2.

The `zeroPt` tensor is optional, and if not set, will be assumed to be zero. Its data type must be identical to the input's data type. `zeroPt` must only contain zero-valued coefficients, because only symmetric quantization is supported. The `scale` value must be a scalar for per-tensor quantization, a 1D tensor for per-channel quantization, or the same rank as the input tensor for block quantization. All `scale` coefficients must have strictly positive values. The size of the 1D `scale` tensor must match the size of the quantization axis. For block quantization, the shape of `scale` tensor must match the shape of the input, except for one dimension (the last or second to last dimension) in which blocking occurs. The size of `zeroPt` must match the size of `scale`.

The subgraph which terminates with the `zeroPt` tensor must be a build-time constant containing only zeros. The output type, if constrained, must be constrained to DataType::kFLOAT, DataType::kHALF, or DataType::kBF16. The input type, if constrained, must be constrained to DataType::kINT8, DataType::kFP8, DataType::kINT4 or DataType::kFP4. The output size is the same as the input size. The quantization axis is in reference to the input tensor's dimensions.

IDequantizeLayer supports DataType::kINT8 (default), DataType::kFP8, DataType::kINT4 or DataType::kFP4. For strongly typed networks, `input` data type must be the same as `zeroPt` data type.

IDequantizeLayer supports DataType::kFLOAT, DataType::kHALF, or DataType::kBF16 output. The output data type must be configured explicitly using `setToType`.

As an example of the operation of this layer, imagine a 4D NCHW activation input which can be quantized using a single scale coefficient (referred to as per-tensor quantization):

```
For each n in N:
    For each c in C:
        For each h in H:
            For each w in W:
                output[n,c,h,w] = (input[n,c,h,w] - zeroPt) * scale
```

Per-channel dequantization is supported only for input that is rooted at an IConstantLayer (i.e. weights). Activations cannot be quantized per-channel. As an example of per-channel operation, imagine a 4D KCRS weights input and K (dimension 0) as the quantization axis. The scale is an array of coefficients, which is the same size as the quantization axis.

```
For each k in K:
    For each c in C:
        For each r in R:
            For each s in S:
                output[k,c,r,s] = (input[k,c,r,s] - zeroPt[k]) * scale[k]
```

Block dequantization is supported for input types DataType::kFP4, DataType::kFP8 and DataType::kINT4. As an example of blocked operation, imagine a 2D RS input with R (dimension 0) as the blocking axis and B as the block size. The scale is a 2D array of coefficients, with dimensions (R//B, S).

```
For each r in R:
    For each s in S:
        output[r,s] = (input[r,s] - zeroPt[r//B, s]) * scale[r//B, s]
```

Note

Only symmetric quantization is supported.

Currently the only allowed build-time constant `zeroPt` subgraphs are:

1. Constant -> Quantize
2. Constant -> Cast -> Quantize

The input tensor for this layer must not be a scalar.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.87.2 Constructor & Destructor Documentation

9.87.2.1 ~IDequantizeLayer()

`nvinfer1::IDequantizeLayer::~IDequantizeLayer ( ) [inline], [protected], [pure virtual], [default], [noexcept]`

9.87.3 Member Function Documentation

9.87.3.1 getAxis()

```
int32_t nvinfer1::IDequantizeLayer::getAxis ( ) const  [inline], [noexcept]
```

Get the quantization axis.

Returns

axis parameter set by setAxis(). The return value is the index of the quantization axis in the input tensor's dimensions. A value of -1 indicates per-tensor quantization. The default value is -1.

9.87.3.2 getBlockShape()

```
TRT_NODISCARD Dims nvinfer1::IDequantizeLayer::getBlockShape ( ) const  [inline], [noexcept]
```

Get the shape of the quantization block.

The default value is empty Dims.

See also

setBlockShape()

9.87.3.3 getToType()

```
DataType nvinfer1::IDequantizeLayer::getToType ( ) const  [inline], [noexcept]
```

Return the Dequantize layer output type.

Returns

toType parameter set during layer creation or by setToType(). The return value is the output type of the quantize layer. The default value is DataType::kFLOAT.

9.87.3.4 setAxis()

```
void nvinfer1::IDequantizeLayer::setAxis (
            int32_t axis )  [inline], [noexcept]
```

Set the quantization axis.

Set the index of the quantization axis (with reference to the input tensor's dimensions). The axis must be a valid axis if the scale tensor has more than one coefficient. The axis value will be ignored if the scale tensor has exactly one coefficient (per-tensor quantization).

9.87.3.5 setBlockShape()

```
bool nvinfer1::IDequantizeLayer::setBlockShape (
            Dims const & blockShape )  [inline], [noexcept]
```

Set the shape of the quantization block.

Parameters

blockShape	The shape of the quantization block.

Set the shape of the quantization block. Allowed values are positive values and -1 which denotes a fully blocked dimension. Returns true if the block shape was set successfully, false if the block shape is invalid. The default value is empty Dims.

See also

getBlockShape()

9.87.3.6 setToType()

```
void nvinfer1::IDequantizeLayer::setToType (
            DataType toType )  [inline], [noexcept]
```

Set the Dequantize layer output type.

Parameters

toType	The DataType of the output tensor.

Set the output type of the dequantize layer. Valid values are DataType::kFLOAT, DataType::kHALF and DataType::kBF16. If the network is strongly typed, setToType must be used to set the output type, and use of setOutputType is an error. Otherwise, types passed to setOutputType and setToType must be the same.

See also

NetworkDefinitionCreationFlag::kSTRONGLY_TYPED

9.87.4 Member Data Documentation

9.87.4.1 mImpl

```
apiv::VDequantizeLayer* nvinfer1::IDequantizeLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

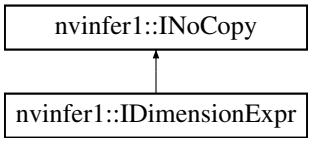
- NvInfer.h

9.88 nvinfer1::IDimensionExpr Class Reference

An IDimensionExpr represents an integer expression constructed from constants, input dimensions, and binary operations. These expressions are can be used in overrides of IPluginV2DynamicExt::getOutputDimensions or IPluginV3OneBuild::getOutputShapes() to define output dimensions in terms of input dimensions.

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::IDimensionExpr:



Public Member Functions

- bool isConstant () const noexcept

  Return true if expression is a build-time constant.
- int64_t getConstantValue () const noexcept

  Get the value of the constant.
- bool isSizeTensor () const noexcept

  Return true if this denotes the value of a size tensor.

Protected Member Functions

- virtual ~IDimensionExpr () noexcept=0

Protected Attributes

- apiv::VDimensionExpr * mImpl

9.88.1 Detailed Description

An IDimensionExpr represents an integer expression constructed from constants, input dimensions, and binary operations. These expressions are can be used in overrides of IPluginV2DynamicExt::getOutputDimensions or IPluginV3OneBuild::getOutputShapes() to define output dimensions in terms of input dimensions.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

See also

DimensionOperation, IPluginV2DynamicExt::getOutputDimensions, IPluginV3OneBuild::getOutputShapes()

9.88.2 Constructor & Destructor Documentation

9.88.2.1 ~IDimensionExpr()

```
nvinfer1::IDimensionExpr::~IDimensionExpr ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.88.3 Member Function Documentation

9.88.3.1 getConstantValue()

```
int64_t nvinfer1::IDimensionExpr::getConstantValue ( ) const  [inline], [noexcept]
```

Get the value of the constant.

If isConstant(), returns value of the constant. If !isConstant(), return std::numeric_limits<int64_t>::min().

9.88.3.2 isConstant()

```
bool nvinfer1::IDimensionExpr::isConstant ( ) const  [inline], [noexcept]
```

Return true if expression is a build-time constant.

9.88.3.3 isSizeTensor()

```
bool nvinfer1::IDimensionExpr::isSizeTensor ( ) const  [inline], [noexcept]
```

Return true if this denotes the value of a size tensor.

Returns

True if this was created with method IExprBuilder::declareSizeTensor, false otherwise

9.88.4 Member Data Documentation

9.88.4.1 mImpl

```
apiv::VDimensionExpr* nvinfer1::IDimensionExpr::mImpl  [protected]
```

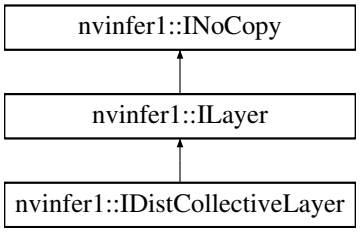
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.89 nvinfer1::IDistCollectiveLayer Class Reference

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IDistCollectiveLayer:



Protected Member Functions

- virtual ~IDistCollectiveLayer () noexcept=0

Protected Attributes

- apiv::VDistCollectiveLayer * mImpl

Additional Inherited Members

9.89.1 Detailed Description

Implements a distributed collective operation.

See INetworkDefinition::addDistCollective for more details.

9.89.2 Constructor & Destructor Documentation

9.89.2.1 ~IDistCollectiveLayer()

```
nvinfer1::IDistCollectiveLayer::~IDistCollectiveLayer ( )  [inline], [protected], [pure virtual],
[default], [noexcept]
```

9.89.3 Member Data Documentation

9.89.3.1 mImpl

```
apiv::VDistCollectiveLayer* nvinfer1::IDistCollectiveLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

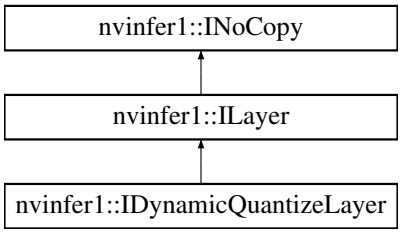
- NvInfer.h

9.90 nvinfer1::IDynamicQuantizeLayer Class Reference

A network layer to perform dynamic quantization.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IDynamicQuantizeLayer:



Public Member Functions

- void setToType (DataType toType) noexcept

  Set DynamicQuantizeLayer's quantized output type.
- DataType getToType () const noexcept

  Return DynamicQuantizeLayer's quantized output type.
- void setScaleType (DataType scaleType) noexcept

  Set the data type of the scale factors used to quantize the data.
- DataType getScaleType () const noexcept

  Return the scale factors data type.
- TRT_DEPRECATED void setAxis (int32_t axis) noexcept

  Set the axis along which block quantization occurs.
- TRT_DEPRECATED int32_t getAxis () const noexcept

  Get the axis along which blocking occurs.
- TRT_DEPRECATED void setBlockSize (int32_t size) noexcept

  Set the size of the quantization block.
- TRT_DEPRECATED int32_t getBlockSize () const noexcept

  Get the size of the quantization block.
- void setBlockShape (Dims const &blockShape) noexcept

  Set the shape of the quantization block.
- Dims getBlockShape () const noexcept

  Get the shape of the quantization block.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~IDynamicQuantizeLayer () noexcept=0

Protected Attributes

- apiv::VDynamicQuantizeLayer * mImpl

9.90.1 Detailed Description

A network layer to perform dynamic quantization.

This layer accepts a floating-point input tensor and computes the block scale factors needed to quantize the input's data. It outputs the quantized tensor as its first output and the scale factors as its second output.

Use ILayer::setInput to add an input for the double-quantization scale factor.

Note

Only symmetric quantization is supported.

The input tensor for this layer must not be a scalar.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.90.2 Constructor & Destructor Documentation

9.90.2.1 ~IDynamicQuantizeLayer()

```
nvinfer1::IDynamicQuantizeLayer::~IDynamicQuantizeLayer ( )  [inline], [protected], [pure virtual],
[default], [noexcept]
```

9.90.3 Member Function Documentation

9.90.3.1 getAxis()

```
TRT_DEPRECATED int32_t nvinfer1::IDynamicQuantizeLayer::getAxis ( ) const  [inline], [noexcept]
```

Get the axis along which blocking occurs.

See also

setAxis()

9.90.3.2 getBlockShape()

```
Dims nvinfer1::IDynamicQuantizeLayer::getBlockShape ( ) const  [inline], [noexcept]
```

Get the shape of the quantization block.

The default value is empty Dims.

See also

setBlockShape()

9.90.3.3 getBlockSize()

```
TRT_DEPRECATED int32_t nvinfer1::IDynamicQuantizeLayer::getBlockSize ( ) const  [inline], [noexcept]
```

Get the size of the quantization block.

See also

setBlockSize()

9.90.3.4 getScaleType()

```
DataType nvinfer1::IDynamicQuantizeLayer::getScaleType ( ) const  [inline], [noexcept]
```

Return the scale factors data type.

Returns

scaleType parameter set during layer creation or by setScaleType().

The return value is the type of the scale factors used to quantize the dynamic data. The default value is DataType::kFP8.

9.90.3.5 getToType()

```
DataType nvinfer1::IDynamicQuantizeLayer::getToType ( ) const  [inline], [noexcept]
```

Return DynamicQuantizeLayer's quantized output type.

Returns

toType parameter set during layer creation or by setToType().

The return value is the type of the quantized output tensor. The default value is DataType::kFP4.

9.90.3.6 setAxis()

```
TRT_DEPRECATED void nvinfer1::IDynamicQuantizeLayer::setAxis (
            int32_t axis )  [inline], [noexcept]
```

Set the axis along which block quantization occurs.

The axis must be the last dimension or second to last dimension. The input's shape along the axis must be constant.

See also

getAxis()

9.90.3.7 setBlockShape()

```
void nvinfer1::IDynamicQuantizeLayer::setBlockShape (
            Dims const & blockShape )  [inline], [noexcept]
```

Set the shape of the quantization block.

Note: The block shape rank must match the input rank. The default value is empty Dims.

See also

getBlockShape()

9.90.3.8 setBlockSize()

```
TRT_DEPRECATED void nvinfer1::IDynamicQuantizeLayer::setBlockSize (
            int32_t size )  [inline], [noexcept]
```

Set the size of the quantization block.

Note: The block size must divide the input in the blocked axis without remainder. Valid values are 16 (NVFP4 quantization) and 32 (MXFP8 quantization).

See also

getBlockSize()

9.90.3.9 setInput()

```
void nvinfer1::ILayer::setInput (
            int32_t index,
            ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor

Input 0 is the input activation tensor. Input 1 is the double-quantization scale factor. This scale is used to quantize the dynamically computed high-precision scale factors that are used to quantize the activation data. Currently this input

must be a positive scalar (a 0D tensor).

9.90.3.10 setScaleType()

```
void nvinfer1::IDynamicQuantizeLayer::setScaleType (
            DataType scaleType )  [inline], [noexcept]
```

Set the data type of the scale factors used to quantize the data.

Parameters

scaleType	The scale factors data type.

Set the scale-factors type. Valid values are DataType::kFP8, DataType::kE8M0 or DataType::kFLOAT.

9.90.3.11 setToType()

```
void nvinfer1::IDynamicQuantizeLayer::setToType (
            DataType toType )  [inline], [noexcept]
```

Set DynamicQuantizeLayer's quantized output type.

Parameters

toType	The data type of the quantized output tensor.

Set the type of the dynamic quantization layer's quantized output.If the network is strongly typed, setToType must be used to set the output type, and use of setOutputType is an error. Otherwise, types passed to setOutputType and setToType must be the same. Valid values for `toType` are DataType::kFP4 (NVFP4 quantization) and DataType::kFP8 (MXFP8 quantization).

See also

NetworkDefinitionCreationFlag::kSTRONGLY_TYPED

9.90.4 Member Data Documentation

9.90.4.1 mImpl

```
apiv::VDynamicQuantizeLayer* nvinfer1::IDynamicQuantizeLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

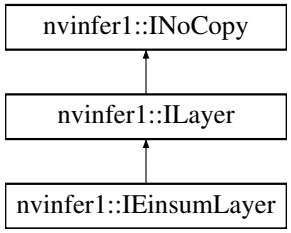
- NvInfer.h

9.91 nvinfer1::IEinsumLayer Class Reference

An Einsum layer in a network.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IEinsumLayer:



Public Member Functions

- bool setEquation (char const *equation) noexcept

  Set the equation. The equation is a comma-separated list of subscript labels, where each label refers to a dimension of the corresponding tensor.
- char const * getEquation () const noexcept

  Return the equation.

Protected Member Functions

- virtual ~IEinsumLayer () noexcept=0

Protected Attributes

- apiv::VEinsumLayer * mImpl

9.91.1 Detailed Description

An Einsum layer in a network.

This layer implements a summation over the elements of the inputs along dimensions specified by the equation parameter, based on the Einstein summation convention. The layer can have one or more inputs of rank >= 0. All the inputs must have type DataType::kFLOAT or DataType::kHALF, not necessarily the same. There is one output of type DataType::kFLOAT. The shape of the output tensor is determined by the equation.

The equation specifies ASCII lower-case letters for each dimension in the inputs in the same order as the dimensions, separated by comma for each input. The dimensions labeled with the same subscript must match or be broadcastable. Repeated subscript labels in one input take the diagonal. Repeating a label across multiple inputs means that those axes will be multiplied. Omitting a label from the output means values along those axes will be summed. In implicit mode, the indices which appear once in the expression will be part of the output in increasing alphabetical order. In explicit mode, the output can be controlled by specifying output subscript labels by adding an arrow ('->') followed by subscripts for the output. For example, "ij,jk->ik" is equivalent to "ij,jk". Ellipsis ('...') can be used in place of subscripts to broadcast the dimensions. See the TensorRT Developer Guide for more details on equation syntax.

Many common operations can be expressed using the Einsum equation. For example: Matrix Transpose: ij->ji Sum: ij-> Matrix-Matrix Multiplication: ik,kj->ij Dot Product: i,i-> Matrix-Vector Multiplication: ik,k->i Batch Matrix Multiplication: ijk,ikl->ijl Batch Diagonal: ...ii->...i

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.91.2 Constructor & Destructor Documentation

9.91.2.1 ~IEinsumLayer()

```
nvinfer1::IEinsumLayer::~IEinsumLayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.91.3 Member Function Documentation

9.91.3.1 getEquation()

```
char const * nvinfer1::IEinsumLayer::getEquation ( ) const  [inline], [noexcept]
```

Return the equation.

See also

setEquation()

9.91.3.2 setEquation()

```
bool nvinfer1::IEinsumLayer::setEquation (
            char const * equation )  [inline], [noexcept]
```

Set the equation. The equation is a comma-separated list of subscript labels, where each label refers to a dimension of the corresponding tensor.

Returns

true if the equation was syntactically valid and set successfully, false otherwise.

See also

setEquation()

9.91.4 Member Data Documentation

9.91.4.1 mImpl

```
apiv::VEinsumLayer* nvinfer1::IEinsumLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

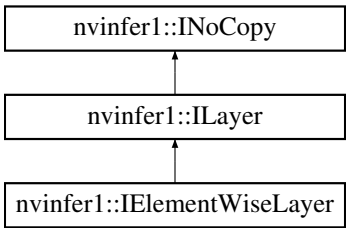
- NvInfer.h

9.92 nvinfer1::IElementWiseLayer Class Reference

A elementwise layer in a network definition.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IElementWiseLayer:



Public Member Functions

- void setOperation (ElementWiseOperation op) noexcept
  Set the binary operation for the layer.
- ElementWiseOperation getOperation () const noexcept
  Get the binary operation for the layer.

Protected Member Functions

- virtual ~IElementWiseLayer () noexcept=0

Protected Attributes

- apiv::VElementWiseLayer * mImpl

9.92.1 Detailed Description

A elementwise layer in a network definition.

This layer applies a per-element binary operation between corresponding elements of two tensors.

The input tensors must have the same rank. For each dimension, their lengths must match, or one of them must be one. In the latter case, the tensor is broadcast along that axis.

The output tensor has the same rank as the inputs. For each output dimension, its length is equal to the lengths of the corresponding input dimensions if they match, otherwise it is equal to the length that is not one.

Warning

When running this layer on the DLA with Int8 data type, the dynamic ranges of two input tensors shall be equal. If the dynamic ranges are generated using calibrator, the largest value shall be used.

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.92.2 Constructor & Destructor Documentation

9.92.2.1 ~IElementWiseLayer()

```
nvinfer1::IElementWiseLayer::~IElementWiseLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.92.3 Member Function Documentation

9.92.3.1 getOperation()

```
ElementWiseOperation nvinfer1::IElementWiseLayer::getOperation ( ) const  [inline], [noexcept]
```

Get the binary operation for the layer.

See also

setOperation(), ElementWiseOperation

setBiasWeights()

9.92.3.2 setOperation()

```
void nvinfer1::IElementWiseLayer::setOperation (
            ElementWiseOperation op )  [inline], [noexcept]
```

Set the binary operation for the layer.

DLA supports only kSUM, kPROD, kMAX, kMIN, and kSUB.

See also

getOperation(), ElementWiseOperation

getBiasWeights()

9.92.4 Member Data Documentation

9.92.4.1 mImpl

```
apiv::VElementWiseLayer* nvinfer1::IElementWiseLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

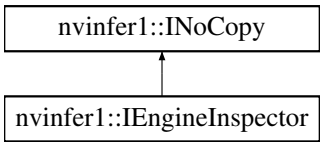
- NvInfer.h

9.93 nvinfer1::IEngineInspector Class Reference

An engine inspector which prints out the layer information of an engine or an execution context.

```
#include <NvInferRuntime.h>
```

Inheritance diagram for nvinfer1::IEngineInspector:



Public Member Functions

- virtual ~IEngineInspector () noexcept=0
- bool setExecutionContext (IExecutionContext const *context) noexcept

  Set an execution context as the inspection source.
- IExecutionContext const * getExecutionContext () const noexcept

  Get the context currently being inspected.
- char const * getLayerInformation (int32_t layerIndex, LayerInformationFormat format) const noexcept

  Get a string describing the information about a specific layer in the current engine or the execution context.
- char const * getEngineInformation (LayerInformationFormat format) const noexcept

  Get a string describing the information about all the layers in the current engine or the execution context.
- void setErrorRecorder (IErrorRecorder *recorder) noexcept

  Set the ErrorRecorder for this interface.
- IErrorRecorder * getErrorRecorder () const noexcept

  Get the ErrorRecorder assigned to this interface.

Protected Attributes

- apiv::VEngineInspector * mImpl

Additional Inherited Members

9.93.1 Detailed Description

An engine inspector which prints out the layer information of an engine or an execution context.

The amount of printed information depends on the profiling verbosity setting of the builder config when the engine is built:

- ProfilingVerbosity::kLAYER_NAMES_ONLY: only layer names will be printed.
- ProfilingVerbosity::kNONE: no layer information will be printed.
- ProfilingVerbosity::kDETAILED: layer names and layer parameters will be printed.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

See also

ProfilingVerbosity, IEngineInspector

9.93.2 Constructor & Destructor Documentation

9.93.2.1 ~IEngineInspector()

```
nvinfer1::IEngineInspector::~IEngineInspector ( )  [inline], [pure virtual], [default], [noexcept]
```

9.93.3 Member Function Documentation

9.93.3.1 getEngineInformation()

```
char const * nvinfer1::IEngineInspector::getEngineInformation (
            LayerInformationFormat format ) const  [inline], [noexcept]
```

Get a string describing the information about all the layers in the current engine or the execution context.

Parameters

format	the format the layer information should be printed in.

Returns

A null-terminated C-style string describing the information about all the layers in the current engine or the execution context.

Warning

The content of the returned string may change when another execution context has been set, or when another getLayerInformation() or getEngineInformation() has been called.

In a multi-threaded environment, this function must be protected from other threads changing the inspection source. If the inspection source changes, the data that is being pointed to can change. Copy the string to another buffer before releasing the lock in order to guarantee consistency.

See also

LayerInformationFormat

9.93.3.2 getErrorRecorder()

```
IErrorRecorder * nvinfer1::IEngineInspector::getErrorRecorder ( ) const  [inline], [noexcept]
```

Get the ErrorRecorder assigned to this interface.

Retrieves the assigned error recorder object for the given class. A nullptr will be returned if an error handler has not been set.

Returns

A pointer to the IErrorRecorder object that has been registered.

See also

setErrorRecorder()

9.93.3.3 getExecutionContext()

```
IExecutionContext const * nvinfer1::IEngineInspector::getExecutionContext ( ) const  [inline],
[noexcept]
```

Get the context currently being inspected.

Returns

The pointer to the context currently being inspected.

See also

setExecutionContext()

9.93.3.4 getLayerInformation()

```
char const * nvinfer1::IEngineInspector::getLayerInformation (
            int32_t layerIndex,
            LayerInformationFormat format ) const  [inline], [noexcept]
```

Get a string describing the information about a specific layer in the current engine or the execution context.

Parameters

layerIndex	the index of the layer. It must lie in range [0, engine.getNbLayers()).
format	the format the layer information should be printed in.

Returns

A null-terminated C-style string describing the information about a specific layer in the current engine or the execution context.

Warning

The content of the returned string may change when another execution context has been set, or when another getLayerInformation() or getEngineInformation() has been called.

In a multi-threaded environment, this function must be protected from other threads changing the inspection source. If the inspection source changes, the data that is being pointed to can change. Copy the string to another buffer before releasing the lock in order to guarantee consistency.

See also

LayerInformationFormat

9.93.3.5 setErrorRecorder()

```
void nvinfer1::IEngineInspector::setErrorRecorder (
            IErrorRecorder * recorder )  [inline], [noexcept]
```

Set the ErrorRecorder for this interface.

Assigns the ErrorRecorder to this interface. The ErrorRecorder will track all errors during execution. This function will call incRefCount of the registered ErrorRecorder at least once. Setting recorder to nullptr unregisters the recorder with the interface, resulting in a call to decRefCount if a recorder has been registered.

If an error recorder is not set, messages will be sent to the global log stream.

Parameters

recorder	The error recorder to register with this interface.

See also

getErrorRecorder()

9.93.3.6 setExecutionContext()

```
bool nvinfer1::IEngineInspector::setExecutionContext (
            IExecutionContext const * context )  [inline], [noexcept]
```

Set an execution context as the inspection source.

Setting the execution context and specifying all the input shapes allows the inspector to calculate concrete dimensions for any dynamic shapes and display their format information. Otherwise, values dependent on input shapes will be displayed as -1 and format information will not be shown.

Passing nullptr will remove any association with an execution context.

Returns

Whether the action succeeds.

9.93.4 Member Data Documentation

9.93.4.1 mImpl

```
apiv::VEngineInspector* nvinfer1::IEngineInspector::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.94 IErrorRecorder Class Reference

Reference counted application-implemented error reporting interface for TensorRT objects.

```
#include <NvInferRuntimeBase.h>
```

9.94.1 Detailed Description

Reference counted application-implemented error reporting interface for TensorRT objects.

The error reporting mechanism is a user-defined object that interacts with the internal state of the object that it is assigned to in order to determine information about abnormalities in execution. The error recorder gets both an error enum that is more descriptive than pass/fail and also a string description that gives more detail on the exact failure modes. In the safety context, the error strings are all limited to 128 bytes or less in length, including the NULL terminator.

The ErrorRecorder gets passed along to any class that is created from another class that has an ErrorRecorder assigned to it. For example, assigning an ErrorRecorder to an IBuilder allows all INetwork's, ILayer's, and ITensor's to use the same error recorder. For functions that have their own ErrorRecorder accessor functions. This allows registering a different error recorder or de-registering of the error recorder for that specific object.

ErrorRecorder objects that are used in the safety runtime must define an implementation-dependent upper limit of errors whose information can be stored, and drop errors above this upper limit. The limit must fit in int32_t. The IErrorRecorder::hasOverflowed() method is used to signal that one or more errors have been dropped.

The ErrorRecorder object implementation must be thread safe. All locking and synchronization is pushed to the interface implementation and TensorRT does not hold any synchronization primitives when calling the interface functions.

The lifetime of the ErrorRecorder object must exceed the lifetime of all TensorRT objects that use it.

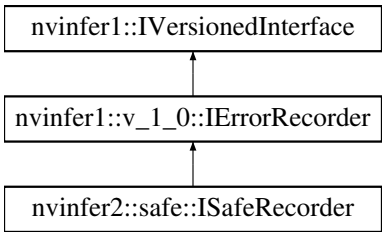
The documentation for this class was generated from the following file:

- NvInferRuntimeBase.h

9.95 nvinfer1::v_1_0::IErrorRecorder Class Reference

`#include <NvInferRuntimeBase.h>`

Inheritance diagram for nvinfer1::v_1_0::IErrorRecorder:



Public Types

- using ErrorDesc = char const *

  A typedef of a C-style string for reporting error descriptions.
- using RefCount = int32_t

  A typedef of a 32-bit integer for reference counting.

Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- IErrorRecorder ()=default
- ~IErrorRecorder () noexcept override=default
- virtual int32_t getNbErrors () const noexcept=0

  Return the number of errors.
- virtual ErrorCode getErrorCode (int32_t errorIdx) const noexcept=0

  Returns the ErrorCode enumeration.
- virtual ErrorDesc getErrorDesc (int32_t errorIdx) const noexcept=0

  Returns a null-terminated C-style string description of the error.
- virtual bool hasOverflowed () const noexcept=0

  Determine if the error stack has overflowed.
- virtual void clear () noexcept=0

  Clear the error stack on the error recorder.
- virtual bool reportError (ErrorCode val, ErrorDesc desc) noexcept=0

  Report an error to the error recorder with the corresponding enum and description.
- virtual RefCount incRefCount () noexcept=0

  Increments the refcount for the current ErrorRecorder.
- virtual RefCount decRefCount () noexcept=0

  Decrements the refcount for the current ErrorRecorder.

Static Public Attributes

- static constexpr size_t kMAX_DESC_LENGTH {127U}

  The length limit for an error description in bytes, excluding the '\0' string terminator. Only applicable to safe runtime. General error recorder implementation can use any size appropriate for the use case.

Additional Inherited Members

9.95.1 Member Typedef Documentation

9.95.1.1 ErrorDesc

```
using nvinfer1::v_1_0::IErrorRecorder::ErrorDesc = char const*
```

A typedef of a C-style string for reporting error descriptions.

9.95.1.2 RefCount

```
using nvinfer1::v_1_0::IErrorRecorder::RefCount = int32_t
```

A typedef of a 32-bit integer for reference counting.

9.95.2 Constructor & Destructor Documentation

9.95.2.1 IErrorRecorder()

```
nvinfer1::v_1_0::IErrorRecorder::IErrorRecorder ( )  [default]
```

9.95.2.2 ~IErrorRecorder()

```
nvinfer1::v_1_0::IErrorRecorder::~IErrorRecorder ( )  [override], [default], [noexcept]
```

9.95.3 Member Function Documentation

9.95.3.1 clear()

```
virtual void nvinfer1::v_1_0::IErrorRecorder::clear ( )  [pure virtual], [noexcept]
```

Clear the error stack on the error recorder.

Removes all the tracked errors by the error recorder. The implementation must guarantee that after this function is called, and as long as no error occurs, the next call to getNbErrors will return zero and hasOverflowed will return false.

See also

getNbErrors(), hasOverflowed()

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.95.3.2 decRefCount()

```
virtual RefCount nvinfer1::v_1_0::IErrorRecorder::decRefCount ( )  [pure virtual], [noexcept]
```

Decrements the refcount for the current ErrorRecorder.

Decrements the reference count for the object by one and returns the current value. This reference count allows the application to know that an object inside of TensorRT has taken a reference to the ErrorRecorder. TensorRT guarantees that every call to IErrorRecorder::decRefCount() will be preceded by a call to IErrorRecorder::incRefCount(). It is undefined behavior to destruct the ErrorRecorder when incRefCount() has been called without a corresponding decRefCount().

Returns

The reference counted value after the decrement completes.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.95.3.3 getErrorCode()

```
virtual ErrorCode nvinfer1::v_1_0::IErrorRecorder::getErrorCode (
            int32_t errorIdx ) const  [pure virtual], [noexcept]
```

Returns the ErrorCode enumeration.

Parameters

errorIdx	A 32-bit integer that indexes into the error array.

The errorIdx specifies what error code from 0 to getNbErrors()-1 that the application wants to analyze and return the error code enum.

Returns

Returns the enum corresponding to errorIdx if errorIdx is in range (between 0 and getNbErrors()-1). ErrorCode::kUNSPECIFIED_ERROR must be returned if errorIdx is not in range.

See also

getErrorDesc(), ErrorCode

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.95.3.4 getErrorDesc()

```
virtual ErrorDesc nvinfer1::v_1_0::IErrorRecorder::getErrorDesc (
            int32_t errorIdx ) const  [pure virtual], [noexcept]
```

Returns a null-terminated C-style string description of the error.

Parameters

errorIdx	A 32-bit integer that indexes into the error array.

For the error specified by the idx value, return the string description of the error. The error string is a null-terminated C-style string. In the safety context there is a constant length requirement to remove any dynamic memory allocations and the error message will be truncated if it exceeds kMAX_DESC_LENGTH bytes. The format of the string is "<EnumAsStr> - <Description>".

Returns

Returns a string representation of the error along with a description of the error if errorIdx is in range (between 0 and getNbErrors()-1). An empty string will be returned if errorIdx is not in range.

See also

getErrorCode()

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.95.3.5 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IErrorRecorder::getInterfaceInfo ( ) const  [inline], [override], [virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.95.3.6 getNbErrors()

```
virtual int32_t nvinfer1::v_1_0::IErrorRecorder::getNbErrors ( ) const  [pure virtual], [noexcept]
```

Return the number of errors.

Determines the number of errors that occurred between the current point in execution and the last time that the clear() was executed. Due to the possibility of asynchronous errors occurring, a TensorRT API can return correct results, but still register errors with the Error Recorder. The value of getNbErrors() must increment by 1 after each reportError() call until clear() is called, or the maximum number of errors that can be stored is exceeded.

Returns

Returns the number of errors detected, or 0 if there are no errors. If the upper bound of errors that can be stored is exceeded, the upper bound value must be returned.

For example, if the error recorder can store up to 16 error descriptions but reportError() has been called 20 times, getNbErrors() must return 16.

See also

clear(), hasOverflowed()

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.95.3.7 hasOverflowed()

```
virtual bool nvinfer1::v_1_0::IErrorRecorder::hasOverflowed ( ) const  [pure virtual], [noexcept]
```

Determine if the error stack has overflowed.

In the case when the number of errors is large, this function is used to query if one or more errors have been dropped due to lack of storage capacity. This is especially important in the automotive safety case where the internal error handling mechanisms cannot allocate memory.

Returns

true if errors have been dropped due to overflowing the error stack.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.95.3.8 incRefCount()

```
virtual RefCount nvinfer1::v_1_0::IErrorRecorder::incRefCount ( )  [pure virtual], [noexcept]
```

Increments the refcount for the current ErrorRecorder.

Increments the reference count for the object by one and returns the current value. This reference count allows the application to know that an object inside of TensorRT has taken a reference to the ErrorRecorder. TensorRT guarantees that every call to IErrorRecorder::incRefCount() will be paired with a call to IErrorRecorder::decRefCount() when the reference is released. It is undefined behavior to destruct the ErrorRecorder when incRefCount() has been called without a corresponding decRefCount().

Returns

The reference counted value after the increment completes.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.95.3.9 reportError()

```
virtual bool nvinfer1::v_1_0::IErrorRecorder::reportError (
            ErrorCode val,
            ErrorDesc desc )  [pure virtual], [noexcept]
```

Report an error to the error recorder with the corresponding enum and description.

Parameters

val	The error code enum that is being reported.
desc	The string description of the error, which will be a NULL-terminated string. For safety use cases its length is limited to kMAX_DESC_LENGTH bytes (excluding the NULL terminator) and descriptions that exceed this limit will be silently truncated.

Report an error to the user that has a given value and human readable description. The function returns false if processing can continue, which implies that the reported error is not fatal. This does not guarantee that processing continues, but provides a hint to TensorRT. The desc C-string data is only valid during the call to reportError and may be immediately deallocated by the caller when reportError returns. The implementation must not store the desc pointer in the ErrorRecorder object or otherwise access the data from desc after reportError returns.

Returns

True if the error is determined to be fatal and processing of the current function must end.

Warning

If the error recorder's maximum number of storable errors is exceeded, the error description will be silently dropped and the value returned by getNbErrors() will not be incremented. However, the return value will still signal whether the error must be considered fatal.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.95.4 Member Data Documentation

9.95.4.1 kMAX_DESC_LENGTH

```
constexpr size_t nvinfer1::v_1_0::IErrorRecorder::kMAX_DESC_LENGTH {127U}  [static], [constexpr]
```

The length limit for an error description in bytes, excluding the '\0' string terminator. Only applicable to safe runtime. General error recorder implementation can use any size appropriate for the use case.

The documentation for this class was generated from the following file:

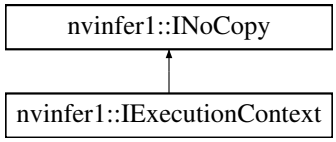
- NvInferRuntimeBase.h

9.96 nvinfer1::IExecutionContext Class Reference

Context for executing inference using an engine, with functionally unsafe features.

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::IExecutionContext:



Public Member Functions

- virtual ~IExecutionContext () noexcept=0
- void setDebugSync (bool sync) noexcept

  Set the debug sync flag.
- bool getDebugSync () const noexcept

  Get the debug sync flag.
- void setProfiler (IProfiler *profiler) noexcept

  Set the profiler.
- IProfiler * getProfiler () const noexcept

  Get the profiler.
- ICudaEngine const & getEngine () const noexcept

  Get the associated engine.
- void setName (char const *name) noexcept

  Set the name of the execution context.
- char const * getName () const noexcept

  Return the name of the execution context.
- void setDeviceMemory (void *memory) noexcept

  Set the device memory for use by this execution context.
- void setDeviceMemoryV2 (void *memory, int64_t size) noexcept

  Set the device memory and its corresponding size for use by this execution context.
- Dims getTensorStrides (char const *tensorName) const noexcept

  Return the strides of the buffer for the given tensor name.
- int32_t getOptimizationProfile () const noexcept

  Get the index of the currently selected optimization profile.
- bool setInputShape (char const *tensorName, Dims const &dims) noexcept

  Set shape of given input.
- Dims getTensorShape (char const *tensorName) const noexcept

  Return the shape of the given input or output.
- bool allInputDimensionsSpecified () const noexcept

  Whether all dynamic dimensions of input tensors have been specified.
- void setErrorRecorder (IErrorRecorder *recorder) noexcept

  Set the ErrorRecorder for this interface.

- IErrorRecorder * getErrorRecorder () const noexcept

  Get the ErrorRecorder assigned to this interface.
- bool executeV2 (void *const *bindings) noexcept

  Synchronously execute a network.
- bool setOptimizationProfileAsync (int32_t profileIndex, cudaStream_t stream) noexcept

  Select an optimization profile for the current context with async semantics.
- void setEnqueueEmitsProfile (bool enqueueEmitsProfile) noexcept

  Set whether enqueue emits layer timing to the profiler.
- bool getEnqueueEmitsProfile () const noexcept

  Get the enqueueEmitsProfile state.
- bool reportToProfiler () const noexcept

  Calculate layer timing info for the current optimization profile in IExecutionContext and update the profiler after one iteration of inference launch.
- bool setTensorAddress (char const *tensorName, void *data) noexcept

  Set memory address for given input or output tensor.
- void const * getTensorAddress (char const *tensorName) const noexcept

  Get memory address bound to given input or output tensor, or nullptr if the provided name does not map to an input or output tensor.
- bool setOutputTensorAddress (char const *tensorName, void *data) noexcept

  Set the memory address for a given output tensor.
- bool setInputTensorAddress (char const *tensorName, void const *data) noexcept

  Set memory address for given input.
- void * getOutputTensorAddress (char const *tensorName) const noexcept

  Get memory address for given output.
- int32_t inferShapes (int32_t nbMaxNames, char const **tensorNames) noexcept

  Run shape calculations.
- size_t updateDeviceMemorySizeForShapes () noexcept

  Recompute the internal activation buffer sizes based on the current input shapes, and return the total amount of memory required.
- bool setInputConsumedEvent (cudaEvent_t event) noexcept

  Mark input as consumed.
- cudaEvent_t getInputConsumedEvent () const noexcept

  The event associated with consuming the input.
- bool setOutputAllocator (char const *tensorName, IOutputAllocator *outputAllocator) noexcept

  Set output allocator to use for output tensor of given name. Pass nullptr to outputAllocator to unset. The allocator is called by enqueueV3().
- IOutputAllocator * getOutputAllocator (char const *tensorName) const noexcept

  Get output allocator associated with output tensor of given name, or nullptr if the provided name does not map to an output tensor.
- int64_t getMaxOutputSize (char const *tensorName) const noexcept

  Get upper bound on an output tensor's size, in bytes, based on the current optimization profile and input dimensions.
- bool setTemporaryStorageAllocator (IGpuAllocator *allocator) noexcept

  Specify allocator to use for internal temporary storage.
- IGpuAllocator * getTemporaryStorageAllocator () const noexcept

  Get allocator set by setTemporaryStorageAllocator.
- bool enqueueV3 (cudaStream_t stream) noexcept

  Enqueue inference on a stream.
- void setPersistentCacheLimit (size_t size) noexcept

Set the maximum size for persistent cache usage.

- size_t getPersistentCacheLimit () const noexcept

  Get the maximum size for persistent cache usage.

- bool setNvtxVerbosity (ProfilingVerbosity verbosity) noexcept

  Set the verbosity of the NVTX markers in the execution context.

- ProfilingVerbosity getNvtxVerbosity () const noexcept

  Get the NVTX verbosity of the execution context.

- void setAuxStreams (cudaStream_t *auxStreams, int32_t nbStreams) noexcept

  Set the auxiliary streams that TensorRT should launch kernels on in the next enqueueV3() call.

- bool setDebugListener (IDebugListener *listener) noexcept

  Set DebugListener for this execution context.

- IDebugListener * getDebugListener () noexcept

  Get the DebugListener of this execution context.

- bool setTensorDebugState (char const *name, bool flag) noexcept

  Set debug state of tensor given the tensor name.

- bool getDebugState (char const *name) const noexcept

  Get the debug state.

- IRuntimeConfig * getRuntimeConfig () const noexcept

  Get the runtime config object used during execution context creation.

- bool setAllTensorsDebugState (bool flag) noexcept

  Turn the debug state of all debug tensors on or off.

- bool setUnfusedTensorsDebugState (bool flag) noexcept

  Turn the debug state of unfused tensors on or off.

- bool getUnfusedTensorsDebugState () const noexcept

  Get the debug state of unfused tensors.

- bool setCommunicator (void *communicator) noexcept

  Set the NCCL communicator for the execution context.

Protected Attributes

- apiv::VExecutionContext * mImpl

Additional Inherited Members

9.96.1 Detailed Description

Context for executing inference using an engine, with functionally unsafe features.

Multiple execution contexts may exist for one ICudaEngine instance, allowing the same engine to be used for the execution of multiple batches simultaneously. If the engine supports dynamic shapes, each execution context in concurrent use must use a separate optimization profile.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.96.2 Constructor & Destructor Documentation

9.96.2.1 ~IExecutionContext()

```
nvinfer1::IExecutionContext::~IExecutionContext ( )  [inline], [pure virtual], [default], [noexcept]
```

9.96.3 Member Function Documentation

9.96.3.1 allInputDimensionsSpecified()

```
bool nvinfer1::IExecutionContext::allInputDimensionsSpecified ( ) const  [inline], [noexcept]
```

Whether all dynamic dimensions of input tensors have been specified.

Returns

True if all dynamic dimensions of input tensors have been specified by calling setInputShape().

Trivially true if network has no dynamically shaped input tensors.

Does not work with name-base interfaces eg. IExecutionContext::setInputShape(). Use IExecutionContext::inferShapes() instead.

9.96.3.2 enqueueV3()

```
bool nvinfer1::IExecutionContext::enqueueV3 (
            cudaStream_t stream )  [inline], [noexcept]
```

Enqueue inference on a stream.

Parameters

stream	A CUDA stream on which the inference kernels will be enqueued.

Returns

True if the kernels were enqueued successfully, false otherwise.

Modifying or releasing memory that has been registered for the tensors before stream synchronization or the event

passed to setInputConsumedEvent has been being triggered results in undefined behavior. Input tensor can be released after the setInputConsumedEvent whereas output tensors require stream synchronization.

Warning

Using default stream may lead to performance issues due to additional cudaDeviceSynchronize() calls by TensorRT to ensure correct synchronizations. Please use non-default stream instead.

If the Engine is streaming weights, enqueueV3 will become synchronous, and the graph will not be capturable.

9.96.3.3 executeV2()

```
bool nvinfer1::IExecutionContext::executeV2 (
            void *const * bindings )  [inline], [noexcept]
```

Synchronously execute a network.

This method requires an array of input and output buffers. The mapping from indices to tensor names can be queried using ICudaEngine::getIOTensorName().

Parameters

bindings	An array of pointers to input and output buffers for the network.

Returns

True if execution succeeded.

See also

ICudaEngine::getIOTensorName()

9.96.3.4 getDebugListener()

```
IDebugListener * nvinfer1::IExecutionContext::getDebugListener ( )  [inline], [noexcept]
```

Get the DebugListener of this execution context.

Returns

DebugListener of this execution context.

9.96.3.5 getDebugState()

```
bool nvinfer1::IExecutionContext::getDebugState (
            char const * name ) const  [inline], [noexcept]
```

Get the debug state.

Parameters

name	Name of target tensor.

Returns

true if there is a debug tensor with the given name and it has debug state turned on.

9.96.3.6 getDebugSync()

```
bool nvinfer1::IExecutionContext::getDebugSync ( ) const  [inline], [noexcept]
```

Get the debug sync flag.

See also

setDebugSync()

9.96.3.7 getEngine()

```
ICudaEngine const & nvinfer1::IExecutionContext::getEngine ( ) const  [inline], [noexcept]
```

Get the associated engine.

See also

ICudaEngine

9.96.3.8 getEnqueueEmitsProfile()

```
bool nvinfer1::IExecutionContext::getEnqueueEmitsProfile ( ) const  [inline], [noexcept]
```

Get the enqueueEmitsProfile state.

Returns

The enqueueEmitsProfile state.

See also

IExecutionContext::setEnqueueEmitsProfile()

9.96.3.9 getErrorRecorder()

```
IErrorRecorder * nvinfer1::IExecutionContext::getErrorRecorder ( ) const  [inline], [noexcept]
```

Get the ErrorRecorder assigned to this interface.

Retrieves the assigned error recorder object for the given class. A nullptr will be returned if an error handler has not been set.

Returns

A pointer to the IErrorRecorder object that has been registered.

See also

setErrorRecorder()

9.96.3.10 getInputConsumedEvent()

```
cudaEvent_t nvinfer1::IExecutionContext::getInputConsumedEvent ( ) const  [inline], [noexcept]
```

The event associated with consuming the input.

Returns

The CUDA event. Nullptr will be returned if the event is not set yet.

9.96.3.11 getMaxOutputSize()

```
int64_t nvinfer1::IExecutionContext::getMaxOutputSize (
            char const * tensorName ) const  [inline], [noexcept]
```

Get upper bound on an output tensor's size, in bytes, based on the current optimization profile and input dimensions.

If the profile or input dimensions are not yet set, or the provided name does not map to an output, returns -1.

Parameters

tensorName	The name of an output tensor.

Returns

Upper bound in bytes.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

9.96.3.12 getName()

```
char const * nvinfer1::IExecutionContext::getName ( ) const  [inline], [noexcept]
```

Return the name of the execution context.

See also

setName()

9.96.3.13 getNvtxVerbosity()

```
ProfilingVerbosity nvinfer1::IExecutionContext::getNvtxVerbosity ( ) const  [inline], [noexcept]
```

Get the NVTX verbosity of the execution context.

Returns

The current NVTX verbosity of the execution context.

See also

setNvtxVerbosity()

9.96.3.14 getOptimizationProfile()

```
int32_t nvinfer1::IExecutionContext::getOptimizationProfile ( ) const  [inline], [noexcept]
```

Get the index of the currently selected optimization profile.

If the profile index has not been set yet (implicitly to 0 if no other execution context has been set to profile 0, or explicitly for all subsequent contexts), an invalid value of -1 will be returned and all calls to enqueueV3()/executeV2() will fail until a valid profile index has been set. This behavior is deprecated in TensorRT 8.6, all profiles will default to optimization profile 0 and -1 will no longer be returned.

9.96.3.15 getOutputAllocator()

```
IOutputAllocator * nvinfer1::IExecutionContext::getOutputAllocator (
            char const * tensorName ) const  [inline], [noexcept]
```

Get output allocator associated with output tensor of given name, or nullptr if the provided name does not map to an output tensor.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

See also

IOutputAllocator

9.96.3.16 getOutputTensorAddress()

```
void * nvinfer1::IExecutionContext::getOutputTensorAddress (
            char const * tensorName ) const  [inline], [noexcept]
```

Get memory address for given output.

Parameters

tensorName	The name of an output tensor.

Returns

Raw output data pointer (void∗) for given output tensor, or nullptr if the provided name does not map to an output tensor.

If only a (void const∗) pointer is needed, an alternative is to call method getTensorAddress().

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

See also

getTensorAddress()

9.96.3.17 getPersistentCacheLimit()

```
size_t nvinfer1::IExecutionContext::getPersistentCacheLimit ( ) const  [inline], [noexcept]
```

Get the maximum size for persistent cache usage.

Returns

The size of the persistent cache limit

See also

setPersistentCacheLimit

9.96.3.18 getProfiler()

```
IProfiler * nvinfer1::IExecutionContext::getProfiler ( ) const  [inline], [noexcept]
```

Get the profiler.

See also

IProfiler setProfiler()

9.96.3.19 getRuntimeConfig()

```
IRuntimeConfig * nvinfer1::IExecutionContext::getRuntimeConfig ( ) const  [inline], [noexcept]
```

Get the runtime config object used during execution context creation.

Returns

The runtime config object.

9.96.3.20 getTemporaryStorageAllocator()

```
IGpuAllocator * nvinfer1::IExecutionContext::getTemporaryStorageAllocator ( ) const  [inline],
[noexcept]
```

Get allocator set by setTemporaryStorageAllocator.

Returns a nullptr if a nullptr was passed with setTemporaryStorageAllocator().

9.96.3.21 getTensorAddress()

```
void const * nvinfer1::IExecutionContext::getTensorAddress (
            char const * tensorName ) const  [inline], [noexcept]
```

Get memory address bound to given input or output tensor, or nullptr if the provided name does not map to an input or output tensor.

Parameters

tensorName	The name of an input or output tensor.

Use method getOutputTensorAddress() if a non-const pointer for an output tensor is required.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

See also

getOutputTensorAddress()

9.96.3.22 getTensorShape()

```
Dims nvinfer1::IExecutionContext::getTensorShape (
            char const * tensorName ) const  [inline], [noexcept]
```

Return the shape of the given input or output.

Parameters

tensorName	The name of an input or output tensor.

Return Dims{-1, {}} if the provided name does not map to an input or output tensor. Otherwise return the shape of the input or output tensor.

A dimension in an input tensor will have a -1 wildcard value if all the following are true:

- setInputShape() has not yet been called for this tensor
- The dimension is a runtime dimension that is not implicitly constrained to be a single value.

A dimension in an output tensor will have a -1 wildcard value if the dimension depends on values of execution tensors OR if all the following are true:

- It is a runtime dimension.
- setInputShape() has NOT been called for some input tensor(s) with a runtime shape.
- setTensorAddress() has NOT been called for some input tensor(s) with isShapeInferenceIO() = true.

An output tensor may also have -1 wildcard dimensions if its shape depends on values of tensors supplied to enqueueV3().

If the request is for the shape of an output tensor with runtime dimensions, all input tensors with isShapeInferenceIO() = true should have their value already set, since these values might be needed to compute the output shape.

Examples of an input dimension that is implicitly constrained to a single value:

- The optimization profile specifies equal min and max values.
- The dimension is named and only one value meets the optimization profile requirements for dimensions with that name.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

9.96.3.23 getTensorStrides()

```
Dims nvinfer1::IExecutionContext::getTensorStrides (
            char const * tensorName ) const  [inline], [noexcept]
```

Return the strides of the buffer for the given tensor name.

The strides are in units of elements, not components or bytes. For example, for TensorFormat::kHWC8, a stride of one spans 8 scalars.

Note that strides can be different for different execution contexts with dynamic shapes.

If the provided name does not map to an input or output tensor, or there are dynamic dimensions that have not been set yet, return Dims{-1, {}}

Parameters

tensorName	The name of an input or output tensor.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

9.96.3.24 getUnfusedTensorsDebugState()

```
bool nvinfer1::IExecutionContext::getUnfusedTensorsDebugState ( ) const  [inline], [noexcept]
```

Get the debug state of unfused tensors.

Returns

true if unfused tensors debug state is on. False if unfused tensors debug state is off.

9.96.3.25 inferShapes()

```
int32_t nvinfer1::IExecutionContext::inferShapes (
            int32_t nbMaxNames,
            char const ** tensorNames )  [inline], [noexcept]
```

Run shape calculations.

Parameters

nbMaxNames	Maximum number of names to write to tensorNames. When the return value is a positive value n and tensorNames != nullptr, the names of min(n,nbMaxNames) insufficiently specified input tensors are written to tensorNames.
tensorNames	Buffer in which to place names of insufficiently specified input tensors.

Returns

0 on success. Positive value n if n input tensors were not sufficiently specified. -1 for other errors.

An input tensor is insufficiently specified if either of the following is true:

- It has dynamic dimensions and its runtime dimensions have not yet been specified via IExecutionContext::setInputShape.
- isShapeInferenceIO(t)=true and the tensor's address has not yet been set.

If an output tensor has isShapeInferenceIO(t)=true and its address has been specified, then its value is written.

Returns -1 if tensorNames == nullptr and nbMaxNames != 0. Returns -1 if nbMaxNames $<$ 0. Returns -1 if a tensor's dimensions are invalid, e.g. a tensor ends up with a negative dimension.

9.96.3.26 reportToProfiler()

```
bool nvinfer1::IExecutionContext::reportToProfiler ( ) const  [inline], [noexcept]
```

Calculate layer timing info for the current optimization profile in IExecutionContext and update the profiler after one iteration of inference launch.

If IExecutionContext::getEnqueueEmitsProfile() returns true, the enqueue function will calculate layer timing implicitly if a profiler is provided. This function returns true and does nothing.

If IExecutionContext::getEnqueueEmitsProfile() returns false, the enqueue function will record the CUDA event timers if a profiler is provided. But it will not perform the layer timing calculation. IExecutionContext::reportToProfiler() needs to be called explicitly to calculate layer timing for the previous inference launch.

In the CUDA graph launch scenario, it will record the same set of CUDA events as in regular enqueue functions if the graph is captured from an IExecutionContext with profiler enabled. This function needs to be called after graph launch to report the layer timing info to the profiler.

Warning

profiling CUDA graphs is only available from CUDA 11.1 onwards.

reportToProfiler uses the stream of the previous enqueue call, so the stream must be live otherwise behavior is undefined.

Returns

true if the call succeeded, else false (e.g. profiler not provided, in CUDA graph capture mode, etc.)

See also

IExecutionContext::setEnqueueEmitsProfile()

IExecutionContext::getEnqueueEmitsProfile()

9.96.3.27 setAllTensorsDebugState()

```
bool nvinfer1::IExecutionContext::setAllTensorsDebugState (
            bool flag )  [inline], [noexcept]
```

Turn the debug state of all debug tensors on or off.

Parameters

flag	true if turning on debug state, false if turning off debug state.

Returns

true if successful, false otherwise.

The default is off.

9.96.3.28 setAuxStreams()

```
void nvinfer1::IExecutionContext::setAuxStreams (
            cudaStream_t * auxStreams,
            int32_t nbStreams )  [inline], [noexcept]
```

Set the auxiliary streams that TensorRT should launch kernels on in the next enqueueV3() call.

If set, TensorRT will launch the kernels that are supposed to run on the auxiliary streams using the streams provided by the user with this API. If this API is not called before the enqueueV3() call, then TensorRT will use the auxiliary streams created by TensorRT internally.

TensorRT will always insert event synchronizations between the main stream provided via enqueueV3() call and the auxiliary streams:

- At the beginning of the enqueueV3() call, TensorRT will make sure that all the auxiliary streams wait on the activities on the main stream.
- At the end of the enqueueV3() call, TensorRT will make sure that the main stream wait on the activities on all the auxiliary streams.

Parameters

auxStreams	The pointer to an array of cudaStream_t with the array length equal to nbStreams.
nbStreams	The number of auxiliary streams provided. If nbStreams is greater than `engine->getNbAuxStreams()`, then only the first `engine->getNbAuxStreams()` streams will be used. If `nbStreams` is less than `engine->getNbAuxStreams()`, such as setting `nbStreams` to 0, then TensorRT will use the provided streams for the first `nbStreams` auxiliary streams, and will create additional streams internally for the rest of the auxiliary streams.

Note

The provided auxiliary streams must not be the default stream and must all be different to avoid deadlocks.

See also

enqueueV3(), IBuilderConfig::setMaxAuxStreams(), ICudaEngine::getNbAuxStreams()

9.96.3.29 setCommunicator()

```
bool nvinfer1::IExecutionContext::setCommunicator (
            void * communicator )  [inline], [noexcept]
```

Set the NCCL communicator for the execution context.

Parameters

communicator	A pointer to the communicator that is used by the execution context. The communicator is expected to be already initialized with `ncclCommInitRank` and castable to `ncclComm_t`.

The communicator must be uniform across all multi-device instances or undefined behavior occurs.

Warning

The lifetime of the communicator must be longer than the execution contexts it is attached to.

Returns

True if the communicator was set successfully, false otherwise.

9.96.3.30 setDebugListener()

```
bool nvinfer1::IExecutionContext::setDebugListener (
            IDebugListener * listener )  [inline], [noexcept]
```

Set DebugListener for this execution context.

Parameters

listener	DebugListener for this execution context.

Returns

true if succeed, false if failure.

9.96.3.31 setDebugSync()

```
void nvinfer1::IExecutionContext::setDebugSync (
            bool sync )  [inline], [noexcept]
```

Set the debug sync flag.

If this flag is set to true, the engine will log the successful execution for each kernel during executeV2(). It has no effect when using enqueueV3().

See also

getDebugSync()

9.96.3.32 setDeviceMemory()

```
void nvinfer1::IExecutionContext::setDeviceMemory (
            void * memory )  [inline], [noexcept]
```

Set the device memory for use by this execution context.

The memory must be aligned with CUDA memory alignment property (using cudaGetDeviceProperties()), and its size must be large enough for performing inference with the given network inputs. getDeviceMemorySizeV2() and getDeviceMemorySizeForProfileV2() report upper bounds of the size. Setting memory to nullptr is acceptable if the reported size is 0. If using enqueueV3() to run the network, the memory is in use from the invocation of enqueueV3() until network execution is complete. If using executeV2(), it is in use until executeV2() returns. Releasing or otherwise using the memory for other purposes, including using it in another execution context running in parallel, during this time will result in undefined behavior.

Deprecated Deprecated in TensorRT 10.1. Superseded by setDeviceMemoryV2().

Warning

Weight streaming related scratch memory will be allocated by TensorRT if the memory is set by this API. Please use setDeviceMemoryV2() instead.

See also

ICudaEngine::getDeviceMemorySizeV2()

ICudaEngine::getDeviceMemorySizeForProfileV2()

ExecutionContextAllocationStrategy

ICudaEngine::createExecutionContext()

9.96.3.33 setDeviceMemoryV2()

```
void nvinfer1::IExecutionContext::setDeviceMemoryV2 (
            void * memory,
            int64_t size )  [inline], [noexcept]
```

Set the device memory and its corresponding size for use by this execution context.

The memory must be aligned with CUDA memory alignment property (using cudaGetDeviceProperties()), and its size must be large enough for performing inference with the given network inputs. getDeviceMemorySizeV2() and getDeviceMemorySizeForProfileV2() report upper bounds of the size. Setting memory to nullptr is acceptable if the reported size is 0. If using enqueueV3() to run the network, the memory is in use from the invocation of enqueueV3() until network execution is complete. If using executeV2(), it is in use until executeV2() returns. Releasing or otherwise using the memory for other purposes, including using it in another execution context running in parallel, during this time will result in undefined behavior.

See also

ICudaEngine::getDeviceMemorySizeV2()

ICudaEngine::getDeviceMemorySizeForProfileV2()

ExecutionContextAllocationStrategy

ICudaEngine::createExecutionContext()

9.96.3.34 setEnqueueEmitsProfile()

```
void nvinfer1::IExecutionContext::setEnqueueEmitsProfile (
            bool enqueueEmitsProfile )  [inline], [noexcept]
```

Set whether enqueue emits layer timing to the profiler.

If set to true (default), enqueue is synchronous and does layer timing profiling implicitly if there is a profiler attached. If set to false, enqueue will be asynchronous if there is a profiler attached. An extra method reportToProfiler() needs to be called to obtain the profiling data and report to the profiler attached.

See also

IExecutionContext::getEnqueueEmitsProfile()

IExecutionContext::reportToProfiler()

9.96.3.35 setErrorRecorder()

```
void nvinfer1::IExecutionContext::setErrorRecorder (
            IErrorRecorder * recorder )  [inline], [noexcept]
```

Set the ErrorRecorder for this interface.

Assigns the ErrorRecorder to this interface. The ErrorRecorder will track all errors during execution. This function will call incRefCount of the registered ErrorRecorder at least once. Setting recorder to nullptr unregisters the recorder with the interface, resulting in a call to decRefCount if a recorder has been registered.

If an error recorder is not set, messages will be sent to the global log stream.

Parameters

recorder	The error recorder to register with this interface.

See also

getErrorRecorder()

9.96.3.36 setInputConsumedEvent()

```
bool nvinfer1::IExecutionContext::setInputConsumedEvent (
            cudaEvent_t event ) [inline], [noexcept]
```

Mark input as consumed.

Parameters

event	The CUDA event that is triggered after all input tensors have been consumed.

Warning

The set event must be valid during the inference.

Returns

True on success, false if error occurred.

Passing event==nullptr removes whatever event was set, if any.

9.96.3.37 setInputShape()

```
bool nvinfer1::IExecutionContext::setInputShape (
            char const * tensorName,
            Dims const & dims ) [inline], [noexcept]
```

Set shape of given input.

Parameters

tensorName	The name of an input tensor.
dims	The shape of an input tensor.

Returns

True on success, false if the provided name does not map to an input tensor, or if some other error occurred.

Each dimension must agree with the network dimension unless the latter was -1.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

9.96.3.38 setInputTensorAddress()

```
bool nvinfer1::IExecutionContext::setInputTensorAddress (
            char const * tensorName,
            void const * data )  [inline], [noexcept]
```

Set memory address for given input.

Parameters

tensorName	The name of an input tensor.
data	The pointer (void const∗) to the const data owned by the user.

Returns

True on success, false if the provided name does not map to an input tensor, does not meet alignment requirements, or some other error occurred.

Input addresses can also be set using method setTensorAddress, which requires a (void∗).

See description of method setTensorAddress() for alignment and data type constraints.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

See also

setTensorAddress()

9.96.3.39 setName()

```
void nvinfer1::IExecutionContext::setName (
            char const * name )  [inline], [noexcept]
```

Set the name of the execution context.

This method copies the name string.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

See also

getName()

9.96.3.40 setNvtxVerbosity()

```
bool nvinfer1::IExecutionContext::setNvtxVerbosity (
            ProfilingVerbosity verbosity )  [inline], [noexcept]
```

Set the verbosity of the NVTX markers in the execution context.

Building with kDETAILED verbosity will generally increase latency in enqueueV3(). Call this method to select NVTX verbosity in this execution context at runtime.

The default is the verbosity with which the engine was built, and the verbosity may not be raised above that level.

This function does not affect how IEngineInspector interacts with the engine.

Parameters

verbosity	The verbosity of the NVTX markers.

Returns

True if the NVTX verbosity is set successfully. False if the provided verbosity level is higher than the profiling verbosity of the corresponding engine.

See also

getNvtxVerbosity()

ICudaEngine::getProfilingVerbosity()

9.96.3.41 setOptimizationProfileAsync()

```
bool nvinfer1::IExecutionContext::setOptimizationProfileAsync (
            int32_t profileIndex,
            cudaStream_t stream )  [inline], [noexcept]
```

Select an optimization profile for the current context with async semantics.

Parameters

profileIndex	Index of the profile. The value must lie between 0 and getEngine().getNbOptimizationProfiles() - 1
stream	A CUDA stream on which the cudaMemcpyAsyncs may be enqueued

When an optimization profile is switched via this API, TensorRT may require that data is copied via cudaMemcpy↩ Async. It is the application's responsibility to guarantee that synchronization between the profile sync stream and the enqueue stream occurs.

The selected profile will be used in subsequent calls to executeV2()/enqueueV3(). If the associated CUDA engine has inputs with dynamic shapes, the optimization profile must be set with its corresponding profileIndex before calling execute or enqueue. The newly created execution context will be assigned optimization profile 0.

If the associated CUDA engine does not have inputs with dynamic shapes, this method need not be called, in which case the default profile index of 0 will be used.

setOptimizationProfileAsync() must be called before calling setInputShape() for all dynamic input tensors or input shape tensors, which in turn must be called before executeV2()/enqueueV3().

Warning

This function will trigger layer resource updates on the next call of executeV2()/enqueueV3(), possibly resulting in performance bottlenecks.

Not synchronizing the stream used at enqueue with the stream used to set optimization profile asynchronously using this API will result in undefined behavior.

Returns

true if the call succeeded, else false (e.g. input out of range)

See also

ICudaEngine::getNbOptimizationProfiles()

9.96.3.42 setOutputAllocator()

```
bool nvinfer1::IExecutionContext::setOutputAllocator (
            char const * tensorName,
            IOutputAllocator * outputAllocator )  [inline], [noexcept]
```

Set output allocator to use for output tensor of given name. Pass nullptr to outputAllocator to unset. The allocator is called by enqueueV3().

Parameters

tensorName	The name of an output tensor.
outputAllocator	IOutputAllocator for the tensors.

Returns

True if success, false if the provided name does not map to an output or, if some other error occurred.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

See also

enqueueV3() IOutputAllocator

9.96.3.43 setOutputTensorAddress()

```
bool nvinfer1::IExecutionContext::setOutputTensorAddress (
            char const * tensorName,
            void * data )  [inline], [noexcept]
```

Set the memory address for a given output tensor.

Parameters

tensorName	The name of an output tensor.
data	The pointer to the buffer to which to write the output.

Returns

True on success, false if the provided name does not map to an output tensor, does not meet alignment requirements, or some other error occurred.

Output addresses can also be set using method setTensorAddress. This method is provided for applications which prefer to use different methods for setting input and output tensors.

See setTensorAddress() for alignment and data type constraints.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

See also

setTensorAddress()

9.96.3.44 setPersistentCacheLimit()

```
void nvinfer1::IExecutionContext::setPersistentCacheLimit (
            size_t size ) [inline], [noexcept]
```

Set the maximum size for persistent cache usage.

This function sets the maximum persistent L2 cache that this execution context may use for activation caching. Activation caching is not supported on all architectures - see "How TensorRT uses Memory" in the developer guide for details

Parameters

size	the size of persistent cache limitation in bytes. The default is 0 Bytes.

See also

getPersistentCacheLimit

9.96.3.45 setProfiler()

```
void nvinfer1::IExecutionContext::setProfiler (
            IProfiler * profiler ) [inline], [noexcept]
```

Set the profiler.

See also

IProfiler getProfiler()

9.96.3.46 setTemporaryStorageAllocator()

```
bool nvinfer1::IExecutionContext::setTemporaryStorageAllocator (
            IGpuAllocator * allocator ) [inline], [noexcept]
```

Specify allocator to use for internal temporary storage.

This allocator is used only by enqueueV3() for temporary storage whose size cannot be predicted ahead of enqueueV3(). It is not used for output tensors, because memory allocation for those is allocated by the allocator set by setOutputAllocator(). All memory allocated is freed by the time enqueueV3() returns.

Parameters

allocator	pointer to allocator to use. Pass nullptr to revert to using TensorRT's default allocator.

Returns

True on success, false if error occurred.

See also

enqueueV3() setOutputAllocator()

9.96.3.47 setTensorAddress()

```
bool nvinfer1::IExecutionContext::setTensorAddress (
            char const * tensorName,
            void * data )  [inline], [noexcept]
```

Set memory address for given input or output tensor.

Parameters

tensorName	The name of an input or output tensor.
data	The pointer (void∗) to the data owned by the user.

Returns

True on success, false if error occurred.

An address defaults to nullptr. Pass data=nullptr to reset to the default state.

Return false if the provided name does not map to an input or output tensor.

If an input pointer has type (void const∗), use setInputTensorAddress() instead.

Before calling enqueueV3(), each input must have a non-null address and each output must have a non-null address or an IOutputAllocator to set it later.

If the TensorLocation of the tensor is kHOST:

- The pointer must point to a host buffer of sufficient size.
- Data representing shape values is not copied until enqueueV3 is invoked.

If the TensorLocation of the tensor is kDEVICE:

- The pointer must point to a device buffer of sufficient size and alignment, or
- Be nullptr if the tensor is an output tensor that will be allocated by IOutputAllocator.

If getTensorShape(name) reports a -1 for any dimension of an output after all input shapes have been set, use setOutputAllocator() to associate an IOutputAllocator to which the dimensions will be reported when known.

Calling both setTensorAddress and setOutputAllocator() for the same output is allowed, and can be useful for preallocating memory, and then reallocating if it's not big enough.

The pointer must have at least 256-byte alignment.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

See also

setInputTensorAddress() setOutputTensorAddress() getTensorShape() setOutputAllocator() IOutputAllocator

9.96.3.48 setTensorDebugState()

```
bool nvinfer1::IExecutionContext::setTensorDebugState (
            char const * name,
            bool flag )  [inline], [noexcept]
```

Set debug state of tensor given the tensor name.

Turn the debug state of a tensor on or off. A tensor with the parameter tensor name must exist in the network, and the tensor must have been marked as a debug tensor during build time. Otherwise, an error is thrown.

Parameters

name	Name of target tensor.
flag	True if turning on debug state, false if turning off debug state of tensor The default is off.

Returns

True if successful, false otherwise.

9.96.3.49 setUnfusedTensorsDebugState()

```
bool nvinfer1::IExecutionContext::setUnfusedTensorsDebugState (
            bool flag )  [inline], [noexcept]
```

Turn the debug state of unfused tensors on or off.

The default is off.

Parameters

flag	true if turning on debug state, false if turning off debug state.

Returns

true if successful, false otherwise.

See also

INetworkDefinition::markUnfusedTensorsAsDebugTensors()

9.96.3.50 updateDeviceMemorySizeForShapes()

```
size_t nvinfer1::IExecutionContext::updateDeviceMemorySizeForShapes ( )  [inline], [noexcept]
```

Recompute the internal activation buffer sizes based on the current input shapes, and return the total amount of memory required.

Users can allocate the device memory based on the size returned and provided the memory to TRT with IExecutionContext::setDeviceMemory(). Must specify all input shapes and the optimization profile to use before calling this function, otherwise the partition will be invalidated.

Returns

Total amount of memory required on success, 0 if error occurred.

See also

IExecutionContext::setDeviceMemory()

9.96.4 Member Data Documentation

9.96.4.1 mImpl

```
apiv::VExecutionContext* nvinfer1::IExecutionContext::mImpl  [protected]
```

The documentation for this class was generated from the following file:

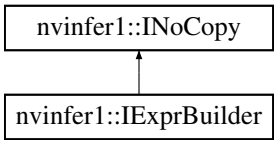
- NvInferRuntime.h

9.97 nvinfer1::IExprBuilder Class Reference

Object for constructing IDimensionExpr.

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::IExprBuilder:



Public Member Functions

- IDimensionExpr const * constant (int64_t value) noexcept

  Return pointer to IDimensionExpr for given value.
- IDimensionExpr const * operation (DimensionOperation op, IDimensionExpr const &first, IDimensionExpr const &second) noexcept

  Get the operation.
- IDimensionExpr const * declareSizeTensor (int32_t outputIndex, IDimensionExpr const &opt, IDimensionExpr const &upper)

  Declare a size tensor at the given output index, with the specified auto-tuning formula and upper bound.

Protected Member Functions

- virtual ~IExprBuilder () noexcept=0

Protected Attributes

- apiv::VExprBuilder * mImpl

9.97.1 Detailed Description

Object for constructing IDimensionExpr.

There is no public way to construct an IExprBuilder. It appears as an argument to method IPluginV2DynamicExt::getOutputDimensions() and IPluginV3OneBuild::getOutputShapes(). Overrides of that method can use that IExprBuilder argument to construct expressions that define output dimensions in terms of input dimensions.

Clients should assume that any values constructed by the IExprBuilder are destroyed after IPluginV2DynamicExt::getOutputDimensions(or IPluginV3OneBuild::getOutputShapes() returns.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

See also

IDimensionExpr

9.97.2 Constructor & Destructor Documentation

9.97.2.1 ~IExprBuilder()

```
nvinfer1::IExprBuilder::~IExprBuilder ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.97.3 Member Function Documentation

9.97.3.1 constant()

```
IDimensionExpr const * nvinfer1::IExprBuilder::constant (
            int64_t value )  [inline], [noexcept]
```

Return pointer to IDimensionExpr for given value.

9.97.3.2 declareSizeTensor()

```
IDimensionExpr const * nvinfer1::IExprBuilder::declareSizeTensor (
            int32_t outputIndex,
            IDimensionExpr const & opt,
            IDimensionExpr const & upper )  [inline]
```

Declare a size tensor at the given output index, with the specified auto-tuning formula and upper bound.

A size tensor allows a plugin to have output dimensions that cannot be computed solely from input dimensions. For example, suppose a plugin implements the equivalent of INonZeroLayer for 2D input. The plugin can have one output for the indices of non-zero elements, and a second output containing the number of non-zero elements. Suppose the input has size [M,N] and has K non-zero elements. The plugin can write K to the second output. When telling TensorRT that the first output has shape [2,K], plugin uses IExprBuilder::constant() and IExprBuilder::declareSizeTensor(1,...) to create the IDimensionExpr that respectively denote 2 and K.

TensorRT also needs to know the value of K to use for auto-tuning and an upper bound on K so that it can allocate memory for the output tensor. In the example, supposed typically half of the plugin's input elements are non-zero, and all the elements might be nonzero. then using M∗N/2 might be a good expression for the opt parameter, and M∗N for the upper bound. IDimensionsExpr for these expressions can be constructed from IDimensionsExpr for the input dimensions.

Parameters

outputIndex	index of a plugin output that is a size tensor.
opt	formula for computing auto-tuning value. Must not depend on a size tensor.
upper	Upper bound on the size tensor.

Returns

IDimensionExpr denoting the value of the size tensor.

See also

IPluginV3OneBuild::getOutputShapes()

9.97.3.3 operation()

```
IDimensionExpr const * nvinfer1::IExprBuilder::operation (
            DimensionOperation op,
            IDimensionExpr const & first,
            IDimensionExpr const & second )  [inline], [noexcept]
```

Get the operation.

Return pointer to IDimensionExpr that represents the given operation applied to first and second. Returns nullptr if op is not a valid DimensionOperation.

9.97.4 Member Data Documentation

9.97.4.1 mImpl

```
apiv::VExprBuilder* nvinfer1::IExprBuilder::mImpl  [protected]
```

The documentation for this class was generated from the following file:

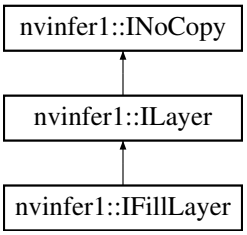
- NvInferRuntime.h

9.98 nvinfer1::IFillLayer Class Reference

Generate a tensor according to a specified mode.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IFillLayer:



Public Member Functions

- void setDimensions (Dims const &dimensions) noexcept

  Set the output tensor's dimensions.
- Dims getDimensions () const noexcept

  Get the output tensor's dimensions.
- void setOperation (FillOperation op) noexcept

  Set the fill operation for the layer.
- FillOperation getOperation () const noexcept

  Get the fill operation for the layer.
- void setAlpha (double alpha) noexcept

  Set the alpha parameter.
- double getAlpha () const noexcept

  Get the value of alpha parameter.
- void setBeta (double beta) noexcept

  Set the beta parameter.
- double getBeta () const noexcept

  Get the value of beta parameter.
- void setAlphaInt64 (int64_t alpha) noexcept

  Set the alpha parameter with int64 datatype.
- int64_t getAlphaInt64 () const noexcept

  Get the value of alpha parameter with int64 datatype.
- void setBetaInt64 (int64_t beta) noexcept

  Set the beta parameter with int64 datatype.
- int64_t getBetaInt64 () const noexcept

  Get the value of beta parameter with int64 datatype.
- bool isAlphaBetaInt64 () const noexcept

  Return true if alpha/beta have type int64, false if they have type double.
- void setToType (DataType toType) noexcept

  Set the fill layer output type.
- DataType getToType () const noexcept

  Get the fill layer output type.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~IFillLayer () noexcept=0

Protected Attributes

- apiv::VFillLayer * mImpl

9.98.1 Detailed Description

Generate a tensor according to a specified mode.

The fill layer generates a tensor with values that are drawn from a random distribution or an affine function of their indices, as specified by the FillMode.

When an IFillLayer is initially added to a network, all of its parameters are static. Each parameter may be changed to dynamic by setting a corresponding input. A parameter is considered dynamic even if that input is the output of an IConstantLayer. The inputs for each parameter are:

- 0: Dimensions
- 1: Alpha
- 2: Beta

The parameter Dimensions describes the shape of the output. If the Dimensions input is provided, it must be a 1D tensor of type Int32 or Int64 whose length is computable by constant folding.

The meanings of Alpha and Beta depend on the mode, as described in IFillLayer::setAlpha(), IFillLayer::setBeta(), and IFillLayer::setInput(). Parameters Alpha and Beta must both be static or both be dynamic.

An IFillLayer can produce a shape tensor if the following restrictions are met:

- The FillOperation is kLINSPACE.
- The output has type Int32, Int64, or Float.
- The volume of the output is within the volume limit imposed on shape tensors.
- If input 0 exists, the values of input 0 must be computable by constant folding.

See also

FillOperation

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.98.2 Constructor & Destructor Documentation

9.98.2.1 ~IFillLayer()

```
nvinfer1::IFillLayer::~IFillLayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.98.3 Member Function Documentation

9.98.3.1 getAlpha()

```
double nvinfer1::IFillLayer::getAlpha ( ) const  [inline], [noexcept]
```

Get the value of alpha parameter.

Returns

A double value of alpha.

If the second input is present and non-null, this function returns -1.0.

See also

setAlpha

9.98.3.2 getAlphaInt64()

```
int64_t nvinfer1::IFillLayer::getAlphaInt64 ( ) const  [inline], [noexcept]
```

Get the value of alpha parameter with int64 datatype.

Returns

A int64 value of alpha.

If the second input is present and non-null, this function returns -1.

See also

setAlphaInt64

9.98.3.3 getBeta()

```
double nvinfer1::IFillLayer::getBeta ( ) const  [inline], [noexcept]
```

Get the value of beta parameter.

Returns

A double value of beta.

If the third input is present and non-null, this function returns -1.0.

See also

setBeta, setBetaInt64

9.98.3.4 getBetaInt64()

```
int64_t nvinfer1::IFillLayer::getBetaInt64 ( ) const  [inline], [noexcept]
```

Get the value of beta parameter with int64 datatype.

Returns

A int64 value of beta.

If the third input is present and non-null, this function returns -1.0.

See also

setBetaInt64

9.98.3.5 getDimensions()

```
Dims nvinfer1::IFillLayer::getDimensions ( ) const  [inline], [noexcept]
```

Get the output tensor's dimensions.

Returns

The output tensor's dimensions, or an invalid Dims structure.

If the first input is present and non-null, this function returns a Dims with nbDims = -1.

See also

setDimensions

9.98.3.6 getOperation()

```
FillOperation nvinfer1::IFillLayer::getOperation ( ) const  [inline], [noexcept]
```

Get the fill operation for the layer.

See also

setOperation(), FillOperation

9.98.3.7 getToType()

```
DataType nvinfer1::IFillLayer::getToType ( ) const  [inline], [noexcept]
```

Get the fill layer output type.

Returns

toType parameter set during layer creation or by setToType(). The return value is the output type of the fill layer. The default value is DataType::kFLOAT.

9.98.3.8 isAlphaBetaInt64()

```
bool nvinfer1::IFillLayer::isAlphaBetaInt64 ( ) const  [inline], [noexcept]
```

Return true if alpha/beta have type int64, false if they have type double.

9.98.3.9 setAlpha()

```
void nvinfer1::IFillLayer::setAlpha (
            double alpha )  [inline], [noexcept]
```

Set the alpha parameter.

Parameters

alpha	has different meanings for each operator:

Operation	Usage
kLINSPACE	the start value, defaults to 0.0;
kRANDOM_UNIFORM	the minimum value, defaults to 0.0;
kRANDOM_NORMAL	the mean of the normal distribution, default is 0.0;

If input 1 exists, it is reset to null by this method.

See also

getAlpha, setAlphaInt64

9.98.3.10 setAlphaInt64()

```
void nvinfer1::IFillLayer::setAlphaInt64 (
            int64_t alpha )  [inline], [noexcept]
```

Set the alpha parameter with int64 datatype.

Parameters

alpha	has different meanings for each operator:

Operation	Usage
kLINSPACE	the start value, defaults to 0;
kRANDOM_UNIFORM	the minimum value, defaults to 0;
kRANDOM_NORMAL	the mean of the normal distribution, default is 0;

If a third input had been used to create this layer, that input is reset to null by this method.

See also

getAlphaInt64

9.98.3.11 setBeta()

```
void nvinfer1::IFillLayer::setBeta (
            double beta )  [inline], [noexcept]
```

Set the beta parameter.

Parameters

beta	has different meanings for each operator:

Operation	Usage
kLINSPACE	the delta value, defaults to 1.0;
kRANDOM_UNIFORM	the maximal value, defaults to 1.0;
kRANDOM_NORMAL	the standard deviation of the normal distribution, default is 1.0;

If input 2 exists, it is reset to null by this method.

See also

getBeta

9.98.3.12 setBetaInt64()

```
void nvinfer1::IFillLayer::setBetaInt64 (
            int64_t beta )  [inline], [noexcept]
```

Set the beta parameter with int64 datatype.

Parameters

beta	has different meanings for each operator:

Operation	Usage
kLINSPACE	the delta value, defaults to 1;
kRANDOM_UNIFORM	the maximal value, defaults to 1;
kRANDOM_NORMAL	the standard deviation of the normal distribution, default is 1;

If a third input had been used to create this layer, that input is reset to null by this method.

See also

getBetaInt64

9.98.3.13 setDimensions()

```
void nvinfer1::IFillLayer::setDimensions (
            Dims const & dimensions )  [inline], [noexcept]
```

Set the output tensor's dimensions.

Parameters

dimensions	The output tensor's dimensions.

If the first input had been used to create this layer, that input is reset to null by this method.

See also

getDimensions

9.98.3.14 setInput()

```
void nvinfer1::ILayer::setInput (
            int32_t index,
            ITensor & tensor )  [inline], [noexcept]
```

Replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to set.
tensor	the new input tensor

The three inputs correspond to these setters of IFillLayer:

- 0: setDimensions
- 1: setAlpha
- 2: setBeta

The following descriptions give more intuitive names for the inputs.

Indices for kLINSPACE are:

- 0: Shape, a 1D shape tensor, specifies the output tensor's dimensions.

- 1: Start, a scalar, specifies the start value.
- 2: Delta, a 1D tensor, specifies the delta value for each dimension.

Indices for kRANDOM_UNIFORM are:

- 0: Shape, a 1D shape tensor, specifies the output tensor's dimensions.
- 1: Minimum, a scalar, specifies the minimum random value.
- 2: Maximum, a scalar, specifies the maximal random value.

Indices for kRANDOM_NORMAL are:

- 0: Shape, a 1D shape tensor, specifies the output tensor's dimensions.
- 1: Mean, a scalar, specifies the mean of the normal distribution,.
- 2: Scale, a scalar, specifies the standard deviation of the normal distribution.

Using the corresponding setter resets the input to null.

If either inputs 1 or 2 is non-null, then both must be non-null and have the same data type.

If this function is called for an index greater or equal to getNbInputs(), then afterwards getNbInputs() returns index + 1, and any missing intervening inputs are set to null.

9.98.3.15 setOperation()

```
void nvinfer1::IFillLayer::setOperation (
            FillOperation op )  [inline], [noexcept]
```

Set the fill operation for the layer.

See also

getOperation(), FillOperation

9.98.3.16 setToType()

```
void nvinfer1::IFillLayer::setToType (
            DataType toType )  [inline], [noexcept]
```

Set the fill layer output type.

Parameters

toType	The DataType of the output tensor.

Set the output type of the fill layer. Valid values are DataType::kFLOAT, DataType::kHALF, DataType::kINT32, and DataType::kINT64. If the network is strongly typed, setToType must be used to set the output type, and use of setOutputType is an error. Otherwise, types passed to setOutputType and setToType must be the same.

See also

NetworkDefinitionCreationFlag::kSTRONGLY_TYPED

FillOperation for more information on which types are supported for each mode.

9.98.4 Member Data Documentation

9.98.4.1 mImpl

```
apiv::VFillLayer* nvinfer1::IFillLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

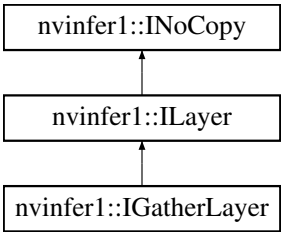
- NvInfer.h

9.99 nvinfer1::IGatherLayer Class Reference

A Gather layer in a network definition. Supports several kinds of gathering.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IGatherLayer:



Public Member Functions

- void setGatherAxis (int32_t axis) noexcept

  Set the axis used by GatherMode::kELEMENTS and GatherMode::kDEFAULT The axis must be less than the number of dimensions in the data input. The axis defaults to 0.
- int32_t getGatherAxis () const noexcept

  Get the axis to gather on.
- void setNbElementWiseDims (int32_t elementWiseDims) noexcept

  Set the number of leading dimensions of indices tensor to be handled elementwise.
- int32_t getNbElementWiseDims () const noexcept

  Get the number of leading dimensions of indices tensor to be handled elementwise.
- void setMode (GatherMode mode) noexcept

  Set the gather mode.
- GatherMode getMode () const noexcept

  Get the gather mode.

Protected Member Functions

- virtual ~IGatherLayer () noexcept=0

Protected Attributes

- apiv::VGatherLayer * mImpl

9.99.1 Detailed Description

A Gather layer in a network definition. Supports several kinds of gathering.

The Gather layer has two input tensors, Data and Indices, and an output tensor Output. Additionally, there are three parameters: mode, nbElementwiseDims, and axis that control how the indices are interpreted.

- Data is a tensor of rank r >= 1 that stores the values to be gathered in Output.
- Indices is a tensor of rank q that determines which locations in Data to gather.
  - GatherMode::kDEFAULT: q >= 0
  - GatherMode::kND: q >= 1 and the last dimension of Indices must be a build time constant.
  - GatherMode::kELEMENT: q = r
- Output stores the gathered results. Its rank s depends on the mode:
  - GatherMode::kDEFAULT: s = q + r - 1 - nbElementwiseDims
  - GatherMode::kND: s = q + r - indices.d[q-1] - 1 - nbElementwiseDims
  - GatherMode::kELEMENT: s = q = r.

The dimensions of the output likewise depends on the mode:

```
GatherMode::kDEFAULT:

    First nbElementwiseDims of output are computed by applying broadcast rules to
    first nbElementwiseDims of indices and data. Note that nbElementwiseDims <= 1.
    Rest of dimensions are computed by copying dimensions of Data, and replacing
    the dimension for axis gatherAxis with the dimensions of indices.

GatherMode::kND:
    If indices.d[q-1] = r - nbElementwiseDims
        output.d = [indices.d[0], ... , indices.d[q-2]]
    Else if indices.d[q-1] < r - nbElementwiseDims
        output.d = [indices.d[0], ... , indices.d[q-1], data.d[nbElementwiseDims + indices.d[q-1] + q],
        data.d[r-1]]
    Else
        This is build time error

GatherMode::kELEMENT:
    The output dimensions match the dimensions of the indices tensor.
```

The types of Data and Output must be the same, and Indices shall be DataType::kINT32 or DataType::kINT64.

How the elements of Data are gathered depends on the mode:

```
GatherMode::kDEFAULT:
    Each index in indices is used to index Data along axis gatherAxis.

GatherMode::kND:
    Indices is a rank q integer tensor, best thought of as a rank (q-1) tensor of
    indices into data, where each element defines a slice of data
    The operation can be formulated as output[i_1, ..., i_{q-1}] = data[indices[i_1, ..., i_{q-1}]]

GatherMode::kELEMENT:

    Here "axis" denotes the result of getGatherAxis().
    For each element X of indices:
        Let J denote a sequence for the subscripts of X
        Let K = sequence J with element [axis] replaced by X
        output[J] = data[K]
```

The handling of nbElementWiseDims depends on the mode:

- GatherMode::kDEFAULT: nbElementWiseDims <= 1. Broadcast is supported across the elementwise dimension if present.
- GatherMode::kND: 0 <= nbElementWiseDims < rank(Data)-1. Broadcast is not supported across the elementwise dimensions.
- GatherMode::kELEMENT: nbElementWiseDims = 0

Notes:

- For modes GatherMode::kND and GatherMode::kELEMENT, the first nbElementWiseDims dimensions of data and index must be equal. If not, an error will be reported at build time or run time.
- If an axis of Data has dynamic length, using a negative index for it has undefined behavior.
- No DLA support
- Zero will be stored for OOB access

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.99.2 Constructor & Destructor Documentation

9.99.2.1 ~IGatherLayer()

```
nvinfer1::IGatherLayer::~IGatherLayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.99.3 Member Function Documentation

9.99.3.1 getGatherAxis()

```
int32_t nvinfer1::IGatherLayer::getGatherAxis ( ) const  [inline], [noexcept]
```

Get the axis to gather on.

Warning

Undefined behavior when used with GatherMode::kND.

See also

setGatherAxis()

9.99.3.2 getMode()

```
GatherMode nvinfer1::IGatherLayer::getMode ( ) const  [inline], [noexcept]
```

Get the gather mode.

See also

setMode()

9.99.3.3 getNbElementWiseDims()

```
int32_t nvinfer1::IGatherLayer::getNbElementWiseDims ( ) const  [inline], [noexcept]
```

Get the number of leading dimensions of indices tensor to be handled elementwise.

See also

setNbElementWiseDims()

9.99.3.4 setGatherAxis()

```
void nvinfer1::IGatherLayer::setGatherAxis (
            int32_t axis )  [inline], [noexcept]
```

Set the axis used by GatherMode::kELEMENTS and GatherMode::kDEFAULT The axis must be less than the number of dimensions in the data input. The axis defaults to 0.

Warning

Undefined behavior when used with GatherMode::kND.

See also

getGatherAxis()

9.99.3.5 setMode()

```
void nvinfer1::IGatherLayer::setMode (
            GatherMode mode )  [inline], [noexcept]
```

Set the gather mode.

See also

getMode()

9.99.3.6 setNbElementWiseDims()

```
void nvinfer1::IGatherLayer::setNbElementWiseDims (
            int32_t elementWiseDims )  [inline], [noexcept]
```

Set the number of leading dimensions of indices tensor to be handled elementwise.

The gathering of indexing starts from the dimension of data[NbElementWiseDims:]. The NbElementWiseDims must be less than the Rank of the data input.

Parameters

elementWiseDims	number of dims to be handled as elementwise.

Default: 0

The value of nbElementWiseDims and GatherMode are checked during network validation:

GatherMode::kDEFAULT: nbElementWiseDims can be 0 or 1. GatherMode::kND: nbElementWiseDims can be between 0 and one less than rank(data). GatherMode::kELEMENT: nbElementWiseDims must be 0

See also

getNbElementWiseDims()

9.99.4 Member Data Documentation

9.99.4.1 mImpl

```
apiv::VGatherLayer* nvinfer1::IGatherLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.100 IGpuAllocator Class Reference

Application-implemented class for controlling allocation on the GPU.

```
#include <NvInferRuntime.h>
```

9.100.1 Detailed Description

Application-implemented class for controlling allocation on the GPU.

Warning

The lifetime of an IGpuAllocator object must exceed that of all objects that use it.

This class is intended as a base class for allocators that implement synchronous allocation. If you want the benefits of asynchronous allocation, you can do either of:

- Derive your class from IGpuAllocator and override all four of its virtual methods for allocation/deallocation, including the two deprecated methods.
- Derive your class from IGpuAsyncAllocator and override its two pure virtual methods for allocation/deallocation.

The latter style is preferred because it does not tie code to deprecated methods.

See also

IGpuAsyncAllocator.

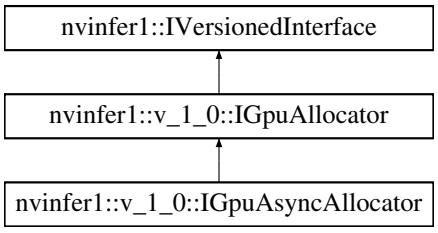
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.101 nvinfer1::v_1_0::IGpuAllocator Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IGpuAllocator:



Public Member Functions

- virtual TRT_DEPRECATED void * allocate (uint64_t const size, uint64_t const alignment, AllocatorFlags const flags) noexcept=0

  A thread-safe callback implemented by the application to handle acquisition of GPU memory.
- ~IGpuAllocator () override=default
- IGpuAllocator ()=default
- virtual void * reallocate (void *const, uint64_t, uint64_t) noexcept

  A thread-safe callback implemented by the application to resize an existing allocation.
- virtual TRT_DEPRECATED bool deallocate (void *const memory) noexcept=0

  A thread-safe callback implemented by the application to handle release of GPU memory.
- virtual void * allocateAsync (uint64_t const size, uint64_t const alignment, AllocatorFlags const flags, cudaStream_t) noexcept

  A thread-safe callback implemented by the application to handle stream-ordered acquisition of GPU memory.
- virtual bool deallocateAsync (void *const memory, cudaStream_t) noexcept

  A thread-safe callback implemented by the application to handle stream-ordered release of GPU memory.
- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.

Additional Inherited Members

9.101.1 Constructor & Destructor Documentation

9.101.1.1 ~IGpuAllocator()

```
nvinfer1::v_1_0::IGpuAllocator::~IGpuAllocator ( )  [override], [default]
```

9.101.1.2 IGpuAllocator()

```
nvinfer1::v_1_0::IGpuAllocator::IGpuAllocator ( )  [default]
```

9.101.2 Member Function Documentation

9.101.2.1 allocate()

```
virtual TRT_DEPRECATED void * nvinfer1::v_1_0::IGpuAllocator::allocate (
            uint64_t const size,
            uint64_t const alignment,
            AllocatorFlags const flags )  [pure virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle acquisition of GPU memory.

Parameters

size	The size of the memory block required (in bytes).
alignment	The required alignment of memory. Alignment will be zero or a power of 2 not exceeding the alignment guaranteed by cudaMalloc. Thus this allocator can be safely implemented with cudaMalloc/cudaFree. An alignment value of zero indicates any alignment is acceptable.
flags	Reserved for future use. In the current release, 0 will be passed.

Returns

If the allocation was successful, the start address of a device memory block of the requested size. If an allocation request of size 0 is made, nullptr must be returned. If an allocation request cannot be satisfied, nullptr must be returned. If a non-null address is returned, it is guaranteed to have the specified alignment.

Note

The implementation must guarantee thread safety for concurrent allocate/reallocate/deallocate requests.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads.

Deprecated Deprecated in TensorRT 10.0. Superseded by allocateAsync

Implemented in nvinfer1::v_1_0::IGpuAsyncAllocator.

9.101.2.2 allocateAsync()

```
virtual void * nvinfer1::v_1_0::IGpuAllocator::allocateAsync (
            uint64_t const size,
            uint64_t const alignment,
            AllocatorFlags const flags,
            cudaStream_t  )  [inline], [virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle stream-ordered acquisition of GPU memory.

The default behavior is to call method allocate(), which is synchronous and thus loses any performance benefits of asynchronous allocation. If you want the benefits of asynchronous allocation, see discussion of IGpuAsyncAllocator vs. IGpuAllocator in the documentation for nvinfer1::IGpuAllocator.

Parameters

size	The size of the memory block required (in bytes).
alignment	The required alignment of memory. Alignment will be zero or a power of 2 not exceeding the alignment guaranteed by cudaMalloc. Thus this allocator can be safely implemented with cudaMalloc/cudaFree. An alignment value of zero indicates any alignment is acceptable.
flags	Reserved for future use. In the current release, 0 will be passed.
stream	specifies the cudaStream for asynchronous usage.

Returns

If the allocation was successful, the start address of a device memory block of the requested size. If an allocation request of size 0 is made, nullptr must be returned. If an allocation request cannot be satisfied, nullptr must be returned. If a non-null address is returned, it is guaranteed to have the specified alignment.

Note

The implementation must guarantee thread safety for concurrent allocate/reallocate/deallocate requests.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads.

Reimplemented in nvinfer1::v_1_0::IGpuAsyncAllocator.

9.101.2.3 deallocate()

```
virtual TRT_DEPRECATED bool nvinfer1::v_1_0::IGpuAllocator::deallocate (
            void *const memory )  [pure virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle release of GPU memory.

TensorRT may pass a nullptr to this function if it was previously returned by allocate().

Parameters

memory	A memory address that was previously returned by an allocate() or reallocate() call of the same allocator object.

Returns

True if the acquired memory is released successfully.

Note

The implementation must guarantee thread safety for concurrent allocate/reallocate/deallocate requests.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads.

  Deprecated Deprecated in TensorRT 10.0. Superseded by deallocateAsync

Implemented in nvinfer1::v_1_0::IGpuAsyncAllocator.

9.101.2.4 deallocateAsync()

```
virtual bool nvinfer1::v_1_0::IGpuAllocator::deallocateAsync (
            void *const memory,
            cudaStream_t  )  [inline], [virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle stream-ordered release of GPU memory.

The default behavior is to call method deallocate(), which is synchronous and thus loses any performance benefits of asynchronous deallocation. If you want the benefits of asynchronous deallocation, see discussion of IGpuAsyncAllocator vs. IGpuAllocator in the documentation for nvinfer1::IGpuAllocator.

TensorRT may pass a nullptr to this function if it was previously returned by allocate().

Parameters

memory	A memory address that was previously returned by an allocate() or reallocate() call of the same allocator object.
stream	specifies the cudaStream for asynchronous usage.

Returns

True if the acquired memory is released successfully.

Note

The implementation must guarantee thread safety for concurrent allocate/reallocate/deallocate requests.

The implementation is not required to be asynchronous. It is permitted to synchronize, albeit doing so will lose the performance advantage of asynchronous deallocation. Either way, it is critical that it not actually free the memory until the current stream position is reached.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads.

Reimplemented in nvinfer1::v_1_0::IGpuAsyncAllocator.

9.101.2.5 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IGpuAllocator::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

Reimplemented in nvinfer1::v_1_0::IGpuAsyncAllocator.

9.101.2.6 reallocate()

```
virtual void * nvinfer1::v_1_0::IGpuAllocator::reallocate (
            void * const ,
            uint64_t ,
            uint64_t  )  [inline], [virtual], [noexcept]
```

A thread-safe callback implemented by the application to resize an existing allocation.

Only allocations which were allocated with AllocatorFlag::kRESIZABLE will be resized.

Options are one of:

- resize in place leaving min(oldSize, newSize) bytes unchanged and return the original address
- move min(oldSize, newSize) bytes to a new location of sufficient size and return its address
- return nullptr, to indicate that the request could not be fulfilled.

If nullptr is returned, TensorRT will assume that resize() is not implemented, and that the allocation at baseAddr is still valid.

This method is made available for use cases where delegating the resize strategy to the application provides an opportunity to improve memory management. One possible implementation is to allocate a large virtual device buffer and progressively commit physical memory with cuMemMap. CU_MEM_ALLOC_GRANULARITY_RECOMMENDED is suggested in this case.

TensorRT may call realloc to increase the buffer by relatively small amounts.

Parameters

baseAddr	the address of the original allocation, which will have been returned by previously calling allocate() or reallocate() on the same object.
alignment	The alignment used by the original allocation. This will be the same value that was previously passed to the allocate() or reallocate() call that returned baseAddr.
newSize	The new memory size required (in bytes).

Returns

The address of the reallocated memory, or nullptr. If a non-null address is returned, it is guaranteed to have the specified alignment.

Note

The implementation must guarantee thread safety for concurrent allocate/reallocate/deallocate requests.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads.

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.102 IGpuAsyncAllocator Class Reference

Application-implemented class for controlling asynchronous (stream ordered) memory allocation on the GPU.

`#include <NvInferRuntime.h>`

9.102.1 Detailed Description

Application-implemented class for controlling asynchronous (stream ordered) memory allocation on the GPU.

> Warning
>
> The lifetime of an IGpuAsyncAllocator object must exceed that of all objects that use it.

The advantage of deriving from IGpuAsyncAllocator instead of IGpuAllocator is that you only have to override two methods: allocateAsync() and deallocateAsync() to implement an allocator with asynchronous capability, whereas deriving from IGpuAllocator requires overriding four methods, including two deprecated methods.

See also

IGpuAllocator

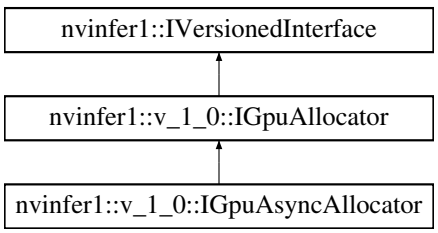
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.103 nvinfer1::v_1_0::IGpuAsyncAllocator Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IGpuAsyncAllocator:



Public Member Functions

- IGpuAsyncAllocator ()=default
- ~IGpuAsyncAllocator () override=default
- void * allocateAsync (uint64_t const size, uint64_t const alignment, AllocatorFlags const flags, cudaStream_t) noexcept override=0

  A thread-safe callback implemented by the application to handle stream-ordered asynchronous acquisition of GPU memory.
- bool deallocateAsync (void *const memory, cudaStream_t) noexcept override=0

  A thread-safe callback implemented by the application to handle stream-ordered asynchronous release of GPU memory.
- TRT_DEPRECATED void * allocate (uint64_t const size, uint64_t const alignment, AllocatorFlags const flags) noexcept override

  A thread-safe callback implemented by the application to handle acquisition of GPU memory.
- TRT_DEPRECATED bool deallocate (void *const memory) noexcept override

  A thread-safe callback implemented by the application to handle release of GPU memory.
- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.

Additional Inherited Members

9.103.1 Constructor & Destructor Documentation

9.103.1.1 IGpuAsyncAllocator()

```
nvinfer1::v_1_0::IGpuAsyncAllocator::IGpuAsyncAllocator ( )  [default]
```

9.103.1.2 ~IGpuAsyncAllocator()

```
nvinfer1::v_1_0::IGpuAsyncAllocator::~IGpuAsyncAllocator ( )  [override], [default]
```

9.103.2 Member Function Documentation

9.103.2.1 allocate()

```
TRT_DEPRECATED void * nvinfer1::v_1_0::IGpuAsyncAllocator::allocate (
            uint64_t const size,
            uint64_t const alignment,
            AllocatorFlags const flags )  [inline], [override], [virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle acquisition of GPU memory.

Parameters

size	The size of the memory block required (in bytes).
alignment	The required alignment of memory. Alignment will be zero or a power of 2 not exceeding the alignment guaranteed by cudaMalloc. Thus this allocator can be safely implemented with cudaMalloc/cudaFree. An alignment value of zero indicates any alignment is acceptable.
flags	Reserved for future use. In the current release, 0 will be passed.

Returns

If the allocation was successful, the start address of a device memory block of the requested size. If an allocation request of size 0 is made, nullptr must be returned. If an allocation request cannot be satisfied, nullptr must be returned. If a non-null address is returned, it is guaranteed to have the specified alignment.

Note

The implementation must guarantee thread safety for concurrent allocateAsync/deallocateAsync/reallocate requests.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads.

    Deprecated Deprecated in TensorRT 10.0. Superseded by allocateAsync

Implements nvinfer1::v_1_0::IGpuAllocator.

9.103.2.2 allocateAsync()

```
void * nvinfer1::v_1_0::IGpuAsyncAllocator::allocateAsync (
            uint64_t const size,
            uint64_t const alignment,
            AllocatorFlags const flags,
            cudaStream_t  )  [override], [pure virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle stream-ordered asynchronous acquisition of GPU memory.

Parameters

size	The size of the memory block required (in bytes).
alignment	The required alignment of memory. Alignment will be zero or a power of 2 not exceeding the alignment guaranteed by cudaMalloc. Thus this allocator can be safely implemented with cudaMalloc/cudaFree. An alignment value of zero indicates any alignment is acceptable.
flags	Reserved for future use. In the current release, 0 will be passed.
stream	Specifies the cudastream for the asynchronous allocation. If nullptr or 0 is passed, the default stream will be used.

Returns

If the allocation was successful, the start address of a device memory block of the requested size. If an allocation request of size 0 is made, nullptr must be returned. If an allocation request cannot be satisfied, nullptr must be returned. If a non-null address is returned, it is guaranteed to have the specified alignment.

Note

The implementation must guarantee thread safety for concurrent allocateAsync/deallocateAsync requests.

The implementation is not required to be asynchronous. It is permitted to synchronize, albeit doing so will lose the performance advantage of asynchronous allocation.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads.

Reimplemented from nvinfer1::v_1_0::IGpuAllocator.

9.103.2.3 deallocate()

```
TRT_DEPRECATED bool nvinfer1::v_1_0::IGpuAsyncAllocator::deallocate (
            void *const memory )  [inline], [override], [virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle release of GPU memory.

TensorRT may pass a nullptr to this function if it was previously returned by allocate().

Parameters

memory	A memory address that was previously returned by an allocate() or reallocate() call of the same allocator object.

Returns

True if the acquired memory is released successfully.

Note

The implementation must guarantee thread safety for concurrent allocate/reallocate/deallocate requests.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads.

  Deprecated Deprecated in TensorRT 10.0. Superseded by deallocateAsync

Implements nvinfer1::v_1_0::IGpuAllocator.

9.103.2.4 deallocateAsync()

```
bool nvinfer1::v_1_0::IGpuAsyncAllocator::deallocateAsync (
            void *const memory,
            cudaStream_t  )  [override], [pure virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle stream-ordered asynchronous release of GPU memory.

TensorRT may pass a nullptr to this function if it was previously returned by allocate().

Parameters

memory	A memory address that was previously returned by an allocate() or reallocate() call of the same allocator object.
stream	Specifies the cudastream for the asynchronous deallocation. If nullptr or 0 is passed, the default stream will be used.

Returns

True if the acquired memory is released successfully.

Note

The implementation must guarantee thread safety for concurrent allocateAsync/deallocateAsync requests.

The implementation is not required to be asynchronous. It is permitted to synchronize, albeit doing so will lose the performance advantage of asynchronous deallocation. Either way, it is critical that it not actually free the memory until the current stream position is reached.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads.

Reimplemented from nvinfer1::v_1_0::IGpuAllocator.

9.103.2.5 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IGpuAsyncAllocator::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Reimplemented from nvinfer1::v_1_0::IGpuAllocator.

The documentation for this class was generated from the following file:

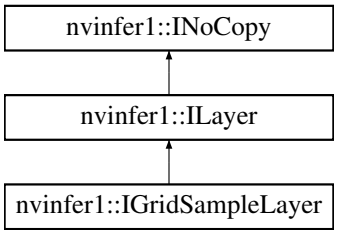
- NvInferRuntime.h

9.104 nvinfer1::IGridSampleLayer Class Reference

A GridSample layer in a network definition.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IGridSampleLayer:



Public Member Functions

- void setInterpolationMode (InterpolationMode mode) noexcept

  Set the grid sample interpolation mode.
- InterpolationMode getInterpolationMode () const noexcept

  Get the grid sample interpolation mode.
- void setAlignCorners (bool alignCorners) noexcept

  Set the align corners mode.
- bool getAlignCorners () const noexcept

  Get the align corners mode.
- bool setSampleMode (SampleMode mode) noexcept

  Set the sample mode.
- SampleMode getSampleMode () const noexcept

  Get the sample mode.

Protected Member Functions

- virtual ~IGridSampleLayer () noexcept=0

Protected Attributes

- apiv::VGridSampleLayer * mImpl

9.104.1 Detailed Description

A GridSample layer in a network definition.

This layer uses an input tensor and a grid tensor to produce an interpolated output tensor. The input and grid tensors must be shape tensors of rank 4. The only supported SampleMode values are SampleMode::kCLAMP, SampleMode::kFILL, and SampleMode::kREFLECT.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.104.2 Constructor & Destructor Documentation

9.104.2.1 ~IGridSampleLayer()

```
nvinfer1::IGridSampleLayer::~IGridSampleLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.104.3 Member Function Documentation

9.104.3.1 getAlignCorners()

```
bool nvinfer1::IGridSampleLayer::getAlignCorners ( ) const  [inline], [noexcept]
```

Get the align corners mode.

See also

setAlignCorners()

Returns

The value specified by setAlignCorners(), or false otherwise.

9.104.3.2 getInterpolationMode()

```
InterpolationMode nvinfer1::IGridSampleLayer::getInterpolationMode ( ) const  [inline], [noexcept]
```

Get the grid sample interpolation mode.

See also

setInterpolationMode()

Returns

The value specified by setInterpolationMode, or InterpolationMode::kLINEAR otherwise.

9.104.3.3 getSampleMode()

```
SampleMode nvinfer1::IGridSampleLayer::getSampleMode ( ) const  [inline], [noexcept]
```

Get the sample mode.

See also

setSampleMode()

Returns

the value specified by a successful call to setSampleMode(), or SampleMode::kFILL otherwise.

9.104.3.4 setAlignCorners()

```
void nvinfer1::IGridSampleLayer::setAlignCorners (
            bool alignCorners )  [inline], [noexcept]
```

Set the align corners mode.

See also

getAlignCorners()

9.104.3.5 setInterpolationMode()

```
void nvinfer1::IGridSampleLayer::setInterpolationMode (
            InterpolationMode mode )  [inline], [noexcept]
```

Set the grid sample interpolation mode.

See also

getInterpolationMode()

9.104.3.6 setSampleMode()

```
bool nvinfer1::IGridSampleLayer::setSampleMode (
            SampleMode mode )  [inline], [noexcept]
```

Set the sample mode.

See also

getSampleMode()

Returns

true if layer's sample mode was set to mode, false otherwise.

9.104.4 Member Data Documentation

9.104.4.1 mImpl

```
apiv::VGridSampleLayer* nvinfer1::IGridSampleLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

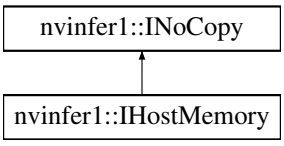
- NvInfer.h

9.105 nvinfer1::IHostMemory Class Reference

Class to handle library allocated memory that is accessible to the user.

```
#include <NvInferRuntime.h>
```

Inheritance diagram for nvinfer1::IHostMemory:



Public Member Functions

- virtual ~IHostMemory () noexcept=0
- void * data () const noexcept

  A pointer to the raw data that is owned by the library.
- std::size_t size () const noexcept

  The size in bytes of the data that was allocated.
- DataType type () const noexcept

  The type of the memory that was allocated.

Protected Attributes

- apiv::VHostMemory * mImpl

Additional Inherited Members

9.105.1 Detailed Description

Class to handle library allocated memory that is accessible to the user.

The memory allocated via the host memory object is owned by the library and will be de-allocated when the destroy method is called.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.105.2 Constructor & Destructor Documentation

9.105.2.1 ~IHostMemory()

```
nvinfer1::IHostMemory::~IHostMemory ( )  [inline], [pure virtual], [default], [noexcept]
```

9.105.3 Member Function Documentation

9.105.3.1 data()

```
void * nvinfer1::IHostMemory::data ( ) const  [inline], [noexcept]
```

A pointer to the raw data that is owned by the library.

9.105.3.2 size()

```
std::size_t nvinfer1::IHostMemory::size ( ) const  [inline], [noexcept]
```

The size in bytes of the data that was allocated.

9.105.3.3 type()

```
DataType nvinfer1::IHostMemory::type ( ) const  [inline], [noexcept]
```

The type of the memory that was allocated.

9.105.4 Member Data Documentation

9.105.4.1 mImpl

```
apiv::VHostMemory* nvinfer1::IHostMemory::mImpl  [protected]
```

The documentation for this class was generated from the following file:

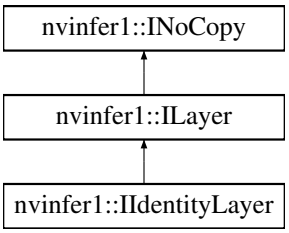
- NvInferRuntime.h

9.106 nvinfer1::IIdentityLayer Class Reference

A layer that represents the identity function.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IIdentityLayer:



Protected Member Functions

- virtual ~IIdentityLayer () noexcept=0

Protected Attributes

- apiv::VIdentityLayer * mImpl

Additional Inherited Members

9.106.1 Detailed Description

A layer that represents the identity function.

For a strongly typed network, the layer is an identity function, i.e. the output tensor elements are identical to the input tensor elements, possibly with a change in layout. For example, if a network consists of a single IIdentityLayer, the network input and output must have the same type, but the input can have NCHW layout and the output can have NHWC layout.

If the network is weakly typed, the layer is additionally permitted some type conversions as described below.

If the output type is explicitly specified via setOutputType, IIdentityLayer can be used to convert from one type to another. Other than conversions between the same type (kFLOAT -> kFLOAT for example), the only valid conversions are:

```
(kFLOAT | kHALF | kINT32 | kBOOL) -> (kFLOAT | kHALF | kINT32 | kBOOL)

(kFLOAT | kHALF) -> kUINT8

kUINT8 -> (kFLOAT | kHALF)
```

Conversion also happens implicitly, without calling setOutputType, if the output tensor is a network output.

Two types are compatible if they are identical, or are both in {kFLOAT, kHALF}. Implicit conversion between incompatible types, i.e. without using setOutputType, was recognized as incorrect as of TensorRT 8.4, but was retained for API compatibility within TensorRT 8.x releases. In TensorRT 10.0 onwards it is an error if the network output tensor type is incompatible with the layer output type. E.g., implicit conversion from kFLOAT to kINT32 is not allowed.

To explicitly convert kFLOAT to kINT32:

- Preferred: use ICastLayer.
- Legacy alternative: use IIdentityLayer and setOutputType(DataType::kINT32).

Similar advice applies for explicit conversion in the other direction.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.106.2 Constructor & Destructor Documentation

9.106.2.1 ~IIdentityLayer()

```
nvinfer1::IIdentityLayer::~IIdentityLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.106.3 Member Data Documentation

9.106.3.1 mImpl

```
apiv::VIdentityLayer* nvinfer1::IIdentityLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

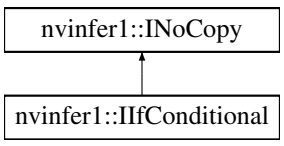
- NvInfer.h

9.107 nvinfer1::IIfConditional Class Reference

Helper for constructing conditionally-executed subgraphs.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IIfConditional:



Public Member Functions

- IConditionLayer * setCondition (ITensor &condition) noexcept

  Set the condition tensor for this If-Conditional construct.
- IIfConditionalOutputLayer * addOutput (ITensor &trueSubgraphOutput, ITensor &falseSubgraphOutput) noexcept

  Add an If-conditional output.
- IIfConditionalInputLayer * addInput (ITensor &input) noexcept

  Add an If-conditional input.
- void setName (char const *name) noexcept

  Set the name of the conditional.
- char const * getName () const noexcept

  Return the name of the conditional.

Protected Member Functions

- virtual ~IIfConditional () noexcept=0

Protected Attributes

- apiv::VIfConditional * mImpl

9.107.1 Detailed Description

Helper for constructing conditionally-executed subgraphs.

An If-conditional conditionally executes part of the network according to the following pseudo-code:

If condition is true then: output = trueSubgraph(trueInputs); Else output = falseSubgraph(falseInputs); Emit output

Condition is a 0D boolean tensor (representing a scalar). trueSubgraph represents a network subgraph that is executed when condition evaluates to True. falseSubgraph represents a network subgraph that is executed when condition evaluates to False.

The following constraints apply to If-conditionals:

- Both the trueSubgraph and falseSubgraph must be defined.
- The number of output tensors in both subgraphs is the same.
- Corresponding output tensors from the true/false subgraphs have the same type and rank.

The subgraphs may directly use tensors defined outside of the IIfConditional.

9.107.2 Constructor & Destructor Documentation

9.107.2.1 ~IIfConditional()

```
nvinfer1::IIfConditional::~IIfConditional ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.107.3 Member Function Documentation

9.107.3.1 addInput()

```
IIfConditionalInputLayer * nvinfer1::IIfConditional::addInput (
            ITensor & input )  [inline], [noexcept]
```

Add an If-conditional input.

Parameters

input	An input to the conditional that can be used by either or both of the conditional's subgraphs.

See also

IIfConditionalInputLayer

9.107.3.2 addOutput()

```
IIfConditionalOutputLayer * nvinfer1::IIfConditional::addOutput (
            ITensor & trueSubgraphOutput,
            ITensor & falseSubgraphOutput )  [inline], [noexcept]
```

Add an If-conditional output.

Parameters

trueSubgraphOutput	The output of the subgraph executed when the conditional evaluates to true.
falseSubgraphOutput	The output of the subgraph executed when the conditional evaluates to false.

Each output layer of an IIfConditional represents a single output of either the true-subgraph or the false-subgraph of an IIfConditional, depending on which subgraph was executed.

The ranks of the two tensors must be equal unless the condition is a build-time constant.

See also

IIfConditionalOutputLayer

9.107.3.3 getName()

```
char const * nvinfer1::IIfConditional::getName ( ) const  [inline], [noexcept]
```

Return the name of the conditional.

See also

setName()

9.107.3.4 setCondition()

```
IConditionLayer * nvinfer1::IIfConditional::setCondition (
            ITensor & condition )  [inline], [noexcept]
```

Set the condition tensor for this If-Conditional construct.

Parameters

condition	The condition tensor that will determine which subgraph to execute.

`condition` tensor must be a 0D execution tensor (scalar) with type DataType::kBOOL.

See also

IConditionLayer

9.107.3.5 setName()

```
void nvinfer1::IIfConditional::setName (
            char const * name )  [inline], [noexcept]
```

Set the name of the conditional.

The name is used in error diagnostics. This method copies the name string.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

See also

getName()

9.107.4 Member Data Documentation

9.107.4.1 mImpl

```
apiv::VIfConditional* nvinfer1::IIfConditional::mImpl  [protected]
```

The documentation for this class was generated from the following file:

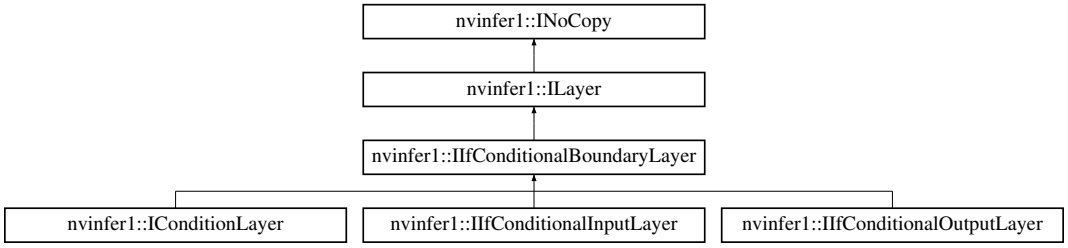
- NvInfer.h

9.108 nvinfer1::IIfConditionalBoundaryLayer Class Reference

This is a base class for Conditional boundary layers.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IIfConditionalBoundaryLayer:



Public Member Functions

- IIfConditional * getConditional () const noexcept

  Get a pointer to the IIfConditional associated with this boundary layer.

Protected Member Functions

- virtual ~IIfConditionalBoundaryLayer () noexcept=0

Protected Attributes

- apiv::VConditionalBoundaryLayer * mBoundary

9.108.1 Detailed Description

This is a base class for Conditional boundary layers.

Boundary layers are used to demarcate the boundaries of Conditionals.

9.108.2 Constructor & Destructor Documentation

9.108.2.1 ~IIfConditionalBoundaryLayer()

```
nvinfer1::IIfConditionalBoundaryLayer::~IIfConditionalBoundaryLayer ( )  [inline], [protected],
[pure virtual], [default], [noexcept]
```

9.108.3 Member Function Documentation

9.108.3.1 getConditional()

```
IIfConditional * nvinfer1::IIfConditionalBoundaryLayer::getConditional ( ) const  [inline], [noexcept]
```

Get a pointer to the IIfConditional associated with this boundary layer.

9.108.4 Member Data Documentation

9.108.4.1 mBoundary

```
apiv::VConditionalBoundaryLayer* nvinfer1::IIfConditionalBoundaryLayer::mBoundary  [protected]
```

The documentation for this class was generated from the following file:

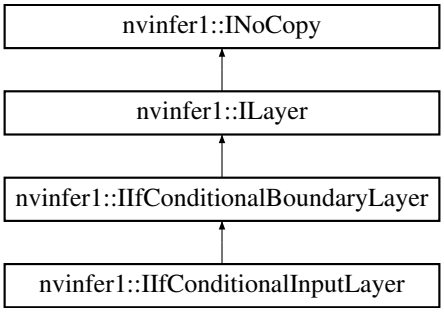
- NvInfer.h

9.109 nvinfer1::IIfConditionalInputLayer Class Reference

This layer represents an input to an IIfConditional.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IIfConditionalInputLayer:



Protected Member Functions

- virtual ~IIfConditionalInputLayer () noexcept=0

Protected Attributes

- apiv::VConditionalInputLayer * mImpl

Additional Inherited Members

9.109.1 Detailed Description

This layer represents an input to an IIfConditional.

9.109.2 Constructor & Destructor Documentation

9.109.2.1 ~IIfConditionalInputLayer()

```
nvinfer1::IIfConditionalInputLayer::~IIfConditionalInputLayer ( )  [inline], [protected], [pure
virtual], [default], [noexcept]
```

9.109.3 Member Data Documentation

9.109.3.1 mImpl

```
apiv::VConditionalInputLayer* nvinfer1::IIfConditionalInputLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

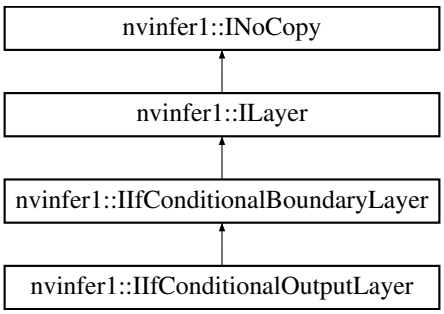
- NvInfer.h

9.110 nvinfer1::IIfConditionalOutputLayer Class Reference

This layer represents an output of an IIfConditional.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IIfConditionalOutputLayer:



Protected Member Functions

- virtual ~IIfConditionalOutputLayer () noexcept=0

Protected Attributes

- apiv::VConditionalOutputLayer * mImpl

Additional Inherited Members

9.110.1 Detailed Description

This layer represents an output of an IIfConditional.

An IIfConditionalOutputLayer has two inputs and one output.

See also

IIfConditional::addOutput

9.110.2 Constructor & Destructor Documentation

9.110.2.1 ~IIfConditionalOutputLayer()

```
nvinfer1::IIfConditionalOutputLayer::~IIfConditionalOutputLayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.110.3 Member Data Documentation

9.110.3.1 mImpl

```
apiv::VConditionalOutputLayer* nvinfer1::IIfConditionalOutputLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

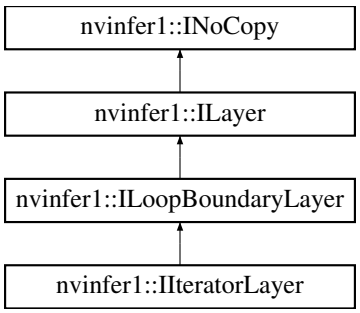
- NvInfer.h

9.111 nvinfer1::IIteratorLayer Class Reference

A layer to do iterations.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IIteratorLayer:



Public Member Functions

- void setAxis (int32_t axis) noexcept

  Set axis to iterate over.
- int32_t getAxis () const noexcept

  Get axis being iterated over.
- void setReverse (bool reverse) noexcept

  Set iteration order to be reverse.
- bool getReverse () const noexcept

  Check if the iteration order is reverse.

Protected Member Functions

- virtual ~IIteratorLayer () noexcept=0

Protected Attributes

- apiv::VIteratorLayer * mImpl

9.111.1 Detailed Description

A layer to do iterations.

The iterator layer iterates over a tensor along the given axis and in the given direction. It enables each loop iteration to inspect a different slice of the tensor.

See also

ILoop::addIterator()

9.111.2 Constructor & Destructor Documentation

9.111.2.1 ~IIteratorLayer()

```
nvinfer1::IIteratorLayer::~IIteratorLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.111.3 Member Function Documentation

9.111.3.1 getAxis()

```
int32_t nvinfer1::IIteratorLayer::getAxis ( ) const  [inline], [noexcept]
```

Get axis being iterated over.

9.111.3.2 getReverse()

```
bool nvinfer1::IIteratorLayer::getReverse ( ) const  [inline], [noexcept]
```

Check if the iteration order is reverse.

Returns

True if and only if reversing input.

9.111.3.3 setAxis()

```
void nvinfer1::IIteratorLayer::setAxis (
            int32_t axis )  [inline], [noexcept]
```

Set axis to iterate over.

9.111.3.4 setReverse()

```
void nvinfer1::IIteratorLayer::setReverse (
            bool reverse )  [inline], [noexcept]
```

Set iteration order to be reverse.

For reverse=false, the layer is equivalent to addGather(tensor, I, 0) where I is a scalar tensor containing the loop iteration number. For reverse=true, the layer is equivalent to addGather(tensor, M-1-I, 0) where M is the trip count computed from TripLimits of kind kCOUNT. The default is reverse=false.

9.111.4 Member Data Documentation

9.111.4.1 mImpl

```
apiv::VIteratorLayer* nvinfer1::IIteratorLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

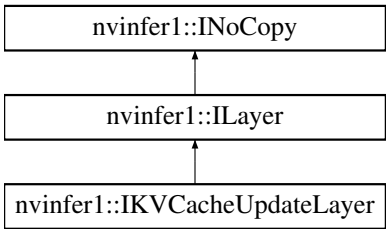
- NvInfer.h

9.112 nvinfer1::IKVCacheUpdateLayer Class Reference

Layer that represents a KVCacheUpdate operation.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IKVCacheUpdateLayer:



Public Member Functions

- bool setCacheMode (KVCacheMode cacheMode) noexcept

  Set the mode of the KVCacheUpdate layer.
- KVCacheMode getCacheMode () const noexcept

  Get the mode of the KVCacheUpdate layer.
- TRT_NODISCARD bool setUpdateForm (AttentionIOForm form) noexcept

  Set the update form.
- TRT_NODISCARD AttentionIOForm getUpdateForm () const noexcept

  Get the update form.
- TRT_NODISCARD bool setUpdateLengths (ITensor *lengths) noexcept

  Set the update lengths tensor.
- TRT_NODISCARD ITensor * getUpdateLengths () const noexcept

  Get the update lengths tensor.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~IKVCacheUpdateLayer () noexcept=0

Protected Attributes

- apiv::VKVCacheUpdateLayer * mImpl

9.112.1 Detailed Description

Layer that represents a KVCacheUpdate operation.

The KVCacheUpdate layer is used to cache the key or value tensors for the attention mechanism. K and V use separate KVCacheUpdate layers.

An IKVCacheUpdateLayer has three inputs (`cache`, `update`, `writeIndices`) and one output. In `kLINEAR` mode, for each batch element i, the layer copies the update tensor into the cache starting at position `writeIndices[i]`. Assuming no out-of-bounds writes occur, the operation for each sequence position s in [0, sequenceLength) is:

```
output[i, :, writeIndices[i] + s, :]  = update[i, :, s, :]
```

The output performs in-place updates on the cache tensor, so they must share the same device memory address.

> **Warning**
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.112.2 Constructor & Destructor Documentation

9.112.2.1 ~IKVCacheUpdateLayer()

```
nvinfer1::IKVCacheUpdateLayer::~IKVCacheUpdateLayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.112.3 Member Function Documentation

9.112.3.1 getCacheMode()

```
KVCacheMode nvinfer1::IKVCacheUpdateLayer::getCacheMode ( ) const  [inline], [noexcept]
```

Get the mode of the KVCacheUpdate layer.

Returns

The mode of the KVCacheUpdate layer.

9.112.3.2 getUpdateForm()

```
TRT_NODISCARD AttentionIOForm nvinfer1::IKVCacheUpdateLayer::getUpdateForm ( ) const  [inline],
[noexcept]
```

Get the update form.

Returns

The update form.

See also

setUpdateForm()

AttentionIOForm

9.112.3.3 getUpdateLengths()

```
TRT_NODISCARD ITensor * nvinfer1::IKVCacheUpdateLayer::getUpdateLengths ( ) const  [inline], [noexcept]
```

Get the update lengths tensor.

Returns

The update lengths tensor, or nullptr if not set.

See also

setUpdateLengths()

9.112.3.4 setCacheMode()

```
bool nvinfer1::IKVCacheUpdateLayer::setCacheMode (
            KVCacheMode cacheMode )  [inline], [noexcept]
```

Set the mode of the KVCacheUpdate layer.

Parameters

cacheMode	The mode of the KVCacheUpdate layer. For TensorRT 10.15, only `kLINEAR` mode is supported.

Returns

True if cache mode is set successfully, false otherwise.

9.112.3.5 setInput()

```
void nvinfer1::ILayer::setInput (
             int32_t index,
             ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor.

The indices are as follows:

Input 0 is the input cache tensor. Input 1 is the input update tensor. Input 2 is the input writeIndices tensor. Input 3 is the optional updateLengths tensor. setUpdateLengths should be used instead of setInput to set this tensor.

9.112.3.6 setUpdateForm()

```
TRT_NODISCARD bool nvinfer1::IKVCacheUpdateLayer::setUpdateForm (
             AttentionIOForm form )  [inline], [noexcept]
```

Set the update form.

Default is kPADDED_BHND. When set to kPACKED_NHD, the update tensor shape is [totalTokens, numHeads, dimHead] instead of [batchSize, numHeads, numTokens, dimHead], and the updateLengths tensor must be provided.

Parameters

form	The update form.

Returns

True if the update form is set successfully, false otherwise.

See also

getUpdateForm()
AttentionIOForm

9.112.3.7 setUpdateLengths()

```
TRT_NODISCARD bool nvinfer1::IKVCacheUpdateLayer::setUpdateLengths (
            ITensor * lengths )  [inline], [noexcept]
```

Set the update lengths tensor.

Only valid when the update form is kPACKED_NHD. Provides a cumulative token counts tensor with shape [batch↩ Size + 1]. The first element should be 0 and the last element equals totalTokens. The number of tokens for batch i is lengths[i + 1] - lengths[i]. Must be set when update form is kPACKED_NHD.

> Warning
>
> Providing a first element that is not 0 results in undefined behavior.

Parameters

lengths	A 1D tensor of type kINT32 with shape [batchSize + 1], or nullptr to clear a previously set update lengths tensor.

Returns

True if the update lengths tensor is set successfully, false otherwise.

See also

getUpdateLengths()

9.112.4 Member Data Documentation

9.112.4.1 mImpl

```
apiv::VKVCacheUpdateLayer* nvinfer1::IKVCacheUpdateLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

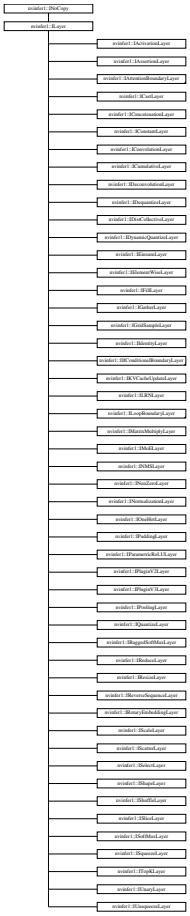
- NvInfer.h

9.113 nvinfer1::ILayer Class Reference

Base class for all layer classes in a network definition.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::ILayer:



Public Member Functions

- LayerType getType () const noexcept

  Return the type of a layer.
- void setName (char const *name) noexcept

  Set the name of a layer.
- char const * getName () const noexcept

  Return the name of a layer.
- int32_t getNbInputs () const noexcept

  Get the number of inputs of a layer.
- ITensor * getInput (int32_t index) const noexcept

  Get the layer input corresponding to the given index.

- int32_t getNbOutputs () const noexcept

  Get the number of outputs of a layer.
- ITensor * getOutput (int32_t index) const noexcept

  Get the layer output corresponding to the given index.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Replace an input of this layer with a specific tensor.
- DataType getOutputType (int32_t index) const noexcept

  get the output type of this layer
- void setMetadata (char const *metadata) noexcept

  Set the metadata for this layer.
- char const * getMetadata () const noexcept

  Get the metadata of the layer.
- bool setNbRanks (int32_t nbRanks) noexcept

  Set the number of ranks for multi-device execution.
- int32_t getNbRanks () const noexcept

  Get the number of ranks for multi-device execution.

Protected Member Functions

- virtual ~ILayer () noexcept=0

Protected Attributes

- apiv::VLayer * mLayer

9.113.1 Detailed Description

Base class for all layer classes in a network definition.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.113.2 Constructor & Destructor Documentation

9.113.2.1 ~ILayer()

```
nvinfer1::ILayer::~ILayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.113.3 Member Function Documentation

9.113.3.1 getInput()

```
ITensor * nvinfer1::ILayer::getInput (
            int32_t index ) const  [inline], [noexcept]
```

Get the layer input corresponding to the given index.

Parameters

index	The index of the input tensor.

Returns

The input tensor, or nullptr if the index is out of range or the tensor is optional (ISliceLayer).

9.113.3.2 getMetadata()

```
char const * nvinfer1::ILayer::getMetadata ( ) const  [inline], [noexcept]
```

Get the metadata of the layer.

Returns

The metadata as a null-terminated C-style string. If setMetadata() has not been called, an empty string "" will be returned as a default value.

See also

setMetadata()

9.113.3.3 getName()

```
char const * nvinfer1::ILayer::getName ( ) const  [inline], [noexcept]
```

Return the name of a layer.

See also

setName()

9.113.3.4 getNbInputs()

```
int32_t nvinfer1::ILayer::getNbInputs ( ) const  [inline], [noexcept]
```

Get the number of inputs of a layer.

9.113.3.5 getNbOutputs()

```
int32_t nvinfer1::ILayer::getNbOutputs ( ) const  [inline], [noexcept]
```

Get the number of outputs of a layer.

9.113.3.6 getNbRanks()

```
int32_t nvinfer1::ILayer::getNbRanks ( ) const  [inline], [noexcept]
```

Get the number of ranks for multi-device execution.

Returns

The number of ranks configured for multi-device execution. Default is 1.

See also

setNbRanks()

9.113.3.7 getOutput()

```
ITensor * nvinfer1::ILayer::getOutput (
            int32_t index ) const  [inline], [noexcept]
```

Get the layer output corresponding to the given index.

Returns

The indexed output tensor, or nullptr if the index is out of range or the tensor is optional.

9.113.3.8 getOutputType()

```
DataType nvinfer1::ILayer::getOutputType (
            int32_t index ) const  [inline], [noexcept]
```

get the output type of this layer

Parameters

index	the index of the output

Returns

the inferred output type.

See also

getOutputType() outputTypeIsSet() resetOutputType()

9.113.3.9 getType()

```
LayerType nvinfer1::ILayer::getType ( ) const  [inline], [noexcept]
```

Return the type of a layer.

See also

LayerType

9.113.3.10 setInput()

```
void nvinfer1::ILayer::setInput (
             int32_t index,
             ITensor & tensor )  [inline], [noexcept]
```

Replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor

Except for IFillLayer, ILoopOutputLayer, INMSLayer, IResizeLayer, IShuffleLayer, and ISliceLayer, this method cannot change the number of inputs to a layer. The index argument must be less than the value of getNbInputs().

See comments for overloads of setInput() for layers with special behavior.

9.113.3.11 setMetadata()

```
void nvinfer1::ILayer::setMetadata (
            char const * metadata )  [inline], [noexcept]
```

Set the metadata for this layer.

The metadata is emitted in the JSON returned by IEngineInspector with ProfilingVerbosity set to kDETAILED.

Parameters

metadata	The per-layer metadata.

Warning

The string name must be null-terminated and be at most 4096 bytes including the terminator.

See also

getMetadata()
getLayerInformation()

9.113.3.12 setName()

```
void nvinfer1::ILayer::setName (
            char const * name )  [inline], [noexcept]
```

Set the name of a layer.

This method copies the name string.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

See also

getName()

9.113.3.13 setNbRanks()

```
bool nvinfer1::ILayer::setNbRanks (
            int32_t nbRanks )  [inline], [noexcept]
```

Set the number of ranks for multi-device execution.

Setting nbRanks > 1 is only allowed for specific layer types that support multi-device execution:

- IDistCollectiveLayer: Determines output shape for kALL_GATHER and kREDUCE_SCATTER operations.

For attention layers, use IAttention::setNbRanks() instead. For all other layer types, nbRanks must be 1.

Parameters

nbRanks	The number of ranks for multi-device execution. Must be >= 1.

Returns

true if successful, false if the layer type does not support the specified nbRanks value.

See also

getNbRanks()

IAttention::setNbRanks()

9.113.4 Member Data Documentation

9.113.4.1 mLayer

```
apiv::VLayer* nvinfer1::ILayer::mLayer  [protected]
```

The documentation for this class was generated from the following file:

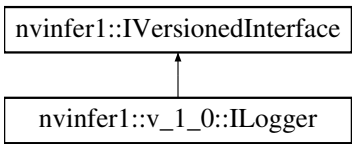
- NvInfer.h

9.114 nvinfer1::v_1_0::ILogger Class Reference

Application-implemented logging interface for the builder, refitter and runtime.

```
#include <NvInferRuntime.h>
```

Inheritance diagram for nvinfer1::v_1_0::ILogger:



Public Types

- enum class Severity : int32_t {
  kINTERNAL_ERROR = 0 , kERROR = 1 , kWARNING = 2 , kINFO = 3 ,
  kVERBOSE = 4 }

  The severity corresponding to a log message.

Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual void log (Severity severity, AsciiChar const *msg) noexcept=0

  A callback implemented by the application to handle logging messages;.
- ILogger ()=default
- ~ILogger () override=default

Additional Inherited Members

9.114.1 Detailed Description

Application-implemented logging interface for the builder, refitter and runtime.

The logger used to create an instance of IBuilder, IRuntime or IRefitter is used for all objects created through that interface. The logger must be valid until all objects created are released.

The Logger object implementation must be thread safe. All locking and synchronization is pushed to the interface implementation and TensorRT does not hold any synchronization primitives when calling the interface functions.

9.114.2 Member Enumeration Documentation

9.114.2.1 Severity

```
enum class nvinfer1::v_1_0::ILogger::Severity : int32_t [strong]
```

The severity corresponding to a log message.

Enumerator

kINTERNAL_ERROR	An internal error has occurred. Execution is unrecoverable.
kERROR	An application error has occurred.
kWARNING	An application error has been discovered, but TensorRT has recovered or fallen back to a default.
kINFO	Informational messages with instructional information.
kVERBOSE	Verbose messages with debugging information.

9.114.3 Constructor & Destructor Documentation

9.114.3.1 ILogger()

```
nvinfer1::v_1_0::ILogger::ILogger ( )  [default]
```

9.114.3.2 ~ILogger()

```
nvinfer1::v_1_0::ILogger::~ILogger ( )  [override], [default]
```

9.114.4 Member Function Documentation

9.114.4.1 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::ILogger::getInterfaceInfo ( ) const  [inline], [override], [virtual],
[noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.114.4.2 log()

```
virtual void nvinfer1::v_1_0::ILogger::log (
            Severity severity,
            AsciiChar const * msg )  [pure virtual], [noexcept]
```

A callback implemented by the application to handle logging messages;.

Parameters

severity	The severity of the message.
msg	A null-terminated log message.

Warning

Loggers used in the safety certified runtime must set a maximum message length and truncate messages exceeding this length. It is up to the implementer of the derived class to define a suitable limit that will prevent buffer overruns, resource exhaustion, and other security vulnerabilities in their implementation. The TensorRT safety certified runtime will never emit messages longer than 1024 bytes.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime, or if the same logger is used for multiple runtimes, builders, or refitters.

The documentation for this class was generated from the following file:

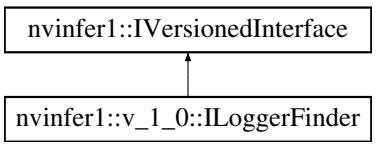
- NvInferRuntime.h

9.115 nvinfer1::v_1_0::ILoggerFinder Class Reference

A virtual base class to find a logger. Allows a plugin to find an instance of a logger if it needs to emit a log message. A pointer to an instance of this class is passed to a plugin shared library on initialization when that plugin is serialized as part of a version-compatible plan. See the plugin chapter in the developer guide for details.

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::ILoggerFinder:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual ILogger * findLogger ()=0

  Get the logger used by the engine or execution context which called the plugin method.

Protected Member Functions

- ~ILoggerFinder () override=default

  Protected: TRT owns ILoggerFinder instances and passes non-owning pointers to plugins.

9.115.1 Detailed Description

A virtual base class to find a logger. Allows a plugin to find an instance of a logger if it needs to emit a log message. A pointer to an instance of this class is passed to a plugin shared library on initialization when that plugin is serialized as part of a version-compatible plan. See the plugin chapter in the developer guide for details.

9.115.2 Constructor & Destructor Documentation

9.115.2.1 ~ILoggerFinder()

```
nvinfer1::v_1_0::ILoggerFinder::~ILoggerFinder ( )  [override], [protected], [default]
```

Protected: TRT owns ILoggerFinder instances and passes non-owning pointers to plugins.

9.115.3 Member Function Documentation

9.115.3.1 findLogger()

```
virtual ILogger * nvinfer1::v_1_0::ILoggerFinder::findLogger ( )  [pure virtual]
```

Get the logger used by the engine or execution context which called the plugin method.

> **Warning**
>
> Must be called from the thread in which the plugin method was called.

Returns

A pointer to the logger.

9.115.3.2 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::ILoggerFinder::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

The documentation for this class was generated from the following file:

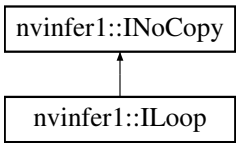
- NvInferRuntime.h

9.116 nvinfer1::ILoop Class Reference

Helper for creating a recurrent subgraph.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::ILoop:



Public Member Functions

- IRecurrenceLayer * addRecurrence (ITensor &initialValue) noexcept

  Create a recurrence layer for this loop with initialValue as its first input.
- ITripLimitLayer * addTripLimit (ITensor &tensor, TripLimit limit) noexcept

  Add a trip-count limiter, based on the given tensor.
- IIteratorLayer * addIterator (ITensor &tensor, int32_t axis=0, bool reverse=false) noexcept

  Return layer that subscripts tensor by loop iteration.
- ILoopOutputLayer * addLoopOutput (ITensor &tensor, LoopOutput outputKind, int32_t axis=0) noexcept

  Make an output for this loop, based on the given tensor.
- void setName (char const *name) noexcept

  Set the name of the loop.
- char const * getName () const noexcept

  Return the name of the loop.

Protected Member Functions

- virtual ~ILoop () noexcept=0

Protected Attributes

- apiv::VLoop * mImpl

9.116.1 Detailed Description

Helper for creating a recurrent subgraph.

An ILoop defines a loop within a network. It supports the implementation of recurrences, which are crucial for iterative computations, such as RNNs for natural language processing and time-series analysis.

The subgraph may directly use tensors defined outside of the ILoop.

9.116.2 Constructor & Destructor Documentation

9.116.2.1 ~ILoop()

```
nvinfer1::ILoop::~ILoop ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.116.3 Member Function Documentation

9.116.3.1 addIterator()

```
IIteratorLayer * nvinfer1::ILoop::addIterator (
            ITensor & tensor,
            int32_t axis = 0,
            bool reverse = false )  [inline], [noexcept]
```

Return layer that subscripts tensor by loop iteration.

For reverse=false, this is equivalent to addGather(tensor, I, 0) where I is a scalar tensor containing the loop iteration number. For reverse=true, this is equivalent to addGather(tensor, M-1-I, 0) where M is the trip count computed from TripLimits of kind kCOUNT.

9.116.3.2 addLoopOutput()

```
ILoopOutputLayer * nvinfer1::ILoop::addLoopOutput (
            ITensor & tensor,
            LoopOutput outputKind,
            int32_t axis = 0 )  [inline], [noexcept]
```

Make an output for this loop, based on the given tensor.

axis is the axis for concatenation (if using outputKind of kCONCATENATE or kREVERSE).

If outputKind is kCONCATENATE or kREVERSE, a second input specifying the concatenation dimension must be added via method ILoopOutputLayer::setInput.

9.116.3.3 addRecurrence()

```
IRecurrenceLayer * nvinfer1::ILoop::addRecurrence (
            ITensor & initialValue )  [inline], [noexcept]
```

Create a recurrence layer for this loop with initialValue as its first input.

IRecurrenceLayer requires exactly two inputs. The 2nd input must be added, via method IRecurrenceLayer::set↩ Input(1,...) before an Engine can be built.

9.116.3.4 addTripLimit()

```
ITripLimitLayer * nvinfer1::ILoop::addTripLimit (
            ITensor & tensor,
            TripLimit limit )  [inline], [noexcept]
```

Add a trip-count limiter, based on the given tensor.

There may be at most one kCOUNT and one kWHILE limiter for a loop. When both trip limits exist, the loop exits when the count is reached or condition is falsified. It is an error to not add at least one trip limiter.

For kCOUNT, the input tensor must be available before the loop starts.

For kWHILE, the input tensor must be the output of a subgraph that contains only layers that are not ITripLimitLayer, IIteratorLayer or ILoopOutputLayer. Any IRecurrenceLayers in the subgraph must belong to the same loop as the ITripLimitLayer. A trivial example of this rule is that the input to the kWHILE is the output of an IRecurrenceLayer for the same loop.

9.116.3.5 getName()

```
char const * nvinfer1::ILoop::getName ( ) const  [inline], [noexcept]
```

Return the name of the loop.

See also

setName()

9.116.3.6 setName()

```
void nvinfer1::ILoop::setName (
            char const * name )  [inline], [noexcept]
```

Set the name of the loop.

The name is used in error diagnostics. This method copies the name string.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

See also

getName()

9.116.4 Member Data Documentation

9.116.4.1 mImpl

```
apiv::VLoop* nvinfer1::ILoop::mImpl  [protected]
```

The documentation for this class was generated from the following file:

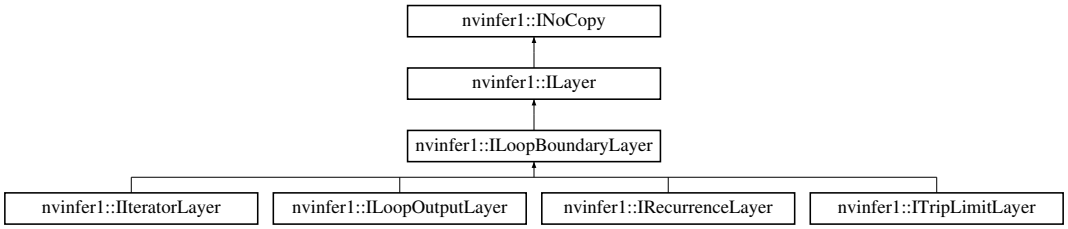
- NvInfer.h

9.117 nvinfer1::ILoopBoundaryLayer Class Reference

This is a base class for Loop boundary layers.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::ILoopBoundaryLayer:



Public Member Functions

- ILoop * getLoop () const noexcept

  Get a pointer to ILoop associated with this boundary layer.

Protected Member Functions

- virtual ~ILoopBoundaryLayer () noexcept=0

Protected Attributes

- apiv::VLoopBoundaryLayer * mBoundary

9.117.1 Detailed Description

This is a base class for Loop boundary layers.

The loop boundary layers are used to define loops within a network, enabling the implementation of recurrences. The boundary layers for a loop are created by class ILoop.

There are four kinds of boundary layers.

- ITripLimitLayer: controls the number of loop iterations.
- IIterationLayer: iterates over an input tensor.
- IRecurrenceLayer: returns an initial value or value from the previous loop iteration.
- ILoopOutputLayer: generates an output tensor from the loop iterations.

9.117.2 Constructor & Destructor Documentation

9.117.2.1 ~ILoopBoundaryLayer()

```
nvinfer1::ILoopBoundaryLayer::~ILoopBoundaryLayer ( )  [inline], [protected], [pure virtual],
[default], [noexcept]
```

9.117.3 Member Function Documentation

9.117.3.1 getLoop()

```
ILoop * nvinfer1::ILoopBoundaryLayer::getLoop ( ) const  [inline], [noexcept]
```

Get a pointer to ILoop associated with this boundary layer.

9.117.4 Member Data Documentation

9.117.4.1 mBoundary

```
apiv::VLoopBoundaryLayer* nvinfer1::ILoopBoundaryLayer::mBoundary  [protected]
```

The documentation for this class was generated from the following file:

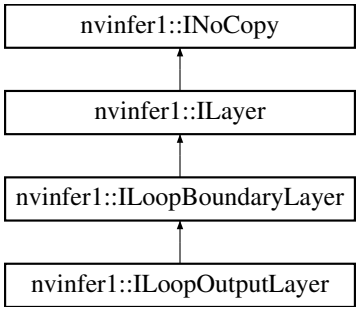
- NvInfer.h

9.118 nvinfer1::ILoopOutputLayer Class Reference

An ILoopOutputLayer is the sole way to get output from a loop.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::ILoopOutputLayer:



Public Member Functions

- LoopOutput getLoopOutput () const noexcept

  Get which kind a loop output has.
- void setAxis (int32_t axis) noexcept

  Set where to insert the contenation axis. Ignored if getLoopOutput() is kLAST_VALUE.
- int32_t getAxis () const noexcept

  Get axis being concatenated over.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~ILoopOutputLayer () noexcept=0

Protected Attributes

- apiv::VLoopOutputLayer * mImpl

9.118.1 Detailed Description

An ILoopOutputLayer is the sole way to get output from a loop.

The first input tensor must be defined inside the loop; the output tensor is outside the loop. The second input tensor, if present, must be defined outside the loop.

If getLoopOutput() is kLAST_VALUE, a single input must be provided, and that input must be from an IRecurrenceLayer in the same loop.

If getLoopOutput() is kCONCATENATE or kREVERSE, a second input must be provided. The second input must be a 0D shape tensor, defined before the loop commences, that specifies the concatenation length of the output.

The output tensor has j more dimensions than the input tensor, where j == 0 if getLoopOutput() is kLAST_VALUE j == 1 if getLoopOutput() is kCONCATENATE or kREVERSE.

9.118.2 Constructor & Destructor Documentation

9.118.2.1 ~ILoopOutputLayer()

```
nvinfer1::ILoopOutputLayer::~ILoopOutputLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.118.3 Member Function Documentation

9.118.3.1 getAxis()

```
int32_t nvinfer1::ILoopOutputLayer::getAxis ( ) const  [inline], [noexcept]
```

Get axis being concatenated over.

9.118.3.2 getLoopOutput()

```
LoopOutput nvinfer1::ILoopOutputLayer::getLoopOutput ( ) const  [inline], [noexcept]
```

Get which kind a loop output has.

9.118.3.3 setAxis()

```
void nvinfer1::ILoopOutputLayer::setAxis (
            int32_t axis )  [inline], [noexcept]
```

Set where to insert the contenation axis. Ignored if getLoopOutput() is kLAST_VALUE.

For example, if the input tensor has dimensions [b,c,d], and getLoopOutput() is kCONCATENATE, the output has four dimensions. Let a be the value of the second input. setAxis(0) causes the output to have dimensions [a,b,c,d]. setAxis(1) causes the output to have dimensions [b,a,c,d]. setAxis(2) causes the output to have dimensions [b,c,a,d]. setAxis(3) causes the output to have dimensions [b,c,d,a]. Default is axis is 0.

9.118.3.4 setInput()

```
void nvinfer1::ILayer::setInput (
            int32_t index,
            ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor Sets the input tensor for the given index. The index must be 0 for a kLAST_VALUE loop output layer. Loop output layer is converted to a kCONCATENATE or kREVERSE loop output layer by calling setInput with an index 1. A kCONCATENATE or kREVERSE loop output layer cannot be converted back to a kLAST_VALUE loop output layer.

For a kCONCATENATE or kREVERSE loop output layer, the values 0 and 1 are valid. The indices in the kCONCATENATE or kREVERSE cases are as follows:

- 0: Contribution to the output tensor. The contribution must come from inside the loop.
- 1: The concatenation length scalar value, must come from outside the loop, as a 0D shape tensor of type Int32 or Int64.

If this function is called with the value 1, then the function getNbInputs() changes from returning 1 to 2.

9.118.4 Member Data Documentation

9.118.4.1 mImpl

```
apiv::VLoopOutputLayer* nvinfer1::ILoopOutputLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

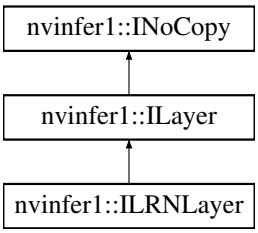
- NvInfer.h

9.119 nvinfer1::ILRNLayer Class Reference

A LRN layer in a network definition.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::ILRNLayer:



Public Member Functions

- void setWindowSize (int64_t windowSize) noexcept
  Set the LRN window size.
- int64_t getWindowSize () const noexcept
  Get the LRN window size.
- void setAlpha (float alpha) noexcept
  Set the LRN alpha value.
- float getAlpha () const noexcept
  Get the LRN alpha value.
- void setBeta (float beta) noexcept
  Set the LRN beta value.
- float getBeta () const noexcept
  Get the LRN beta value.
- void setK (float k) noexcept
  Set the LRN K value.
- float getK () const noexcept
  Get the LRN K value.

Protected Member Functions

- virtual ~ILRNLayer () noexcept=0

Protected Attributes

- apiv::VLRNLayer * mImpl

9.119.1 Detailed Description

A LRN layer in a network definition.

The output size is the same as the input size.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.119.2 Constructor & Destructor Documentation

9.119.2.1 ~ILRNLayer()

```
nvinfer1::ILRNLayer::~ILRNLayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.119.3 Member Function Documentation

9.119.3.1 getAlpha()

```
float nvinfer1::ILRNLayer::getAlpha ( ) const  [inline], [noexcept]
```

Get the LRN alpha value.

See also

setAlpha()

9.119.3.2 getBeta()

```
float nvinfer1::ILRNLayer::getBeta ( ) const  [inline], [noexcept]
```

Get the LRN beta value.

See also

setBeta()

9.119.3.3 getK()

```
float nvinfer1::ILRNLayer::getK ( ) const  [inline], [noexcept]
```

Get the LRN K value.

See also

setK()

9.119.3.4 getWindowSize()

```
int64_t nvinfer1::ILRNLayer::getWindowSize ( ) const  [inline], [noexcept]
```

Get the LRN window size.

See also

getWindowStride()

9.119.3.5 setAlpha()

```
void nvinfer1::ILRNLayer::setAlpha (
            float alpha )  [inline], [noexcept]
```

Set the LRN alpha value.

The valid range is [-1e20, 1e20].

See also

getAlpha()

9.119.3.6 setBeta()

```
void nvinfer1::ILRNLayer::setBeta (
            float beta )  [inline], [noexcept]
```

Set the LRN beta value.

The valid range is [0.01, 1e5f].

See also

getBeta()

9.119.3.7 setK()

```
void nvinfer1::ILRNLayer::setK (
            float k )  [inline], [noexcept]
```

Set the LRN K value.

The valid range is [1e-5, 1e10].

See also

getK()

9.119.3.8 setWindowSize()

```
void nvinfer1::ILRNLayer::setWindowSize (
            int64_t windowSize )  [inline], [noexcept]
```

Set the LRN window size.

The window size must be odd and in the range of [1, 15].

If executing this layer on the DLA, only values in the set, [3, 5, 7, 9], are valid.

See also

setWindowStride()

9.119.4 Member Data Documentation

9.119.4.1 mImpl

```
apiv::VLRNLayer* nvinfer1::ILRNLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

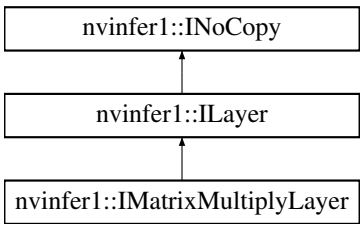
- NvInfer.h

9.120 nvinfer1::IMatrixMultiplyLayer Class Reference

Layer that represents a Matrix Multiplication.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IMatrixMultiplyLayer:



Public Member Functions

- void setOperation (int32_t index, MatrixOperation op) noexcept

  Set the operation for an input tensor.
- MatrixOperation getOperation (int32_t index) const noexcept

  Get the operation for an input tensor.

Protected Member Functions

- virtual ~IMatrixMultiplyLayer () noexcept=0

Protected Attributes

- apiv::VMatrixMultiplyLayer * mImpl

9.120.1 Detailed Description

Layer that represents a Matrix Multiplication.

Let A be op(getInput(0)) and B be op(getInput(1)) where op(x) denotes the corresponding MatrixOperation.

When A and B are matrices or vectors, computes the inner product A * B:

```
matrix * matrix -> matrix
matrix * vector -> vector
vector * matrix -> vector
vector * vector -> scalar
```

Inputs of higher rank are treated as collections of matrices or vectors. The output will be a corresponding collection of matrices, vectors, or scalars.

For a dimension that is not one of the matrix or vector dimensions: If the dimension is 1 for one of the tensors but not the other tensor, the former tensor is broadcast along that dimension to match the dimension of the latter tensor. The number of these extra dimensions for A and B must match.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.120.2 Constructor & Destructor Documentation

9.120.2.1 ~IMatrixMultiplyLayer()

```
nvinfer1::IMatrixMultiplyLayer::~IMatrixMultiplyLayer ( )  [inline], [protected], [pure virtual],
[default], [noexcept]
```

9.120.3 Member Function Documentation

9.120.3.1 getOperation()

```
MatrixOperation nvinfer1::IMatrixMultiplyLayer::getOperation (
            int32_t index ) const  [inline], [noexcept]
```

Get the operation for an input tensor.

Parameters

index	Input tensor number (0 or 1).

See also

setOperation()

9.120.3.2 setOperation()

```
void nvinfer1::IMatrixMultiplyLayer::setOperation (
            int32_t index,
            MatrixOperation op )  [inline], [noexcept]
```

Set the operation for an input tensor.

Parameters

index	Input tensor number (0 or 1).
op	New operation.

See also

getOperation()

9.120.4 Member Data Documentation

9.120.4.1 mImpl

```
apiv::VMatrixMultiplyLayer* nvinfer1::IMatrixMultiplyLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

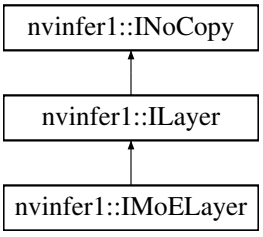
- NvInfer.h

9.121 nvinfer1::IMoELayer Class Reference

A MoE layer in a network definition. Mixture of Experts (MoE) is a collection of experts with each expert specializing in processing different subsets of input data. The key innovation lies in using a Router that selectively activates only the specific experts needed for a given input, rather than engaging the entire neural network for every task.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IMoELayer:



Public Member Functions

- void setGatedWeights (ITensor &fcGateWeights, ITensor &fcUpWeights, ITensor &fcDownWeights, MoEActType activationType) noexcept

  Set the weights of the experts when each expert is a GLU (gated linear unit). In each GLU, there are 3 linear layers and 1 activation function, so this function requires 3 weight tensors and 1 activation type.
- void setGatedBiases (ITensor &fcGateBiases, ITensor &fcUpBiases, ITensor &fcDownBiases) noexcept

  Set the biases of the experts when each expert is a GLU (gated linear unit). In each GLU, there are 3 linear layers, so this function requires 3 bias tensors.
- void setActivationType (MoEActType activationType) noexcept

  Set the activation type for the MoE layer.
- MoEActType getActivationType () const noexcept

  Get the activation type for the MoE layer.
- void setQuantizationStatic (ITensor &fcDownActivationScale, DataType dataType) noexcept

  Configure static quantization after the mul op.
- void setQuantizationDynamicDblQ (ITensor &fcDownActivationDblQScale, DataType dataType, Dims const &blockShape, DataType dynQOutputScaleType) noexcept

  Configure dynamic quantization (with double quantization) after the mul op.
- void setQuantizationToType (DataType type) noexcept

  Set the data type the mul output is quantized to.
- DataType getQuantizationToType () const noexcept

  Get the data type the mul in MoE layer is quantized to.
- void setQuantizationBlockShape (Dims const &blockShape) noexcept

  Set the block shape for the quantization of the Mul output.
- Dims getQuantizationBlockShape () const noexcept

  Get the block shape for the quantization of the Mul output.
- void setDynQOutputScaleType (DataType type) noexcept

  Set the dynamic quantization output scale type.
- DataType getDynQOutputScaleType () const noexcept

  Get the dynamic quantization output scale type.
- void setSwigluParams (float limit, float alpha, float beta) noexcept

  Set the SwiGLU parameters.
- void setSwigluParamLimit (float limit) noexcept

  Set the SwiGLU parameter limit.
- float getSwigluParamLimit () const noexcept

  Get the SwiGLU parameter limit.
- void setSwigluParamAlpha (float alpha) noexcept

  Set the SwiGLU parameter alpha.
- float getSwigluParamAlpha () const noexcept

  Get the SwiGLU parameter alpha.
- void setSwigluParamBeta (float beta) noexcept

  Set the SwiGLU parameter beta.
- float getSwigluParamBeta () const noexcept

  Get the SwiGLU parameter beta.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Set the input of the MoE layer.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Replace an input of this layer with a specific tensor.

Protected Member Functions

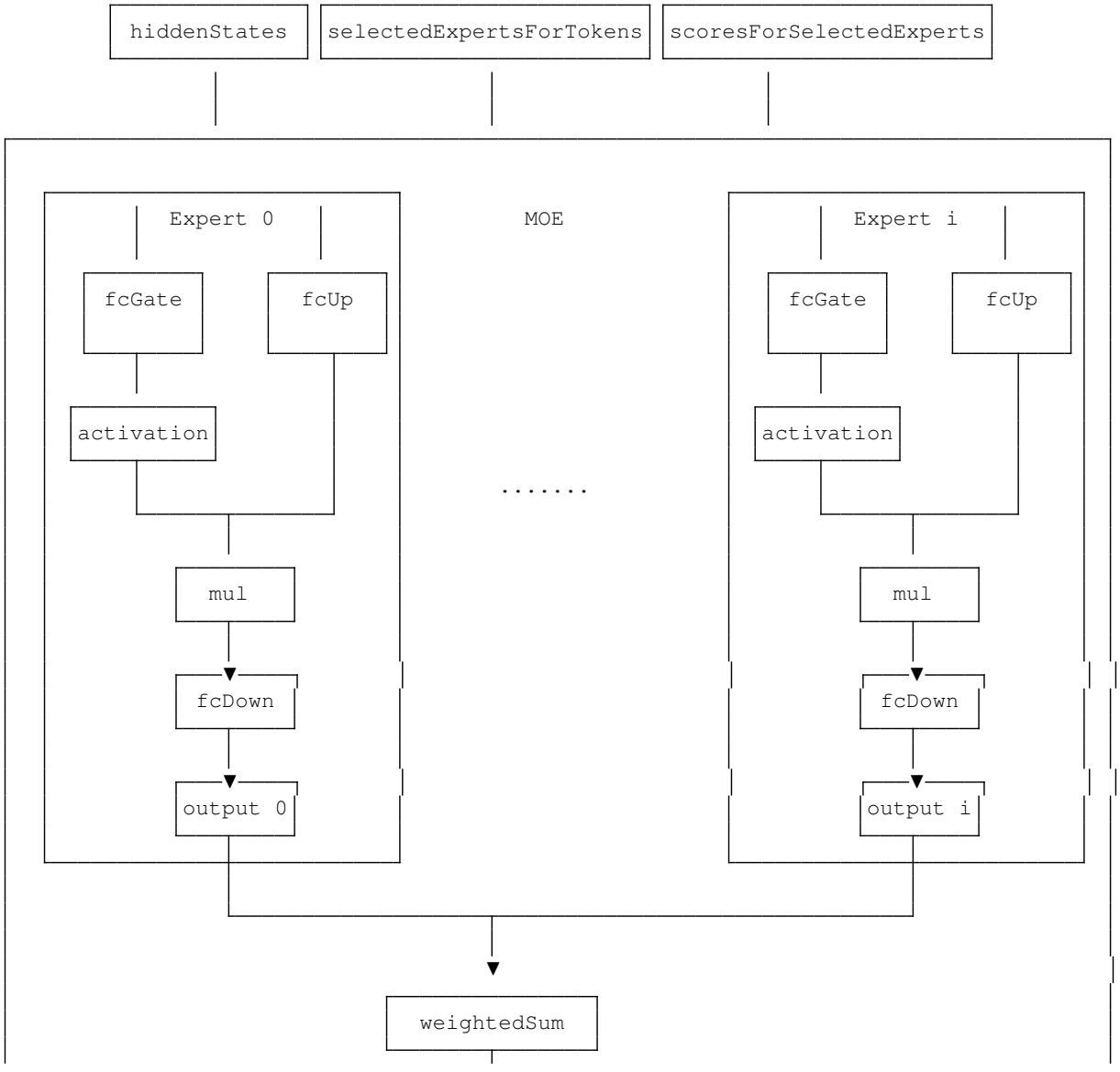
- virtual ~IMoELayer () noexcept=0

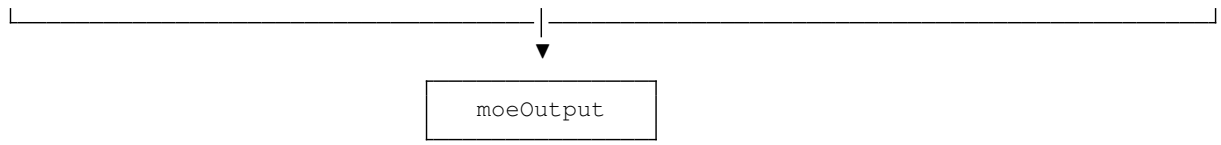
Protected Attributes

- apiv::VMoELayer * mImpl

9.121.1 Detailed Description

A MoE layer in a network definition. Mixture of Experts (MoE) is a collection of experts with each expert specializing in processing different subsets of input data. The key innovation lies in using a Router that selectively activates only the specific experts needed for a given input, rather than engaging the entire neural network for every task.





Definition in the MoE layer: `fcDown`, `fcGate`, `fcUp` are three linear layers. fc(x) = x $*$ w + b, where x is the input, w is the weight, b is the bias, $*$ is the matrix multiplication. `activation` is the activation function. `mul` is the multiplication between the output of fc_up and the output of fc_gate. `weightedSum` is the weighted sum of the output of the experts. `moeOutput` is the output of the MoE layer.

MoE is a collection of experts. Each expert is a GLU (gated linear unit), which consists by `fcGate`, `fcUp`, `fcDown`, `activation`, `mul`.

Definitions and Abbreviations: `batchSize`: batch size `seqLen`: sequence length `hiddenSize`: the size of the hidden states `numExperts`: the number of experts in the MoE layer `moeInterSize`: the intermediate size of the MoE layer `topK`: the number of experts to select for each token

This layer takes several activation inputs:

1. `hiddenStates`: the hidden states of the layer, with shape [batchSize, seqLen, hiddenSize]
2. `selectedExpertsForTokens`: the top K experts selected for each token, with shape [batchSize, seqLen, topK]
3. `scoresForSelectedExperts`: the scales for the selected experts per token, with shape [batchSize, seq↩ Len, topK] The MoE will take the selected experts and the corresponding scales for the selected experts to compute the output.

The weights in the MoE layer:

1. `fcGateWeights` with shape [numExperts, hiddenSize, moeInterSize]: the weight matrix for fcGate
2. `fcUpWeights` with shape [numExperts, hiddenSize, moeInterSize]: the weight matrix for fcUp
3. `fcDownWeights` with shape [numExperts, moeInterSize, hiddenSize]: the weight matrix for fcDown

Several optional inputs are supported:

1. `fcGateBias`: the bias for the fcGate, with shape [numExperts, moeInterSize]
2. `fcUpBias`: the bias for the fcUp, with shape [numExperts, moeInterSize]
3. `fcDownBias`: the bias for the fcDown, with shape [numExperts, hiddenSize] All the bias are none by default. You must either set all the bias or none of them.
4. `activation`: the activation type for the MoE layer, currently only support SILU.

MoE computation process description: For each token, the MoE layer computation process is as follows:

1. Input processing:
    - Receive `hiddenStates`:

- Receive `selectedExpertsForTokens`:
- Receive `scoresForSelectedExperts`:

2. Expert computation for each token:
   - output_i = fcDown(fcUp(hiddenStates) ∗ activation(fcGate(hiddenStates)))
3. Expert output aggregation: For each token, firstly select all the experts that need to be activated to do the computation.
   - calculate the selected expert's output according to expert id in `selectedExpertsForTokens` for each token
   - Weighted sum of each expert's output according to weights in `scoresForSelectedExperts` for each token
   - Final output for the token: moeOutput = Σ(score_i ∗ output_i) The output of MoE has the same shape as the input `hiddenStates`.

Warning

MoE requires Blackwell or Thor GPU architecture (SM 10.x or SM 11.x). SM 12.x is not currently supported. And performance is limited when seqLen > 16.

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.121.2 Constructor & Destructor Documentation

9.121.2.1 ~IMoELayer()

```
nvinfer1::IMoELayer::~IMoELayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.121.3 Member Function Documentation

9.121.3.1 getActivationType()

```
MoEActType nvinfer1::IMoELayer::getActivationType ( ) const  [inline], [noexcept]
```

Get the activation type for the MoE layer.

See also

setActivationType()

Returns

the activation type for the MoE layer.

9.121.3.2 getDynQOutputScaleType()

```
DataType nvinfer1::IMoELayer::getDynQOutputScaleType ( ) const  [inline], [noexcept]
```

Get the dynamic quantization output scale type.

See also

setDynQOutputScaleType()

Returns

the dynamic quantization output scale type.

9.121.3.3 getQuantizationBlockShape()

```
Dims nvinfer1::IMoELayer::getQuantizationBlockShape ( ) const  [inline], [noexcept]
```

Get the block shape for the quantization of the Mul output.

See also

setQuantizationBlockShape()

Returns

the block shape for the quantization of the Mul output.

9.121.3.4 getQuantizationToType()

```
DataType nvinfer1::IMoELayer::getQuantizationToType ( ) const  [inline], [noexcept]
```

Get the data type the mul in MoE layer is quantized to.

See also

setQuantizationToType()

Returns

the data type the mul in MoE layer is quantized to.

9.121.3.5 getSwigluParamAlpha()

```
float nvinfer1::IMoELayer::getSwigluParamAlpha ( ) const  [inline], [noexcept]
```

Get the SwiGLU parameter alpha.

See also

setSwigluParamAlpha()

Returns

the SwiGLU parameter alpha.

9.121.3.6 getSwigluParamBeta()

```
float nvinfer1::IMoELayer::getSwigluParamBeta ( ) const  [inline], [noexcept]
```

Get the SwiGLU parameter beta.

See also

setSwigluParamBeta()

Returns

the SwiGLU parameter beta.

9.121.3.7 getSwigluParamLimit()

```
float nvinfer1::IMoELayer::getSwigluParamLimit ( ) const  [inline], [noexcept]
```

Get the SwiGLU parameter limit.

See also

setSwigluParamLimit()

Returns

the SwiGLU parameter limit.

9.121.3.8 setActivationType()

```
void nvinfer1::IMoELayer::setActivationType (
            MoEActType activationType )  [inline], [noexcept]
```

Set the activation type for the MoE layer.

Parameters

activationType	the activation type for the MoE layer.

See also

getActivationType()

9.121.3.9 setDynQOutputScaleType()

```
void nvinfer1::IMoELayer::setDynQOutputScaleType (
             DataType type ) [inline], [noexcept]
```

Set the dynamic quantization output scale type.

Parameters

type	the dynamic quantization output scale type.

See also

getDynQOutputScaleType()

9.121.3.10 setGatedBiases()

```
void nvinfer1::IMoELayer::setGatedBiases (
             ITensor & fcGateBiases,
             ITensor & fcUpBiases,
             ITensor & fcDownBiases ) [inline], [noexcept]
```

Set the biases of the experts when each expert is a GLU (gated linear unit). In each GLU, there are 3 linear layers, so this function requires 3 bias tensors.

Parameters

fcGateBiases	The biases for the gate-projection layer of all experts in MoE. Shape: [numExperts, moeInterSize].
fcUpBiases	The biases for the up-projection layer of all experts in MoE. Shape: [numExperts, moeInterSize].
fcDownBiases	The biases for the down-projection layer of all experts in MoE. Shape: [numExperts, hiddenSize].

9.121.3.11 setGatedWeights()

```
void nvinfer1::IMoELayer::setGatedWeights (
            ITensor & fcGateWeights,
            ITensor & fcUpWeights,
            ITensor & fcDownWeights,
            MoEActType activationType )  [inline], [noexcept]
```

Set the weights of the experts when each expert is a GLU (gated linear unit). In each GLU, there are 3 linear layers and 1 activation function, so this function requires 3 weight tensors and 1 activation type.

Parameters

fcGateWeights	The weights for the gate-projection layer of all experts in MoE. Shape: [numExperts, hiddenSize, moeInterSize].
fcUpWeights	The weights for the up-projection layer of all experts in MoE. Shape: [numExperts, hiddenSize, moeInterSize].
fcDownWeights	The weights for the down-projection layer of all experts in MoE. Shape: [numExperts, moeInterSize, hiddenSize].
activationType	The activation function to use for the MoE layer. Currently only kSILU is supported.

See also

setActivationType()

getActivationType()

9.121.3.12 setInput() [1/2]

```
void nvinfer1::IMoELayer::setInput (
            int32_t index,
            ITensor & tensor )  [inline], [noexcept]
```

Set the input of the MoE layer.

Parameters

index	the index of the input to modify.
tensor	the new input tensor

The indices are as follows:

Input 0: hiddenStates: the input activations, with shape [batchSize, seqLen, hiddenSize] Input 1: selectedExpertsFor↩ Tokens: the selected experts for tokens, with shape [batchSize, seqLen, topK] Input 2: scoresForSelectedExperts: the scores for selected experts, with shape [batchSize, seqLen, topK]

9.121.3.13 setInput() [2/2]

```
void nvinfer1::ILayer::setInput (
            int32_t index,
            ITensor & tensor )  [inline], [noexcept]
```

Replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor

Except for IFillLayer, ILoopOutputLayer, INMSLayer, IResizeLayer, IShuffleLayer, and ISliceLayer, this method cannot change the number of inputs to a layer. The index argument must be less than the value of getNbInputs().

See comments for overloads of setInput() for layers with special behavior.

9.121.3.14 setQuantizationBlockShape()

```
void nvinfer1::IMoELayer::setQuantizationBlockShape (
            Dims const & blockShape )  [inline], [noexcept]
```

Set the block shape for the quantization of the Mul output.

Parameters

blockShape	the block shape for the quantization of the Mul output.

The shape must have rank 4 and the dimensions representing block sizes for Mul output dimensions (batchSize, seq↩ Len, topK, moeInterSize) respectively. For example, a shape of [1, 1, 1, 16] means block quantization on the last (moeInterSize) axis. -1 means a fully blocked dimension.

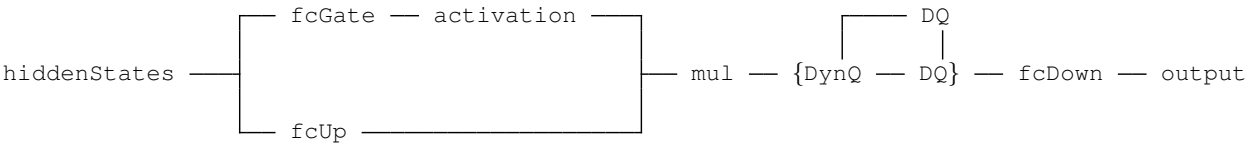
See also

getQuantizationBlockShape()

9.121.3.15 setQuantizationDynamicDblQ()

```
void nvinfer1::IMoELayer::setQuantizationDynamicDblQ (
            ITensor & fcDownActivationDblQScale,
            DataType dataType,
            Dims const & blockShape,
            DataType dynQOutputScaleType )  [inline], [noexcept]
```

Configure dynamic quantization (with double quantization) after the mul op.



When using mul output dynamic quantization (with double quantization), the user must provide:

Parameters

fcDownActivationDblQScale	the double quantization scale tensor.
dataType	the type that the activation is quantized to.
blockShape	the blockShape used in quantization.
dynQOutputScaleType	the data type of the scale tensor. In addition, the user should also insert DynQ/DQ/DQ before the hiddenStates input of the MoE layer. The quantization method must be the same as the quantization method here.

If setQuantizationStatic is called, then previous calls to this function are overridden. If setQuantizationToType, set↩ QuantizationBlockShape or setDynQOutputScaleType is called, previous parameters set by this function are overridden.

See also

setQuantizationToType()

getQuantizationToType()

setQuantizationBlockShape()

getQuantizationBlockShape()

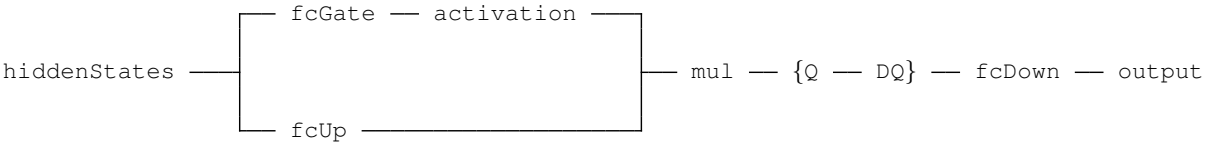
setDynQOutputScaleType()

getDynQOutputScaleType()

9.121.3.16 setQuantizationStatic()

```
void nvinfer1::IMoELayer::setQuantizationStatic (
            ITensor & fcDownActivationScale,
            DataType dataType ) [inline], [noexcept]
```

Configure static quantization after the mul op.



When using mul output static quantization, the user must provide:

Parameters

fcDownActivationScale	the scale tensor.
dataType	the type that the activation is quantized to. In addition, the user should also insert Q/DQ before the hiddenStates input of the MoE layer. The quantization method must be the same as the quantization method here.

If setQuantizationDynamicDblQ is called, then previous calls to this function are overridden. If setQuantizationToType is called, previous parameters set by this function are overridden.

See also

setQuantizationToType()

getQuantizationToType()

9.121.3.17 setQuantizationToType()

```
void nvinfer1::IMoELayer::setQuantizationToType (
             DataType type ) [inline], [noexcept]
```

Set the data type the mul output is quantized to.

Parameters

type	the data type the mul output is quantized to. The type must be one of DataType::kFP8, DataType::kFP4.

Default: `DataType::kFLOAT` which means the MoE layer is not quantized.

See also

getQuantizationToType()

9.121.3.18 setSwigluParamAlpha()

```
void nvinfer1::IMoELayer::setSwigluParamAlpha (
             float alpha ) [inline], [noexcept]
```

Set the SwiGLU parameter alpha.

Parameters

alpha	the SwiGLU parameter alpha.

Default: 1.0

See also

getSwigluParamAlpha()

9.121.3.19 setSwigluParamBeta()

```
void nvinfer1::IMoELayer::setSwigluParamBeta (
            float beta )  [inline], [noexcept]
```

Set the SwiGLU parameter beta.

Parameters

beta	the SwiGLU parameter beta.

Default: 0.0

See also

getSwigluParamBeta()

9.121.3.20 setSwigluParamLimit()

```
void nvinfer1::IMoELayer::setSwigluParamLimit (
            float limit )  [inline], [noexcept]
```

Set the SwiGLU parameter limit.

Parameters

limit	the SwiGLU parameter limit.

Default: +inf

See also

getSwigluParamLimit()

9.121.3.21 setSwigluParams()

```
void nvinfer1::IMoELayer::setSwigluParams (
            float limit,
            float alpha,
            float beta )  [inline], [noexcept]
```

Set the SwiGLU parameters.

Parameters

limit	the SwiGLU parameter limit.
alpha	the SwiGLU parameter alpha.
beta	the SwiGLU parameter beta.

Default: +inf, 1.0, 0.0

See also

setSwigluParamLimit()
getSwigluParamLimit()
setSwigluParamAlpha()
getSwigluParamAlpha()
setSwigluParamBeta()
getSwigluParamBeta()

9.121.4 Member Data Documentation

9.121.4.1 mImpl

```
apiv::VMoELayer* nvinfer1::IMoELayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

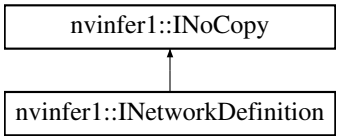
- NvInfer.h

9.122 nvinfer1::INetworkDefinition Class Reference

A network definition for input to the builder.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::INetworkDefinition:



Public Member Functions

- virtual ~INetworkDefinition () noexcept=0
- ITensor * addInput (char const *name, DataType type, Dims const &dimensions) noexcept

  Add an input tensor to the network.
- void markOutput (ITensor &tensor) noexcept

  Mark a tensor as a network output.
- bool markDebug (ITensor &tensor) noexcept

  Mark a tensor as a debug tensor.
- bool unmarkDebug (ITensor &tensor) noexcept

  Unmark a tensor as a debug tensor.
- bool isDebugTensor (ITensor const &tensor) const noexcept

  Check if a tensor is marked as debug tensor.
- bool markUnfusedTensorsAsDebugTensors () noexcept

  Mark unfused tensors as debug tensors.
- bool unmarkUnfusedTensorsAsDebugTensors () noexcept

  Undo the marking of unfused tensors as debug tensors.
- IActivationLayer * addActivation (ITensor &input, ActivationType type) noexcept

  Add an activation layer to the network.
- ILRNLayer * addLRN (ITensor &input, int64_t window, float alpha, float beta, float k) noexcept

  Add a LRN layer to the network.
- IScaleLayer * addScale (ITensor &input, ScaleMode mode, Weights shift, Weights scale, Weights power) noexcept

  Add a Scale layer to the network.
- ISoftMaxLayer * addSoftMax (ITensor &input) noexcept

  Add a SoftMax layer to the network.
- IConcatenationLayer * addConcatenation (ITensor *const *inputs, int32_t nbInputs) noexcept

  Add a concatenation layer to the network.
- IElementWiseLayer * addElementWise (ITensor &input1, ITensor &input2, ElementWiseOperation op) noexcept

  Add an elementwise layer to the network.
- IUnaryLayer * addUnary (ITensor &input, UnaryOperation operation) noexcept

  Add a unary layer to the network.

- IShuffleLayer * addShuffle (ITensor &input) noexcept

  Add a shuffle layer to the network.
- IOneHotLayer * addOneHot (ITensor &indices, ITensor &values, ITensor &depth, int32_t axis) noexcept

  Add a OneHot layer to the network.
- int32_t getNbLayers () const noexcept

  Get the number of layers in the network.
- ILayer * getLayer (int32_t index) const noexcept

  Get the layer specified by the given index.
- int32_t getNbInputs () const noexcept

  Get the number of inputs in the network.
- ITensor * getInput (int32_t index) const noexcept

  Get the input tensor specified by the given index.
- int32_t getNbOutputs () const noexcept

  Get the number of outputs in the network.
- ITensor * getOutput (int32_t index) const noexcept

  Get the output tensor specified by the given index.
- IReduceLayer * addReduce (ITensor &input, ReduceOperation operation, uint32_t reduceAxes, bool keep↩ Dimensions) noexcept

  Add a reduce layer to the network.
- TRT_DEPRECATED ITopKLayer * addTopK (ITensor &input, TopKOperation op, int32_t k, uint32_t reduce↩ Axes) noexcept

  Add a TopK layer to the network.
- ITopKLayer * addTopK (ITensor &input, TopKOperation op, int32_t k, uint32_t reduceAxes, DataType indices↩ Type) noexcept

  Add a TopK layer to the network.
- IGatherLayer * addGather (ITensor &data, ITensor &indices, int32_t axis) noexcept

  Add gather with mode GatherMode::kDEFAULT and specified axis and nbElementWiseDims=0.
- IGatherLayer * addGatherV2 (ITensor &data, ITensor &indices, GatherMode mode) noexcept

  Add gather with specified mode, axis=0 and nbElementWiseDims=0.
- IRaggedSoftMaxLayer * addRaggedSoftMax (ITensor &input, ITensor &bounds) noexcept

  Add a RaggedSoftMax layer to the network.
- IMatrixMultiplyLayer * addMatrixMultiply (ITensor &input0, MatrixOperation op0, ITensor &input1, MatrixOperation op1) noexcept

  Add a MatrixMultiply layer to the network.
- TRT_DEPRECATED INonZeroLayer * addNonZero (ITensor &input) noexcept

  Add a nonzero layer to the network.
- INonZeroLayer * addNonZero (ITensor &input, DataType indicesType) noexcept

  Add a nonzero layer to the network.
- IConstantLayer * addConstant (Dims const &dimensions, Weights weights) noexcept

  Add a constant layer to the network.
- IIdentityLayer * addIdentity (ITensor &input) noexcept

  Add an identity layer.
- ICastLayer * addCast (ITensor &input, DataType toType) noexcept

  Add a cast layer.
- void removeTensor (ITensor &tensor) noexcept

  remove a tensor from the network definition.
- void unmarkOutput (ITensor &tensor) noexcept

unmark a tensor as a network output.

- TRT_DEPRECATED IPluginV2Layer * addPluginV2 (ITensor *const *inputs, int32_t nbInputs, IPluginV2 &plugin) noexcept

  Add a plugin layer to the network using the IPluginV2 interface.
- IPluginV3Layer * addPluginV3 (ITensor *const *inputs, int32_t nbInputs, ITensor *const *shapeInputs, int32_t nbShapeInputs, IPluginV3 &plugin) noexcept

  Add a plugin layer implementing the IPluginV3 interface to the network.
- ISliceLayer * addSlice (ITensor &input, Dims const &start, Dims const &size, Dims const &stride) noexcept

  Add a slice layer to the network.
- void setName (char const *name) noexcept

  Sets the name of the network.
- char const * getName () const noexcept

  Returns the name associated with the network.
- IShapeLayer * addShape (ITensor &input) noexcept

  Add a shape layer to the network.
- TRT_DEPRECATED bool hasImplicitBatchDimension () const noexcept

  Query whether the network was created with an implicit batch dimension.
- NetworkDefinitionCreationFlags getFlags () const noexcept

  Get the network definition creation flags for this network definition object. Defaults to 0.
- bool getFlag (NetworkDefinitionCreationFlag networkDefinitionCreationFlag) const noexcept

  Returns true if the network definition creation flag is set.
- bool markOutputForShapes (ITensor &tensor) noexcept

  Enable tensor's value to be computed by IExecutionContext::getShapeBinding.
- bool unmarkOutputForShapes (ITensor &tensor) noexcept

  Undo markOutputForShapes.
- IParametricReLULayer * addParametricReLU (ITensor &input, ITensor &slope) noexcept

  Add a parametric ReLU layer to the network.
- IConvolutionLayer * addConvolutionNd (ITensor &input, int64_t nbOutputMaps, Dims const &kernelSize, Weights kernelWeights, Weights biasWeights) noexcept

  Add a multi-dimension convolution layer to the network.
- IPoolingLayer * addPoolingNd (ITensor &input, PoolingType type, Dims const &windowSize) noexcept

  Add a multi-dimension pooling layer to the network.
- IDeconvolutionLayer * addDeconvolutionNd (ITensor &input, int64_t nbOutputMaps, Dims kernelSize, Weights kernelWeights, Weights biasWeights) noexcept

  Add a multi-dimension deconvolution layer to the network.
- IScaleLayer * addScaleNd (ITensor &input, ScaleMode mode, Weights shift, Weights scale, Weights power, int32_t channelAxis) noexcept

  Add a multi-dimension scale layer to the network.
- IResizeLayer * addResize (ITensor &input) noexcept

  Add a resize layer to the network.
- ILoop * addLoop () noexcept

  Add a loop to the network.
- IIfConditional * addIfConditional () noexcept

  Add an if-then-else to the network.
- ISelectLayer * addSelect (ITensor &condition, ITensor &thenInput, ITensor &elseInput) noexcept

  Add a select layer to the network.
- IAssertionLayer * addAssertion (ITensor &condition, char const *message) noexcept

  Add an assertion layer to the network.

- IFillLayer * addFill (Dims const &dimensions, FillOperation op, DataType outputType) noexcept

  Add a fill layer to the network.
- IPaddingLayer * addPaddingNd (ITensor &input, Dims const &prePadding, Dims const &postPadding) noexcept

  Add a padding layer to the network. Only 2D padding is currently supported.
- bool setWeightsName (Weights weights, char const *name) noexcept

  Associate a name with all current uses of the given weights.
- void setErrorRecorder (IErrorRecorder *recorder) noexcept

  Set the ErrorRecorder for this interface.
- IErrorRecorder * getErrorRecorder () const noexcept

  get the ErrorRecorder assigned to this interface.
- IDequantizeLayer * addDequantize (ITensor &input, ITensor &scale, DataType outputType) noexcept

  Add a dequantization layer to the network.
- IScatterLayer * addScatter (ITensor &data, ITensor &indices, ITensor &updates, ScatterMode mode) noexcept

  Add a Scatter layer to the network with specified mode and axis=0.
- IQuantizeLayer * addQuantize (ITensor &input, ITensor &scale, DataType outputType) noexcept

  Add a quantization layer to the network.
- TRT_DEPRECATED IDynamicQuantizeLayer * addDynamicQuantize (ITensor &input, int32_t axis, int32_t blockSize, DataType outputType, DataType scaleType) noexcept

  Add a dynamic quantization layer to the network.
- IDynamicQuantizeLayer * addDynamicQuantizeV2 (ITensor &input, Dims const &blockShape, DataType outputType, DataType scaleType) noexcept

  Add a dynamic quantization layer to the network.
- IEinsumLayer * addEinsum (ITensor *const *inputs, int32_t nbInputs, char const *equation) noexcept

  Add an Einsum layer to the network.
- IGridSampleLayer * addGridSample (ITensor &input, ITensor &grid) noexcept

  Add a GridSample layer to the network.
- TRT_DEPRECATED INMSLayer * addNMS (ITensor &boxes, ITensor &scores, ITensor &maxOutputBoxesPerClass) noexcept

  Add a non-maximum suppression layer to the network.
- INMSLayer * addNMS (ITensor &boxes, ITensor &scores, ITensor &maxOutputBoxesPerClass, DataType indicesType) noexcept

  Add a non-maximum suppression layer to the network.
- IReverseSequenceLayer * addReverseSequence (ITensor &input, ITensor &sequenceLens) noexcept

  Add a ReverseSequence layer to the network.
- TRT_DEPRECATED INormalizationLayer * addNormalization (ITensor &input, ITensor &scale, ITensor &bias, uint32_t axesMask) noexcept

  Add a normalization layer to the network.
- ICumulativeLayer * addCumulative (ITensor &input, ITensor &axis, CumulativeOperation operation, bool exclusive, bool reverse) noexcept

  Add a cumulative layer to the network.
- TRT_DEPRECATED IAttention * addAttention (ITensor &query, ITensor &key, ITensor &value, AttentionNormalizationOp normOp, bool causal) noexcept

  Add an attention to the network.
- IAttention * addAttentionV2 (ITensor &query, ITensor &key, ITensor &value, AttentionNormalizationOp normOp, CausalMaskKind causalKind) noexcept

  Add an attention to the network with explicit causal mask kind.
- IRotaryEmbeddingLayer * addRotaryEmbedding (ITensor &input, ITensor &cosCache, ITensor &sinCache, bool interleaved, int32_t rotaryEmbeddingDim) noexcept

Add a Rotary Position Embedding (RoPE) layer to the network.

- IKVCacheUpdateLayer * addKVCacheUpdate (ITensor &cache, ITensor &update, ITensor &writeIndices, KVCacheMode cacheMode) noexcept

  Add a KVCacheUpdate layer to the network.
- IMoELayer * addMoE (ITensor &hiddenStates, ITensor &selectedExpertsForTokens, ITensor &scoresFor↩ SelectedExperts) noexcept

  Add a MoE (Mixture of Experts) layer to the network.
- TRT_NODISCARD IDistCollectiveLayer * addDistCollective (ITensor &input, CollectiveOperation dist↩ CollectiveOp, ReduceOperation reduceOp, int64_t root, int64_t *groups, int64_t groupSize) noexcept

  Add a DistCollective layer to the network.
- virtual IBuilder & getBuilder () const noexcept

  Return the builder from which this INetworkDefinition was created.
- bool markWeightsRefittable (char const *name) noexcept

  Mark weights as refittable when the builder flag kREFIT_INDIVIDUAL is set.
- bool unmarkWeightsRefittable (char const *name) noexcept

  Unmark weights as refittable when the builder flag kREFIT_INDIVIDUAL is set.
- bool areWeightsMarkedRefittable (char const *name) const noexcept

  Whether the weight has been marked as refittable.
- ISqueezeLayer * addSqueeze (ITensor &input, ITensor &axes) noexcept

  Add a squeeze layer to the network.
- IUnsqueezeLayer * addUnsqueeze (ITensor &input, ITensor &axes) noexcept

  Add an unsqueeze layer to the network.
- TRT_NODISCARD INormalizationLayer * addNormalizationV2 (ITensor &input, ITensor &scale, ITensor &bias, uint32_t axesMask) noexcept

  Add a normalization layer to the network.

Protected Attributes

- apiv::VNetworkDefinition * mImpl

Additional Inherited Members

9.122.1 Detailed Description

A network definition for input to the builder.

A network definition defines the structure of the network, and combined with a IBuilderConfig, is built into an engine using an IBuilder. An INetworkDefinition can have all dimensions explicit, full dims mode, in the network definition. The former mode, i.e. the implicit batch size mode, has been deprecated.

A network with implicit batch dimensions returns the dimensions of a layer without the implicit dimension, and instead the batch is specified at execute/enqueue time. If the network has all dimensions specified, then the first dimension follows elementwise broadcast rules: if it is 1 for some inputs and is some value N for all other inputs, then the first dimension of each output is N, and the inputs with 1 for the first dimension are broadcast. Having divergent batch sizes across inputs to a layer is not supported.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.122.2 Constructor & Destructor Documentation

9.122.2.1 ~INetworkDefinition()

```
nvinfer1::INetworkDefinition::~INetworkDefinition ( )  [inline], [pure virtual], [default], [noexcept]
```

9.122.3 Member Function Documentation

9.122.3.1 addActivation()

```
IActivationLayer * nvinfer1::INetworkDefinition::addActivation (
            ITensor & input,
            ActivationType type )  [inline], [noexcept]
```

Add an activation layer to the network.

Parameters

input	The input tensor to the layer.
type	The type of activation function to apply.

Note that the setAlpha() and setBeta() methods must be used on the output for activations that require these parameters.

See also

IActivationLayer ActivationType

Warning

Int32 and Int64 are valid only for activation type kRELU.

Returns

The new activation layer, or nullptr if it could not be created.

9.122.3.2 addAssertion()

```
IAssertionLayer * nvinfer1::INetworkDefinition::addAssertion (
            ITensor & condition,
            char const * message )  [inline], [noexcept]
```

Add an assertion layer to the network.

Parameters

condition	The input tensor to the layer.
message	A message to print if the assertion fails.

See also

IAssertionLayer

Returns

The new assertion layer, or nullptr if it could not be created.

The input tensor must be a boolean shape tensor.

9.122.3.3 addAttention()

```
TRT_DEPRECATED IAttention * nvinfer1::INetworkDefinition::addAttention (
            ITensor & query,
            ITensor & key,
            ITensor & value,
            AttentionNormalizationOp normOp,
            bool causal )  [inline], [noexcept]
```

Add an attention to the network.

Parameters

query	A 3D or 4D input query tensor to the layer.
key	A 3D or 4D input key tensor to the layer.
value	A 3D or 4D input value tensor to the layer.
normOp	The normalization operation to perform.
causal	Use causal inference or not. When true, uses kUPPER_LEFT causal masking.

For padded (BHND) form, query must have shape [batchSize, numHeadsQuery, sequenceLengthQuery, dimHead]. For packed (NHD) form, query must have shape [totalTokens, numHeadsQuery, dimHead]. key and value follow the same convention based on their form. Use IAttention::setQueryForm() and IAttention::setKeyValueForm() to configure the tensor layout. normOp defaults to kSOFTMAX isCausal defaults to false.

By default, IAttention is not decomposable and TensorRT will try to use a single fused kernel, which may be more efficient than if the subgraph is expressed without IAttention. Setting the IAttention to decomposable=True can allow IAttention to be to use multiple kernels if no fused kernel support found.

See also

IAttention

Deprecated Deprecated in TensorRT 10.16. Superseded by addAttentionV2 with CausalMaskKind parameter.

Returns

The new attention, or nullptr if it could not be created.

9.122.3.4 addAttentionV2()

```
IAttention * nvinfer1::INetworkDefinition::addAttentionV2 (
             ITensor & query,
             ITensor & key,
             ITensor & value,
             AttentionNormalizationOp normOp,
             CausalMaskKind causalKind )  [inline], [noexcept]
```

Add an attention to the network with explicit causal mask kind.

Parameters

query	A 4d input query tensor to the layer.
key	A 4d input key tensor to the layer.
value	A 4d input value tensor to the layer.
normOp	The normalization operation to perform.
causalKind	The causal mask alignment orientation. Use kNONE for no causal masking, kUPPER_LEFT for diagonal anchored at upper-left corner (legacy default), or kLOWER_RIGHT for diagonal anchored at lower-right corner (for LLM generation with s_q != s_kv).

query must have shape [batchSize, numHeadsQuery, sequenceLengthQuery, dimHead]. key and value must have shape [batchSize, numHeadsKeyValue, sequenceLengthKeyValue, dimHead]. pastKey and pastValue must have shape [batchSize, numHeadsKeyValue, sequenceLengthKeyValue, dimHead]. normOp defaults to kSOFTMAX, causalKind defaults to kNONE.

By default, IAttention is not decomposable and TensorRT will try to use a single fused kernel, which may be more efficient than if the subgraph is expressed without IAttention. Setting the IAttention to decomposable=True can allow IAttention to be to use multiple kernels if no fused kernel support found.

See also

IAttention, CausalMaskKind

Returns

The new attention, or nullptr if it could not be created.

9.122.3.5 addCast()

```
ICastLayer * nvinfer1::INetworkDefinition::addCast (
            ITensor & input,
            DataType toType )  [inline], [noexcept]
```

Add a cast layer.

Parameters

input	The input tensor to the layer.
toType	The DataType of the output tensor

See also

ICastLayer

Returns

The new cast layer, or nullptr if it could not be created.

9.122.3.6 addConcatenation()

```
IConcatenationLayer * nvinfer1::INetworkDefinition::addConcatenation (
            ITensor *const * inputs,
            int32_t nbInputs )  [inline], [noexcept]
```

Add a concatenation layer to the network.

Parameters

inputs	The input tensors to the layer.
nbInputs	The number of input tensors.

See also

IConcatenationLayer

Returns

The new concatenation layer, or nullptr if it could not be created.

Warning

All tensors must have the same dimensions except along the concatenation axis.

9.122.3.7 addConstant()

```
IConstantLayer * nvinfer1::INetworkDefinition::addConstant (
            Dims const & dimensions,
            Weights weights )  [inline], [noexcept]
```

Add a constant layer to the network.

Parameters

dimensions	The dimensions of the constant.
weights	The constant value, represented as weights.

See also

IConstantLayer

Returns

The new constant layer, or nullptr if it could not be created.

If weights.type is DataType::kINT32, the output is a tensor of 32-bit indices. Otherwise the output is a tensor of real values and the output type will be follow TensorRT's normal precision rules.

If a wildcard dimension is used, the volume of the runtime dimensions must equal the number of weights specified.

Warning

DataType::kUINT8 not supported.

9.122.3.8 addConvolutionNd()

```
IConvolutionLayer * nvinfer1::INetworkDefinition::addConvolutionNd (
            ITensor & input,
            int64_t nbOutputMaps,
            Dims const & kernelSize,
            Weights kernelWeights,
            Weights biasWeights )  [inline], [noexcept]
```

Add a multi-dimension convolution layer to the network.

Parameters

input	The input tensor to the convolution.
nbOutputMaps	The number of output feature maps for the convolution.
kernelSize	The multi-dimensions of the convolution kernel.
kernelWeights	The kernel weights for the convolution.
biasWeights	The bias weights for the convolution. Weights{} represents no bias.

See also

IConvolutionLayer

Warning

It is an error to specify a wildcard value for the 'C' dimension of the input tensor.

Int32 tensors are not valid input tensors.

Only 2D or 3D convolution is supported.

Returns

The new convolution layer, or nullptr if it could not be created.

9.122.3.9 addCumulative()

```
ICumulativeLayer * nvinfer1::INetworkDefinition::addCumulative (
            ITensor & input,
            ITensor & axis,
            CumulativeOperation operation,
            bool exclusive,
            bool reverse )  [inline], [noexcept]
```

Add a cumulative layer to the network.

Parameters

input	The input tensor to the layer.
axis	The axis tensor to apply the cumulative operation on. Currently, it must be a build-time constant 0D shape tensor and must be in the range [-rank(input), rank(input)-1]. Negative value means counting dimensions from the back.
operation	The reduction operation to perform.
exclusive	The boolean that specifies whether it is an exclusive cumulative or inclusive cumulative.
reverse	The boolean that specifies whether the cumulative operation should be applied backward.

The cumulative layer works by performing the specified cumulative `operation` to the tensor `input` on the axis specified by `axis`.

See also

ICumulativeLayer

Returns

The new cumulative layer, or nullptr if it could not be created.

9.122.3.10 addDeconvolutionNd()

```
IDeconvolutionLayer * nvinfer1::INetworkDefinition::addDeconvolutionNd (
            ITensor & input,
            int64_t nbOutputMaps,
            Dims kernelSize,
            Weights kernelWeights,
            Weights biasWeights )  [inline], [noexcept]
```

Add a multi-dimension deconvolution layer to the network.

Parameters

input	The input tensor to the layer.
nbOutputMaps	The number of output feature maps.
kernelSize	The multi-dimensions of the deconvolution kernel.
kernelWeights	The kernel weights for the deconvolution.
biasWeights	The bias weights for the deconvolution. Weights{} represents no bias.

See also

IDeconvolutionLayer

Warning

It is an error to specify a wildcard value for the 'C' dimension of the input tensor.

Int32 tensors are not valid input tensors.

Only 2D or 3D deconvolution is supported.

Returns

The new deconvolution layer, or nullptr if it could not be created.

9.122.3.11 addDequantize()

```
IDequantizeLayer * nvinfer1::INetworkDefinition::addDequantize (
            ITensor & input,
            ITensor & scale,
            DataType outputType ) [inline], [noexcept]
```

Add a dequantization layer to the network.

Parameters

input	The input tensor to be dequantized.
scale	A tensor with the scale value.
outputType	Output tensor data type.

See also

IDequantizeLayer

`input` tensor data type must be DataType::kINT8, DataType::kFP8, DataType::kINT4 or DataType::kFP4. `scale` tensor data type must be one of the following: DataType::kFLOAT (default), DataType::kHALF, DataType::kBF16 or DataType::kE8M0 (for MXFP8 quantization). `outputType` output tensor data type must be DataType::kFLOAT (default), DataType::kHALF or DataType::kBF16. Future calls to set output type using setToType or setOutputType must be consistent. For strongly typed networks, if the scale type is DataType::kHALF or DataType::kBF16 the output type must match.

Returns

The new quantization layer, or nullptr if it could not be created.

9.122.3.12 addDistCollective()

```
TRT_NODISCARD IDistCollectiveLayer * nvinfer1::INetworkDefinition::addDistCollective (
            ITensor & input,
            CollectiveOperation distCollectiveOp,
            ReduceOperation reduceOp,
            int64_t root,
            int64_t * groups,
            int64_t groupSize )  [inline], [noexcept]
```

Add a DistCollective layer to the network.

Parameters

input	The input tensor to the layer.
distCollectiveOp	The collective operation to perform. See CollectiveOperation for valid values.
reduceOp	The reduction operation to perform, in case the collective operation is reduction type: kREDUCE, kREDUCE_SCATTER or kALL_REDUCE. See ReduceOperation for valid values. Use ReduceOperation::kNONE for a CollectiveOperation which does not need a ReduceOperation
root	The root rank of the collective operation. Some CollectiveOperations require specifying a root rank, with the following semantics: • kBROADCAST: the root rank sends, all other ranks receive data • kREDUCE: the root rank receives reduced data, the other ranks send data • kGATHER: the root rank receives data gathered from all ranks • kSCATTER: the root rank distributes data to all ranks For operations that do not use a root rank (kALL_REDUCE, kALL_GATHER, kREDUCE_SCATTER, kALL_TO_ALL), the `root` parameter is ignored. Use `root = -1` as the recommended sentinel value when constructing the layer to make this explicit.
groups	Pointer to a flat array of rank IDs in the communicator that defines a single group for this layer. The DistCollective runner treats this array as the ordered list of participating ranks; only those ranks take part in the collective, and the order defines the group-local rank (used to remap the root for root-based ops).
groupSize	The number of elements in the groups array. If groupSize is 0, all ranks participate and groups can be nullptr.

See also

IDistCollectiveLayer

Returns

The new DistCollective layer, or nullptr if it could not be created.

9.122.3.13 addDynamicQuantize()

```
TRT_DEPRECATED IDynamicQuantizeLayer * nvinfer1::INetworkDefinition::addDynamicQuantize (
            ITensor & input,
            int32_t axis,
            int32_t blockSize,
            DataType outputType,
            DataType scaleType )  [inline], [noexcept]
```

Add a dynamic quantization layer to the network.

This layer performs dynamic block quantization of its input tensor and outputs the quantized data and the computed block scale-factors. The blocked axis dimension size must be divisible by the block size.

Parameters

input	The input tensor to be quantized. Its data type must be one of DataType::kFLOAT, DataType::kHALF, or DataType::kBF16. Currently only 2D and 3D inputs are supported.
axis	The axis that is sliced into blocks. The axis must be the last or second to last dimension.
blockSize	The number of elements that are quantized using a shared scale factor. Valid values are 16 (NVFP4 quantization) and 32 (MXFP8 quantization).
outputType	The data type of the quantized output tensor, must be DataType::kFP4 (NVFP4 quantization) or DataType::kFP8 (MXFP8 quantization). Future calls to set output type using setToType or setOutputType must be consistent.
scaleType	The data type of the scale factor used for quantizing the input data, must be DataType::kFP8 (NVFP4 quantization) or DataType::kE8M0 (MXFP8 quantization).

Returns

The new dynamic quantization layer, or nullptr if it could not be created.

See also

IDynamicQuantizeLayer

9.122.3.14 addDynamicQuantizeV2()

```
IDynamicQuantizeLayer * nvinfer1::INetworkDefinition::addDynamicQuantizeV2 (
            ITensor & input,
            Dims const & blockShape,
            DataType outputType,
            DataType scaleType )  [inline], [noexcept]
```

Add a dynamic quantization layer to the network.

This layer performs dynamic block quantization of its input tensor and outputs the quantized data and the computed block scale factors.

Parameters

input	The input tensor to be quantized. Its data type must be one of DataType::kFLOAT, DataType::kHALF, or DataType::kBF16.
blockShape	Defines the block shape for the quantization. Must match the input tensor rank.
outputType	The data type of the quantized output tensor, must be DataType::kFP4, DataType::kFP8 or DataType::kINT8. Future calls to set output type using setToType or setOutputType must be consistent.
scaleType	The data type of the scale factor used for quantizing the input data, must be DataType::kFP8, DataType::kE8M0 or DataType::kFLOAT.

Returns

The new dynamic quantization layer, or nullptr if it could not be created.

See also

IDynamicQuantizeLayer

9.122.3.15 addEinsum()

```
IEinsumLayer * nvinfer1::INetworkDefinition::addEinsum (
            ITensor *const * inputs,
            int32_t nbInputs,
            char const * equation )  [inline], [noexcept]
```

Add an Einsum layer to the network.

Parameters

inputs	The input tensors to the layer.
nbInputs	The number of input tensors.
equation	The equation of the layer

See also

IEinsumLayer

Returns

The new Einsum layer, or nullptr if it could not be created.

9.122.3.16 addElementWise()

```
IElementWiseLayer * nvinfer1::INetworkDefinition::addElementWise (
            ITensor & input1,
            ITensor & input2,
            ElementWiseOperation op )  [inline], [noexcept]
```

Add an elementwise layer to the network.

Parameters

input1	The first input tensor to the layer.
input2	The second input tensor to the layer.
op	The binary operation that the layer applies.

The input tensors must have the same rank and compatible type. Two types are compatible if they are the same type or are both in the set {kFLOAT, kHALF}. For each dimension, their lengths must match, or one of them must be one. In the latter case, the tensor is broadcast along that axis.

The output tensor has the same rank as the inputs. For each dimension, its length is the maximum of the lengths of the corresponding input dimension.

The inputs are shape tensors if the output is a shape tensor.

See also

IElementWiseLayer

Returns

The new elementwise layer, or nullptr if it could not be created.

9.122.3.17 addFill()

```
IFillLayer * nvinfer1::INetworkDefinition::addFill (
            Dims const & dimensions,
            FillOperation op,
            DataType outputType )  [inline], [noexcept]
```

Add a fill layer to the network.

Parameters

dimensions	The output tensor dimensions if input 0 is missing.
op	The fill operation that the layer applies.
outputType	Optional output tensor data type, must be DataType::kFLOAT, DataType::kHALF, DataType::kINT32, or DataType::kINT64. This parameter is only used for static alpha/beta. Future calls to set output type using setToType or setOutputType must be consistent.

Warning

For FillOperation::kLINSPACE, dimensions.nbDims must be 1 for static start/delta. If delta is provided as a 1D tensor, the length of delta must match dimensions.nbDims.

This layer is non-deterministic across subsequent calls as the same inputs will produce different output tensors if `op` is either FillOperation::kRANDOM_UNIFORM or FillOperation::kRANDOM_NORMAL due to random state being shared across calls. The output tensors generated are deterministic when starting from the same initial state.

See also

IFillLayer

Returns

The new fill layer, or nullptr if it could not be created.

9.122.3.18 addGather()

```
IGatherLayer * nvinfer1::INetworkDefinition::addGather (
            ITensor & data,
            ITensor & indices,
            int32_t axis )  [inline], [noexcept]
```

Add gather with mode GatherMode::kDEFAULT and specified axis and nbElementWiseDims=0.

Parameters

data	The tensor to gather values from.
indices	The tensor to get indices from to populate the output tensor.
axis	The axis in the data tensor to gather on.

See also

IGatherLayer

Returns

The new gather layer, or nullptr if it could not be created.

9.122.3.19 addGatherV2()

```
IGatherLayer * nvinfer1::INetworkDefinition::addGatherV2 (
            ITensor & data,
            ITensor & indices,
            GatherMode mode )  [inline], [noexcept]
```

Add gather with specified mode, axis=0 and nbElementWiseDims=0.

Parameters

data	The tensor to gather values from.
indices	The tensor to get indices from to populate the output tensor.
mode	The gather mode.

See also

IGatherLayer

Returns

The new gather layer, or nullptr if it could not be created.

9.122.3.20 addGridSample()

```
IGridSampleLayer * nvinfer1::INetworkDefinition::addGridSample (
            ITensor & input,
            ITensor & grid )  [inline], [noexcept]
```

Add a GridSample layer to the network.

Parameters

input	The input tensor to the layer.
grid	The grid tensor to the layer.

See also

IGridSampleLayer

Creates a GridSample layer with a InterpolationMode::kLINEAR, unaligned corners, and SampleMode::kFILL for 4d-shape input tensors.

Returns

The new GridSample layer, or nullptr if it could not be created.

9.122.3.21 addIdentity()

```
IIdentityLayer * nvinfer1::INetworkDefinition::addIdentity (
            ITensor & input )  [inline], [noexcept]
```

Add an identity layer.

Parameters

input	The input tensor to the layer.

See also

IIdentityLayer

Returns

The new identity layer, or nullptr if it could not be created.

9.122.3.22 addIfConditional()

```
IIfConditional * nvinfer1::INetworkDefinition::addIfConditional ( )  [inline], [noexcept]
```

Add an if-then-else to the network.

An IIfConditional provides a way to conditionally execute parts of the network.

Returns

Pointer to the IIfConditional that can be used to add conditional-boundary layers for the if-then-else.

See also

IIfConditional

9.122.3.23 addInput()

```
ITensor * nvinfer1::INetworkDefinition::addInput (
            char const * name,
            DataType type,
            Dims const & dimensions )  [inline], [noexcept]
```

Add an input tensor to the network.

Each input and output tensor must have a unique name.

For networks with wildcard dimensions, the volume is based on the maxima specified by an IOptimizationProfile.↩ Dimensions are normally non-negative integers. The exception is that in networks with all explicit dimensions, -1 can be used as a wildcard for a dimension to be specified at runtime. Input tensors with such a wildcard must have a corresponding entry in the IOptimizationProfiles indicating the permitted extrema, and the input dimensions must be set by IExecutionContext::setInputShape. Different IExecutionContext instances can have different dimensions. Wildcard dimensions are only supported for EngineCapability::kSTANDARD. They are not supported in safety contexts. DLA does not support Wildcard dimensions.

Tensor dimensions are specified independent of format. For example, if a tensor is formatted in "NHWC" or a vectorized format, the dimensions are still specified in the order{N, C, H, W}. For 2D images with a channel dimension, the last three dimensions are always {C,H,W}. For 3D images with a channel dimension, the last four dimensions are always {C,D,H,W}.

Parameters

name	The name of the tensor.
type	The type of the data held in the tensor.
dimensions	The dimensions of the tensor.

Warning

It is an error to specify a wildcard value on a dimension that is determined by trained parameters.

If run on DLA with explicit dimensions, only leading dimension can be a wildcard. And provided profile must have same minimum, optimum, and maximum dimensions.

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

See also

ITensor

Returns

The new tensor or nullptr if there is an error.

9.122.3.24 addKVCacheUpdate()

```
IKVCacheUpdateLayer * nvinfer1::INetworkDefinition::addKVCacheUpdate (
            ITensor & cache,
            ITensor & update,
            ITensor & writeIndices,
            KVCacheMode cacheMode )  [inline], [noexcept]
```

Add a KVCacheUpdate layer to the network.

Parameters

cache	The key/value cache tensor for the layer. The user is responsible for properly allocating and binding the tensor memory.
update	The newly updated key/value tensor for the layer.
writeIndices	The write indices tensor for key/value cache updates.
cacheMode	The mode of the KVCacheUpdate layer. For TensorRT 10.15, only `kLINEAR` mode is supported.

The expected tensor shapes are as follows:

- `cache`: [batchSize, numHeads, maxSequenceLength, headSize]
- `update`: [batchSize, numHeads, sequenceLength, headSize]
- `writeIndices`: [batchSize]

The `cache` and `update` tensors must have the same data type, which can be DataType::kFLOAT, DataType::kHALF, or DataType::kBF16. Quantized data types are not supported. The `writeIndices` tensor must be DataType::kINT32 or DataType::kINT64.

The layer performs in-place updates on the cache tensor. Therefore, the user must ensure that the `cache` tensor and the corresponding output tensor share the same device memory address before execution.

Warning

In `kLINEAR` mode, each update must satisfy the condition `writeIndices[i] + sequenceLength <= maxSequenceLength`. Out-of-bound updates will be ignored silently.

See also

IKVCacheUpdateLayer

Returns

The new KVCacheUpdate layer, or nullptr if it could not be created.

9.122.3.25 addLoop()

```
ILoop * nvinfer1::INetworkDefinition::addLoop ( )  [inline], [noexcept]
```

Add a loop to the network.

An ILoop provides a way to specify a recurrent subgraph.

Returns

Pointer to ILoop that can be used to add loop-boundary layers for the loop.

See also

ILoop

9.122.3.26 addLRN()

```
ILRNLayer * nvinfer1::INetworkDefinition::addLRN (
            ITensor & input,
            int64_t window,
            float alpha,
            float beta,
            float k )  [inline], [noexcept]
```

Add a LRN layer to the network.

Parameters

input	The input tensor to the layer.
window	The size of the window.
alpha	The alpha value for the LRN computation.
beta	The beta value for the LRN computation.
k	The k value for the LRN computation.

See also

ILRNLayer

Warning

Int32 tensors are not valid input tensors.

Returns

The new LRN layer, or nullptr if it could not be created.

9.122.3.27 addMatrixMultiply()

```
IMatrixMultiplyLayer * nvinfer1::INetworkDefinition::addMatrixMultiply (
            ITensor & input0,
            MatrixOperation op0,
            ITensor & input1,
            MatrixOperation op1 )  [inline], [noexcept]
```

Add a MatrixMultiply layer to the network.

Parameters

input0	The first input tensor (commonly A).
op0	The operation to apply to input0.
input1	The second input tensor (commonly B).
op1	The operation to apply to input1.

The inputs are shape tensors if the output is a shape tensor.

See also

IMatrixMultiplyLayer

Warning

Int32 tensors are not valid input tensors.

Returns

The new matrix multiply layer, or nullptr if it could not be created.

9.122.3.28 addMoE()

```
IMoELayer * nvinfer1::INetworkDefinition::addMoE (
            ITensor & hiddenStates,
            ITensor & selectedExpertsForTokens,
            ITensor & scoresForSelectedExperts )  [inline], [noexcept]
```

Add a MoE (Mixture of Experts) layer to the network.

Parameters

hiddenStates	The hidden states tensor input to the MoE layer. Shape: [batchSize, seqLen, hiddenSize].
selectedExpertsForTokens	The tensor containing expert indices selected for each token. Shape: [batchSize, seqLen, topK].
scoresForSelectedExperts	The tensor containing scores computed for the selected experts. Shape: [batchSize, seqLen, topK].

See also

IMoELayer

Warning

MoE requires Blackwell or Thor GPU architecture (SM 10.x or SM 11.x). SM 12.x is not currently supported. And performance is limited when seqLen $>$ 16.

The number of selected experts per token could be inferred from the input `selectedExpertsForTokens` and should be consistent with the topK in the `scoresForSelectedExperts`.

Returns

The new MoE layer, or nullptr if it could not be created.

9.122.3.29 addNMS() [1/2]

```
TRT_DEPRECATED INMSLayer * nvinfer1::INetworkDefinition::addNMS (
            ITensor & boxes,
            ITensor & scores,
            ITensor & maxOutputBoxesPerClass )  [inline], [noexcept]
```

Add a non-maximum suppression layer to the network.

The default indices tensor (the first output) data type is DataType::kINT32.

Parameters

boxes	The input boxes tensor to the layer.
scores	The input scores tensor to the layer.
maxOutputBoxesPerClass	The input maxOutputBoxesPerClass tensor to the layer.

See also

INMSLayer

Returns

The new NMS layer, or nullptr if it could not be created.

Deprecated Deprecated in TensorRT 10.14. Superseded by four-argument addNMS.

9.122.3.30 addNMS() [2/2]

```
INMSLayer * nvinfer1::INetworkDefinition::addNMS (
            ITensor & boxes,
            ITensor & scores,
            ITensor & maxOutputBoxesPerClass,
            DataType indicesType ) [inline], [noexcept]
```

Add a non-maximum suppression layer to the network.

Parameters

boxes	The input boxes tensor to the layer.
scores	The input scores tensor to the layer.
maxOutputBoxesPerClass	The input maxOutputBoxesPerClass tensor to the layer.
indicesType	Indices tensor (the first output) data type, must be DataType::kINT32 or DataType::kINT64.

See also

INMSLayer

Returns

The new NMS layer, or nullptr if it could not be created.

9.122.3.31 addNonZero() [1/2]

```
TRT_DEPRECATED INonZeroLayer * nvinfer1::INetworkDefinition::addNonZero (
            ITensor & input ) [inline], [noexcept]
```

Add a nonzero layer to the network.

The default indices tensor (the first output) data type is DataType::kINT32.

Parameters

input	The input tensor to the layer.

See also

INonZeroLayer

Returns

The new nonzero layer, or nullptr if it could not be created.

Deprecated Deprecated in TensorRT 10.14. Superseded by two-argument addNonZero.

9.122.3.32 addNonZero() [2/2]

```
INonZeroLayer * nvinfer1::INetworkDefinition::addNonZero (
            ITensor & input,
            DataType indicesType )  [inline], [noexcept]
```

Add a nonzero layer to the network.

Parameters

input	The input tensor to the layer.
indicesType	Indices tensor (the first output) data type, must be DataType::kINT32 or DataType::kINT64.

See also

INonZeroLayer

Returns

The new nonzero layer, or nullptr if it could not be created.

9.122.3.33 addNormalization()

```
TRT_DEPRECATED INormalizationLayer * nvinfer1::INetworkDefinition::addNormalization (
            ITensor & input,
```

```
ITensor & scale,
ITensor & bias,
uint32_t axesMask )  [inline], [noexcept]
```

Add a normalization layer to the network.

Parameters

input	The input tensor to the layer.
scale	The scale tensor used to scale the normalized output.
bias	The bias tensor used to scale the normalized output.
axesMask	The axes on which to perform mean calculations. The bit in position i of bitmask axesMask corresponds to explicit dimension i of the result. E.g., the least significant bit corresponds to the first explicit dimension and the next to least significant bit corresponds to the second explicit dimension.

The normalization layer works by performing normalization of the tensor `input` on the specified `axesMask`. The result is then scaled by multiplying with `scale` and adding `bias`.

The shapes of `scale` and `bias` must be the same, and must have the same rank and be unidirectionally broadcastable to the shape of `input`. Given a 4D NCHW input tensor, the expected shapes for `scale` and `bias` are:

- [1, C, 1, 1] for InstanceNormalization
- [1, G, 1, 1] for GroupNormalization. Use addNormalizationV2() instead if [1, C, 1, 1] shapes for `scale` and `bias` are required.

See also

INormalizationLayer

Returns

The new normalization layer, or nullptr if it could not be created.

Deprecated Deprecated in TensorRT 10.15. Superseded by addNormalizationV2().

9.122.3.34 addNormalizationV2()

```
TRT_NODISCARD INormalizationLayer * nvinfer1::INetworkDefinition::addNormalizationV2 (
            ITensor & input,
            ITensor & scale,
            ITensor & bias,
            uint32_t axesMask )  [inline], [noexcept]
```

Add a normalization layer to the network.

Parameters

input	The input tensor to the layer.
scale	The scale tensor used to scale the normalized output.
bias	The bias tensor used to scale the normalized output.
axesMask	The axes on which to perform mean calculations. The bit in position i of bitmask axesMask corresponds to explicit dimension i of the result. E.g., the least significant bit corresponds to the first explicit dimension and the next to least significant bit corresponds to the second explicit dimension.

The normalization layer works by performing normalization of the tensor `input` on the specified `axesMask`. The result is then scaled by multiplying with `scale` and adding `bias`.

The shapes of `scale` and `bias` are expected the be the same, and must have the same rank and be unidirectionally broadcastable to the shape of `input`. In the case of InstanceNorm or GroupNorm, the shapes of `scale` and `bias` are expected to be [1, C, 1, 1] in the case of a 4D NCHW input tensor.

See also

INormalizationLayer

Returns

The new normalization layer, or nullptr if it could not be created.

9.122.3.35 addOneHot()

```
IOneHotLayer * nvinfer1::INetworkDefinition::addOneHot (
            ITensor & indices,
            ITensor & values,
            ITensor & depth,
            int32_t axis )  [inline], [noexcept]
```

Add a OneHot layer to the network.

Parameters

indices	- tensor containing indices where on_value should be set.
values	- a 2-element tensor, consisting of [off_value, on_value].
depth	- a shape tensor containing the width of the added one-hot dimension.
axis	- the axis to add the one-hot encoding to.

See also

IOneHotLayer

Returns

The new OneHot layer, or nullptr if it could not be created.

9.122.3.36 addPaddingNd()

```
IPaddingLayer * nvinfer1::INetworkDefinition::addPaddingNd (
            ITensor & input,
            Dims const & prePadding,
            Dims const & postPadding )  [inline], [noexcept]
```

Add a padding layer to the network. Only 2D padding is currently supported.

Parameters

input	The input tensor to the layer.
prePadding	The padding to apply to the start of the tensor.
postPadding	The padding to apply to the end of the tensor.

See also

IPaddingLayer

Returns

The new padding layer, or nullptr if it could not be created.

9.122.3.37 addParametricReLU()

```
IParametricReLULayer * nvinfer1::INetworkDefinition::addParametricReLU (
            ITensor & input,
            ITensor & slope )  [inline], [noexcept]
```

Add a parametric ReLU layer to the network.

Parameters

input	The input tensor to the layer.
slope	The slope tensor to the layer. This tensor should be unidirectionally broadcastable to the input tensor.

See also

IParametricReLULayer

Warning

Tensors of type Int32, Int64, Bool, or UInt8 are not allowed as inputs.

Returns

The new parametric ReLU layer, or nullptr if it could not be created.

9.122.3.38 addPluginV2()

```
TRT_DEPRECATED IPluginV2Layer * nvinfer1::INetworkDefinition::addPluginV2 (
            ITensor *const * inputs,
            int32_t nbInputs,
            IPluginV2 & plugin )  [inline], [noexcept]
```

Add a plugin layer to the network using the IPluginV2 interface.

Parameters

inputs	The input tensors to the layer.
nbInputs	The number of input tensors.
plugin	The layer plugin.

See also

IPluginV2Layer

Warning

Dimension wildcard are only supported with IPluginV2DynamicExt or IPluginV2IOExt plugins.

Int32 tensors are not valid input tensors.

Returns

The new plugin layer, or nullptr if it could not be created.

Deprecated Deprecated in TensorRT 10.8. Superseded by addPluginV3.

9.122.3.39 addPluginV3()

```
IPluginV3Layer * nvinfer1::INetworkDefinition::addPluginV3 (
            ITensor *const * inputs,
            int32_t nbInputs,
            ITensor *const * shapeInputs,
            int32_t nbShapeInputs,
            IPluginV3 & plugin )  [inline], [noexcept]
```

Add a plugin layer implementing the IPluginV3 interface to the network.

Parameters

inputs	The input tensors to the layer.
nbInputs	The number of input tensors.
shapeInputs	Shape tensor inputs to the layer.
nbShapeInputs	The number of shape tensor inputs.
plugin	The layer plugin.

See also

IPluginV3Layer

Returns

The new plugin layer, or nullptr if it could not be created.

9.122.3.40 addPoolingNd()

```
IPoolingLayer * nvinfer1::INetworkDefinition::addPoolingNd (
            ITensor & input,
            PoolingType type,
            Dims const & windowSize )  [inline], [noexcept]
```

Add a multi-dimension pooling layer to the network.

Parameters

input	The input tensor to the layer.
type	The type of pooling to apply.
windowSize	The size of the pooling window.

See also

IPoolingLayer PoolingType

Warning

Int32 tensors are not valid input tensors.

Only 2D or 3D pooling is supported.

Returns

The new pooling layer, or nullptr if it could not be created.

9.122.3.41 addQuantize()

```
IQuantizeLayer * nvinfer1::INetworkDefinition::addQuantize (
            ITensor & input,
            ITensor & scale,
            DataType outputType )  [inline], [noexcept]
```

Add a quantization layer to the network.

Parameters

input	The input tensor to be quantized.
scale	A tensor with the scale value.
outputType	Output tensor data type.

See also

IQuantizeLayer

`input` tensor data type must be DataType::kFLOAT, DataType::kHALF or DataType::kBF16. `scale` tensor data type must be one of the following: DataType::kFLOAT (default), DataType::kHALF, DataType::kBF16 or DataType::kE8M0 (for MXFP8 quantization). `outputType` output tensor data type must be DataType::kINT8 (default), DataType::kFP8, DataType::kINT4 or DataType::kFP4. Future calls to set output type using setTo↩ Type or setOutputType must be consistent. For strongly typed networks, if the scale type is DataType::kHALF or DataType::kBF16 the output type must match.

Returns

The new quantization layer, or nullptr if it could not be created.

9.122.3.42 addRaggedSoftMax()

```
IRaggedSoftMaxLayer * nvinfer1::INetworkDefinition::addRaggedSoftMax (
            ITensor & input,
            ITensor & bounds )  [inline], [noexcept]
```

Add a RaggedSoftMax layer to the network.

Parameters

input	The ZxS input tensor.
bounds	The Zx1 bounds tensor.

See also

IRaggedSoftMaxLayer

Warning

The bounds tensor cannot have the last dimension be the wildcard character.

Int32 tensors are not valid input tensors.

The input and bounds tensors should be 3D tensors.

Returns

The new RaggedSoftMax layer, or nullptr if it could not be created.

9.122.3.43 addReduce()

```
IReduceLayer * nvinfer1::INetworkDefinition::addReduce (
            ITensor & input,
            ReduceOperation operation,
            uint32_t reduceAxes,
            bool keepDimensions )  [inline], [noexcept]
```

Add a reduce layer to the network.

Parameters

input	The input tensor to the layer.
operation	The reduction operation to perform.
reduceAxes	The reduction dimensions. The bit in position i of bitmask reduceAxes corresponds to explicit dimension i if result. E.g., the least significant bit corresponds to the first explicit dimension and the next to least significant bit corresponds to the second explicit dimension.
keepDimensions	The boolean that specifies whether or not to keep the reduced dimensions in the output of the layer.

The reduce layer works by performing an operation specified by `operation` to reduce the tensor `input` across the axes specified by `reduceAxes`.

See also

IReduceLayer

Warning

If output is an Int32 or Int64 shape tensor, ReduceOperation::kAVG is unsupported.

Returns

The new reduce layer, or nullptr if it could not be created.

9.122.3.44 addResize()

```
IResizeLayer * nvinfer1::INetworkDefinition::addResize (
            ITensor & input )  [inline], [noexcept]
```

Add a resize layer to the network.

Parameters

input	The input tensor to the layer.

See also

IResizeLayer

Warning

Int32 tensors are not valid input tensors.

Returns

The new resize layer, or nullptr if it could not be created.

9.122.3.45 addReverseSequence()

```
IReverseSequenceLayer * nvinfer1::INetworkDefinition::addReverseSequence (
            ITensor & input,
            ITensor & sequenceLens )  [inline], [noexcept]
```

Add a ReverseSequence layer to the network.

Parameters

input	The input tensor to the layer. Must have rank >= 2.
sequenceLens	1D tensor specifying lengths of sequences to reverse in a batch. The length of the sequenceLens tensor must be equal to the size of the dimension in input tensor specified by batchAxis.

See also

IReverseSequenceLayer

Returns

The new ReverseSequence layer, or nullptr if it could not be created.

9.122.3.46 addRotaryEmbedding()

```
IRotaryEmbeddingLayer * nvinfer1::INetworkDefinition::addRotaryEmbedding (
            ITensor & input,
            ITensor & cosCache,
            ITensor & sinCache,
            bool interleaved,
            int32_t rotaryEmbeddingDim )  [inline], [noexcept]
```

Add a Rotary Position Embedding (RoPE) layer to the network.

Parameters

input	The input activation tensor to the layer. The shape must be (batchSize, numHeads, sequenceLength, headSize).
cosCache	The cosine cache tensor for use in RoPE computation. See the following explanation for the shape requirement.
sinCache	The sine cache tensor for use in RoPE computation. See the following explanation for the shape requirement.
interleaved	Whether the `input` is in interleaved format, i.e., whether the 2-d vectors rotated are taken from adjacent 2 elements in the hidden dimension.
rotaryEmbeddingDim	The hidden dimension that participates in RoPE.

The RotaryEmbedding layer applies RoPE to the `input`, using `cosCache` and `sinCache`. An optional input, positionIds, can be provided using setInput with index 3. If provided, it is used to index into `cosCache` and `sin↩Cache`.

If `positionIds` is not provided, `cosCache` and `sinCache` must have shape (batchSize, sequenceLength, head↩Size / 2) if `rotaryEmbeddingDim` is 0, or (batchSize, sequenceLength, rotaryEmbeddingDim / 2) otherwise. If `positionIds` is provided, `cosCache` and `sinCache` must have shape (maxPositionId+1, headSize / 2) if `rotaryEmbeddingDim` is 0, or (maxPositionId+1, rotaryEmbeddingDim / 2) otherwise. `positionIds`, if provided, must have shape (batchSize, sequenceLength).

See also

IRotaryEmbeddingLayer

Returns

The new RotaryEmbedding layer, or nullptr if it could not be created.

9.122.3.47 addScale()

```
IScaleLayer * nvinfer1::INetworkDefinition::addScale (
            ITensor & input,
            ScaleMode mode,
            Weights shift,
            Weights scale,
            Weights power )  [inline], [noexcept]
```

Add a Scale layer to the network.

Parameters

input	The input tensor to the layer. This tensor must have at least 4 dimensions.
mode	The scaling mode.
shift	The shift value.
scale	The scale value.
power	The power value.

If the weights are available, then the size of weights are dependent on the ScaleMode. For ScaleMode::kUNIFORM, the number of weights equals 1. For ScaleMode::kCHANNEL, the number of weights equals the channel dimension. For ScaleMode::kELEMENTWISE, the number of weights equals the product of the last three dimensions of the input.

See also

addScaleNd

IScaleLayer

Warning

Int32 tensors are not valid input tensors.

Returns

The new Scale layer, or nullptr if it could not be created.

9.122.3.48 addScaleNd()

```
IScaleLayer * nvinfer1::INetworkDefinition::addScaleNd (
            ITensor & input,
            ScaleMode mode,
            Weights shift,
            Weights scale,
            Weights power,
            int32_t channelAxis )  [inline], [noexcept]
```

Add a multi-dimension scale layer to the network.

Parameters

input	The input tensor to the layer.
mode	The scaling mode.
shift	The shift value.
scale	The scale value.
power	The power value.
channelAxis	The channel axis.

If the weights are available, then the size of weights are dependent on the ScaleMode. For ScaleMode::kUNIFORM, the number of weights equals 1. For ScaleMode::kCHANNEL, the number of weights equals the channel dimension. For ScaleMode::kELEMENTWISE, the number of weights equals the product of all input dimensions at channelAxis and beyond.

For example, if the inputs dimensions are [A,B,C,D,E,F], and channelAxis=2: For ScaleMode::kUNIFORM, the number of weights is equal to 1. For ScaleMode::kCHANNEL, the number of weights is C. For ScaleMode::kELEMENTWISE, the number of weights is C∗D∗E∗F.

channelAxis can also be set explicitly using setChannelAxis().

See also

IScaleLayer

setChannelAxis()

Warning

Int32 tensors are not valid input tensors.

Only 2D or 3D scale is supported.

Returns

The new Scale layer, or nullptr if it could not be created.

9.122.3.49 addScatter()

```
IScatterLayer * nvinfer1::INetworkDefinition::addScatter (
            ITensor & data,
            ITensor & indices,
            ITensor & updates,
            ScatterMode mode )  [inline], [noexcept]
```

Add a Scatter layer to the network with specified mode and axis=0.

Parameters

data	The input tensor to be updated with additional values.
indices	indices of the elements to be updated.
updates	values to be used for updates.
mode	scatter mode.

See also

IScatterLayer

`indices` tensor data type must be DataType::kINT32. `updates` tensor data type must be the same as `data`

Returns

The new Scatter layer, or nullptr if it could not be created.

9.122.3.50 addSelect()

```
ISelectLayer * nvinfer1::INetworkDefinition::addSelect (
            ITensor & condition,
            ITensor & thenInput,
            ITensor & elseInput )  [inline], [noexcept]
```

Add a select layer to the network.

Parameters

condition	The condition tensor to the layer. Must have type DataType::kBOOL.
thenInput	The ”then” input tensor to the layer.
elseInput	The ”else” input tensor to the layer.

All three input tensors must have the same rank, and along each axis must have the same length or a length of one. If the length is one, the tensor is broadcast along that axis. The output tensor has the dimensions of the inputs AFTER the broadcast rule is applied. For example, given:

dimensions of condition: [1,1,5,9] dimensions of thenInput: [1,1,5,9] dimensions of elseInput: [1,3,1,9]

the output dimensions are [1,3,5,9], and the output contents are defined by:

```
output[0,i,j,k] = condition[0,0,j,k] ? thenInput[0,0,j,k] : elseInput[0,i,0,k]
```

The output dimensions are not necessarily the max of the input dimensions if any input is an empty tensor. For example, if in the preceding example, 5 is changed to 0:

dimensions of condition: [1,1,0,9] dimensions of thenInput: [1,1,0,9] dimensions of elseInput: [1,3,1,9]

then the output dimensions are [1,3,0,9].

The inputs are shape tensors if the output is a shape tensor.

See also

ISelectLayer

Returns

The new select layer, or nullptr if it could not be created.

9.122.3.51 addShape()

```
IShapeLayer * nvinfer1::INetworkDefinition::addShape (
            ITensor & input )  [inline], [noexcept]
```

Add a shape layer to the network.

Parameters

input	The input tensor to the layer.

See also

IShapeLayer

Warning

addShape is only supported when hasImplicitBatchDimensions is false.

Returns

The new shape layer, or nullptr if it could not be created.

9.122.3.52 addShuffle()

```
IShuffleLayer * nvinfer1::INetworkDefinition::addShuffle (
            ITensor & input )  [inline], [noexcept]
```

Add a shuffle layer to the network.

Parameters

input	The input tensor to the layer.

See also

IShuffleLayer

Returns

The new shuffle layer, or nullptr if it could not be created.

9.122.3.53 addSlice()

```
ISliceLayer * nvinfer1::INetworkDefinition::addSlice (
            ITensor & input,
            Dims const & start,
            Dims const & size,
            Dims const & stride )  [inline], [noexcept]
```

Add a slice layer to the network.

Parameters

input	The input tensor to the layer.
start	The start offset
size	The output dimension
stride	The slicing stride

Positive, negative, zero stride values, and combinations of them in different dimensions are allowed.

See also

ISliceLayer

Returns

The new slice layer, or nullptr if it could not be created.

9.122.3.54 addSoftMax()

```
ISoftMaxLayer * nvinfer1::INetworkDefinition::addSoftMax (
            ITensor & input )  [inline], [noexcept]
```

Add a SoftMax layer to the network.

See also

ISoftMaxLayer

Warning

Int32 tensors are not valid input tensors.

Returns

The new SoftMax layer, or nullptr if it could not be created.

9.122.3.55 addSqueeze()

```
ISqueezeLayer * nvinfer1::INetworkDefinition::addSqueeze (
            ITensor & input,
            ITensor & axes )  [inline], [noexcept]
```

Add a squeeze layer to the network.

Parameters

input	The input tensor to the layer.
axes	The axes to remove unit dimensions on.

See also

ISqueezeLayer

Axes must be resolvable to a constant Int32 or Int64 1D shape tensor. Values in axes must be unique and in the range of [-r, r-1], where r is the rank of the input tensor. For each axis value, the corresponding dimension in the input tensor must be one.

Returns

The new Squeeze layer, or nullptr if it could not be created.

9.122.3.56 addTopK() [1/2]

```
TRT_DEPRECATED ITopKLayer * nvinfer1::INetworkDefinition::addTopK (
            ITensor & input,
            TopKOperation op,
            int32_t k,
            uint32_t reduceAxes )  [inline], [noexcept]
```

Add a TopK layer to the network.

The TopK layer has two outputs of the same dimensions. The first contains data values, the second contains index positions for the values. Output values are sorted, largest first for operation kMAX and smallest first for operation kMIN.

Currently only values of K up to 3840 are supported.

The default indices tensor (the second output) data type is DataType::kINT32.

Parameters

input	The input tensor to the layer.
op	Operation to perform.
k	The number of elements to keep. For dynamic k, use the setInput() method to pass in k as a tensor instead, which will override the static k value passed here in calculations.
reduceAxes	The reduction dimensions. The bit in position i of bitmask reduceAxes corresponds to explicit dimension i of the result. E.g., the least significant bit corresponds to the first explicit dimension and the next to least significant bit corresponds to the second explicit dimension. Currently reduceAxes must specify exactly one dimension, and it must be one of the last four dimensions.

See also

ITopKLayer

Returns

The new TopK layer, or nullptr if it could not be created.

Deprecated Deprecated in TensorRT 10.14. Superseded by five-argument addTopK.

9.122.3.57 addTopK() [2/2]

```
ITopKLayer * nvinfer1::INetworkDefinition::addTopK (
            ITensor & input,
            TopKOperation op,
            int32_t k,
```

```
        uint32_t reduceAxes,
        DataType indicesType )  [inline], [noexcept]
```

Add a TopK layer to the network.

The TopK layer has two outputs of the same dimensions. The first contains data values, the second contains index positions for the values. Output values are sorted, largest first for operation kMAX and smallest first for operation kMIN.

Currently only values of K up to 3840 are supported.

Parameters

input	The input tensor to the layer.
op	Operation to perform.
k	The number of elements to keep. For dynamic k, use the setInput() method to pass in k as a tensor instead, which will override the static k value passed here in calculations.
reduceAxes	The reduction dimensions. The bit in position i of bitmask reduceAxes corresponds to explicit dimension i of the result. E.g., the least significant bit corresponds to the first explicit dimension and the next to least significant bit corresponds to the second explicit dimension. Currently reduceAxes must specify exactly one dimension, and it must be one of the last four dimensions.
indicesType	Indices tensor (the second output) data type, must be DataType::kINT32 or DataType::kINT64.

See also

ITopKLayer

Returns

The new TopK layer, or nullptr if it could not be created.

9.122.3.58 addUnary()

```
IUnaryLayer * nvinfer1::INetworkDefinition::addUnary (
        ITensor & input,
        UnaryOperation operation )  [inline], [noexcept]
```

Add a unary layer to the network.

Parameters

input	The input tensor to the layer.
operation	The operation to apply.

See also

IUnaryLayer

Generally the input must have a floating-point type (or kINT8 as a quantized float), except for the following operations:

- kSIGN accepts a floating-point or Int32 tensor.
- kNOT requires a Bool tensor.

The input is a shape tensor if the output is a shape tensor.

Returns

The new unary layer, or nullptr if it could not be created

9.122.3.59 addUnsqueeze()

```
IUnsqueezeLayer * nvinfer1::INetworkDefinition::addUnsqueeze (
            ITensor & input,
            ITensor & axes )  [inline], [noexcept]
```

Add an unsqueeze layer to the network.

Parameters

input	The input tensor to the layer.
axes	The axes to add unit dimensions.

See also

IUnsqueezeLayer

Axes must be resolvable to a constant Int32 or Int64 shape tensor. Values in axes must be unique and in the range of [-r_final, r_final-1], where r_final is the sum of rank(input) and len(axes).

r_final must be less than Dims::MAX_DIMS.

Returns

The new Unsqueeze layer, or nullptr if it could not be created

9.122.3.60 areWeightsMarkedRefittable()

```
bool nvinfer1::INetworkDefinition::areWeightsMarkedRefittable (
            char const * name ) const  [inline], [noexcept]
```

Whether the weight has been marked as refittable.

Parameters

name	The name of the weights to check.

Returns

True if the weights are marked as refittable, false if the weights do not exist or are marked as non-refittable.

9.122.3.61 getBuilder()

```
virtual IBuilder & nvinfer1::INetworkDefinition::getBuilder ( ) const  [inline], [virtual], [noexcept]
```

Return the builder from which this INetworkDefinition was created.

See also

IBuilder::createNetworkV2

Returns

the builder

9.122.3.62 getErrorRecorder()

```
IErrorRecorder * nvinfer1::INetworkDefinition::getErrorRecorder ( ) const  [inline], [noexcept]
```

get the ErrorRecorder assigned to this interface.

Retrieves the assigned error recorder object for the given class. A nullptr will be returned if setErrorRecorder has not been called.

Returns

A pointer to the IErrorRecorder object that has been registered.

See also

setErrorRecorder()

9.122.3.63 getFlag()

```
bool nvinfer1::INetworkDefinition::getFlag (
            NetworkDefinitionCreationFlag networkDefinitionCreationFlag ) const  [inline], [noexcept]
```

Returns true if the network definition creation flag is set.

See also

getFlags()

Returns

True if flag is set, false if unset.

9.122.3.64 getFlags()

```
NetworkDefinitionCreationFlags nvinfer1::INetworkDefinition::getFlags ( ) const  [inline], [noexcept]
```

Get the network definition creation flags for this network definition object. Defaults to 0.

Returns

The network definition creation options as a bitmask.

9.122.3.65 getInput()

```
ITensor * nvinfer1::INetworkDefinition::getInput (
            int32_t index ) const  [inline], [noexcept]
```

Get the input tensor specified by the given index.

Parameters

index	The index of the input tensor.

Returns

The input tensor, or nullptr if the index is out of range.

Note

adding inputs invalidates indexing here

See also

getNbInputs()

9.122.3.66 getLayer()

```
ILayer * nvinfer1::INetworkDefinition::getLayer (
            int32_t index ) const  [inline], [noexcept]
```

Get the layer specified by the given index.

Parameters

index	The index of the layer.

Returns

The layer, or nullptr if the index is out of range.

See also

getNbLayers()

9.122.3.67 getName()

```
char const * nvinfer1::INetworkDefinition::getName ( ) const  [inline], [noexcept]
```

Returns the name associated with the network.

The memory pointed to by getName() is owned by the INetworkDefinition object.

See also

INetworkDefinition::setName()

Returns

A null-terminated C-style string representing the name of the network.

9.122.3.68 getNbInputs()

```
int32_t nvinfer1::INetworkDefinition::getNbInputs ( ) const  [inline], [noexcept]
```

Get the number of inputs in the network.

Returns

The number of inputs in the network.

See also

getInput()

9.122.3.69 getNbLayers()

```
int32_t nvinfer1::INetworkDefinition::getNbLayers ( ) const  [inline], [noexcept]
```

Get the number of layers in the network.

Returns

The number of layers in the network.

See also

getLayer()

9.122.3.70 getNbOutputs()

```
int32_t nvinfer1::INetworkDefinition::getNbOutputs ( ) const  [inline], [noexcept]
```

Get the number of outputs in the network.

The outputs include those marked by markOutput or markOutputForShapes.

Returns

The number of outputs in the network.

See also

getOutput()

9.122.3.71 getOutput()

```
ITensor * nvinfer1::INetworkDefinition::getOutput (
            int32_t index ) const  [inline], [noexcept]
```

Get the output tensor specified by the given index.

Parameters

index	The index of the output tensor.

Returns

The output tensor, or nullptr if the index is out of range.

Note

adding inputs invalidates indexing here

See also

getNbOutputs()

9.122.3.72 hasImplicitBatchDimension()

```
TRT_DEPRECATED bool nvinfer1::INetworkDefinition::hasImplicitBatchDimension ( ) const  [inline],
[noexcept]
```

Query whether the network was created with an implicit batch dimension.

Returns

Always false since TensorRT 10.0 does not support an implicit batch dimension.

See also

createNetworkV2

Deprecated Deprecated in TensorRT 10.0. Implicit batch is not supported since TensorRT 10.0.

9.122.3.73 isDebugTensor()

```
bool nvinfer1::INetworkDefinition::isDebugTensor (
            ITensor const & tensor ) const  [inline], [noexcept]
```

Check if a tensor is marked as debug tensor.

Returns

true if tensor is marked as debug tensor, false otherwise.

9.122.3.74 markDebug()

```
bool nvinfer1::INetworkDefinition::markDebug (
            ITensor & tensor )  [inline], [noexcept]
```

Mark a tensor as a debug tensor.

A debug tensor can be optionally emitted at runtime. Note that tensor names are required to specify debug tensors at runtime.

Parameters

tensor	Tensor to be marked as debug

Returns

True if tensor successfully marked (or was already marked), false otherwise.

See also

unmarkDebug(), IExecutionContext::setDebugListener(), ITensor::setName()

9.122.3.75 markOutput()

```
void nvinfer1::INetworkDefinition::markOutput (
             ITensor & tensor )  [inline], [noexcept]
```

Mark a tensor as a network output.

Parameters

tensor	The tensor to mark as an output tensor.

Warning

It is an error to mark a network input as an output.

It is an error to mark a tensor inside an ILoop or an IIfConditional as an output.

9.122.3.76 markOutputForShapes()

```
bool nvinfer1::INetworkDefinition::markOutputForShapes (
             ITensor & tensor )  [inline], [noexcept]
```

Enable tensor's value to be computed by IExecutionContext::getShapeBinding.

Returns

True if successful, false if tensor is already marked as an output.

The tensor must be of type DataType::kINT32 and have no more than one dimension.

Warning

The tensor must have dimensions that can be determined to be constants at build time.

It is an error to mark a network input as a shape output.

9.122.3.77 markUnfusedTensorsAsDebugTensors()

```
bool nvinfer1::INetworkDefinition::markUnfusedTensorsAsDebugTensors ( )  [inline], [noexcept]
```

Mark unfused tensors as debug tensors.

Debug tensors can be optionally emitted at runtime. Tensors that are fused by the optimizer will not be emitted. Tensors marked this way will not prevent fusion like markDebug() does, thus preserving performance.

Warning

Tensors marked this way cannot be detected by isDebugTensor().

DebugListener can only get internal tensor names instead of the original tensor names in the Network↩ Definition for tensors marked this way. But the names correspond to the names obtained by IEngineInspector.

There is no guarantee that all unfused tensors are marked.

Returns

True if tensors were successfully marked (or were already marked), false otherwise.

See also

unmarkUnfusedTensorsAsDebugTensors(), markDebug(), IExecutionContext::setDebugListener()

9.122.3.78 markWeightsRefittable()

```
bool nvinfer1::INetworkDefinition::markWeightsRefittable (
            char const * name )  [inline], [noexcept]
```

Mark weights as refittable when the builder flag kREFIT_INDIVIDUAL is set.

Parameters

name	The name of the weights.

Returns

True if the weights were successfully marked as refittable, false if the weights do not exist or cannot be refitted.

9.122.3.79 removeTensor()

```
void nvinfer1::INetworkDefinition::removeTensor (
            ITensor & tensor )  [inline], [noexcept]
```

remove a tensor from the network definition.

Parameters

tensor	the tensor to remove

It is illegal to remove a tensor that is the input or output of a layer. if this method is called with such a tensor, a warning will be emitted on the log and the call will be ignored. Its intended use is to remove detached tensors after e.g. concatenating two networks with Layer::setInput().

9.122.3.80 setErrorRecorder()

```
void nvinfer1::INetworkDefinition::setErrorRecorder (
            IErrorRecorder * recorder )  [inline], [noexcept]
```

Set the ErrorRecorder for this interface.

Assigns the ErrorRecorder to this interface. The ErrorRecorder will track all errors during execution. This function will call incRefCount of the registered ErrorRecorder at least once. Setting recorder to nullptr unregisters the recorder with the interface, resulting in a call to decRefCount if a recorder has been registered.

If an error recorder is not set, messages will be sent to the global log stream.

Parameters

recorder	The error recorder to register with this interface.

See also

getErrorRecorder()

9.122.3.81 setName()

```
void nvinfer1::INetworkDefinition::setName (
            char const * name )  [inline], [noexcept]
```

Sets the name of the network.

Parameters

name	The name to assign to this network.

Set the name of the network so that it can be associated with a built engine. The `name` must be a null-terminated C-style string. TensorRT makes no use of this string except storing it as part of the engine so that it may be retrieved at runtime. A name unique to the builder will be generated by default.

This method copies the name string.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

See also

INetworkDefinition::getName(), ISafeCudaEngine::getName()

Returns

none

9.122.3.82 setWeightsName()

```
bool nvinfer1::INetworkDefinition::setWeightsName (
            Weights weights,
            char const * name )  [inline], [noexcept]
```

Associate a name with all current uses of the given weights.

The name must be set after the Weights are used in the network. Lookup is associative. The name applies to all Weights with matching type, value pointer, and count. If Weights with a matching value pointer, but different type or count exists in the network, an error message is issued, the name is rejected, and return false. If the name has already been used for other weights, return false. A nullptr causes the weights to become unnamed, i.e. clears any previous name.

Parameters

weights	The weights to be named.
name	The name to associate with the weights.

Returns

true on success.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

9.122.3.83 unmarkDebug()

```
bool nvinfer1::INetworkDefinition::unmarkDebug (
            ITensor & tensor )  [inline], [noexcept]
```

Unmark a tensor as a debug tensor.

Remove the marking of a tensor as a debug tensor.

Parameters

tensor	Tensor to be unmarked as debug.

Returns

True if tensor successfully unmarked (or was already unmarked), false otherwise.

See also

markDebug(), IExecutionContext::setDebugListener()

9.122.3.84 unmarkOutput()

```
void nvinfer1::INetworkDefinition::unmarkOutput (
            ITensor & tensor )  [inline], [noexcept]
```

unmark a tensor as a network output.

Parameters

tensor	The tensor to unmark as an output tensor.

see markOutput()

9.122.3.85 unmarkOutputForShapes()

```
bool nvinfer1::INetworkDefinition::unmarkOutputForShapes (
            ITensor & tensor )  [inline], [noexcept]
```

Undo markOutputForShapes.

> Warning
>
> inputs to addShape cannot contain wildcard dimension values.

Returns

True if successful, false if tensor is not marked as an output.

9.122.3.86 unmarkUnfusedTensorsAsDebugTensors()

```
bool nvinfer1::INetworkDefinition::unmarkUnfusedTensorsAsDebugTensors ( )  [inline], [noexcept]
```

Undo the marking of unfused tensors as debug tensors.

This has no effect on tensors marked by markDebug().

Returns

True if tensor successfully unmarked (or was already unmarked), false otherwise.

See also

markUnfusedTensorsAsDebugTensors(), IExecutionContext::setDebugListener()

9.122.3.87 unmarkWeightsRefittable()

```
bool nvinfer1::INetworkDefinition::unmarkWeightsRefittable (
            char const * name )  [inline], [noexcept]
```

Unmark weights as refittable when the builder flag kREFIT_INDIVIDUAL is set.

Parameters

name	The name of the weights.

Returns

True if the weights were successfully marked as unrefittable, false if the weights do not exist.

9.122.4 Member Data Documentation

9.122.4.1 mImpl

```
apiv::VNetworkDefinition* nvinfer1::INetworkDefinition::mImpl  [protected]
```

The documentation for this class was generated from the following file:

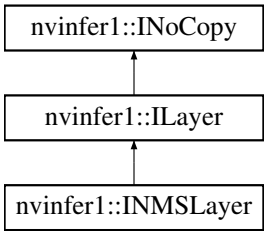
- NvInfer.h

9.123 nvinfer1::INMSLayer Class Reference

A non-maximum suppression layer in a network definition.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::INMSLayer:



Public Member Functions

- void setBoundingBoxFormat (BoundingBoxFormat fmt) noexcept
  Set the bounding box format parameter for the layer.
- BoundingBoxFormat getBoundingBoxFormat () const noexcept
  Get the bounding box format parameter for the layer.
- void setTopKBoxLimit (int32_t limit) noexcept
  Set the TopK box limit parameter for the layer.
- int32_t getTopKBoxLimit () const noexcept
  Get the TopK box limit parameter for the layer.
- bool setIndicesType (DataType type) noexcept
  Set the indices type for the layer.
- DataType getIndicesType () const noexcept
  Return the NMS layer indices type.
- void setInput (int32_t index, ITensor &tensor) noexcept
  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~INMSLayer () noexcept=0

Protected Attributes

- apiv::VNMSLayer * mImpl

9.123.1 Detailed Description

A non-maximum suppression layer in a network definition.

The NMS algorithm iterates through a set of bounding boxes and their confidence scores, in decreasing order of score. Boxes are selected if their score is above a given threshold, and their intersection-over-union (IoU) with previously selected boxes is less than or equal to a given threshold. This layer implements NMS per batch item and per class.

Per batch item, boxes are initially sorted by their scores without regard to class. Only boxes up to a maximum of the TopK limit are considered for selection (per batch). During selection, only overlapping boxes of the same class are compared, so that overlapping boxes of different classes do not suppress each other.

For each batch item, the ordering of candidate bounding boxes with the same score is unspecified, but the ordering will be consistent across different runs for the same inputs.

The layer has the following inputs, in order of input index:

- Boxes contains the input bounding boxes. It is a linear tensor of type kFLOAT or kHALF. It has shape [batchSize, numInputBoundingBoxes, numClasses, 4] if the boxes are per class, or [batchSize, numInputBoundingBoxes, 4] if the same boxes are to be used for each class.
- Scores contains the per-box scores. It is a linear tensor of the same type as Boxes. It has shape [batchSize, numInputBoundingBoxes, numClasses].
- MaxOutputBoxesPerClass is the maximum number of output boxes per batch item per class. It is a scalar (0D tensor) of type kINT32.
- IoUThreshold is the maximum IoU for selected boxes. It is a scalar (0D tensor) of type kFLOAT in the range [0.0f, 1.0f]. It is an optional input with default 0.0f.
- ScoreThreshold is the value that a box score must exceed in order to be selected. It is a scalar (0D tensor) of type kFLOAT. It is an optional input with default 0.0f.

The layer has the following outputs, in order of output index:

- SelectedIndices contains the indices of the selected boxes. It is a linear tensor of type kINT32 or kINT64. It has shape [NumOutputBoxes, 3]. Each row contains a (batchIndex, classIndex, boxIndex) tuple. The output boxes are sorted in order of increasing batchIndex and then in order of decreasing score within each batchIndex. For each batchIndex, the ordering of output boxes with the same score is unspecified. If MaxOutputBoxesPerClass is a constant input, the maximum number of output boxes is batchSize * numClasses * min(numInputBounding↩ Boxes, MaxOutputBoxesPerClass). Otherwise, the maximum number of output boxes is batchSize * numClasses * numInputBoundingBoxes. The maximum number of output boxes is used to determine the upper-bound on allocated memory for this output tensor.
- NumOutputBoxes is the number of output boxes in SelectedIndices. It is a scalar (0D tensor) of type kINT32.

Warning

There is a hardware-dependent limit K such that only the K highest scoring boxes in each batch item will be considered for selection. The value of K is 2000 for SM 5.3 and 6.2 devices, and 5000 otherwise.

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.123.2 Constructor & Destructor Documentation

9.123.2.1 ~INMSLayer()

```
nvinfer1::INMSLayer::~INMSLayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.123.3 Member Function Documentation

9.123.3.1 getBoundingBoxFormat()

```
BoundingBoxFormat nvinfer1::INMSLayer::getBoundingBoxFormat ( ) const  [inline], [noexcept]
```

Get the bounding box format parameter for the layer.

See also

BoundingBoxFormat

setBoundingBoxFormat()

9.123.3.2 getIndicesType()

```
DataType nvinfer1::INMSLayer::getIndicesType ( ) const  [inline], [noexcept]
```

Return the NMS layer indices type.

Returns

indices type set during layer creation or by setIndicesType(). The return value is the indices type of the NMS layer. The default value is DataType::kINT32.

9.123.3.3 getTopKBoxLimit()

```
int32_t nvinfer1::INMSLayer::getTopKBoxLimit ( ) const  [inline], [noexcept]
```

Get the TopK box limit parameter for the layer.

See also

setTopKBoxLimit()

9.123.3.4 setBoundingBoxFormat()

```
void nvinfer1::INMSLayer::setBoundingBoxFormat (
            BoundingBoxFormat fmt )  [inline], [noexcept]
```

Set the bounding box format parameter for the layer.

The default value for the bounding box format parameter is kCORNER_PAIRS.

See also

BoundingBoxFormat

getBoundingBoxFormat()

9.123.3.5 setIndicesType()

```
bool nvinfer1::INMSLayer::setIndicesType (
            DataType type )  [inline], [noexcept]
```

Set the indices type for the layer.

Parameters

type	The DataType of the indices tensor.

Returns

true if set successfully, false otherwise.

Set the indices (the first output) type of the NMS layer. Valid values are DataType::kINT32 and DataType::kINT64, otherwise an error occurs and the type is not updated.

9.123.3.6 setInput()

```
void nvinfer1::ILayer::setInput (
             int32_t index,
             ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor

The indices are as follows:

- 0: The required Boxes tensor.
- 1: The required Scores tensor.
- 2: The required MaxOutputBoxesPerClass tensor.
- 3: The optional IoUThreshold tensor.
- 4: The optional ScoreThreshold tensor.

If this function is called for an index greater or equal to getNbInputs(), then afterwards getNbInputs() returns index + 1, and any missing intervening inputs are set to null. Note that only optional inputs can be missing.

9.123.3.7 setTopKBoxLimit()

```
void nvinfer1::INMSLayer::setTopKBoxLimit (
             int32_t limit )  [inline], [noexcept]
```

Set the TopK box limit parameter for the layer.

The TopK box limit is the maximum number of filtered boxes considered for selection per batch item. The default value for the TopK box limit parameter is 2000 for SM 5.3 and 6.2 devices, and 5000 otherwise. The TopK box limit must be less than or equal to {2000 for SM 5.3 and 6.2 devices, 5000 otherwise}.

See also

getTopKBoxLimit()

9.123.4 Member Data Documentation

9.123.4.1 mImpl

```
apiv::VNMSLayer* nvinfer1::INMSLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

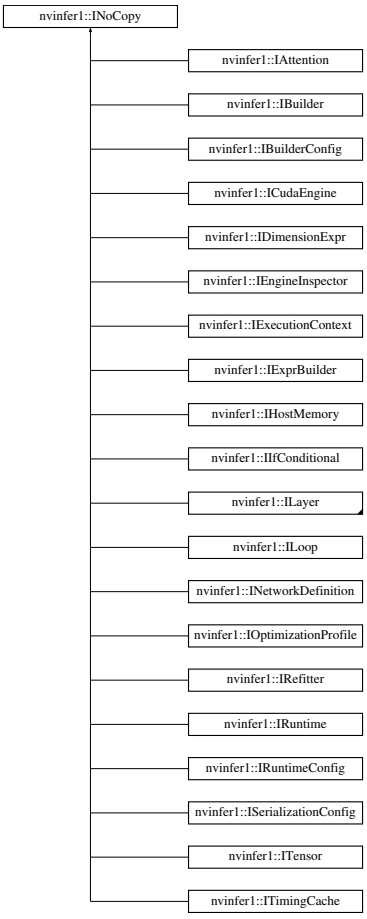
- NvInfer.h

9.124 nvinfer1::INoCopy Class Reference

Forward declaration of IEngineInspector for use by other interfaces.

```
#include <NvInferRuntime.h>
```

Inheritance diagram for nvinfer1::INoCopy:



Protected Member Functions

- INoCopy ()=default
- virtual ~INoCopy ()=default
- INoCopy (INoCopy const &other)=delete
- INoCopy & operator= (INoCopy const &other)=delete
- INoCopy (INoCopy &&other)=delete
- INoCopy & operator= (INoCopy &&other)=delete

9.124.1 Detailed Description

Forward declaration of IEngineInspector for use by other interfaces.

Base class for all TensorRT interfaces that are implemented by the TensorRT libraries

Objects of such classes are not movable or copyable, and should only be manipulated via pointers.

9.124.2 Constructor & Destructor Documentation

9.124.2.1 INoCopy() [1/3]

```
nvinfer1::INoCopy::INoCopy ( )  [protected], [default]
```

9.124.2.2 ~INoCopy()

```
virtual nvinfer1::INoCopy::~INoCopy ( )  [protected], [virtual], [default]
```

9.124.2.3 INoCopy() [2/3]

```
nvinfer1::INoCopy::INoCopy (
            INoCopy const & other )  [protected], [delete]
```

9.124.2.4 INoCopy() [3/3]

```
nvinfer1::INoCopy::INoCopy (
            INoCopy && other )  [protected], [delete]
```

9.124.3 Member Function Documentation

9.124.3.1 operator=() [1/2]

```
INoCopy & nvinfer1::INoCopy::operator= (
            INoCopy && other )  [protected], [delete]
```

9.124.3.2 operator=() [2/2]

```
INoCopy & nvinfer1::INoCopy::operator= (
            INoCopy const & other )  [protected], [delete]
```

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.125 INonZero Class Reference

A NonZero layer in a network.

`#include <NvInfer.h>`

9.125.1 Detailed Description

A NonZero layer in a network.

This layer gets the positions of elements that are non-zero in the input. For boolean input, ”non-zero” means ”true”. Semantics are similar to ONNX NonZero.

The input may have type kFLOAT, kHALF, kINT32, or kBOOL.

The output is a matrix of type kINT32 or kINT64. For an input with dimensions [L1, L2, ..., Lm], the output has dimensions [m,n], where n is the number of non-zero elements. I.e., each column denotes a m-D position.

The columns are lexically ordered. E.g., a column with [3,2,4,7] precedes a column with [3,2,5,6].

Tip: ”compress” can be implemented with INonZero+IShuffle+Gather. For example, to compress a tensor x over axis k using mask vector v, use nonzero(v) to compute the subscripts, shuffle with reshape dimensions = [-1] to make the subscripts 1D, and then gather with the subscripts.

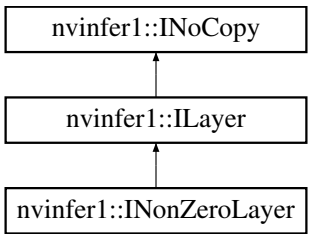
The documentation for this class was generated from the following file:

- NvInfer.h

9.126 nvinfer1::INonZeroLayer Class Reference

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::INonZeroLayer:



Public Member Functions

- bool setIndicesType (DataType type) noexcept

  Set the indices type for the layer.
- DataType getIndicesType () const noexcept

  Return the NonZero layer indices type.

Protected Member Functions

- virtual ~INonZeroLayer () noexcept=0

Protected Attributes

- apiv::VNonZeroLayer * mImpl

9.126.1 Constructor & Destructor Documentation

9.126.1.1 ~INonZeroLayer()

`nvinfer1::INonZeroLayer::~INonZeroLayer ( ) [inline], [protected], [pure virtual], [default], [noexcept]`

9.126.2 Member Function Documentation

9.126.2.1 getIndicesType()

```
DataType nvinfer1::INonZeroLayer::getIndicesType ( ) const  [inline], [noexcept]
```

Return the NonZero layer indices type.

Returns

indices type set during layer creation or by setIndicesType(). The return value is the indices type of the NonZero layer. The default value is DataType::kINT32.

9.126.2.2 setIndicesType()

```
bool nvinfer1::INonZeroLayer::setIndicesType (
            DataType type )  [inline], [noexcept]
```

Set the indices type for the layer.

Parameters

type	The DataType of the indices tensor.

Returns

true if set successfully, false otherwise.

Set the indices (the first output) type of the NonZero layer. Valid values are DataType::kINT32 and DataType::kINT64, otherwise an error occurs and the type is not updated.

9.126.3 Member Data Documentation

9.126.3.1 mImpl

```
apiv::VNonZeroLayer* nvinfer1::INonZeroLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

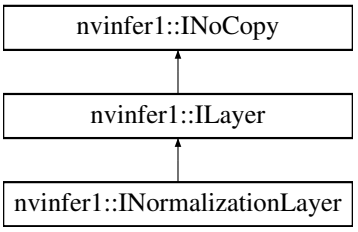
- NvInfer.h

9.127 nvinfer1::INormalizationLayer Class Reference

A normalization layer in a network definition.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::INormalizationLayer:



Public Member Functions

- void setEpsilon (float eps) noexcept

  Set the epsilon value used for the normalization calculation.
- float getEpsilon () const noexcept

  Get the epsilon value used for the normalization calculation.
- void setAxes (uint32_t axesMask) noexcept

  Set the reduction axes for the normalization calculation.
- uint32_t getAxes () const noexcept

  Get the axes value used for the normalization calculation.
- void setNbGroups (int64_t nbGroups) noexcept

  Set the number of groups used to split the channels in the normalization calculation.
- int64_t getNbGroups () const noexcept

  Get the number of groups used to split the channels for the normalization calculation.
- TRT_NODISCARD bool isV2 () const noexcept

  Returns true if this layer was created through addNormalizationV2().

Protected Member Functions

- virtual ~INormalizationLayer () noexcept=0

Protected Attributes

- apiv::VNormalizationLayer * mImpl

9.127.1 Detailed Description

A normalization layer in a network definition.

The normalization layer performs the following operation:

X - input Tensor Y - output Tensor S - scale Tensor B - bias Tensor

Y = (X - Mean(X, axes)) / Sqrt(Variance(X) + epsilon) * S + B

Where Mean(X, axes) is a reduction over a set of axes, and Variance(X) = Mean((X - Mean(X, axes)) ^ 2, axes).

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.127.2 Constructor & Destructor Documentation

9.127.2.1 ~INormalizationLayer()

```
nvinfer1::INormalizationLayer::~INormalizationLayer ( )  [inline], [protected], [pure virtual],
[default], [noexcept]
```

9.127.3 Member Function Documentation

9.127.3.1 getAxes()

```
uint32_t nvinfer1::INormalizationLayer::getAxes ( ) const  [inline], [noexcept]
```

Get the axes value used for the normalization calculation.

Returns

The axes used for the normalization calculation.

9.127.3.2 getEpsilon()

```
float nvinfer1::INormalizationLayer::getEpsilon ( ) const  [inline], [noexcept]
```

Get the epsilon value used for the normalization calculation.

Returns

The epsilon value used for the normalization calculation.

9.127.3.3 getNbGroups()

```
int64_t nvinfer1::INormalizationLayer::getNbGroups ( ) const  [inline], [noexcept]
```

Get the number of groups used to split the channels for the normalization calculation.

Returns

The number of groups used to split the channel used for the normalization calculation.

9.127.3.4 isV2()

```
TRT_NODISCARD bool nvinfer1::INormalizationLayer::isV2 ( ) const  [inline], [noexcept]
```

Returns true if this layer was created through addNormalizationV2().

Returns

Whether the layer was created through addNormalizationV2().

9.127.3.5 setAxes()

```
void nvinfer1::INormalizationLayer::setAxes (
            uint32_t axesMask )  [inline], [noexcept]
```

Set the reduction axes for the normalization calculation.

Parameters

axesMask	The axes used for the normalization calculation.

9.127.3.6 setEpsilon()

```
void nvinfer1::INormalizationLayer::setEpsilon (
            float eps )  [inline], [noexcept]
```

Set the epsilon value used for the normalization calculation.

The default value of `eps` is 1e-5F.

Parameters

eps	The epsilon value used for the normalization calculation.

9.127.3.7 setNbGroups()

```
void nvinfer1::INormalizationLayer::setNbGroups (
             int64_t nbGroups ) [inline], [noexcept]
```

Set the number of groups used to split the channels in the normalization calculation.

The input tensor channels are divided into `nbGroups` groups, and normalization is performed per group. The channel dimension is considered to be the second dimension in a [N, C, H, W, ...] formatted tensor.

The default `nbGroups` is 1.

> Warning
>
> It is an error to set `nbGroups` to a value that does not evenly divide into the number of channels of the input tensor.
>
> When `nbGroups` is != 1, it is expected that the provided axesMask will have all bits corresponding to dimensions after the channel dimension set to 1, with all other bits set to 0.

Parameters

nbGroups	The number of groups to split the channels into for the normalization calculation.

9.127.4 Member Data Documentation

9.127.4.1 mImpl

```
apiv::VNormalizationLayer* nvinfer1::INormalizationLayer::mImpl [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.128 nvinfer1::InterfaceInfo Class Reference

Version information associated with a TRT interface.

`#include <NvInferRuntimeBase.h>`

Public Attributes

- InterfaceKind kind
- int32_t major
- int32_t minor

9.128.1 Detailed Description

Version information associated with a TRT interface.

9.128.2 Member Data Documentation

9.128.2.1 kind

```
InterfaceKind nvinfer1::InterfaceInfo::kind
```

9.128.2.2 major

```
int32_t nvinfer1::InterfaceInfo::major
```

9.128.2.3 minor

```
int32_t nvinfer1::InterfaceInfo::minor
```

The documentation for this class was generated from the following file:

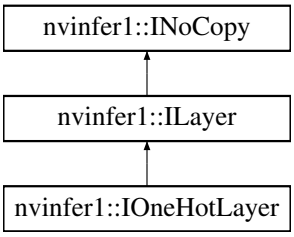
- NvInferRuntimeBase.h

9.129 nvinfer1::IOneHotLayer Class Reference

A OneHot layer in a network definition.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IOneHotLayer:



Public Member Functions

- void setAxis (int32_t axis) noexcept
  Set the axis parameter.
- int32_t getAxis () const noexcept
  Get the value of the axis parameter.

Protected Member Functions

- virtual ~IOneHotLayer () noexcept=0

Protected Attributes

- apiv::VOneHotLayer * mImpl

9.129.1 Detailed Description

A OneHot layer in a network definition.

The OneHot layer has three input tensors: Indices, Values, and Depth, one output tensor: Output, and an axis attribute.

- Indices is an Int32 tensor that determines which locations in Output to set as on_value.
- Values is a two-element (rank=1) tensor that consists of [off_value, on_value]
- Depth is an 0D tensor of type Int32 or Int64, which contains the depth (number of classes) of the one-hot encoding. The depth tensor must be a positive build-time constant.
- Output is a tensor with rank = rank(indices)+1, where the added dimension contains the one-hot encoding. The data types of Output is equal to the Values data type.
- Axis is a scalar specifying to which dimension of the output one-hot encoding is added. Valid range for axis is -rank(indices)-1 <= axis <= rank(indices).

The output is computed by copying off_values to all output elements, then setting on_value on the indices specified by the indices tensor. when axis = 0: output[indices[i, j, k], i, j, k] = on_value for all i, j, k and off_value otherwise.

when axis = -1: output[i, j, k, indices[i, j, k]] = on_value for all i, j, k and off_value otherwise.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.129.2 Constructor & Destructor Documentation

9.129.2.1 ~IOneHotLayer()

```
nvinfer1::IOneHotLayer::~IOneHotLayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.129.3 Member Function Documentation

9.129.3.1 getAxis()

```
int32_t nvinfer1::IOneHotLayer::getAxis ( ) const  [inline], [noexcept]
```

Get the value of the axis parameter.

9.129.3.2 setAxis()

```
void nvinfer1::IOneHotLayer::setAxis (
             int32_t axis )  [inline], [noexcept]
```

Set the axis parameter.

See also

IOneHotLayer

9.129.4 Member Data Documentation

9.129.4.1 mImpl

```
apiv::VOneHotLayer* nvinfer1::IOneHotLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.130 nvonnxparser::IOnnxConfig Class Reference

Configuration Manager Class.

`#include <NvOnnxConfig.h>`

Public Types

- typedef int32_t Verbosity
  Defines Verbosity level.

Public Member Functions

- virtual ~IOnnxConfig () noexcept=0
- virtual void setModelDtype (const nvinfer1::DataType) noexcept=0
  Set the Model Data Type.
- virtual nvinfer1::DataType getModelDtype () const noexcept=0
  Get the Model Data Type.
- virtual char const * getModelFileName () const noexcept=0
  Get the Model FileName.
- virtual void setModelFileName (char const *onnxFilename) noexcept=0
  Set the Model File Name.
- virtual Verbosity getVerbosityLevel () const noexcept=0
  Get the Verbosity Level.
- virtual void addVerbosity () noexcept=0
  Increase the Verbosity Level.
- virtual void reduceVerbosity () noexcept=0
  Reduce the Verbosity Level.
- virtual void setVerbosityLevel (Verbosity) noexcept=0
  Set to specific verbosity Level.
- virtual char const * getTextFileName () const noexcept=0
  Returns the File Name of the Network Description as a Text File.
- virtual void setTextFileName (char const *textFileName) noexcept=0
  Set the File Name of the Network Description as a Text File.
- virtual char const * getFullTextFileName () const noexcept=0
  Get the File Name of the Network Description as a Text File, including the weights.
- virtual void setFullTextFileName (char const *fullTextFileName) noexcept=0
  Set the File Name of the Network Description as a Text File, including the weights.
- virtual bool getPrintLayerInfo () const noexcept=0
  Get whether the layer information will be printed.
- virtual void setPrintLayerInfo (bool) noexcept=0
  Set whether the layer information will be printed.

9.130.1 Detailed Description

Configuration Manager Class.

9.130.2 Member Typedef Documentation

9.130.2.1 Verbosity

`nvonnxparser::IOnnxConfig::Verbosity`

Defines Verbosity level.

9.130.3 Constructor & Destructor Documentation

9.130.3.1 ~IOnnxConfig()

`nvonnxparser::IOnnxConfig::~IOnnxConfig ( )  [inline], [pure virtual], [default], [noexcept]`

9.130.4 Member Function Documentation

9.130.4.1 addVerbosity()

`virtual void nvonnxparser::IOnnxConfig::addVerbosity ( )  [pure virtual], [noexcept]`

Increase the Verbosity Level.

Returns

The Verbosity Level.

See also

reduceVerbosity(), setVerbosity(Verbosity)

9.130.4.2 getFullTextFileName()

```
virtual char const * nvonnxparser::IOnnxConfig::getFullTextFileName ( ) const  [pure virtual],
[noexcept]
```

Get the File Name of the Network Description as a Text File, including the weights.

Returns

Return the name of the file containing the network description converted to a plain text, used for debugging purposes.

See also

setFullTextFilename()

9.130.4.3 getModelDtype()

```
virtual nvinfer1::DataType nvonnxparser::IOnnxConfig::getModelDtype ( ) const  [pure virtual],
[noexcept]
```

Get the Model Data Type.

Returns

the data type of the model.

See also

setModelDtype() and DataType

9.130.4.4 getModelFileName()

```
virtual char const * nvonnxparser::IOnnxConfig::getModelFileName ( ) const  [pure virtual], [noexcept]
```

Get the Model FileName.

Returns

Return the Model Filename, as a null-terminated C-style string.

See also

setModelFileName()

9.130.4.5 getPrintLayerInfo()

```
virtual bool nvonnxparser::IOnnxConfig::getPrintLayerInfo ( ) const  [pure virtual], [noexcept]
```

Get whether the layer information will be printed.

Returns

Returns whether the layer information will be printed.

See also

setPrintLayerInfo()

9.130.4.6 getTextFileName()

```
virtual char const * nvonnxparser::IOnnxConfig::getTextFileName ( ) const  [pure virtual], [noexcept]
```

Returns the File Name of the Network Description as a Text File.

Returns

Return the name of the file containing the network description converted to a plain text, used for debugging purposes.

See also

setTextFilename()

9.130.4.7 getVerbosityLevel()

```
virtual Verbosity nvonnxparser::IOnnxConfig::getVerbosityLevel ( ) const  [pure virtual], [noexcept]
```

Get the Verbosity Level.

Returns

The Verbosity Level.

See also

addVerbosity(), reduceVerbosity()

9.130.4.8 reduceVerbosity()

```
virtual void nvonnxparser::IOnnxConfig::reduceVerbosity ( )  [pure virtual], [noexcept]
```

Reduce the Verbosity Level.

See also

addVerbosity(), setVerbosity(Verbosity)

9.130.4.9 setFullTextFileName()

```
virtual void nvonnxparser::IOnnxConfig::setFullTextFileName (
            char const * fullTextFileName )  [pure virtual], [noexcept]
```

Set the File Name of the Network Description as a Text File, including the weights.

This API allows setting a file name for the network description in plain text, equivalent of the ONNX protobuf.

This method copies the name string.

Parameters

fullTextFileName	Name of the file.

See also

getFullTextFilename()

9.130.4.10 setModelDtype()

```
virtual void nvonnxparser::IOnnxConfig::setModelDtype (
            const nvinfer1::DataType  )  [pure virtual], [noexcept]
```

Set the Model Data Type.

Sets the Model DataType, one of the following: float -d 32 (default), half precision -d 16, and int8 -d 8 data types.

See also

getModelDtype()

9.130.4.11 setModelFileName()

```
virtual void nvonnxparser::IOnnxConfig::setModelFileName (
            char const * onnxFilename )  [pure virtual], [noexcept]
```

Set the Model File Name.

The Model File name contains the Network Description in ONNX pb format.

This method copies the name string.

Parameters

onnxFilename	The name.

See also

getModelFileName()

9.130.4.12 setPrintLayerInfo()

```
virtual void nvonnxparser::IOnnxConfig::setPrintLayerInfo (
            bool  )  [pure virtual], [noexcept]
```

Set whether the layer information will be printed.

See also

getPrintLayerInfo()

9.130.4.13 setTextFileName()

```
virtual void nvonnxparser::IOnnxConfig::setTextFileName (
            char const * textFileName )  [pure virtual], [noexcept]
```

Set the File Name of the Network Description as a Text File.

This API allows setting a file name for the network description in plain text, equivalent of the ONNX protobuf.

This method copies the name string.

Parameters

textFileName	Name of the file.

See also

getTextFilename()

9.130.4.14 setVerbosityLevel()

```
virtual void nvonnxparser::IOnnxConfig::setVerbosityLevel (
            Verbosity  )  [pure virtual], [noexcept]
```

Set to specific verbosity Level.

See also

addVerbosity(), reduceVerbosity()

The documentation for this class was generated from the following file:

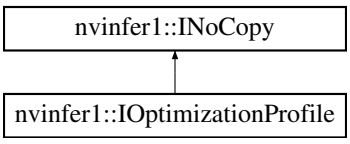
- NvOnnxConfig.h

9.131 nvinfer1::IOptimizationProfile Class Reference

Optimization profile for dynamic input dimensions and shape tensors.

```
#include <NvInferRuntime.h>
```

Inheritance diagram for nvinfer1::IOptimizationProfile:



Public Member Functions

- bool setDimensions (char const *inputName, OptProfileSelector select, Dims const &dims) noexcept

  Set the minimum / optimum / maximum dimensions for a dynamic input tensor.
- Dims getDimensions (char const *inputName, OptProfileSelector select) const noexcept

  Get the minimum / optimum / maximum dimensions for a dynamic input tensor.
- int32_t getNbShapeValues (char const *inputName) const noexcept

  Get the number of values for an input shape tensor.
- bool setExtraMemoryTarget (float target) noexcept

  Set a target for extra GPU memory that may be used by this profile.
- float getExtraMemoryTarget () const noexcept

  Get the extra memory target that has been defined for this profile.
- bool isValid () const noexcept

  Check whether the optimization profile can be passed to an IBuilderConfig object.
- bool setShapeValuesV2 (char const *inputName, OptProfileSelector select, int64_t const *values, int32_t nb↩ Values) noexcept

  Set the minimum / optimum / maximum values for an input shape tensor.
- int64_t const * getShapeValuesV2 (char const *inputName, OptProfileSelector select) const noexcept

  Get the minimum / optimum / maximum values for an input shape tensor.

Protected Member Functions

- virtual ~IOptimizationProfile () noexcept=0

Protected Attributes

- apiv::VOptimizationProfile * mImpl

9.131.1 Detailed Description

Optimization profile for dynamic input dimensions and shape tensors.

When building an ICudaEngine from an INetworkDefinition that has dynamically resizable inputs (at least one input tensor has one or more of its dimensions specified as -1) or shape input tensors, users need to specify at least one optimization profile. Optimization profiles are numbered 0, 1, ... The first optimization profile that has been defined (with index 0) will be used by the ICudaEngine whenever no optimization profile has been selected explicitly. If none of the inputs are dynamic, the default optimization profile will be generated automatically unless it is explicitly provided by the user (this is possible but not required in this case). If more than a single optimization profile is defined, users may set a target how much additional weight space should be maximally allocated to each additional profile (as a fraction of the maximum, unconstrained memory).

Users set optimum input tensor dimensions, as well as minimum and maximum input tensor dimensions. The builder selects the kernels that result in the lowest runtime for the optimum input tensor dimensions, and are valid for all input tensor sizes in the valid range between minimum and maximum dimensions. A runtime error will be raised if the input tensor dimensions fall outside the valid range for this profile. Likewise, users provide minimum, optimum, and maximum values for all shape tensor input values.

See also

IBuilderConfig::addOptimizationProfile()

9.131.2 Constructor & Destructor Documentation

9.131.2.1 ~IOptimizationProfile()

```
nvinfer1::IOptimizationProfile::~IOptimizationProfile ( )  [inline], [protected], [pure virtual],
[default], [noexcept]
```

9.131.3 Member Function Documentation

9.131.3.1 getDimensions()

```
Dims nvinfer1::IOptimizationProfile::getDimensions (
            char const * inputName,
            OptProfileSelector select ) const  [inline], [noexcept]
```

Get the minimum / optimum / maximum dimensions for a dynamic input tensor.

If the dimensions have not been previously set via setDimensions(), return an invalid Dims with nbDims == -1.

> Warning
>
> The string inputName must be null-terminated, and be at most 4096 bytes including the terminator.

9.131.3.2 getExtraMemoryTarget()

```
float nvinfer1::IOptimizationProfile::getExtraMemoryTarget ( ) const  [inline], [noexcept]
```

Get the extra memory target that has been defined for this profile.

This defaults to 1.0F.

Returns

the valid value set by setExtraMemoryTarget or 1.0F.

9.131.3.3 getNbShapeValues()

```
int32_t nvinfer1::IOptimizationProfile::getNbShapeValues (
            char const * inputName ) const  [inline], [noexcept]
```

Get the number of values for an input shape tensor.

This will return the number of shape values if setShapeValuesV2() has been called before for this input tensor. Otherwise, return -1.

Warning

The string inputName must be null-terminated, and be at most 4096 bytes including the terminator.

9.131.3.4 getShapeValuesV2()

```
int64_t const * nvinfer1::IOptimizationProfile::getShapeValuesV2 (
            char const * inputName,
            OptProfileSelector select ) const  [inline], [noexcept]
```

Get the minimum / optimum / maximum values for an input shape tensor.

If the shape values have not been set previously with setShapeValuesV2(), this returns nullptr.

Warning

The string inputName must be null-terminated, and be at most 4096 bytes including the terminator.

9.131.3.5 isValid()

```
bool nvinfer1::IOptimizationProfile::isValid ( ) const  [inline], [noexcept]
```

Check whether the optimization profile can be passed to an IBuilderConfig object.

This function performs partial validation, by e.g. checking that whenever one of the minimum, optimum, or maximum dimensions of a tensor have been set, the others have also been set and have the same rank, as well as checking that the optimum dimensions are always as least as large as the minimum dimensions, and that the maximum dimensions are at least as large as the optimum dimensions. Some validation steps require knowledge of the network definition and are deferred to engine build time.

Returns

true if the optimization profile is valid and may be passed to an IBuilderConfig, else false.

9.131.3.6 setDimensions()

```
bool nvinfer1::IOptimizationProfile::setDimensions (
            char const * inputName,
            OptProfileSelector select,
            Dims const & dims )  [inline], [noexcept]
```

Set the minimum / optimum / maximum dimensions for a dynamic input tensor.

This function must be called three times (for the minimum, optimum, and maximum) for any network input tensor that has dynamic dimensions. If minDims, optDims, and maxDims are the minimum, optimum, and maximum dimensions, and networkDims are the dimensions for this input tensor that are provided to the INetworkDefinition object, then the following conditions must all hold:

(1) minDims.nbDims == optDims.nbDims == maxDims.nbDims == networkDims.nbDims (2) 0 <= minDims.d[i] <= optDims.d[i] <= maxDims.d[i] for i = 0, ..., networkDims.nbDims-1 (3) if networkDims.d[i] != -1, then minDims.d[i] == optDims.d[i] == maxDims.d[i] == networkDims.d[i]

This function may (but need not be) called for an input tensor that does not have dynamic dimensions. In this case, the third argument must always equal networkDims.

Parameters

inputName	The input tensor name
select	Whether to set the minimum, optimum, or maximum dimensions
dims	The minimum, optimum, or maximum dimensions for this input tensor

Returns

false if an inconsistency was detected (e.g. the rank does not match another dimension that was previously set for the same input), true if no inconsistency was detected. Note that inputs can be validated only partially; a full validation is performed at engine build time.

Warning

If run on DLA, minimum, optimum, and maximum dimensions must to be the same.

The string inputName must be null-terminated, and be at most 4096 bytes including the terminator.

9.131.3.7 setExtraMemoryTarget()

```
bool nvinfer1::IOptimizationProfile::setExtraMemoryTarget (
            float target ) [inline], [noexcept]
```

Set a target for extra GPU memory that may be used by this profile.

Parameters

target	Additional memory that the builder should aim to maximally allocate for this profile, as a fraction of the memory it would use if the user did not impose any constraints on memory. This unconstrained case is the default; it corresponds to target == 1.0. If target == 0.0, the builder aims to create the new optimization profile without allocating any additional weight memory. Valid inputs lie between 0.0 and 1.0. This parameter is only a hint, and TensorRT does not guarantee that the target will be reached. This parameter is ignored for the first (default) optimization profile that is defined.

Returns

true if the input is in the valid range (between 0 and 1 inclusive), else false.

9.131.3.8 setShapeValuesV2()

```
bool nvinfer1::IOptimizationProfile::setShapeValuesV2 (
            char const * inputName,
```

```
OptProfileSelector select,
int64_t const * values,
int32_t nbValues )  [inline], [noexcept]
```

Set the minimum / optimum / maximum values for an input shape tensor.

This function must be called three times for every input tensor t that is a shape tensor (t.isShape() == true). This implies that the dimensions of t are fixed at network definition time and the volume does not exceed 64. This function must not be called for any input tensor that is not a shape tensor.

Each time this function is called for the same input tensor, the same nbValues must be supplied (either 1 if the tensor rank is 0, or dims.d[0] if the rank is 1). Furthermore, if minVals, optVals, maxVals are the minimum, optimum, and maximum values, it must be true that minVals[i] <= optVals[i] <= maxVals[i] for i = 0, ..., nbValues - 1. Execution of the network must be valid for the optVals.

Shape tensors are tensors that contribute to shape calculations in some way. While input shape tensors can be type kINT32 or kINT64, the values used to set the minimum, optimum, and maximum values must fit in int64_t.

Examples:

- A shape tensor used as the second input to IShuffleLayer can contain a -1 wildcard. The corresponding minVal[i] should be -1.
- A shape tensor used as the stride input to ISliceLayer can contain any valid strides. The values could be positive, negative, or zero.
- A shape tensor subtracted from zero to compute the size input of an ISliceLayer can contain any non-positive values that yield a valid slice operation.

Tightening the minVals and maxVals bounds to cover only values that are necessary may help optimization.

Parameters

inputName	The input tensor name
select	Whether to set the minimum, optimum, or maximum input values.
values	An array of length nbValues containing the minimum, optimum, or maximum shape tensor elements. For multidimensional tensors, the array is in row-major order.
nbValues	The length of the value array, which must equal the number of shape tensor elements (>= 1)

Returns

false if an inconsistency was detected (e.g. nbValues does not match a previous call for the same tensor), else true. As for setDimensions(), a full validation can only be performed at engine build time.

Warning

If run on DLA, minimum, optimum, and maximum shape values must to be the same.

The string inputName must be null-terminated, and be at most 4096 bytes including the terminator.

9.131.4 Member Data Documentation

9.131.4.1 mImpl

```
apiv::VOptimizationProfile* nvinfer1::IOptimizationProfile::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.132 IOutputAllocator Class Reference

Callback from ExecutionContext::enqueueV3()

`#include <NvInferRuntime.h>`

9.132.1 Detailed Description

Callback from ExecutionContext::enqueueV3()

See also

IExecutionContext::enqueueV3()

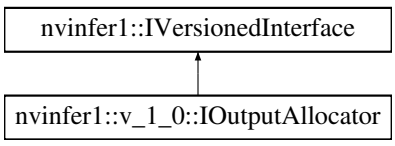
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.133 nvinfer1::v_1_0::IOutputAllocator Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IOutputAllocator:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual TRT_DEPRECATED void * reallocateOutput (char const *, void *, uint64_t, uint64_t) noexcept

  Return a pointer to memory for an output tensor, or nullptr if memory cannot be allocated. If the requested memory size exceeds the currentMemory size, the currentMemory can be freed as well. If currentMemory is known to be big enough, one option is to return currentMemory.
- virtual void * reallocateOutputAsync (char const *tensorName, void *currentMemory, uint64_t size, uint64_t alignment, cudaStream_t)

  Return a pointer to memory for an output tensor, or nullptr if memory cannot be allocated. If the requested memory size exceeds the currentMemory size, the currentMemory can be freed as well. If currentMemory is known to be big enough, one option is to return currentMemory.
- virtual void notifyShape (char const *tensorName, Dims const &dims) noexcept=0

  Called by TensorRT when the shape of the output tensor is known.

Additional Inherited Members

9.133.1 Member Function Documentation

9.133.1.1 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IOutputAllocator::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.133.1.2 notifyShape()

```
virtual void nvinfer1::v_1_0::IOutputAllocator::notifyShape (
            char const * tensorName,
            Dims const & dims )  [pure virtual], [noexcept]
```

Called by TensorRT when the shape of the output tensor is known.

Called by TensorRT sometime between when it calls reallocateOutput and enqueueV3 returns.

Parameters

dims	dimensions of the output
tensorName	name of the tensor

9.133.1.3 reallocateOutput()

```
virtual TRT_DEPRECATED void * nvinfer1::v_1_0::IOutputAllocator::reallocateOutput (
            char const * ,
            void * ,
            uint64_t ,
            uint64_t  )  [inline], [virtual], [noexcept]
```

Return a pointer to memory for an output tensor, or nullptr if memory cannot be allocated. If the requested memory size exceeds the currentMemory size, the currentMemory can be freed as well. If currentMemory is known to be big enough, one option is to return currentMemory.

Parameters

tensorName	name of the output tensor.
currentMemory	points to the address set by IExecutionContext::setTensorAddress.
size	number of bytes required. Always positive, even for an empty tensor.
alignment	required alignment of the allocation.

Returns

A pointer to memory to use for the output tensor or nullptr.

To preallocate memory and have the engine fail if the preallocation is not big enough, use IExecutionContext::setTensorAddress to set a pointer to the preallocated memory, and have reallocateOutput return nullptr if that memory is not big enough.

Deprecated Deprecated in TensorRT 10.0. Superseded by reallocateOutputAsync with cudaStream_t argument

9.133.1.4 reallocateOutputAsync()

```
virtual void * nvinfer1::v_1_0::IOutputAllocator::reallocateOutputAsync (
            char const * tensorName,
            void * currentMemory,
            uint64_t size,
            uint64_t alignment,
            cudaStream_t  )  [inline], [virtual]
```

Return a pointer to memory for an output tensor, or nullptr if memory cannot be allocated. If the requested memory size exceeds the currentMemory size, the currentMemory can be freed as well. If currentMemory is known to be big enough, one option is to return currentMemory.

Parameters

tensorName	name of the output tensor.
currentMemory	points to the address set by IExecutionContext::setTensorAddress.
size	number of bytes required. Always positive, even for an empty tensor.
alignment	required alignment of the allocation.
stream	The stream in which to execute the kernels.

Returns

A pointer to memory to use for the output tensor or nullptr.

To preallocate memory and have the engine fail if the preallocation is not big enough, use IExecutionContext::setTensorAddress to set a pointer to the preallocated memory, and have reallocateOutputAsync return nullptr if that memory is not big enough.

The default definition exists for sake of backward compatibility with earlier versions of TensorRT. Eventually this method will become a pure virtual method that requires an override, and method reallocateOutput() will disappear. Code moving away from TensorRT 9.x should override method reallocateOutputAsync() and NOT override method reallocateOutput().

The documentation for this class was generated from the following file:

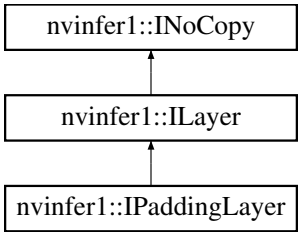
- NvInferRuntime.h

9.134 nvinfer1::IPaddingLayer Class Reference

Layer that represents a padding operation.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IPaddingLayer:



Public Member Functions

- void setPrePaddingNd (Dims const &padding) noexcept

  Set the padding that is applied at the start of the tensor.
- Dims getPrePaddingNd () const noexcept

  Get the padding that is applied at the start of the tensor.
- void setPostPaddingNd (Dims const &padding) noexcept

  Set the padding that is applied at the end of the tensor.
- Dims getPostPaddingNd () const noexcept

  Get the padding that is applied at the end of the tensor.

Protected Member Functions

- virtual ~IPaddingLayer () noexcept=0

Protected Attributes

- apiv::VPaddingLayer * mImpl

9.134.1 Detailed Description

Layer that represents a padding operation.

The padding layer adds zero-padding at the start and end of the input tensor. It supports padding only the last two dimensions. Applying negative padding results in cropping of the input.

To pad across any subset of dimensions, use ISliceLayer with SampleMode::kFILL.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.134.2 Constructor & Destructor Documentation

9.134.2.1 ~IPaddingLayer()

```
nvinfer1::IPaddingLayer::~IPaddingLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.134.3 Member Function Documentation

9.134.3.1 getPostPaddingNd()

```
Dims nvinfer1::IPaddingLayer::getPostPaddingNd ( ) const  [inline], [noexcept]
```

Get the padding that is applied at the end of the tensor.

> Warning
>
> Only 2 dimensional padding is currently supported.

See also

setPostPaddingNd

9.134.3.2 getPrePaddingNd()

```
Dims nvinfer1::IPaddingLayer::getPrePaddingNd ( ) const  [inline], [noexcept]
```

Get the padding that is applied at the start of the tensor.

Warning

Only 2 dimensional padding is currently supported.

See also

setPrePaddingNd

9.134.3.3 setPostPaddingNd()

```
void nvinfer1::IPaddingLayer::setPostPaddingNd (
            Dims const & padding )  [inline], [noexcept]
```

Set the padding that is applied at the end of the tensor.

Negative padding results in trimming the edge by the specified amount

Warning

Only 2 dimensional padding is currently supported.

See also

getPostPaddingNd

9.134.3.4 setPrePaddingNd()

```
void nvinfer1::IPaddingLayer::setPrePaddingNd (
            Dims const & padding )  [inline], [noexcept]
```

Set the padding that is applied at the start of the tensor.

Negative padding results in trimming the edge by the specified amount.

Warning

Only 2 dimensional padding is currently supported.

See also

getPrePaddingNd

9.134.4 Member Data Documentation

9.134.4.1 mImpl

```
apiv::VPaddingLayer* nvinfer1::IPaddingLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

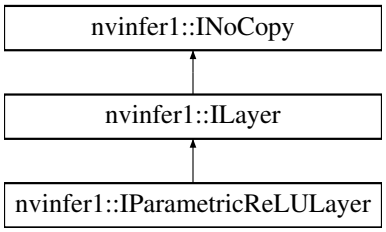
- NvInfer.h

9.135 nvinfer1::IParametricReLULayer Class Reference

Layer that represents a parametric ReLU operation.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IParametricReLULayer:



Protected Member Functions

- virtual ~IParametricReLULayer () noexcept=0

Protected Attributes

- apiv::VParametricReLULayer * mImpl

Additional Inherited Members

9.135.1 Detailed Description

Layer that represents a parametric ReLU operation.

When running this layer on DLA, the slopes input must be a build-time constant.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.135.2 Constructor & Destructor Documentation

9.135.2.1 ~IParametricReLULayer()

```
nvinfer1::IParametricReLULayer::~IParametricReLULayer ( )  [inline], [protected], [pure virtual],
[default], [noexcept]
```

9.135.3 Member Data Documentation

9.135.3.1 mImpl

```
apiv::VParametricReLULayer* nvinfer1::IParametricReLULayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.136 nvonnxparser::IParser Class Reference

an object for parsing ONNX models into a TensorRT network definition

```
#include <NvOnnxParser.h>
```

Public Member Functions

- virtual bool parse (void const *serialized_onnx_model, size_t serialized_onnx_model_size, const char *model_path=nullptr) noexcept=0

  Parse a serialized ONNX model into the TensorRT network. This method has very limited diagnostics. If parsing the serialized model fails for any reason (e.g. unsupported IR version, unsupported opset, etc.) it the user responsibility to intercept and report the error. To obtain a better diagnostic, use the parseFromFile method below.
- virtual bool parseFromFile (char const *onnxModelFile, int verbosity) noexcept=0

  Parse an onnx model file, which can be a binary protobuf or a text onnx model calls parse method inside.
- virtual bool supportsOperator (const char *op_name) const noexcept=0

  Returns whether the specified operator may be supported by the parser.
- virtual int getNbErrors () const noexcept=0

  Get the number of errors that occurred during prior calls to `parse`.
- virtual IParserError const * getError (int index) const noexcept=0

  Get an error that occurred during prior calls to `parse`.
- virtual void clearErrors () noexcept=0

  Clear errors from prior calls to `parse`.
- virtual ~IParser () noexcept=default
- virtual char const *const * getUsedVCPluginLibraries (int64_t &nbPluginLibs) const noexcept=0

  Query the plugin libraries needed to implement operations used by the parser in a version-compatible engine.
- virtual void setFlags (OnnxParserFlags onnxParserFlags) noexcept=0

  Set the parser flags.
- virtual OnnxParserFlags getFlags () const noexcept=0

  Get the parser flags. Defaults to 0.
- virtual void clearFlag (OnnxParserFlag onnxParserFlag) noexcept=0

  clear a parser flag.
- virtual void setFlag (OnnxParserFlag onnxParserFlag) noexcept=0

  Set a single parser flag.
- virtual bool getFlag (OnnxParserFlag onnxParserFlag) const noexcept=0

  Returns true if the parser flag is set.
- virtual nvinfer1::ITensor const * getLayerOutputTensor (char const *name, int64_t i) noexcept=0

  Return the i-th output ITensor object for the ONNX layer "name".
- virtual bool supportsModelV2 (void const *serializedOnnxModel, size_t serializedOnnxModelSize, char const *modelPath=nullptr) noexcept=0

  Check whether TensorRT supports a particular ONNX model. If the function returns True, one can proceed to engine building without having to call `parse` or `parseFromFile`. Results can be queried through `getNbSubgraphs`, `isSubgraphSupported`, `getSubgraphNodes`.
- virtual int64_t getNbSubgraphs () noexcept=0

  Get the number of subgraphs. Calling this function before calling `supportsModelV2` results in undefined behavior.
- virtual bool isSubgraphSupported (int64_t const index) noexcept=0

  Returns whether the subgraph is supported. Calling this function before calling `supportsModelV2` results in undefined behavior.
- virtual int64_t * getSubgraphNodes (int64_t const index, int64_t &subgraphLength) noexcept=0

  Get the nodes of the specified subgraph. Calling this function before calling `supportsModelV2` results in undefined behavior.
- virtual bool loadModelProto (void const *serializedOnnxModel, size_t serializedOnnxModelSize, char const *modelPath=nullptr) noexcept=0

  Load a serialized ONNX model into the parser. Unlike the parse() or parseFromFile() functions, this function does not immediately convert the model into a TensorRT INetworkDefinition. Using this function allows users to provide their own initializers for the ONNX model through the loadInitializer() function.

- virtual bool loadInitializer (char const *name, void const *data, size_t size) noexcept=0

  Prompt the ONNX parser to load an initializer with user-provided binary data. The lifetime of the data must exceed the lifetime of the parser.
- virtual bool parseModelProto () noexcept=0

  Begin the parsing and conversion process of the loaded ONNX model into a TensorRT INetworkDefinition.
- virtual bool setBuilderConfig (const nvinfer1::IBuilderConfig *const builderConfig) noexcept=0

  Set the BuilderConfig for the parser.

9.136.1 Detailed Description

an object for parsing ONNX models into a TensorRT network definition

Warning

If the ONNX model has a graph output with the same name as a graph input, the output will be renamed by prepending "__".

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.136.2 Constructor & Destructor Documentation

9.136.2.1 ~IParser()

```
virtual nvonnxparser::IParser::~IParser ( )  [virtual], [default], [noexcept]
```

9.136.3 Member Function Documentation

9.136.3.1 clearErrors()

```
virtual void nvonnxparser::IParser::clearErrors ( )  [pure virtual], [noexcept]
```

Clear errors from prior calls to `parse`.

See also

getNbErrors() getError() IParserError

9.136.3.2 clearFlag()

```
virtual void nvonnxparser::IParser::clearFlag (
            OnnxParserFlag onnxParserFlag )  [pure virtual], [noexcept]
```

clear a parser flag.

clears the parser flag from the enabled flags.

See also

setFlags()

9.136.3.3 getError()

```
virtual IParserError const * nvonnxparser::IParser::getError (
            int index ) const  [pure virtual], [noexcept]
```

Get an error that occurred during prior calls to `parse`.

See also

getNbErrors() clearErrors() IParserError

9.136.3.4 getFlag()

```
virtual bool nvonnxparser::IParser::getFlag (
            OnnxParserFlag onnxParserFlag ) const  [pure virtual], [noexcept]
```

Returns true if the parser flag is set.

See also

getFlags()

Returns

True if flag is set, false if unset.

9.136.3.5 getFlags()

```
virtual OnnxParserFlags nvonnxparser::IParser::getFlags ( ) const  [pure virtual], [noexcept]
```

Get the parser flags. Defaults to 0.

Returns

The parser flags as a bitmask.

See also

setFlags()

9.136.3.6 getLayerOutputTensor()

```
virtual nvinfer1::ITensor const * nvonnxparser::IParser::getLayerOutputTensor (
            char const * name,
            int64_t i )  [pure virtual], [noexcept]
```

Return the i-th output ITensor object for the ONNX layer ”name”.

Return the i-th output ITensor object for the ONNX layer ”name”. If ”name” is not found or i is out of range, return nullptr. In the case of multiple nodes sharing the same name this function will return the output tensors of the first instance of the node in the ONNX graph.

Parameters

name	The name of the ONNX layer.
i	The index of the output. i must be in range [0, layer.num_outputs).

9.136.3.7 getNbErrors()

```
virtual int nvonnxparser::IParser::getNbErrors ( ) const  [pure virtual], [noexcept]
```

Get the number of errors that occurred during prior calls to `parse`.

See also

getError() clearErrors() IParserError

9.136.3.8 getNbSubgraphs()

```
virtual int64_t nvonnxparser::IParser::getNbSubgraphs ( )  [pure virtual], [noexcept]
```

Get the number of subgraphs. Calling this function before calling `supportsModelV2` results in undefined behavior.

Returns

Number of subgraphs.

9.136.3.9 getSubgraphNodes()

```
virtual int64_t * nvonnxparser::IParser::getSubgraphNodes (
             int64_t const index,
             int64_t & subgraphLength )  [pure virtual], [noexcept]
```

Get the nodes of the specified subgraph. Calling this function before calling `supportsModelV2` results in undefined behavior.

Parameters

index	Index of the subgraph.
subgraphLength	Returns the length of the subgraph as reference.

Returns

Pointer to the subgraph nodes array. This pointer is owned by the Parser.

9.136.3.10 getUsedVCPluginLibraries()

```
virtual char const *const * nvonnxparser::IParser::getUsedVCPluginLibraries (
             int64_t & nbPluginLibs ) const  [pure virtual], [noexcept]
```

Query the plugin libraries needed to implement operations used by the parser in a version-compatible engine.

This provides a list of plugin libraries on the filesystem needed to implement operations in the parsed network. If you are building a version-compatible engine using this network, provide this list to IBuilderConfig::setPluginsToSerialize to serialize these plugins along with the version-compatible engine, or, if you want to ship these plugin libraries externally to the engine, ensure that IPluginRegistry::loadLibrary is used to load these libraries in the appropriate runtime before deserializing the corresponding engine.

Parameters

out	*nbPluginLibs*	Returns the number of plugin libraries in the array, or -1 if there was an error.

Returns

Array of `nbPluginLibs` C-strings describing plugin library paths on the filesystem if nbPluginLibs > 0, or nullptr otherwise. This array is owned by the IParser, and the pointers in the array are only valid until the next call to parse() or parseFromFile().

9.136.3.11 isSubgraphSupported()

```
virtual bool nvonnxparser::IParser::isSubgraphSupported (
            int64_t const index )  [pure virtual], [noexcept]
```

Returns whether the subgraph is supported. Calling this function before calling `supportsModelV2` results in undefined behavior.

Parameters

index	Index of the subgraph.

Returns

Whether the subgraph is supported.

9.136.3.12 loadInitializer()

```
virtual bool nvonnxparser::IParser::loadInitializer (
            char const * name,
            void const * data,
            size_t size )  [pure virtual], [noexcept]
```

Prompt the ONNX parser to load an initializer with user-provided binary data. The lifetime of the data must exceed the lifetime of the parser.

All user-provided initializers must be provided prior to calling refitModelProto().

This function can be called multiple times to specify the names of multiple initializers.

Calling this function with an initializer previously specified will overwrite the previous instance.

This function will return false if initializer validation fails. Possible validation errors are:

- This function was called prior to loadModelProto().
- The requested initializer was not found in the model.
- The size of the data provided is different from the corresponding initializer in the model.

Parameters

name	Name of the initializer.
data	Binary data containing the values of the initializer.
size	Size of the initializer in bytes.

Returns

true if the initializer was loaded successfully

See also

loadModelProto()

9.136.3.13 loadModelProto()

```
virtual bool nvonnxparser::IParser::loadModelProto (
            void const * serializedOnnxModel,
            size_t serializedOnnxModelSize,
            char const * modelPath = nullptr )  [pure virtual], [noexcept]
```

Load a serialized ONNX model into the parser. Unlike the parse() or parseFromFile() functions, this function does not immediately convert the model into a TensorRT INetworkDefinition. Using this function allows users to provide their own initializers for the ONNX model through the loadInitializer() function.

Only one model can be loaded at a time. Subsequent calls to loadModelProto() will result in an error.

To begin the conversion of the model into a TensorRT INetworkDefinition, use parseModelProto().

Parameters

serializedOnnxModel	Pointer to the serialized ONNX model. Can be freed after this function returns.
serializedOnnxModelSize	Size of the serialized ONNX model in bytes.
modelPath	Absolute path to the model file for loading external weights if required.

Returns

true if the model was loaded successfully

See also

getNbErrors() getError()

9.136.3.14 parse()

```
virtual bool nvonnxparser::IParser::parse (
            void const * serialized_onnx_model,
            size_t serialized_onnx_model_size,
            const char * model_path = nullptr )  [pure virtual], [noexcept]
```

Parse a serialized ONNX model into the TensorRT network. This method has very limited diagnostics. If parsing the serialized model fails for any reason (e.g. unsupported IR version, unsupported opset, etc.) it the user responsibility to intercept and report the error. To obtain a better diagnostic, use the parseFromFile method below.

Parameters

serialized_onnx_model	Pointer to the serialized ONNX model. Can be freed after this function returns.
serialized_onnx_model_size	Size of the serialized ONNX model in bytes
model_path	Absolute path to the model file for loading external weights if required

Returns

true if the model was parsed successfully

See also

getNbErrors() getError()

9.136.3.15 parseFromFile()

```
virtual bool nvonnxparser::IParser::parseFromFile (
            char const * onnxModelFile,
            int verbosity )  [pure virtual], [noexcept]
```

Parse an onnx model file, which can be a binary protobuf or a text onnx model calls parse method inside.

Parameters

onnxModelFile	name
verbosity	Level

Returns

true if the model was parsed successfully

9.136.3.16 parseModelProto()

```
virtual bool nvonnxparser::IParser::parseModelProto ( )  [pure virtual], [noexcept]
```

Begin the parsing and conversion process of the loaded ONNX model into a TensorRT INetworkDefinition.

Returns

true if conversion was successful

See also

getNbErrors() getError() loadModelProto() loadModelProtoFromFile()

9.136.3.17 setBuilderConfig()

```
virtual bool nvonnxparser::IParser::setBuilderConfig (
            const nvinfer1::IBuilderConfig *const builderConfig )  [pure virtual], [noexcept]
```

Set the BuilderConfig for the parser.

Returns

true if the IBuilderConfig was set successfully, false otherwise.

9.136.3.18 setFlag()

```
virtual void nvonnxparser::IParser::setFlag (
            OnnxParserFlag onnxParserFlag )  [pure virtual], [noexcept]
```

Set a single parser flag.

Add the input parser flag to the already enabled flags.

See also

setFlags()

9.136.3.19 setFlags()

```
virtual void nvonnxparser::IParser::setFlags (
            OnnxParserFlags onnxParserFlags )  [pure virtual], [noexcept]
```

Set the parser flags.

The flags are listed in the OnnxParserFlag enum.

Parameters

OnnxParserFlags	The flags used when parsing an ONNX model.

Note

This function will override the previous set flags, rather than bitwise ORing the new flag.

See also

getFlags()

9.136.3.20 supportsModelV2()

```
virtual bool nvonnxparser::IParser::supportsModelV2 (
            void const * serializedOnnxModel,
            size_t serializedOnnxModelSize,
            char const * modelPath = nullptr )  [pure virtual], [noexcept]
```

Check whether TensorRT supports a particular ONNX model. If the function returns True, one can proceed to engine building without having to call `parse` or `parseFromFile`. Results can be queried through `getNbSubgraphs`, `isSubgraphSupported`, `getSubgraphNodes`.

Parameters

serializedOnnxModel	Pointer to the serialized ONNX model. Can be freed after this function returns.
serializedOnnxModelSize	Size of the serialized ONNX model in bytes
modelPath	Absolute path to the model file for loading external weights if required

Returns

true if the model is supported

9.136.3.21 supportsOperator()

```
virtual bool nvonnxparser::IParser::supportsOperator (
            const char * op_name ) const  [pure virtual], [noexcept]
```

Returns whether the specified operator may be supported by the parser.

Note that a result of true does not guarantee that the operator will be supported in all cases (i.e., this function may return false-positives).

Parameters

op_name	The name of the ONNX operator to check for support

The documentation for this class was generated from the following file:

- NvOnnxParser.h

9.137 nvonnxparser::IParserError Class Reference

an object containing information about an error

`#include <NvOnnxParser.h>`

Public Member Functions

- virtual ErrorCode code () const =0

  the error code.
- virtual char const * desc () const =0

  description of the error.
- virtual char const * file () const =0

  source file in which the error occurred.
- virtual int line () const =0

  source line at which the error occurred.
- virtual char const * func () const =0

  source function in which the error occurred.
- virtual int node () const =0

  index of the ONNX model node in which the error occurred.
- virtual char const * nodeName () const =0

  name of the node in which the error occurred.
- virtual char const * nodeOperator () const =0

  name of the node operation in which the error occurred.
- virtual char const *const * localFunctionStack () const =0

  A list of the local function names, from the top level down, constituting the current stack trace in which the error occurred. A top-level node that is not inside any local function would return a nullptr.
- virtual int32_t localFunctionStackSize () const =0

  The size of the stack of local functions at the point where the error occurred. A top-level node that is not inside any local function would correspond to.

Protected Member Functions

- virtual ~IParserError ()

9.137.1 Detailed Description

an object containing information about an error

9.137.2 Constructor & Destructor Documentation

9.137.2.1 ~IParserError()

```
virtual nvonnxparser::IParserError::~IParserError ( )  [inline], [protected], [virtual]
```

9.137.3 Member Function Documentation

9.137.3.1 code()

```
virtual ErrorCode nvonnxparser::IParserError::code ( ) const  [pure virtual]
```

the error code.

9.137.3.2 desc()

```
virtual char const * nvonnxparser::IParserError::desc ( ) const  [pure virtual]
```

description of the error.

9.137.3.3 file()

```
virtual char const * nvonnxparser::IParserError::file ( ) const  [pure virtual]
```

source file in which the error occurred.

9.137.3.4 func()

```
virtual char const * nvonnxparser::IParserError::func ( ) const  [pure virtual]
```

source function in which the error occurred.

9.137.3.5 line()

```
virtual int nvonnxparser::IParserError::line ( ) const  [pure virtual]
```

source line at which the error occurred.

9.137.3.6 localFunctionStack()

```
virtual char const *const * nvonnxparser::IParserError::localFunctionStack ( ) const  [pure virtual]
```

A list of the local function names, from the top level down, constituting the current stack trace in which the error occurred. A top-level node that is not inside any local function would return a nullptr.

9.137.3.7 localFunctionStackSize()

```
virtual int32_t nvonnxparser::IParserError::localFunctionStackSize ( ) const  [pure virtual]
```

The size of the stack of local functions at the point where the error occurred. A top-level node that is not inside any local function would correspond to.

9.137.3.8 node()

```
virtual int nvonnxparser::IParserError::node ( ) const  [pure virtual]
```

index of the ONNX model node in which the error occurred.

9.137.3.9 nodeName()

```
virtual char const * nvonnxparser::IParserError::nodeName ( ) const  [pure virtual]
```

name of the node in which the error occurred.

9.137.3.10 nodeOperator()

```
virtual char const * nvonnxparser::IParserError::nodeOperator ( ) const  [pure virtual]
```

name of the node operation in which the error occurred.

The documentation for this class was generated from the following file:

- NvOnnxParser.h

9.138 nvonnxparser::IParserRefitter Class Reference

An interface designed to refit weights from an ONNX model.

`#include <NvOnnxParser.h>`

Public Member Functions

- virtual bool refitFromBytes (void const *serializedOnnxModel, size_t serializedOnnxModelSize, char const *modelPath=nullptr) noexcept=0

  Load a serialized ONNX model from memory and perform weight refit.
- virtual bool refitFromFile (char const *onnxModelFile) noexcept=0

  Load and parse a ONNX model from disk and perform weight refit.
- virtual int32_t getNbErrors () const noexcept=0

  Get the number of errors that occurred during prior calls to `refitFromBytes` or `refitFromFile`.
- virtual IParserError const * getError (int32_t index) const noexcept=0

  Get an error that occurred during prior calls to `refitFromBytes` or `refitFromFile`.
- virtual void clearErrors ()=0

  Clear errors from prior calls to `refitFromBytes` or `refitFromFile`.
- virtual ~IParserRefitter () noexcept=default
- virtual bool loadModelProto (void const *serializedOnnxModel, size_t serializedOnnxModelSize, char const *modelPath=nullptr) noexcept=0

  Load a serialized ONNX model into the parser. Unlike the refit(), or refitFromFile() functions, this function does not immediately begin the refit process. Using this function allows users to provide their own initializers for the ONNX model through the loadInitializer() function.
- virtual bool loadInitializer (char const *name, void const *data, size_t size) noexcept=0

  Prompt the ONNX refitter to load an initializer with user-provided binary data. The lifetime of the data must exceed the lifetime of the refitter.
- virtual bool refitModelProto () noexcept=0

  Begin the refit process from the loaded ONNX model.

9.138.1 Detailed Description

An interface designed to refit weights from an ONNX model.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.138.2 Constructor & Destructor Documentation

9.138.2.1 ~IParserRefitter()

```
virtual nvonnxparser::IParserRefitter::~IParserRefitter ( )  [virtual], [default], [noexcept]
```

9.138.3 Member Function Documentation

9.138.3.1 clearErrors()

```
virtual void nvonnxparser::IParserRefitter::clearErrors ( )  [pure virtual]
```

Clear errors from prior calls to `refitFromBytes` or `refitFromFile`.

See also

getNbErrors() getError() IParserError

9.138.3.2 getError()

```
virtual IParserError const * nvonnxparser::IParserRefitter::getError (
            int32_t index ) const  [pure virtual], [noexcept]
```

Get an error that occurred during prior calls to `refitFromBytes` or `refitFromFile`.

See also

getNbErrors() IParserError

9.138.3.3 getNbErrors()

```
virtual int32_t nvonnxparser::IParserRefitter::getNbErrors ( ) const  [pure virtual], [noexcept]
```

Get the number of errors that occurred during prior calls to `refitFromBytes` or `refitFromFile`.

See also

getError() IParserError

9.138.3.4 loadInitializer()

```
virtual bool nvonnxparser::IParserRefitter::loadInitializer (
            char const * name,
            void const * data,
            size_t size ) [pure virtual], [noexcept]
```

Prompt the ONNX refitter to load an initializer with user-provided binary data. The lifetime of the data must exceed the lifetime of the refitter.

All user-provided initializers must be provided prior to calling refitModelProto().

This function can be called multiple times to specify the names of multiple initializers.

Calling this function with an initializer previously specified will overwrite the previous instance.

This function will return false if initializer validation fails. Possible validation errors are:

- This function was called prior to loadModelProto()
- The requested initializer was not found in the model.
- The size of the data provided is different from the corresponding initializer in the model.

Parameters

name	Name of the initializer.
data	Binary data containing the values of the initializer.
size	Size of the initializer in bytes.

Returns

true if the initializer was loaded successfully

See also

loadModelProto()

9.138.3.5 loadModelProto()

```
virtual bool nvonnxparser::IParserRefitter::loadModelProto (
            void const * serializedOnnxModel,
            size_t serializedOnnxModelSize,
            char const * modelPath = nullptr )  [pure virtual], [noexcept]
```

Load a serialized ONNX model into the parser. Unlike the refit(), or refitFromFile() functions, this function does not immediately begin the refit process. Using this function allows users to provide their own initializers for the ONNX model through the loadInitializer() function.

Only one model can be loaded at a time. Subsequent calls to loadModelProto() will result in an error.

To begin the refit process, use refitModelProto().

Parameters

serializedOnnxModel	Pointer to the serialized ONNX model. Can be freed after this function returns.
serializedOnnxModelSize	Size of the serialized ONNX model in bytes.
modelPath	Absolute path to the model file for loading external weights if required.

Returns

true if the model was loaded successfully

See also

getNbErrors() getError()

9.138.3.6 refitFromBytes()

```
virtual bool nvonnxparser::IParserRefitter::refitFromBytes (
            void const * serializedOnnxModel,
            size_t serializedOnnxModelSize,
            char const * modelPath = nullptr )  [pure virtual], [noexcept]
```

Load a serialized ONNX model from memory and perform weight refit.

Parameters

serializedOnnxModel	Pointer to the serialized ONNX model
serializedOnnxModelSize	Size of the serialized ONNX model in bytes
modelPath	Absolute path to the model file for loading external weights if required

Returns

true if all the weights in the engine were refit successfully.

The serialized ONNX model must be identical to the one used to generate the engine that will be refit.

9.138.3.7 refitFromFile()

```
virtual bool nvonnxparser::IParserRefitter::refitFromFile (
            char const * onnxModelFile )  [pure virtual], [noexcept]
```

Load and parse a ONNX model from disk and perform weight refit.

Parameters

onnxModelFile	Path to the ONNX model to load from disk.

Returns

true if the model was loaded successfully, and if all the weights in the engine were refit successfully.

The provided ONNX model must be identical to the one used to generate the engine that will be refit.

9.138.3.8 refitModelProto()

```
virtual bool nvonnxparser::IParserRefitter::refitModelProto ( )  [pure virtual], [noexcept]
```

Begin the refit process from the loaded ONNX model.

Returns

true if refit was successful

See also

getNbErrors() getError() loadModelProto()

The documentation for this class was generated from the following file:

- NvOnnxParser.h

9.139 IPluginCapability Class Reference

Base class for plugin capability interfaces.

`#include <NvInferPluginBase.h>`

9.139.1 Detailed Description

Base class for plugin capability interfaces.

IPluginCapability represents a split in TensorRT V3 plugins to sub-objects that expose different types of capabilites a plugin may have, as opposed to a single interface which defines all capabilities and behaviors of a plugin.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

See also

PluginCapabilityType

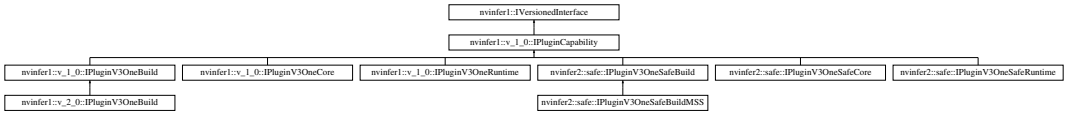
The documentation for this class was generated from the following file:

- NvInferPluginBase.h

9.140 nvinfer1::v_1_0::IPluginCapability Class Reference

`#include <NvInferPluginBase.h>`

Inheritance diagram for nvinfer1::v_1_0::IPluginCapability:



Additional Inherited Members

The documentation for this class was generated from the following file:

- NvInferPluginBase.h

9.141 nvinfer2::safe::consistency::IPluginChecker Class Reference

Interface for validating a plugin used in a TensorRT engine.

`#include <NvInferConsistency.h>`

Public Member Functions

- virtual bool validate (std::vector< TensorDescriptor > const &Inputs, std::vector< TensorDescriptor > const &Outputs, PluginFieldCollection *fc) noexcept=0

  Validate a plugin.
- virtual ~IPluginChecker ()=default

  De-allocates any internally allocated memory.

Protected Member Functions

- IPluginChecker ()=default

  Default constructor for IPluginChecker.
- IPluginChecker (IPluginChecker const &other)=delete

  Deleted copy constructor to maintain non-copyability.
- IPluginChecker & operator= (IPluginChecker const &other) &=delete

  Deleted copy assignment operator to maintain non-copyability.
- IPluginChecker (IPluginChecker &&other)=delete

  Deleted move constructor to maintain non-movability.
- IPluginChecker & operator= (IPluginChecker &&other) &=delete

  Deleted move assignment operator to maintain non-movability.

9.141.1 Detailed Description

Interface for validating a plugin used in a TensorRT engine.

Used by the consistency checker when validating an engine blob that contains plugins. Implementations check that plugin inputs, outputs, and field collection are consistent and valid. There is no public constructor; only derived types or the consistency checker create instances.

9.141.2 Constructor & Destructor Documentation

9.141.2.1 ~IPluginChecker()

`virtual nvinfer2::safe::consistency::IPluginChecker::~IPluginChecker ( ) [virtual], [default]`

De-allocates any internally allocated memory.

9.141.2.2 IPluginChecker() [1/3]

```
nvinfer2::safe::consistency::IPluginChecker::IPluginChecker ( )  [protected], [default]
```

Default constructor for IPluginChecker.

Constructs an object in a default state. Access is protected; only derived types may create instances. The object is not yet bound to any plugin validation context until used by the consistency checker.

Precondition

none

Postcondition

Object is in default-constructed state and is not yet validating any plugin.

9.141.2.3 IPluginChecker() [2/3]

```
nvinfer2::safe::consistency::IPluginChecker::IPluginChecker (
            IPluginChecker const & other )  [protected], [delete]
```

Deleted copy constructor to maintain non-copyability.

Precondition

none

Postcondition

none

9.141.2.4 IPluginChecker() [3/3]

```
nvinfer2::safe::consistency::IPluginChecker::IPluginChecker (
            IPluginChecker && other )  [protected], [delete]
```

Deleted move constructor to maintain non-movability.

Precondition

none

Postcondition

none

9.141.3 Member Function Documentation

9.141.3.1 operator=() [1/2]

```
IPluginChecker & nvinfer2::safe::consistency::IPluginChecker::operator= (
            IPluginChecker && other ) &  [protected], [delete]
```

Deleted move assignment operator to maintain non-movability.

Precondition

none

Postcondition

none

9.141.3.2 operator=() [2/2]

```
IPluginChecker & nvinfer2::safe::consistency::IPluginChecker::operator= (
            IPluginChecker const & other ) &  [protected], [delete]
```

Deleted copy assignment operator to maintain non-copyability.

Precondition

none

Postcondition

none

9.141.3.3 validate()

```
virtual bool nvinfer2::safe::consistency::IPluginChecker::validate (
            std::vector< TensorDescriptor > const & Inputs,
            std::vector< TensorDescriptor > const & Outputs,
            PluginFieldCollection * fc )  [pure virtual], [noexcept]
```

Validate a plugin.

Parameters

Inputs	The input tensors of the plugin.
Outputs	The output tensors of the plugin.
fc	The plugin field collection.

The documentation for this class was generated from the following file:

- NvInferConsistency.h

9.142 IPluginCreator Class Reference

Plugin creator class for user implemented layers.

`#include <NvInferRuntimePlugin.h>`

9.142.1 Detailed Description

Plugin creator class for user implemented layers.

See also

IPlugin and IPluginFactory

Deprecated Deprecated in TensorRT 10.0. Please implement IPluginCreatorV3One along with IPluginV3 plugins instead.

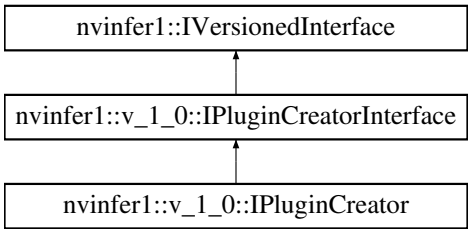
The documentation for this class was generated from the following file:

- NvInferRuntimePlugin.h

9.143 nvinfer1::v_1_0::IPluginCreator Class Reference

`#include <NvInferRuntimePlugin.h>`

Inheritance diagram for nvinfer1::v_1_0::IPluginCreator:



Public Member Functions

- virtual AsciiChar const * getPluginName () const noexcept=0

  Return the plugin name.
- virtual AsciiChar const * getPluginVersion () const noexcept=0

  Return the plugin version.
- virtual PluginFieldCollection const * getFieldNames () noexcept=0

  Return a list of fields that need to be passed to createPlugin.
- virtual IPluginV2 * createPlugin (AsciiChar const *name, PluginFieldCollection const *fc) noexcept=0

  Return a plugin object. Return nullptr in case of error.
- virtual IPluginV2 * deserializePlugin (AsciiChar const *name, void const *serialData, size_t serialLength) noexcept=0

  Called during deserialization of plugin layer. Return a plugin object.
- virtual void setPluginNamespace (AsciiChar const *pluginNamespace) noexcept=0

  Set the namespace of the plugin creator based on the plugin library it belongs to. This can be set while registering the plugin creator.
- virtual AsciiChar const * getPluginNamespace () const noexcept=0

  Return the namespace of the plugin creator object.
- IPluginCreator ()=default
- ~IPluginCreator () override=default
- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.

Additional Inherited Members

9.143.1 Constructor & Destructor Documentation

9.143.1.1 IPluginCreator()

```
nvinfer1::v_1_0::IPluginCreator::IPluginCreator ( )  [default]
```

9.143.1.2 ~IPluginCreator()

```
nvinfer1::v_1_0::IPluginCreator::~IPluginCreator ( )  [override], [default]
```

9.143.2 Member Function Documentation

9.143.2.1 createPlugin()

```
virtual IPluginV2 * nvinfer1::v_1_0::IPluginCreator::createPlugin (
            AsciiChar const * name,
            PluginFieldCollection const * fc )  [pure virtual], [noexcept]
```

Return a plugin object. Return nullptr in case of error.

Parameters

name	A NULL-terminated name string of length 1024 or less, including the NULL terminator.
fc	A pointer to a collection of fields needed for constructing the plugin.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when deserializing multiple engines concurrently sharing plugins.

9.143.2.2 deserializePlugin()

```
virtual IPluginV2 * nvinfer1::v_1_0::IPluginCreator::deserializePlugin (
            AsciiChar const * name,
            void const * serialData,
            size_t serialLength ) [pure virtual], [noexcept]
```

Called during deserialization of plugin layer. Return a plugin object.

Parameters

name	A NULL-terminated name string of length 1024 or less, including the NULL terminator.
serialData	The start address of a byte array with the serialized plugin representation.
serialLength	The length in bytes of the byte array with the serialized plugin representation.

Returns

A deserialized plugin object

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when deserializing multiple engines concurrently sharing plugins.

9.143.2.3 getFieldNames()

```
virtual PluginFieldCollection const * nvinfer1::v_1_0::IPluginCreator::getFieldNames ( ) [pure
virtual], [noexcept]
```

Return a list of fields that need to be passed to createPlugin.

See also

PluginFieldCollection

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when deserializing multiple engines concurrently sharing plugins.

9.143.2.4 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IPluginCreator::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.143.2.5 getPluginName()

```
virtual AsciiChar const * nvinfer1::v_1_0::IPluginCreator::getPluginName ( ) const  [pure virtual],
[noexcept]
```

Return the plugin name.

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when deserializing multiple engines concurrently sharing plugins.

9.143.2.6 getPluginNamespace()

```
virtual AsciiChar const * nvinfer1::v_1_0::IPluginCreator::getPluginNamespace ( ) const  [pure
virtual], [noexcept]
```

Return the namespace of the plugin creator object.

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when deserializing multiple engines concurrently sharing plugins.

9.143.2.7 getPluginVersion()

```
virtual AsciiChar const * nvinfer1::v_1_0::IPluginCreator::getPluginVersion ( ) const  [pure virtual],
[noexcept]
```

Return the plugin version.

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when deserializing multiple engines concurrently sharing plugins.

9.143.2.8 setPluginNamespace()

```
virtual void nvinfer1::v_1_0::IPluginCreator::setPluginNamespace (
            AsciiChar const * pluginNamespace )  [pure virtual], [noexcept]
```

Set the namespace of the plugin creator based on the plugin library it belongs to. This can be set while registering the plugin creator.

Parameters

pluginNamespace	A NULL-terminated namespace string of length 1024 or less, including the NULL terminator

See also

IPluginRegistry::registerCreator()

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when deserializing multiple engines concurrently sharing plugins.

The documentation for this class was generated from the following file:

- NvInferRuntimePlugin.h

9.144 IPluginCreatorInterface Class Reference

Base class for all plugin creator versions.

`#include <NvInferPluginBase.h>`

9.144.1 Detailed Description

Base class for all plugin creator versions.

See also

IPluginCreator and IPluginRegistry

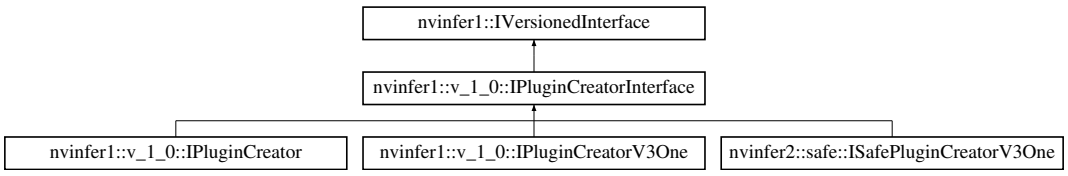
The documentation for this class was generated from the following file:

- NvInferPluginBase.h

9.145 nvinfer1::v_1_0::IPluginCreatorInterface Class Reference

`#include <NvInferPluginBase.h>`

Inheritance diagram for nvinfer1::v_1_0::IPluginCreatorInterface:



Public Member Functions

- ~IPluginCreatorInterface () noexcept override=default

Protected Member Functions

- IPluginCreatorInterface ()=default
- IPluginCreatorInterface (IPluginCreatorInterface const &)=default
- IPluginCreatorInterface (IPluginCreatorInterface &&)=default
- IPluginCreatorInterface & operator= (IPluginCreatorInterface const &) &=default
- IPluginCreatorInterface & operator= (IPluginCreatorInterface &&) &=default

9.145.1 Constructor & Destructor Documentation

9.145.1.1 ~IPluginCreatorInterface()

```
nvinfer1::v_1_0::IPluginCreatorInterface::~IPluginCreatorInterface ( )  [override], [default],
[noexcept]
```

9.145.1.2 IPluginCreatorInterface() [1/3]

```
nvinfer1::v_1_0::IPluginCreatorInterface::IPluginCreatorInterface ( )  [protected], [default]
```

9.145.1.3 IPluginCreatorInterface() [2/3]

```
nvinfer1::v_1_0::IPluginCreatorInterface::IPluginCreatorInterface (
            IPluginCreatorInterface const &  )  [protected], [default]
```

9.145.1.4 IPluginCreatorInterface() [3/3]

```
nvinfer1::v_1_0::IPluginCreatorInterface::IPluginCreatorInterface (
            IPluginCreatorInterface &&  )  [protected], [default]
```

9.145.2 Member Function Documentation

9.145.2.1 operator=() [1/2]

```
IPluginCreatorInterface & nvinfer1::v_1_0::IPluginCreatorInterface::operator= (
            IPluginCreatorInterface &&  ) &  [protected], [default]
```

9.145.2.2 operator=() [2/2]

```
IPluginCreatorInterface & nvinfer1::v_1_0::IPluginCreatorInterface::operator= (
            IPluginCreatorInterface const &  ) &  [protected], [default]
```

The documentation for this class was generated from the following file:

- NvInferPluginBase.h

9.146 IPluginCreatorV3One Class Reference

A plugin creator class capable of producing IPluginV3 objects.

```
#include <NvInferRuntime.h>
```

9.146.1 Detailed Description

A plugin creator class capable of producing IPluginV3 objects.

See also

IPluginV3

IPluginRegistry

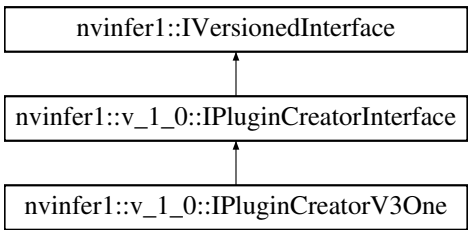
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.147 nvinfer1::v_1_0::IPluginCreatorV3One Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IPluginCreatorV3One:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual IPluginV3 * createPlugin (AsciiChar const *name, PluginFieldCollection const *fc, TensorRTPhase phase) noexcept=0

  Return a plugin object. Return nullptr in case of error.
- virtual PluginFieldCollection const * getFieldNames () noexcept=0

  Return a list of fields that need to be passed to createPlugin() when creating a plugin for use in the TensorRT build phase.
- virtual AsciiChar const * getPluginName () const noexcept=0

  Return the plugin name.
- virtual AsciiChar const * getPluginVersion () const noexcept=0

  Return the plugin version.
- virtual AsciiChar const * getPluginNamespace () const noexcept=0

  Return the plugin namespace.
- IPluginCreatorV3One ()=default
- virtual ~IPluginCreatorV3One ()=default

Protected Member Functions

- IPluginCreatorV3One (IPluginCreatorV3One const &)=default
- IPluginCreatorV3One (IPluginCreatorV3One &&)=default
- IPluginCreatorV3One & operator= (IPluginCreatorV3One const &) &=default
- IPluginCreatorV3One & operator= (IPluginCreatorV3One &&) &=default

9.147.1 Constructor & Destructor Documentation

9.147.1.1 IPluginCreatorV3One() [1/3]

```
nvinfer1::v_1_0::IPluginCreatorV3One::IPluginCreatorV3One ( )  [default]
```

9.147.1.2 ~IPluginCreatorV3One()

```
virtual nvinfer1::v_1_0::IPluginCreatorV3One::~IPluginCreatorV3One ( )  [virtual], [default]
```

9.147.1.3 IPluginCreatorV3One() [2/3]

```
nvinfer1::v_1_0::IPluginCreatorV3One::IPluginCreatorV3One (
            IPluginCreatorV3One const &  )  [protected], [default]
```

9.147.1.4 IPluginCreatorV3One() [3/3]

```
nvinfer1::v_1_0::IPluginCreatorV3One::IPluginCreatorV3One (
            IPluginCreatorV3One &&  )  [protected], [default]
```

9.147.2 Member Function Documentation

9.147.2.1 createPlugin()

```
virtual IPluginV3 * nvinfer1::v_1_0::IPluginCreatorV3One::createPlugin (
            AsciiChar const * name,
            PluginFieldCollection const * fc,
            TensorRTPhase phase )  [pure virtual], [noexcept]
```

Return a plugin object. Return nullptr in case of error.

Parameters

name	A NULL-terminated name string of length 1024 or less, including the NULL terminator.
fc	A pointer to a collection of fields needed for constructing the plugin.
phase	The TensorRT phase in which the plugin is being created

When the phase is TensorRTPhase::kRUNTIME, the PluginFieldCollection provided for serialization by the plugin's runtime interface will be passed as fc.

Note

The returned plugin object must be in an initialized state

If invoked by the user (e.g. with TensorRTPhase::kBUILD, to add to the network defintion with addPlugin↩ V3()), it is the user's responsibility to delete the plugin object. If invoked by TensorRT (e.g. during engine deserialization), TensorRT will delete any objects it creates.

9.147.2.2 getFieldNames()

```
virtual PluginFieldCollection const * nvinfer1::v_1_0::IPluginCreatorV3One::getFieldNames ( )  [pure
virtual], [noexcept]
```

Return a list of fields that need to be passed to createPlugin() when creating a plugin for use in the TensorRT build phase.

See also

PluginFieldCollection

9.147.2.3 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IPluginCreatorV3One::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.147.2.4 getPluginName()

```
virtual AsciiChar const * nvinfer1::v_1_0::IPluginCreatorV3One::getPluginName ( ) const  [pure
virtual], [noexcept]
```

Return the plugin name.

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

9.147.2.5 getPluginNamespace()

```
virtual AsciiChar const * nvinfer1::v_1_0::IPluginCreatorV3One::getPluginNamespace ( ) const  [pure virtual], [noexcept]
```

Return the plugin namespace.

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

9.147.2.6 getPluginVersion()

```
virtual AsciiChar const * nvinfer1::v_1_0::IPluginCreatorV3One::getPluginVersion ( ) const  [pure virtual], [noexcept]
```

Return the plugin version.

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

9.147.2.7 operator=() [1/2]

```
IPluginCreatorV3One & nvinfer1::v_1_0::IPluginCreatorV3One::operator= (
            IPluginCreatorV3One &&  ) &  [protected], [default]
```

9.147.2.8 operator=() [2/2]

```
IPluginCreatorV3One & nvinfer1::v_1_0::IPluginCreatorV3One::operator= (
            IPluginCreatorV3One const &  ) &  [protected], [default]
```

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.148 nvinfer1::IPluginRegistry Class Reference

Single registration point for all plugins in an application. It is used to find plugin implementations during engine deserialization. Internally, the plugin registry is considered to be a singleton so all plugins in an application are part of the same global registry. Note that the plugin registry is only supported for plugins of type IPluginV2 and should also have a corresponding IPluginCreator implementation.

`#include <NvInferRuntimeCommon.h>`

Public Types

- using PluginLibraryHandle = void *

  Pointer for plugin library handle.

Public Member Functions

- virtual void setErrorRecorder (IErrorRecorder *const recorder) noexcept=0

  Set the ErrorRecorder for this interface.
- virtual IErrorRecorder * getErrorRecorder () const noexcept=0

  Get the ErrorRecorder assigned to this interface.
- virtual bool isParentSearchEnabled () const =0

  Return whether the parent registry will be searched if a plugin is not found in this registry default: true.
- virtual void setParentSearchEnabled (bool const enabled)=0

  Set whether the parent registry will be searched if a plugin is not found in this registry.
- virtual PluginLibraryHandle loadLibrary (AsciiChar const *pluginPath) noexcept=0

  Load and register a shared library of plugins.
- virtual void deregisterLibrary (PluginLibraryHandle handle) noexcept=0

  Deregister plugins associated with a library. Any resources acquired when the library was loaded will be released.
- virtual bool registerCreator (IPluginCreatorInterface &creator, AsciiChar const *const pluginNamespace) noexcept=0

  Register a plugin creator. Returns false if a plugin creator with the same type is already registered.
- virtual IPluginCreatorInterface *const * getAllCreators (int32_t *const numCreators) const noexcept=0

  Return all registered plugin creators. Returns nullptr if none found.
- virtual IPluginCreatorInterface * getCreator (AsciiChar const *const pluginName, AsciiChar const *const pluginVersion, AsciiChar const *const pluginNamespace="") noexcept=0

  Return a registered plugin creator based on plugin name, version, and namespace associated with the plugin during network creation.
- virtual bool deregisterCreator (IPluginCreatorInterface const &creator) noexcept=0

  Deregister a previously registered plugin creator.
- virtual IPluginResource * acquirePluginResource (AsciiChar const *key, IPluginResource *resource) noexcept=0

  Get a plugin resource.
- virtual int32_t releasePluginResource (AsciiChar const *key) noexcept=0

  Decrement reference count for the resource with this key If reference count goes to zero after decrement, release() will be invoked on the resource, the key will be deregistered and the resource object will be deleted.
- virtual IPluginCreatorInterface *const * getAllCreatorsRecursive (int32_t *const numCreators) noexcept=0

  Return all registered plugin creators by searching starting from the current registry and following parent registries recursively as long as isParentSearchEnabled() returns true.

Protected Member Functions

- virtual ~IPluginRegistry () noexcept=0

9.148.1 Detailed Description

Single registration point for all plugins in an application. It is used to find plugin implementations during engine deserialization. Internally, the plugin registry is considered to be a singleton so all plugins in an application are part of the same global registry. Note that the plugin registry is only supported for plugins of type IPluginV2 and should also have a corresponding IPluginCreator implementation.

See also

IPluginV2 and IPluginCreator

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

In the automotive safety context, be sure to call IPluginRegistry::setErrorRecorder() to register an error recorder with the registry before using other methods in the registry.

9.148.2 Member Typedef Documentation

9.148.2.1 PluginLibraryHandle

```
using nvinfer1::IPluginRegistry::PluginLibraryHandle = void*
```

Pointer for plugin library handle.

9.148.3 Constructor & Destructor Documentation

9.148.3.1 ~IPluginRegistry()

```
nvinfer1::IPluginRegistry::~IPluginRegistry ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.148.4 Member Function Documentation

9.148.4.1 acquirePluginResource()

```
virtual IPluginResource * nvinfer1::IPluginRegistry::acquirePluginResource (
            AsciiChar const * key,
            IPluginResource * resource )  [pure virtual], [noexcept]
```

Get a plugin resource.

Parameters

key	Key for identifying the resource. Cannot be null.
resource	A plugin resource object. The object will only need to be valid until this method returns, as only a clone of this object will be registered by TRT. Cannot be null.

Returns

Registered plugin resource object

- Allowed context for the API call
  - Thread-safe: Yes; calls to this method will be synchronized by a mutex.

9.148.4.2 deregisterCreator()

```
virtual bool nvinfer1::IPluginRegistry::deregisterCreator (
            IPluginCreatorInterface const & creator )  [pure virtual], [noexcept]
```

Deregister a previously registered plugin creator.

Since there may be a desire to limit the number of plugins, this function provides a mechanism for removing plugin creators registered in TensorRT. The plugin creator that is specified by `creator` is removed from TensorRT and no longer tracked.

Returns

True if the plugin creator was deregistered, false if it was not found in the registry or otherwise could not be deregistered.

- Allowed context for the API call
  - Thread-safe: Yes

9.148.4.3 deregisterLibrary()

```
virtual void nvinfer1::IPluginRegistry::deregisterLibrary (
            PluginLibraryHandle handle )  [pure virtual], [noexcept]
```

Deregister plugins associated with a library. Any resources acquired when the library was loaded will be released.

Parameters

handle	the plugin library handle to deregister.

9.148.4.4 getAllCreators()

```
virtual IPluginCreatorInterface *const * nvinfer1::IPluginRegistry::getAllCreators (
            int32_t *const numCreators ) const  [pure virtual], [noexcept]
```

Return all registered plugin creators. Returns nullptr if none found.

Warning

If any plugin creators are registered or deregistered after calling this function, the returned pointer is not guaranteed to be valid thereafter.

- Allowed context for the API call
  - Thread-safe: No

9.148.4.5 getAllCreatorsRecursive()

```
virtual IPluginCreatorInterface *const * nvinfer1::IPluginRegistry::getAllCreatorsRecursive (
            int32_t *const numCreators )  [pure virtual], [noexcept]
```

Return all registered plugin creators by searching starting from the current registry and following parent registries recursively as long as isParentSearchEnabled() returns true.

Parameters

out	*numCreators*	Pointer to an integer where the number of registered plugin creators will be stored.

Returns

A pointer to an array of IPluginCreatorInterface pointers. Returns nullptr if no creators are found.

Warning

If any plugin creators are registered or deregistered after calling this function, the returned pointer is not guaranteed to remain valid.

- Allowed context for the API call

- Thread-safe: No

9.148.4.6 getCreator()

```
virtual IPluginCreatorInterface * nvinfer1::IPluginRegistry::getCreator (
            AsciiChar const *const pluginName,
            AsciiChar const *const pluginVersion,
            AsciiChar const *const pluginNamespace = "" )  [pure virtual], [noexcept]
```

Return a registered plugin creator based on plugin name, version, and namespace associated with the plugin during network creation.

> **Warning**
>
> The strings pluginName, pluginVersion, and pluginNamespace must be 1024 bytes or less including the NULL terminator and must be NULL terminated.

- Allowed context for the API call
  - Thread-safe: Yes

9.148.4.7 getErrorRecorder()

```
virtual IErrorRecorder * nvinfer1::IPluginRegistry::getErrorRecorder ( ) const  [pure virtual],
[noexcept]
```

Get the ErrorRecorder assigned to this interface.

Retrieves the assigned error recorder object for the given class. A default error recorder does not exist, so a nullptr will be returned if setErrorRecorder has not been called, or an ErrorRecorder has not been inherited.

Returns

A pointer to the IErrorRecorder object that has been registered.

See also

setErrorRecorder()

- Allowed context for the API call
  - Thread-safe: Yes

9.148.4.8 isParentSearchEnabled()

```
virtual bool nvinfer1::IPluginRegistry::isParentSearchEnabled ( ) const  [pure virtual]
```

Return whether the parent registry will be searched if a plugin is not found in this registry default: true.

Returns

bool variable indicating whether parent search is enabled.

See also

setParentSearchEnabled

9.148.4.9 loadLibrary()

```
virtual PluginLibraryHandle nvinfer1::IPluginRegistry::loadLibrary (
            AsciiChar const * pluginPath )  [pure virtual], [noexcept]
```

Load and register a shared library of plugins.

Parameters

pluginPath	the plugin library path.

Returns

The loaded plugin library handle. The call will fail and return nullptr if any of the plugins are already registered.

9.148.4.10 registerCreator()

```
virtual bool nvinfer1::IPluginRegistry::registerCreator (
            IPluginCreatorInterface & creator,
            AsciiChar const *const pluginNamespace )  [pure virtual], [noexcept]
```

Register a plugin creator. Returns false if a plugin creator with the same type is already registered.

Warning

The string pluginNamespace must be 1024 bytes or less including the NULL terminator and must be NULL terminated.

- Allowed context for the API call
  - Thread-safe: Yes; calls to this method will be synchronized by a mutex.

9.148.4.11 releasePluginResource()

```
virtual int32_t nvinfer1::IPluginRegistry::releasePluginResource (
            AsciiChar const * key )  [pure virtual], [noexcept]
```

Decrement reference count for the resource with this key If reference count goes to zero after decrement, release() will be invoked on the resource, the key will be deregistered and the resource object will be deleted.

Parameters

key	Key that was used to register the resource. Cannot be null.

Returns

0 for success, else non-zero

- Allowed context for the API call
  - Thread-safe: Yes; calls to this method will be synchronized by a mutex.

9.148.4.12 setErrorRecorder()

```
virtual void nvinfer1::IPluginRegistry::setErrorRecorder (
            IErrorRecorder *const recorder )  [pure virtual], [noexcept]
```

Set the ErrorRecorder for this interface.

Assigns the ErrorRecorder to this interface. The ErrorRecorder will track all errors during execution. This function will call incRefCount of the registered ErrorRecorder at least once. Setting recorder to nullptr unregisters the recorder with the interface, resulting in a call to decRefCount if a recorder has been registered.

Parameters

recorder	The error recorder to register with this interface.

See also

getErrorRecorder()

- Allowed context for the API call
  - Thread-safe: No

9.148.4.13 setParentSearchEnabled()

```
virtual void nvinfer1::IPluginRegistry::setParentSearchEnabled (
            bool const enabled )  [pure virtual]
```

Set whether the parent registry will be searched if a plugin is not found in this registry.

Parameters

enabled	The bool variable indicating whether parent search is enabled.

See also

isParentSearchEnabled

The documentation for this class was generated from the following file:

- NvInferRuntimeCommon.h

9.149 IPluginResource Class Reference

Interface for plugins to define custom resources that could be shared through the plugin registry.

```
#include <NvInferPluginBase.h>
```

9.149.1 Detailed Description

Interface for plugins to define custom resources that could be shared through the plugin registry.

See also

IPluginRegistry::acquirePluginResource

IPluginRegistry::releasePluginResource

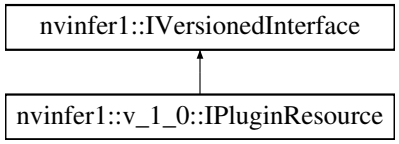
The documentation for this class was generated from the following file:

- NvInferPluginBase.h

9.150 nvinfer1::v_1_0::IPluginResource Class Reference

`#include <NvInferPluginBase.h>`

Inheritance diagram for nvinfer1::v_1_0::IPluginResource:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual int32_t release () noexcept=0

  Free the underlying resource.
- virtual IPluginResource * clone () noexcept=0

  Clone the resource object.
- ~IPluginResource () noexcept override=default
- IPluginResource ()=default
- IPluginResource (IPluginResource const &)=default
- IPluginResource (IPluginResource &&)=default
- IPluginResource & operator= (IPluginResource const &) &=default
- IPluginResource & operator= (IPluginResource &&) &=default

Additional Inherited Members

9.150.1 Constructor & Destructor Documentation

9.150.1.1 ~IPluginResource()

`nvinfer1::v_1_0::IPluginResource::~IPluginResource ( )  [override], [default], [noexcept]`

9.150.1.2 IPluginResource() [1/3]

`nvinfer1::v_1_0::IPluginResource::IPluginResource ( )  [default]`

9.150.1.3 IPluginResource() [2/3]

```
nvinfer1::v_1_0::IPluginResource::IPluginResource (
            IPluginResource const &  )  [default]
```

9.150.1.4 IPluginResource() [3/3]

```
nvinfer1::v_1_0::IPluginResource::IPluginResource (
            IPluginResource &&  )  [default]
```

9.150.2 Member Function Documentation

9.150.2.1 clone()

```
virtual IPluginResource * nvinfer1::v_1_0::IPluginResource::clone ( )  [pure virtual], [noexcept]
```

Clone the resource object.

Note

Resource initialization (if any) may be skipped for non-cloned objects since only clones will be registered by TensorRT

Returns

Pointer to cloned object. nullptr if there was an issue.

- Allowed context for the API call
  - Thread-safe: Yes; this method is required to be thread-safe and may be called from multiple threads.

9.150.2.2 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IPluginResource::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.150.2.3 operator=() [1/2]

```
IPluginResource & nvinfer1::v_1_0::IPluginResource::operator= (
            IPluginResource &&  ) &  [default]
```

9.150.2.4 operator=() [2/2]

```
IPluginResource & nvinfer1::v_1_0::IPluginResource::operator= (
            IPluginResource const &  ) &  [default]
```

9.150.2.5 release()

```
virtual int32_t nvinfer1::v_1_0::IPluginResource::release ( )  [pure virtual], [noexcept]
```

Free the underlying resource.

This will only be called for IPluginResource objects that were produced from IPluginResource::clone()

The IPluginResource object on which release() is called must still be in a clone-able state after release() returns

Returns

0 for success, else non-zero

- Allowed context for the API call
  - Thread-safe: No; this method is not required to be thread-safe

The documentation for this class was generated from the following file:

- NvInferPluginBase.h

9.151 nvinfer1::IPluginResourceContext Class Reference

Interface for plugins to access per context resources provided by TensorRT.

```
#include <NvInferRuntime.h>
```

Public Member Functions

- virtual IGpuAllocator * getGpuAllocator () const noexcept=0

  Get the GPU allocator associated with the resource context.
- virtual IErrorRecorder * getErrorRecorder () const noexcept=0

  Get the error recorder associated with the resource context.
- virtual ~IPluginResourceContext () noexcept=0

Protected Member Functions

- IPluginResourceContext ()=default
- IPluginResourceContext (IPluginResourceContext const &)=default
- IPluginResourceContext (IPluginResourceContext &&)=default
- IPluginResourceContext & operator= (IPluginResourceContext const &) &=default
- IPluginResourceContext & operator= (IPluginResourceContext &&) &=default

9.151.1 Detailed Description

Interface for plugins to access per context resources provided by TensorRT.

There is no public way to construct an IPluginResourceContext. It appears as an argument to IPluginV3OneRuntime::attachToContext(). Overrides of that method can use the IPluginResourceContext object to access any available per context resources.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

See also

IPluginV3OneRuntime::attachToContext()

9.151.2 Constructor & Destructor Documentation

9.151.2.1 ~IPluginResourceContext()

```
nvinfer1::IPluginResourceContext::~IPluginResourceContext ( )  [inline], [pure virtual], [default],
[noexcept]
```

9.151.2.2 IPluginResourceContext() [1/3]

```
nvinfer1::IPluginResourceContext::IPluginResourceContext ( )  [protected], [default]
```

9.151.2.3 IPluginResourceContext() [2/3]

```
nvinfer1::IPluginResourceContext::IPluginResourceContext (
            IPluginResourceContext const &  )  [protected], [default]
```

9.151.2.4 IPluginResourceContext() [3/3]

```
nvinfer1::IPluginResourceContext::IPluginResourceContext (
            IPluginResourceContext &&  )  [protected], [default]
```

9.151.3 Member Function Documentation

9.151.3.1 getErrorRecorder()

```
virtual IErrorRecorder * nvinfer1::IPluginResourceContext::getErrorRecorder ( ) const  [pure virtual],
[noexcept]
```

Get the error recorder associated with the resource context.

See also

IPluginV3OneRuntime::attachToContext()

9.151.3.2 getGpuAllocator()

```
virtual IGpuAllocator * nvinfer1::IPluginResourceContext::getGpuAllocator ( ) const  [pure virtual],
[noexcept]
```

Get the GPU allocator associated with the resource context.

See also

IPluginV3OneRuntime::attachToContext()

9.151.3.3 operator=() [1/2]

```
IPluginResourceContext & nvinfer1::IPluginResourceContext::operator= (
            IPluginResourceContext &&  ) &  [protected], [default]
```

9.151.3.4 operator=() [2/2]

```
IPluginResourceContext & nvinfer1::IPluginResourceContext::operator= (
            IPluginResourceContext const &  ) &  [protected], [default]
```

The documentation for this class was generated from the following file:

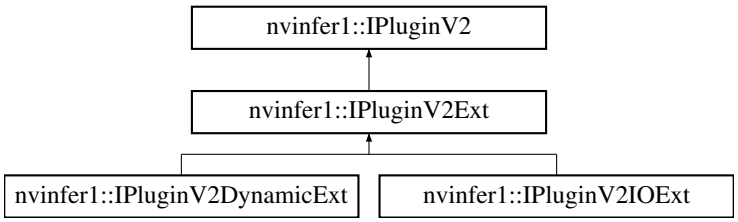
- NvInferRuntime.h

9.152 nvinfer1::IPluginV2 Class Reference

Plugin class for user-implemented layers.

`#include <NvInferRuntimePlugin.h>`

Inheritance diagram for nvinfer1::IPluginV2:



Public Member Functions

- virtual int32_t getTensorRTVersion () const noexcept

  Return the API version with which this plugin was built.
- virtual AsciiChar const * getPluginType () const noexcept=0

  Return the plugin type. Should match the plugin name returned by the corresponding plugin creator.
- virtual AsciiChar const * getPluginVersion () const noexcept=0

  Return the plugin version. Should match the plugin version returned by the corresponding plugin creator.
- virtual int32_t getNbOutputs () const noexcept=0

  Get the number of outputs from the layer.
- virtual Dims getOutputDimensions (int32_t index, Dims const *inputs, int32_t nbInputDims) noexcept=0

  Get the dimension of an output tensor.
- virtual bool supportsFormat (DataType type, PluginFormat format) const noexcept=0

Check format support.

- virtual void configureWithFormat (Dims const *inputDims, int32_t nbInputs, Dims const *outputDims, int32_t nbOutputs, DataType type, PluginFormat format, int32_t maxBatchSize) noexcept=0

  Configure the layer.

- virtual int32_t initialize () noexcept=0

  Initialize the layer for execution. This is called when the engine is created.

- virtual void terminate () noexcept=0

  Release resources acquired during plugin layer initialization. This is called when the engine is destroyed.

- virtual size_t getWorkspaceSize (int32_t maxBatchSize) const noexcept=0

  Find the workspace size required by the layer.

- virtual int32_t enqueue (int32_t batchSize, void const *const *inputs, void *const *outputs, void *workspace, cudaStream_t stream) noexcept=0

  Execute the layer.

- virtual size_t getSerializationSize () const noexcept=0

  Find the size of the serialization buffer required to store the plugin configuration in a binary file.

- virtual void serialize (void *buffer) const noexcept=0

  Serialize the layer.

- virtual void destroy () noexcept=0

  Destroy the plugin object. This will be called when the network, builder or engine is destroyed.

- virtual IPluginV2 * clone () const noexcept=0

  Clone the plugin object. This copies over internal plugin parameters and returns a new plugin object with these parameters.

- virtual void setPluginNamespace (AsciiChar const *pluginNamespace) noexcept=0

  Set the namespace that this plugin object belongs to. Ideally, all plugin objects from the same plugin library must have the same namespace.

- virtual AsciiChar const * getPluginNamespace () const noexcept=0

  Return the namespace of the plugin object.

9.152.1 Detailed Description

Plugin class for user-implemented layers.

Plugins are a mechanism for applications to implement custom layers. When combined with IPluginCreator it provides a mechanism to register plugins and look up the Plugin Registry during de-serialization.

See also

IPluginCreator

IPluginRegistry

Deprecated Deprecated in TensorRT 8.5. Implement IPluginV3 instead.

9.152.2 Member Function Documentation

9.152.2.1 clone()

```
virtual IPluginV2 * nvinfer1::IPluginV2::clone ( ) const  [pure virtual], [noexcept]
```

Clone the plugin object. This copies over internal plugin parameters and returns a new plugin object with these parameters.

The TensorRT runtime calls clone() to clone the plugin when an execution context is created for an engine, after the engine has been created. The runtime does not call initialize() on the cloned plugin, so the cloned plugin must be created in an initialized state.

Returns

A cloned plugin object in an initialized state with the same parameters as the current object. nullptr must be returned if the cloning fails, e.g. because of resource exhaustion.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when creating multiple execution contexts.

Implemented in nvinfer1::IPluginV2DynamicExt, and nvinfer1::IPluginV2Ext.

9.152.2.2 configureWithFormat()

```
virtual void nvinfer1::IPluginV2::configureWithFormat (
            Dims const * inputDims,
            int32_t nbInputs,
            Dims const * outputDims,
            int32_t nbOutputs,
            DataType type,
            PluginFormat format,
            int32_t maxBatchSize )  [pure virtual], [noexcept]
```

Configure the layer.

This function is called by the builder prior to initialize(). It provides an opportunity for the layer to make algorithm choices on the basis of its weights, dimensions, and maximum batch size.

Parameters

inputDims	The input tensor dimensions. Will be the start address of a Dims array of length nbInputs.
nbInputs	The number of inputs. Will be a non-negative integer.
outputDims	The output tensor dimensions. Will be the start address of a Dims array of length nbOutputs.
nbOutputs	The number of outputs. Will be a positive integer identical to the return value of getNbOutputs().
type	The data type selected for the engine.
format	The format selected for the engine.
maxBatchSize	The maximum batch size. Will be a positive integer.

The dimensions passed here do not include the outermost batch size (i.e. for 2D image networks, they will be 3-dimensional CHW dimensions).

Warning

for the format field, the values PluginFormat::kCHW4, PluginFormat::kCHW16, and PluginFormat::kCHW32 will not be passed in, this is to keep backward compatibility with TensorRT 5.x series. Use PluginV2IOExt or PluginV2DynamicExt for other PluginFormats.

DataType:kBOOL and DataType::kUINT8 are not supported.

See also

clone()

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin. However, TensorRT will not call this method from two threads simultaneously on a given clone of a plugin.

Implemented in nvinfer1::IPluginV2Ext.

9.152.2.3 destroy()

```
virtual void nvinfer1::IPluginV2::destroy ( )  [pure virtual], [noexcept]
```

Destroy the plugin object. This will be called when the network, builder or engine is destroyed.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.152.2.4 enqueue()

```
virtual int32_t nvinfer1::IPluginV2::enqueue (
            int32_t batchSize,
            void const *const * inputs,
            void *const * outputs,
            void * workspace,
            cudaStream_t stream )  [pure virtual], [noexcept]
```

Execute the layer.

Parameters

batchSize	The number of inputs in the batch.
inputs	The memory for the input tensors. Will be an array of device addresses corresponding to input tensors of length nbInputs, where nbInputs is the second parameter passed to configureWithFormat(). The i-th input tensor will have the dimensions inputDims[i], where inputDims is the first parameter that was passed to configureWithFormat().
outputs	The memory for the output tensors. Will be an array of device addresses corresponding to output tensors of length getNbOutputs().
workspace	Workspace for execution. Will be the start address of a device buffer whose length will be at least getWorkspaceSize(batchSize).
stream	The stream in which to execute the kernels. This will be a valid CUDA stream.

Returns

0 for success, else non-zero (which will cause engine termination).

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.152.2.5 getNbOutputs()

```
virtual int32_t nvinfer1::IPluginV2::getNbOutputs ( ) const  [pure virtual], [noexcept]
```

Get the number of outputs from the layer.

Returns

The number of outputs, which is a positive integer.

This function is called by the implementations of INetworkDefinition and IBuilder. In particular, it is called prior to any call to initialize().

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.152.2.6 getOutputDimensions()

```
virtual Dims nvinfer1::IPluginV2::getOutputDimensions (
            int32_t index,
            Dims const * inputs,
            int32_t nbInputDims )  [pure virtual], [noexcept]
```

Get the dimension of an output tensor.

Parameters

index	The index of the output tensor. Will lie in the valid range (between 0 and getNbOutputs()-1 inclusive).
inputs	The input tensor dimensions. Will be the start address of a Dims array of length nbInputDims.
nbInputDims	The number of input tensors. Will be a non-negative integer.

Returns

The output tensor dimensions if the index is in the valid range. An invalid value of Dims{-1, {}} must be returned if the index is not in the valid range.

This function is called by the implementations of INetworkDefinition and IBuilder. In particular, it is called prior to any call to initialize().

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

Note

In any non-IPluginV2DynamicExt plugin, batch size must not be included in the returned dimensions, even if the plugin is expected to be run in a network with explicit batch mode enabled. Please see the TensorRT Developer Guide for more details on how plugin inputs and outputs behave.

9.152.2.7 getPluginNamespace()

```
virtual AsciiChar const * nvinfer1::IPluginV2::getPluginNamespace ( ) const  [pure virtual], [noexcept]
```

Return the namespace of the plugin object.

Returns

The namespace string that was passed to setPluginNamespace(), possibly after truncation to 1024 bytes if a longer string was passed. An empty string must be returned as default value.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.152.2.8 getPluginType()

```
virtual AsciiChar const * nvinfer1::IPluginV2::getPluginType ( ) const  [pure virtual], [noexcept]
```

Return the plugin type. Should match the plugin name returned by the corresponding plugin creator.

See also

IPluginCreator::getPluginName()

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.152.2.9 getPluginVersion()

```
virtual AsciiChar const * nvinfer1::IPluginV2::getPluginVersion ( ) const  [pure virtual], [noexcept]
```

Return the plugin version. Should match the plugin version returned by the corresponding plugin creator.

See also

IPluginCreator::getPluginVersion()

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.152.2.10 getSerializationSize()

```
virtual size_t nvinfer1::IPluginV2::getSerializationSize ( ) const  [pure virtual], [noexcept]
```

Find the size of the serialization buffer required to store the plugin configuration in a binary file.

Returns

The size of the serialization buffer in bytes.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.152.2.11 getTensorRTVersion()

```
virtual int32_t nvinfer1::IPluginV2::getTensorRTVersion ( ) const  [inline], [virtual], [noexcept]
```

Return the API version with which this plugin was built.

Do not override this method as it is used by the TensorRT library to maintain backwards-compatibility with plugins.

Returns

The TensorRT version in the format (major $*$ 100 + minor) $*$ 100 + patch.

- Allowed context for the API call
  - Thread-safe: Yes, the implementation provided here is safe to call from any thread.

Reimplemented in nvinfer1::IPluginV2DynamicExt, nvinfer1::IPluginV2Ext, and nvinfer1::IPluginV2IOExt.

9.152.2.12 getWorkspaceSize()

```
virtual size_t nvinfer1::IPluginV2::getWorkspaceSize (
            int32_t maxBatchSize ) const  [pure virtual], [noexcept]
```

Find the workspace size required by the layer.

This function is called during engine startup, after initialize(). The workspace size returned must be sufficient for any batch size up to the maximum.

Parameters

maxBatchSize	The maximum batch size, which will be a positive integer.

Returns

The workspace size in bytes, i.e. the device memory size that the plugin requires for its internal computations.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin. However, TensorRT will not call this method from two threads simultaneously on a given clone of a plugin.

9.152.2.13 initialize()

```
virtual int32_t nvinfer1::IPluginV2::initialize ( )  [pure virtual], [noexcept]
```

Initialize the layer for execution. This is called when the engine is created.

Returns

0 for success, else non-zero (which will cause engine termination).

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when using multiple execution contexts using this plugin.

9.152.2.14 serialize()

```
virtual void nvinfer1::IPluginV2::serialize (
            void * buffer ) const  [pure virtual], [noexcept]
```

Serialize the layer.

Parameters

buffer	A pointer to a host buffer to serialize data. Size of buffer will be at least as large as the value returned by getSerializationSize.

See also

getSerializationSize()

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.152.2.15 setPluginNamespace()

```
virtual void nvinfer1::IPluginV2::setPluginNamespace (
            AsciiChar const * pluginNamespace )  [pure virtual], [noexcept]
```

Set the namespace that this plugin object belongs to. Ideally, all plugin objects from the same plugin library must have the same namespace.

Parameters

pluginNamespace	The namespace for the plugin object.

Warning

The string pluginNamespace will be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.152.2.16 supportsFormat()

```
virtual bool nvinfer1::IPluginV2::supportsFormat (
            DataType type,
            PluginFormat format ) const  [pure virtual], [noexcept]
```

Check format support.

Parameters

type	DataType requested.
format	PluginFormat requested.

Returns

true if the plugin supports the type-format combination.

This function is called by the implementations of INetworkDefinition, IBuilder, and safe::ICudaEngine/ICudaEngine. In particular, it is called when creating an engine and when deserializing an engine.

Warning

for the format field, the values PluginFormat::kCHW4, PluginFormat::kCHW16, and PluginFormat::k↩ CHW32 will not be passed in, this is to keep backward compatibility with TensorRT 5.x series. Use Plugin↩ V2IOExt or PluginV2DynamicExt for other PluginFormats.

DataType:kBOOL and DataType::kUINT8 are not supported.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.152.2.17 terminate()

```
virtual void nvinfer1::IPluginV2::terminate ( )  [pure virtual], [noexcept]
```

Release resources acquired during plugin layer initialization. This is called when the engine is destroyed.

See also

initialize()

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when using multiple execution contexts using this plugin. However, TensorRT will not call this method from two threads simultaneously on a given clone of a plugin.

The documentation for this class was generated from the following file:

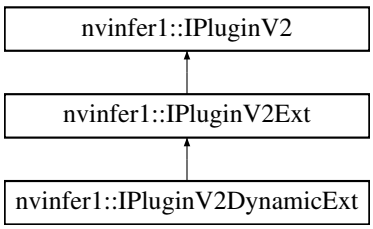
- NvInferRuntimePlugin.h

9.153 nvinfer1::IPluginV2DynamicExt Class Reference

Similar to IPluginV2Ext, but with support for dynamic shapes.

```
#include <NvInferRuntime.h>
```

Inheritance diagram for nvinfer1::IPluginV2DynamicExt:



Public Member Functions

- IPluginV2DynamicExt ∗ clone () const noexcept override=0

  Clone the plugin object. This copies over internal plugin parameters as well and returns a new plugin object with these parameters. If the source plugin is pre-configured with configurePlugin(), the returned object must also be pre-configured. The returned object must allow attachToContext() with a new execution context. Cloned plugin objects can share the same per-engine immutable resource (e.g. weights) with the source object (e.g. via ref-counting) to avoid duplication.
- virtual DimsExprs getOutputDimensions (int32_t outputIndex, DimsExprs const ∗inputs, int32_t nbInputs, IExprBuilder &exprBuilder) noexcept=0

  Get expressions for computing dimensions of an output tensor from dimensions of the input tensors.
- virtual bool supportsFormatCombination (int32_t pos, PluginTensorDesc const ∗inOut, int32_t nbInputs, int32_t nbOutputs) noexcept=0

  Return true if plugin supports the format and datatype for the input/output indexed by pos.
- virtual void configurePlugin (DynamicPluginTensorDesc const ∗in, int32_t nbInputs, DynamicPluginTensorDesc const ∗out, int32_t nbOutputs) noexcept=0

  Configure the plugin.
- virtual size_t getWorkspaceSize (PluginTensorDesc const ∗inputs, int32_t nbInputs, PluginTensorDesc const ∗outputs, int32_t nbOutputs) const noexcept=0

  Find the workspace size required by the layer.
- virtual int32_t enqueue (PluginTensorDesc const ∗inputDesc, PluginTensorDesc const ∗outputDesc, void const ∗const ∗inputs, void ∗const ∗outputs, void ∗workspace, cudaStream_t stream) noexcept=0

  Execute the layer.

Static Public Attributes

- static constexpr int32_t kFORMAT_COMBINATION_LIMIT = 100

  Limit on number of format combinations accepted.

Protected Member Functions

- int32_t getTensorRTVersion () const noexcept override

  Return the API version with which this plugin was built. The upper byte reserved by TensorRT and is used to differentiate this from IPluginV2.
- virtual ∼IPluginV2DynamicExt () noexcept

9.153.1 Detailed Description

Similar to IPluginV2Ext, but with support for dynamic shapes.

Clients should override the public methods, including the following inherited methods:

- virtual int32_t getNbOutputs() const noexcept = 0;
- virtual DataType getOutputDataType(int32_t index, DataType const∗ inputTypes, int32_t nbInputs) const noexcept = 0;
- virtual size_t getSerializationSize() const noexcept = 0;

- virtual void serialize(void* buffer) const noexcept = 0;
- virtual void destroy() noexcept = 0;
- virtual void setPluginNamespace(char const* pluginNamespace) noexcept = 0;
- virtual char const* getPluginNamespace() const noexcept = 0;

For weakly typed networks, the inputTypes will always be DataType::kFLOAT or DataType::kINT32, and the returned type is canonicalized to DataType::kFLOAT if it is DataType::kHALF or DataType:kINT8. For strongly typed networks, inputTypes are inferred from previous operations, and getOutputDataType specifies the returned type based on the inputTypes. Details about the floating-point precision are elicited later by method supportsFormatCombination.

Deprecated Deprecated in TensorRT 10.0. Please implement IPluginV3 instead.

9.153.2 Constructor & Destructor Documentation

9.153.2.1 ~IPluginV2DynamicExt()

```
virtual nvinfer1::IPluginV2DynamicExt::~IPluginV2DynamicExt ( )  [inline], [protected], [virtual],
[noexcept]
```

9.153.3 Member Function Documentation

9.153.3.1 clone()

```
IPluginV2DynamicExt * nvinfer1::IPluginV2DynamicExt::clone ( ) const  [override], [pure virtual],
[noexcept]
```

Clone the plugin object. This copies over internal plugin parameters as well and returns a new plugin object with these parameters. If the source plugin is pre-configured with configurePlugin(), the returned object must also be pre-configured. The returned object must allow attachToContext() with a new execution context. Cloned plugin objects can share the same per-engine immutable resource (e.g. weights) with the source object (e.g. via ref-counting) to avoid duplication.

Returns

A pointer to a cloned plugin object if cloning was successful, otherwise nullptr.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

Implements nvinfer1::IPluginV2Ext.

9.153.3.2 configurePlugin()

```
virtual void nvinfer1::IPluginV2DynamicExt::configurePlugin (
            DynamicPluginTensorDesc const * in,
            int32_t nbInputs,
            DynamicPluginTensorDesc const * out,
            int32_t nbOutputs )  [pure virtual], [noexcept]
```

Configure the plugin.

configurePlugin() can be called multiple times in both the build and execution phases. The build phase happens before initialize() is called and only occurs during creation of an engine by IBuilder. The execution phase happens after initialize() is called and occurs during both creation of an engine by IBuilder and execution of an engine by IExecutionContext.

Build phase: IPluginV2DynamicExt->configurePlugin is called when a plugin is being prepared for profiling but not for any specific input size. This provides an opportunity for the plugin to make algorithmic choices on the basis of input and output formats, along with the bound of possible dimensions. The min and max value of the DynamicPluginTensorDesc correspond to the kMIN and kMAX value of the current profile that the plugin is being profiled for, with the desc.dims field corresponding to the dimensions of plugin specified at network creation. Wildcard dimensions will exist during this phase in the desc.dims field.

Execution phase: IPluginV2DynamicExt->configurePlugin is called when a plugin is being prepared for executing the plugin for a specific dimensions. This provides an opportunity for the plugin to change algorithmic choices based on the explicit input dimensions stored in desc.dims field.

- IBuilder will call this function once per profile, with desc.dims resolved to the values specified by the kOPT field of the current profile. Wildcard dimensions will not exist during this phase.
- IExecutionContext will call this during the next subsequent instance enqueue[V2]() or execute[V2]() if:
  - The optimization profile is changed via setOptimizationProfileAsync().
  - An input execution binding is changed via setInputShape().

> Warning
>
> The execution phase is timing critical during IExecutionContext but is not part of the timing loop when called from IBuilder. Performance bottlenecks of configurePlugin won't show up during engine building but will be visible during execution after calling functions that trigger layer resource updates.

Parameters

in	The input tensors attributes that are used for configuration.
nbInputs	Number of input tensors.
out	The output tensors attributes that are used for configuration.
nbOutputs	Number of output tensors.

9.153.3.3 enqueue()

```
virtual int32_t nvinfer1::IPluginV2DynamicExt::enqueue (
            PluginTensorDesc const * inputDesc,
            PluginTensorDesc const * outputDesc,
            void const *const * inputs,
            void *const * outputs,
            void * workspace,
            cudaStream_t stream )  [pure virtual], [noexcept]
```

Execute the layer.

Parameters

inputDesc	how to interpret the memory for the input tensors.
outputDesc	how to interpret the memory for the output tensors.
inputs	The memory for the input tensors.
outputs	The memory for the output tensors.
workspace	Workspace for execution.
stream	The stream in which to execute the kernels.

Returns

0 for success, else non-zero (which will cause engine termination).

9.153.3.4 getOutputDimensions()

```
virtual DimsExprs nvinfer1::IPluginV2DynamicExt::getOutputDimensions (
            int32_t outputIndex,
            DimsExprs const * inputs,
            int32_t nbInputs,
            IExprBuilder & exprBuilder )  [pure virtual], [noexcept]
```

Get expressions for computing dimensions of an output tensor from dimensions of the input tensors.

Parameters

outputIndex	The index of the output tensor
inputs	Expressions for dimensions of the input tensors
nbInputs	The number of input tensors
exprBuilder	Object for generating new expressions

This function is called by the implementations of IBuilder during analysis of the network.

Example #1: A plugin has a single output that transposes the last two dimensions of the plugin's single input. The body of the override of getOutputDimensions can be:

```
DimsExprs output(inputs[0]);
std::swap(output.d[output.nbDims-1], output.d[output.nbDims-2]);
return output;
```

Example #2: A plugin concatenates its two inputs along the first dimension. The body of the override of getOutput↩ Dimensions can be:

```
DimsExprs output(inputs[0]);
output.d[0] = exprBuilder.operation(DimensionOperation::kSUM, *inputs[0].d[0], *inputs[1].d[0]);
return output;
```

9.153.3.5 getTensorRTVersion()

```
int32_t nvinfer1::IPluginV2DynamicExt::getTensorRTVersion ( ) const  [inline], [override], [protected],
[virtual], [noexcept]
```

Return the API version with which this plugin was built. The upper byte reserved by TensorRT and is used to differentiate this from IPluginV2.

Do not override this method as it is used by the TensorRT library to maintain backwards-compatibility with plugins.

Reimplemented from nvinfer1::IPluginV2.

9.153.3.6 getWorkspaceSize()

```
virtual size_t nvinfer1::IPluginV2DynamicExt::getWorkspaceSize (
            PluginTensorDesc const * inputs,
            int32_t nbInputs,
            PluginTensorDesc const * outputs,
            int32_t nbOutputs ) const  [pure virtual], [noexcept]
```

Find the workspace size required by the layer.

This function is called after the plugin is configured, and possibly during execution. The result should be a sufficient workspace size to deal with inputs and outputs of the given size or any smaller problem.

Returns

The workspace size.

9.153.3.7 supportsFormatCombination()

```
virtual bool nvinfer1::IPluginV2DynamicExt::supportsFormatCombination (
            int32_t pos,
            PluginTensorDesc const * inOut,
            int32_t nbInputs,
            int32_t nbOutputs )  [pure virtual], [noexcept]
```

Return true if plugin supports the format and datatype for the input/output indexed by pos.

For this method inputs are numbered 0..(nbInputs-1) and outputs are numbered nbInputs..(nbInputs+nbOutputs-1). Using this numbering, pos is an index into InOut, where 0 <= pos < nbInputs+nbOutputs.

TensorRT invokes this method to ask if the input/output indexed by pos supports the format/datatype specified by inOut[pos].format and inOut[pos].type. The override should return true if that format/datatype at inOut[pos] are supported by the plugin. If support is conditional on other input/output formats/datatypes, the plugin can make its result conditional on the formats/datatypes in inOut[0..pos-1], which will be set to values that the plugin supports. The override should not inspect inOut[pos+1..nbInputs+nbOutputs-1], which will have invalid values. In other words, the decision for pos must be based on inOut[0..pos] only.

Some examples:

- A definition for a plugin that supports only FP16 NCHW:

  ```
  return inOut[pos].format == TensorFormat::kLINEAR && inOut[pos].type == DataType::kHALF;
  ```

- A definition for a plugin that supports only FP16 NCHW for its two inputs, and FP32 NCHW for its single output:

  ```
  return inOut[pos].format == TensorFormat::kLINEAR && (inOut[pos].type == (pos < 2 ? DataType::kHALF :
  DataType::kFLOAT));
  ```

- A definition for a "polymorphic" plugin with two inputs and one output that supports any format or type, but the inputs and output must have the same format and type:

  ```
  return pos == 0 || (inOut[pos].format == inOut.format[0] && inOut[pos].type == inOut[0].type);
  ```

  Warning: TensorRT will stop asking for formats once it finds kFORMAT_COMBINATION_LIMIT on combinations.

9.153.4 Member Data Documentation

9.153.4.1 kFORMAT_COMBINATION_LIMIT

```
constexpr int32_t nvinfer1::IPluginV2DynamicExt::kFORMAT_COMBINATION_LIMIT = 100  [static], [constexpr]
```

Limit on number of format combinations accepted.

The documentation for this class was generated from the following file:

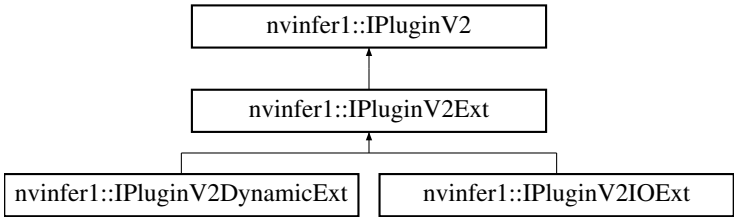
- NvInferRuntime.h

9.154 nvinfer1::IPluginV2Ext Class Reference

Plugin class for user-implemented layers.

`#include <NvInferRuntimePlugin.h>`

Inheritance diagram for nvinfer1::IPluginV2Ext:



Public Member Functions

- virtual nvinfer1::DataType getOutputDataType (int32_t index, nvinfer1::DataType const *inputTypes, int32_t nbInputs) const noexcept=0

  Return the DataType of the plugin output at the requested index.
- virtual void configurePlugin (Dims const *inputDims, int32_t nbInputs, Dims const *outputDims, int32_t nbOutputs, DataType const *inputTypes, DataType const *outputTypes, bool const *inputIsBroadcast, bool const *outputIsBroadcast, PluginFormat floatFormat, int32_t maxBatchSize) noexcept=0

  Configure the layer with input and output data types.
- IPluginV2Ext ()=default
- ~IPluginV2Ext () override=default
- virtual void attachToContext (cudnnContext *, cublasContext *, IGpuAllocator *) noexcept

  Attach the plugin object to an execution context and grant the plugin the access to some context resources.
- virtual void detachFromContext () noexcept

  Detach the plugin object from its execution context.
- IPluginV2Ext * clone () const noexcept override=0

  Clone the plugin object. This copies over internal plugin parameters as well and returns a new plugin object with these parameters. If the source plugin is pre-configured with configurePlugin(), the returned object must also be pre-configured. The returned object must allow attachToContext() with a new execution context. Cloned plugin objects can share the same per-engine immutable resource (e.g. weights) with the source object (e.g. via ref-counting) to avoid duplication.

Protected Member Functions

- int32_t getTensorRTVersion () const noexcept override

  Return the API version with which this plugin was built. The upper byte reserved by TensorRT and is used to differentiate this from IPluginV2.
- void configureWithFormat (Dims const *, int32_t, Dims const *, int32_t, DataType, PluginFormat, int32_t) noexcept override

  Derived classes must not implement this. In a C++11 API it would be final.

9.154.1 Detailed Description

Plugin class for user-implemented layers.

Plugins are a mechanism for applications to implement custom layers. This interface provides additional capabilities to the IPluginV2 interface by supporting different output data types and broadcast across batches.

See also

IPluginV2

Deprecated Deprecated in TensorRT 8.5. Implement IPluginV3 instead.

9.154.2 Constructor & Destructor Documentation

9.154.2.1 IPluginV2Ext()

```
nvinfer1::IPluginV2Ext::IPluginV2Ext ( )  [default]
```

9.154.2.2 ~IPluginV2Ext()

```
nvinfer1::IPluginV2Ext::~IPluginV2Ext ( )  [override], [default]
```

9.154.3 Member Function Documentation

9.154.3.1 attachToContext()

```
virtual void nvinfer1::IPluginV2Ext::attachToContext (
            cudnnContext * ,
            cublasContext * ,
            IGpuAllocator *  )  [inline], [virtual], [noexcept]
```

Attach the plugin object to an execution context and grant the plugin the access to some context resources.

Parameters

cudnn	The cuDNN context handle of the execution context. Always nullptr.
cublas	The cuBLAS context handle of the execution context. Always nullptr.
allocator	The allocator used by the execution context

This function is called automatically for each plugin when a new execution context is created. If the context was created without resources, this method is not called until the resources are assigned. It is also called if new resources are assigned to the context.

If the plugin needs per-context resource, it can be allocated here.

Note

The cuDNN and cuBLAS handles are always nullptr. Plugins that need cuDNN or cuBLAS should create their own handles. The allocator pointer is unique to each building or execution context instance having overlapping lifetimes. It can be used as a key to manage resources across plugin instances sharing the same context. Plugins attached to different contexts will have different handles as their execution will not overlap.

See also

TacticSources

getPluginCudnnHandle(void* executionContextIdentifier)

getPluginCublasHandle(void* excecutionContextIdentifier)

Note

In the automotive safety context, the cuDNN and cuBLAS parameters will be nullptr because cuDNN and cuBLAS are not used by the safe runtime.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.154.3.2 clone()

```
IPluginV2Ext * nvinfer1::IPluginV2Ext::clone ( ) const  [override], [pure virtual], [noexcept]
```

Clone the plugin object. This copies over internal plugin parameters as well and returns a new plugin object with these parameters. If the source plugin is pre-configured with configurePlugin(), the returned object must also be pre-configured. The returned object must allow attachToContext() with a new execution context. Cloned plugin objects can share the same per-engine immutable resource (e.g. weights) with the source object (e.g. via ref-counting) to avoid duplication.

Returns

A pointer to a cloned plugin object if cloning was successful, otherwise nullptr.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

Implements nvinfer1::IPluginV2.

Implemented in nvinfer1::IPluginV2DynamicExt.

9.154.3.3 configurePlugin()

```
virtual void nvinfer1::IPluginV2Ext::configurePlugin (
            Dims const * inputDims,
            int32_t nbInputs,
            Dims const * outputDims,
            int32_t nbOutputs,
            DataType const * inputTypes,
            DataType const * outputTypes,
            bool const * inputIsBroadcast,
            bool const * outputIsBroadcast,
            PluginFormat floatFormat,
            int32_t maxBatchSize ) [pure virtual], [noexcept]
```

Configure the layer with input and output data types.

This function is called by the builder prior to initialize(). It provides an opportunity for the layer to make algorithm choices on the basis of its weights, dimensions, data types and maximum batch size.

Parameters

inputDims	The input tensor dimensions. Will be an array of length nbInputs.
nbInputs	The number of inputs. Will be a non-negative integer.
outputDims	The output tensor dimensions. Will be an array of length nbOutputs.
nbOutputs	The number of outputs. Will be a positive integer.
inputTypes	The data types selected for the plugin inputs. Will be an array of length nbInputs.
outputTypes	The data types selected for the plugin outputs. Will be an array of length nbOutputs.
inputIsBroadcast	True for each input that the plugin must broadcast across the batch. Will be an array of length nbInputs.
outputIsBroadcast	True for each output that TensorRT will broadcast across the batch. Will be an array of length nbOutputs.
floatFormat	The format selected for the engine for the floating point inputs/outputs.
maxBatchSize	The maximum batch size. Will be a positive integer.

The dimensions passed here do not include the outermost batch size (i.e. for 2D image networks, they will be 3-dimensional CHW dimensions). When inputIsBroadcast or outputIsBroadcast is true, the outermost batch size for that input or output must be treated as if it is one. Index 'i' of inputIsBroadcast is true only if the input is semantically broadcast across the batch. Index 'i' of outputIsBroadcast is true only if the output is semantically broadcast across the batch.

Warning

for the floatFormat field, the values PluginFormat::kCHW4, PluginFormat::kCHW16, and PluginFormat↩ ::kCHW32 will not be passed in, this is to keep backward compatibility with TensorRT 5.x series. Use PluginV2IOExt or PluginV2DynamicExt for other PluginFormats.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin. However, TensorRT will not call this method from two threads simultaneously on a given clone of a plugin.

9.154.3.4 configureWithFormat()

```
void nvinfer1::IPluginV2Ext::configureWithFormat (
            Dims const * ,
            int32_t ,
            Dims const * ,
            int32_t ,
            DataType ,
            PluginFormat ,
            int32_t  )  [inline], [override], [protected], [virtual], [noexcept]
```

Derived classes must not implement this. In a C++11 API it would be final.

IPluginV2Ext::configureWithFormat() is a NOP operation for all classes derived from IPluginV2Ext. These classes call configurePlugin() instead.

Implements nvinfer1::IPluginV2.

9.154.3.5 detachFromContext()

```
virtual void nvinfer1::IPluginV2Ext::detachFromContext ( )  [inline], [virtual], [noexcept]
```

Detach the plugin object from its execution context.

This function is called automatically for each plugin when an execution context is destroyed or the context resources are unassigned from the context.

If the plugin owns per-context resource, it can be released here.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.154.3.6 getOutputDataType()

```
virtual nvinfer1::DataType nvinfer1::IPluginV2Ext::getOutputDataType (
            int32_t index,
            nvinfer1::DataType const * inputTypes,
            int32_t nbInputs ) const  [pure virtual], [noexcept]
```

Return the DataType of the plugin output at the requested index.

Parameters

index	The output tensor index in the valid range between 0 and getNbOutputs()-1.
inputTypes	The data types of the input tensors, stored in an array of length nbInputs.
nbInputs	The number of input tensors. Will be a non-negative integer.

Returns

The data type of the output tensor with the provided index if the input tensors have the data types provided in inputTypes, provided the output tensor index is in the valid range. DataType::kFLOAT must be returned if the index is not in the valid range.

The default behavior must be to return the type of the first input, or DataType::kFLOAT if the layer has no inputs. The returned data type must have a format that is supported by the plugin.

See also

supportsFormat()

Warning

DataType:kBOOL and DataType::kUINT8 are not supported.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.154.3.7 getTensorRTVersion()

```
int32_t nvinfer1::IPluginV2Ext::getTensorRTVersion ( ) const  [inline], [override], [protected], [virtual], [noexcept]
```

Return the API version with which this plugin was built. The upper byte reserved by TensorRT and is used to differentiate this from IPluginV2.

Returns

In the lower three bytes, the TensorRT version in the format (major $*$ 100 + minor) $*$ 100 + patch. In the upper byte, the value 1.

Do not override this method as it is used by the TensorRT library to maintain backwards-compatibility with plugins.

- Allowed context for the API call
  - Thread-safe: Yes, the implementation provided here is safe to call from any thread.

Reimplemented from nvinfer1::IPluginV2.

Reimplemented in nvinfer1::IPluginV2IOExt.

The documentation for this class was generated from the following file:

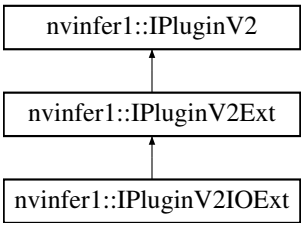
- NvInferRuntimePlugin.h

9.155 nvinfer1::IPluginV2IOExt Class Reference

Plugin class for user-implemented layers.

`#include <NvInferRuntimePlugin.h>`

Inheritance diagram for nvinfer1::IPluginV2IOExt:



Public Member Functions

- virtual void configurePlugin (PluginTensorDesc const ∗in, int32_t nbInput, PluginTensorDesc const ∗out, int32↩ _t nbOutput) noexcept=0

  Configure the layer.
- virtual bool supportsFormatCombination (int32_t pos, PluginTensorDesc const ∗inOut, int32_t nbInputs, int32_t nbOutputs) const noexcept=0

  Return true if plugin supports the format and datatype for the input/output indexed by pos.

Protected Member Functions

- int32_t getTensorRTVersion () const noexcept override

  Return the API version with which this plugin was built. The upper byte is reserved by TensorRT and is used to differentiate this from IPluginV2 and IPluginV2Ext.

9.155.1 Detailed Description

Plugin class for user-implemented layers.

Plugins are a mechanism for applications to implement custom layers. This interface provides additional capabilities to the IPluginV2Ext interface by extending different I/O data types and tensor formats.

See also

IPluginV2Ext

Deprecated Deprecated in TensorRT 10.0. Implement IPluginV3 instead.

9.155.2 Member Function Documentation

9.155.2.1 configurePlugin()

```
virtual void nvinfer1::IPluginV2IOExt::configurePlugin (
            PluginTensorDesc const * in,
            int32_t nbInput,
            PluginTensorDesc const * out,
            int32_t nbOutput )  [pure virtual], [noexcept]
```

Configure the layer.

This function is called by the builder prior to initialize(). It provides an opportunity for the layer to make algorithm choices on the basis of the provided I/O PluginTensorDesc.

Parameters

in	The input tensors attributes that are used for configuration.
nbInput	Number of input tensors.
out	The output tensors attributes that are used for configuration.
nbOutput	Number of output tensors.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin. However, TensorRT will not call this method from two threads simultaneously on a given clone of a plugin.

9.155.2.2 getTensorRTVersion()

```
int32_t nvinfer1::IPluginV2IOExt::getTensorRTVersion ( ) const  [inline], [override], [protected],
[virtual], [noexcept]
```

Return the API version with which this plugin was built. The upper byte is reserved by TensorRT and is used to differentiate this from IPluginV2 and IPluginV2Ext.

Do not override this method as it is used by the TensorRT library to maintain backwards-compatibility with plugins.

- Allowed context for the API call
  - Thread-safe: Yes, the implementation provided here is safe to call from any thread.

Reimplemented from nvinfer1::IPluginV2Ext.

9.155.2.3 supportsFormatCombination()

```
virtual bool nvinfer1::IPluginV2IOExt::supportsFormatCombination (
            int32_t pos,
            PluginTensorDesc const * inOut,
            int32_t nbInputs,
            int32_t nbOutputs ) const  [pure virtual], [noexcept]
```

Return true if plugin supports the format and datatype for the input/output indexed by pos.

For this method inputs are numbered 0..(nbInputs-1) and outputs are numbered nbInputs..(nbInputs+nbOutputs-1). Using this numbering, pos is an index into InOut, where 0 <= pos < nbInputs+nbOutputs.

TensorRT invokes this method to ask if the input/output indexed by pos supports the format/datatype specified by in↩ Out[pos].format and inOut[pos].type. The override must return true if that format/datatype at inOut[pos] are supported by the plugin. If support is conditional on other input/output formats/datatypes, the plugin can make its result conditional on the formats/datatypes in inOut[0..pos-1], which will be set to values that the plugin supports. The override must not inspect inOut[pos+1..nbInputs+nbOutputs-1], which will have invalid values. In other words, the decision for pos must be based on inOut[0..pos] only.

Some examples:

- A definition for a plugin that supports only FP16 NCHW:

  ```
  return inOut.format[pos] == TensorFormat::kLINEAR && inOut.type[pos] == DataType::kHALF;
  ```

- A definition for a plugin that supports only FP16 NCHW for its two inputs, and FP32 NCHW for its single output:

  ```
  return inOut.format[pos] == TensorFormat::kLINEAR &&
         (inOut.type[pos] == (pos < 2 ?  DataType::kHALF : DataType::kFLOAT));
  ```

- A definition for a "polymorphic" plugin with two inputs and one output that supports any format or type, but the inputs and output must have the same format and type:

  ```
  return pos == 0 || (inOut.format[pos] == inOut.format[0] && inOut.type[pos] == inOut.type[0]);
  ```

  Warning: TensorRT will stop asking for formats once it finds kFORMAT_COMBINATION_LIMIT on combinations.

- Allowed context for the API call

  – Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

The documentation for this class was generated from the following file:

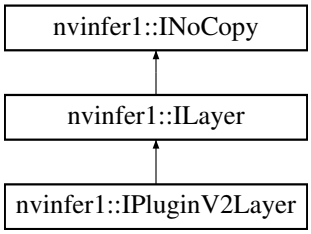
- NvInferRuntimePlugin.h

9.156 nvinfer1::IPluginV2Layer Class Reference

Layer type for pluginV2.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IPluginV2Layer:



Public Member Functions

- IPluginV2 & getPlugin () noexcept

  Get the plugin for the layer.

Protected Member Functions

- virtual ~IPluginV2Layer () noexcept=0

Protected Attributes

- apiv::VPluginV2Layer * mImpl

9.156.1 Detailed Description

Layer type for pluginV2.

See also

IPluginV2

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

Deprecated Deprecated in TensorRT 10.8. Superseded by IPluginV3Layer.

9.156.2 Constructor & Destructor Documentation

9.156.2.1 ~IPluginV2Layer()

```
nvinfer1::IPluginV2Layer::~IPluginV2Layer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.156.3 Member Function Documentation

9.156.3.1 getPlugin()

```
IPluginV2 & nvinfer1::IPluginV2Layer::getPlugin ( )  [inline], [noexcept]
```

Get the plugin for the layer.

See also

IPluginV2

9.156.4 Member Data Documentation

9.156.4.1 mImpl

```
apiv::VPluginV2Layer* nvinfer1::IPluginV2Layer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.157 IPluginV3 Class Reference

Plugin class for the V3 generation of user-implemented layers.

```
#include <NvInferPluginBase.h>
```

9.157.1 Detailed Description

Plugin class for the V3 generation of user-implemented layers.

IPluginV3 acts as a wrapper around the plugin capability interfaces that define the actual behavior of the plugin.

See also

IPluginCapability
IPluginCreatorV3One
IPluginRegistry

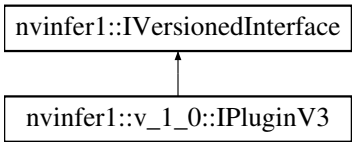
The documentation for this class was generated from the following file:

- NvInferPluginBase.h

9.158 nvinfer1::v_1_0::IPluginV3 Class Reference

`#include <NvInferPluginBase.h>`

Inheritance diagram for nvinfer1::v_1_0::IPluginV3:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual IPluginCapability * getCapabilityInterface (PluginCapabilityType type) noexcept=0

  Return a pointer to plugin object implementing the specified PluginCapabilityType.
- virtual IPluginV3 * clone () noexcept=0

  Clone the plugin object. This copies over internal plugin parameters and returns a new plugin object with these parameters. The cloned object must be in a fully initialized state.

Additional Inherited Members

9.158.1 Member Function Documentation

9.158.1.1 clone()

```
virtual IPluginV3 * nvinfer1::v_1_0::IPluginV3::clone ( )  [pure virtual], [noexcept]
```

Clone the plugin object. This copies over internal plugin parameters and returns a new plugin object with these parameters. The cloned object must be in a fully initialized state.

Note

The cloned object must return valid objects through getCapabilityInterface() for at least the same PluginCapabilityTypes as the original object.

Returns

A cloned plugin object in an initialized state with the same parameters as the current object. nullptr must be returned if the cloning fails.

9.158.1.2 getCapabilityInterface()

```
virtual IPluginCapability * nvinfer1::v_1_0::IPluginV3::getCapabilityInterface (
            PluginCapabilityType type )  [pure virtual], [noexcept]
```

Return a pointer to plugin object implementing the specified PluginCapabilityType.

Note

IPluginV3 objects added for the build phase (through addPluginV3()) must return valid objects for PluginCapabilityType::kCORE, PluginCapabilityType::kBUILD and PluginCapabilityType::kRUNTIME.

IPluginV3 objects added for the runtime phase must return valid objects for PluginCapabilityType::kCORE and PluginCapabilityType::kRUNTIME.

See also

TensorRTPhase

IPluginCreatorV3One::createPlugin()

9.158.1.3 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IPluginV3::getInterfaceInfo ( ) const  [inline], [override], [virtual],
[noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

The documentation for this class was generated from the following file:

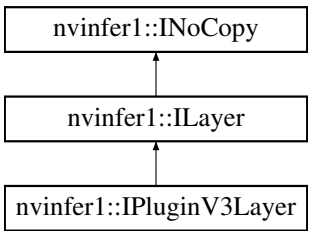
- NvInferPluginBase.h

9.159 nvinfer1::IPluginV3Layer Class Reference

Layer type for V3 plugins.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IPluginV3Layer:



Public Member Functions

- IPluginV3 & getPlugin () noexcept
  Get the plugin for the layer.

Protected Member Functions

- virtual ~IPluginV3Layer () noexcept=0

Protected Attributes

- apiv::VPluginV3Layer * mImpl

9.159.1 Detailed Description

Layer type for V3 plugins.

See also

IPluginV3

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.159.2 Constructor & Destructor Documentation

9.159.2.1 ~IPluginV3Layer()

```
nvinfer1::IPluginV3Layer::~IPluginV3Layer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.159.3 Member Function Documentation

9.159.3.1 getPlugin()

```
IPluginV3 & nvinfer1::IPluginV3Layer::getPlugin ( )  [inline], [noexcept]
```

Get the plugin for the layer.

See also

IPluginV3

9.159.4 Member Data Documentation

9.159.4.1 mImpl

```
apiv::VPluginV3Layer* nvinfer1::IPluginV3Layer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.160 IPluginV3OneBuild Class Reference

A plugin capability interface that enables the build capability (PluginCapabilityType::kBUILD). Exposes methods that allow the expression of the build time properties and behavior of a plugin.

```
#include <NvInferRuntime.h>
```

9.160.1 Detailed Description

A plugin capability interface that enables the build capability (PluginCapabilityType::kBUILD). Exposes methods that allow the expression of the build time properties and behavior of a plugin.

See also

IPluginCapability

PluginCapabilityType

IPluginV3::getCapabilityInterface()

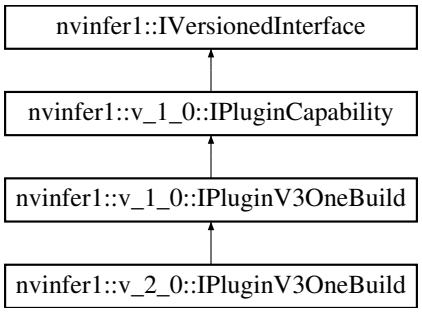
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.161 nvinfer1::v_1_0::IPluginV3OneBuild Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IPluginV3OneBuild:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual int32_t configurePlugin (DynamicPluginTensorDesc const ∗in, int32_t nbInputs, DynamicPluginTensorDesc const ∗out, int32_t nbOutputs) noexcept=0

  Configure the plugin.
- virtual int32_t getOutputDataTypes (DataType ∗outputTypes, int32_t nbOutputs, const DataType ∗inputTypes, int32_t nbInputs) const noexcept=0

  Provide the data types of the plugin outputs if the input tensors have the data types provided.
- virtual int32_t getOutputShapes (DimsExprs const ∗inputs, int32_t nbInputs, DimsExprs const ∗shapeInputs, int32_t nbShapeInputs, DimsExprs ∗outputs, int32_t nbOutputs, IExprBuilder &exprBuilder) noexcept=0

  Provide expressions for computing dimensions of the output tensors from dimensions of the input tensors.

- virtual bool supportsFormatCombination (int32_t pos, DynamicPluginTensorDesc const *inOut, int32_t nb↩ Inputs, int32_t nbOutputs) noexcept=0

  Return true if plugin supports the format and datatype for the input/output indexed by pos.
- virtual int32_t getNbOutputs () const noexcept=0

  Get the number of outputs from the plugin.
- virtual size_t getWorkspaceSize (DynamicPluginTensorDesc const *, int32_t, DynamicPluginTensorDesc const *, int32_t) const noexcept

  Find the workspace size required by the layer.
- virtual int32_t getValidTactics (int32_t *, int32_t) noexcept

  Query for any custom tactics that the plugin intends to use.
- virtual int32_t getNbTactics () noexcept

  Query for the number of custom tactics the plugin intends to use.
- virtual char const * getTimingCacheID () noexcept

  Called to query the suffix to use for the timing cache ID. May be called anytime after plugin creation.
- virtual int32_t getFormatCombinationLimit () noexcept

  Return the maximum number of format combinations that will be timed by TensorRT during the build phase.
- virtual char const * getMetadataString () noexcept

  Query for a string representing the configuration of the plugin. May be called anytime after plugin creation.

Static Public Attributes

- static constexpr int32_t kDEFAULT_FORMAT_COMBINATION_LIMIT = 100

  The default maximum number of format combinations that will be timed by TensorRT during the build phase.

Additional Inherited Members

9.161.1 Member Function Documentation

9.161.1.1 configurePlugin()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneBuild::configurePlugin (
            DynamicPluginTensorDesc const * in,
            int32_t nbInputs,
            DynamicPluginTensorDesc const * out,
            int32_t nbOutputs )  [pure virtual], [noexcept]
```

Configure the plugin.

configurePlugin() can be called multiple times in the build phase during creation of an engine by IBuilder.

configurePlugin() is called when a plugin is being prepared for profiling but not for any specific input size. This provides an opportunity for the plugin to make algorithmic choices on the basis of input and output formats, along with the bound of possible dimensions. The min, opt and max value of the DynamicPluginTensorDesc correspond to the kMIN, kOPT and kMAX value of the current profile that the plugin is being profiled for, with the desc.dims field corresponding to the dimensions of plugin specified at network creation. Wildcard dimensions may exist during this phase in the desc.dims field.

Parameters

in	The input tensors attributes that are used for configuration.
nbInputs	Number of input tensors.
out	The output tensors attributes that are used for configuration.
nbOutputs	Number of output tensors.

Returns

0 for success, else non-zero (which will cause engine termination, if invoked by TensorRT).

9.161.1.2 getFormatCombinationLimit()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneBuild::getFormatCombinationLimit ( )  [inline], [virtual],
[noexcept]
```

Return the maximum number of format combinations that will be timed by TensorRT during the build phase.

9.161.1.3 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IPluginV3OneBuild::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

Reimplemented in nvinfer1::v_2_0::IPluginV3OneBuild.

9.161.1.4 getMetadataString()

```
virtual char const * nvinfer1::v_1_0::IPluginV3OneBuild::getMetadataString ( )  [inline], [virtual],
[noexcept]
```

Query for a string representing the configuration of the plugin. May be called anytime after plugin creation.

Returns

A string representing the plugin's creation state, especially with regard to its attribute values.

9.161.1.5 getNbOutputs()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneBuild::getNbOutputs ( ) const  [pure virtual], [noexcept]
```

Get the number of outputs from the plugin.

Returns

The number of outputs, which must be a positive integer.

9.161.1.6 getNbTactics()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneBuild::getNbTactics ( )  [inline], [virtual], [noexcept]
```

Query for the number of custom tactics the plugin intends to use.

9.161.1.7 getOutputDataTypes()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneBuild::getOutputDataTypes (
            DataType * outputTypes,
            int32_t nbOutputs,
            const DataType * inputTypes,
            int32_t nbInputs ) const  [pure virtual], [noexcept]
```

Provide the data types of the plugin outputs if the input tensors have the data types provided.

Parameters

outputTypes	Pre-allocated array to which the output data types should be written.
nbOutputs	The number of output tensors. This matches the value returned from getNbOutputs().
inputTypes	The input data types.
nbInputs	The number of input tensors.

Returns

0 for success, else non-zero (which will cause engine termination). The returned code will be reported through the error recorder.

Note

Provide DataType::kFLOATs if the layer has no inputs. The data type for any size tensor outputs must be DataType::kINT32. The returned data types must each have a format that is supported by the plugin.

Warning

DataType:kBOOL and DataType::kUINT8 are not supported.

9.161.1.8 getOutputShapes()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneBuild::getOutputShapes (
            DimsExprs const * inputs,
            int32_t nbInputs,
            DimsExprs const * shapeInputs,
            int32_t nbShapeInputs,
            DimsExprs * outputs,
            int32_t nbOutputs,
            IExprBuilder & exprBuilder ) [pure virtual], [noexcept]
```

Provide expressions for computing dimensions of the output tensors from dimensions of the input tensors.

Parameters

inputs	Expressions for dimensions of the input tensors
nbInputs	The number of input tensors
shapeInputs	Expressions for values of the shape tensor inputs
nbShapeInputs	The number of shape tensor inputs
outputs	Pre-allocated array to which the output dimensions must be written
nbOutputs	Number of outputs.
exprBuilder	Object for generating new dimension expressions

Note

Any size tensor outputs must be declared to be 0D.

The declaration of shapeInputs as DimsExprs is slightly abusive, because the ”dimensions” are actually the values of the shape tensor. For example, if the input shape tensor is a 2x3 matrix, the DimsExprs will have six ”dimensions”: the three values from the first row of the matrix followed by the three values from the second row of the matrix.

Returns

0 for success, else non-zero (which will cause engine termination). Returned code will be reported through the error recorder.

9.161.1.9 getTimingCacheID()

```
virtual char const * nvinfer1::v_1_0::IPluginV3OneBuild::getTimingCacheID ( )  [inline], [virtual],
[noexcept]
```

Called to query the suffix to use for the timing cache ID. May be called anytime after plugin creation.

Returns

Suffix to use for timing cache ID, considering only the creation state of the plugin. Returning nullptr will disable timing caching for the plugin altogether.

Note

If timing caching is enabled for the plugin (by returning non-null), the I/O shape and format information will be automatically considered to form the prefix of the timing cache ID. Therefore, only other factors determining the creation state of the plugin, such as its attribute values, should be considered to compose the return value.

9.161.1.10 getValidTactics()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneBuild::getValidTactics (
            int32_t * ,
            int32_t  )  [inline], [virtual], [noexcept]
```

Query for any custom tactics that the plugin intends to use.

This method queries for the set of tactics T(f) supported by the plugin for the format combination f indicated by the immediately preceding call to configurePlugin(). It is guaranteed to be called after configurePlugin().

For each format combination provided through configurePlugin(), up to a maximum of getFormatCombinationLimit(), the plugin will be timed for each tactic advertised through this method for that format combination. i.e. The plugin will be timed $N = \sum_{i=0}^{i<getFormatCombinationLimit()}(T(f[i]))$ times. If $N = 1$, the plugin may not be timed. In pseudocode, the timing protocol appears as the following:

```
counter = 0
for each supported format combination
    ++counter
    if counter > getFormatCombinationLimit()
        goto done
    configurePlugin(...)
    for each tactic in getValidTactics(...)
        time tactic
done:
```

Parameters

tactics	Pre-allocated buffer to which the tactic values should be written
nbTactics	The number of tactics advertised through getNbTactics()

Note

The provided tactic values must be unique and non-zero. The tactic value 0 is reserved for the default tactic attached to each format combination.

Returns

0 for success, else non-zero (which will cause engine termination). The returned code will be reported through the error recorder.

9.161.1.11 getWorkspaceSize()

```
virtual size_t nvinfer1::v_1_0::IPluginV3OneBuild::getWorkspaceSize (
            DynamicPluginTensorDesc const * ,
            int32_t ,
            DynamicPluginTensorDesc const * ,
            int32_t  ) const  [inline], [virtual], [noexcept]
```

Find the workspace size required by the layer.

This function is called after the plugin is configured, and possibly during execution. The result should be a sufficient workspace size to deal with inputs and outputs of the given size or any smaller problem.

Returns

The workspace size.

9.161.1.12 supportsFormatCombination()

```
virtual bool nvinfer1::v_1_0::IPluginV3OneBuild::supportsFormatCombination (
            int32_t pos,
            DynamicPluginTensorDesc const * inOut,
            int32_t nbInputs,
            int32_t nbOutputs )  [pure virtual], [noexcept]
```

Return true if plugin supports the format and datatype for the input/output indexed by pos.

For this method inputs are numbered 0.. (nbInputs - 1) and outputs are numbered nbInputs.. (nbInputs + nbOutputs

- 1). Using this numbering, pos is an index into InOut, where 0 <= pos < nbInputs + nbOutputs - 1.

TensorRT invokes this method to ask if the input/output indexed by pos supports the format/datatype specified by inOut[pos].format and inOut[pos].type. The override should return true if that format/datatype at inOut[pos] are supported by the plugin. If support is conditional on other input/output formats/datatypes, the plugin can make its result conditional on the formats/datatypes in inOut[0.. pos - 1], which will be set to values that the plugin supports. The override should not inspect inOut[pos1.. nbInputs + nbOutputs - 1], which will have invalid values. In other words, the decision for pos must be based on inOut[0..pos] only.

Some examples:

- A definition for a plugin that supports only FP16 NCHW:

```
return inOut.format[pos] == TensorFormat::kLINEAR && inOut.type[pos] == DataType::kHALF;
```

- A definition for a plugin that supports only FP16 NCHW for its two inputs, and FP32 NCHW for its single output:

```
return inOut.format[pos] == TensorFormat::kLINEAR && (inOut.type[pos] == pos < 2 ?  DataType::kHALF :
DataType::kFLOAT);
```

- A definition for a "polymorphic" plugin with two inputs and one output that supports any format or type, but the inputs and output must have the same format and type:

```
return pos == 0 || (inOut.format[pos] == inOut.format[0] && inOut.type[pos] == inOut.type[0]);
```

> **Warning**
>
> TensorRT will stop querying once it finds getFormatCombinationLimit() of combinations.

See also

getFormatCombinationLimit

9.161.2 Member Data Documentation

9.161.2.1 kDEFAULT_FORMAT_COMBINATION_LIMIT

```
constexpr int32_t nvinfer1::v_1_0::IPluginV3OneBuild::kDEFAULT_FORMAT_COMBINATION_LIMIT = 100  [static],
[constexpr]
```

The default maximum number of format combinations that will be timed by TensorRT during the build phase.

See also

getFormatCombinationLimit

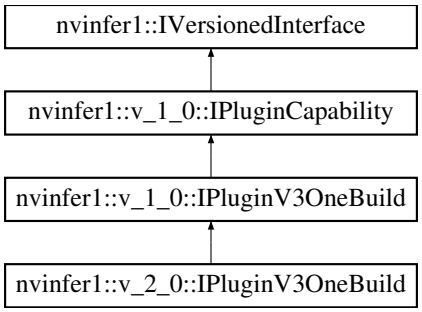
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.162 nvinfer1::v_2_0::IPluginV3OneBuild Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_2_0::IPluginV3OneBuild:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual int32_t getAliasedInput (int32_t) noexcept

  Communicates to TensorRT that the output at the specified output index is aliased to the input at the returned index.

Additional Inherited Members

9.162.1 Member Function Documentation

9.162.1.1 getAliasedInput()

```
virtual int32_t nvinfer1::v_2_0::IPluginV3OneBuild::getAliasedInput (
            int32_t  )  [inline], [virtual], [noexcept]
```

Communicates to TensorRT that the output at the specified output index is aliased to the input at the returned index.

Enables read-modify-write behavior in plugins. TensorRT may insert copies to facilitate this capability.

Returns

An integer denoting the index of the input which is aliased to the output at outputIndex. Returning -1 indicates that the output is not aliased to any input. Otherwise, the valid range for return value is [0, nbInputs - 1].

Note

A given plugin input can only be aliased to a single plugin output.

This API will only be called and have an effect when PreviewFeature::kALIASED_PLUGIN_IO_10_03 is turned on.

Warning

If an input is not shallow copyable, a copy inserted by TensorRT may not work as intended. Therefore, using this feature with tensors requiring deep copies is not supported.

If a given tensor is requested to be aliased by two different plugins, this may result in divergent copies of the tensor after writes from each plugin. e.g. In the below example, t1 and t2 could be divergent.

```
   +-----+            +--------+
+->|Copy +--> t* ---->|Plugin0 +--> t1
|  +-----+            +--------+
t
|  +-----+            +--------+
+->|Copy +--> t** --->|Plugin1 +--> t2
   +-----+            +--------+
```

9.162.1.2 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_2_0::IPluginV3OneBuild::getInterfaceInfo ( ) const  [inline], [override], [virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Reimplemented from nvinfer1::v_1_0::IPluginV3OneBuild.

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.163 IPluginV3OneBuildV2 Class Reference

A plugin capability interface that extends IPluginV3OneBuild by providing I/O aliasing functionality.

```
#include <NvInferRuntime.h>
```

9.163.1 Detailed Description

A plugin capability interface that extends IPluginV3OneBuild by providing I/O aliasing functionality.

See also

IPluginV3OneBuild

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.164 IPluginV3OneCore Class Reference

A plugin capability interface that enables the core capability (PluginCapabilityType::kCORE).

`#include <NvInferRuntime.h>`

9.164.1 Detailed Description

A plugin capability interface that enables the core capability (PluginCapabilityType::kCORE).

See also

IPluginCapability

PluginCapabilityType

IPluginV3::getCapabilityInterface()

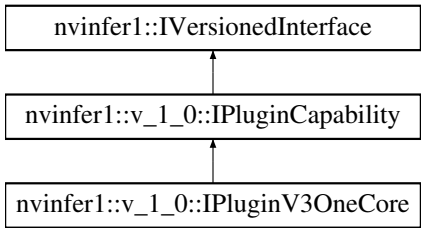
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.165 nvinfer1::v_1_0::IPluginV3OneCore Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IPluginV3OneCore:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual AsciiChar const * getPluginName () const noexcept=0

  Return the plugin name. Should match the plugin name returned by the corresponding plugin creator.
- virtual AsciiChar const * getPluginVersion () const noexcept=0

  Return the plugin version. Should match the plugin version returned by the corresponding plugin creator.
- virtual AsciiChar const * getPluginNamespace () const noexcept=0

  Return the namespace of the plugin object. Should match the plugin namespace returned by the corresponding plugin creator.

Additional Inherited Members

9.165.1 Member Function Documentation

9.165.1.1 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IPluginV3OneCore::getInterfaceInfo ( ) const  [inline], [override], [virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.165.1.2 getPluginName()

```
virtual AsciiChar const * nvinfer1::v_1_0::IPluginV3OneCore::getPluginName ( ) const  [pure virtual], [noexcept]
```

Return the plugin name. Should match the plugin name returned by the corresponding plugin creator.

See also

IPluginCreatorV3One::getPluginName()

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

9.165.1.3 getPluginNamespace()

```
virtual AsciiChar const * nvinfer1::v_1_0::IPluginV3OneCore::getPluginNamespace ( ) const  [pure virtual], [noexcept]
```

Return the namespace of the plugin object. Should match the plugin namespace returned by the corresponding plugin creator.

See also

IPluginCreatorV3One::getPluginNamespace()

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

9.165.1.4 getPluginVersion()

```
virtual AsciiChar const * nvinfer1::v_1_0::IPluginV3OneCore::getPluginVersion ( ) const  [pure virtual], [noexcept]
```

Return the plugin version. Should match the plugin version returned by the corresponding plugin creator.

See also

IPluginCreatorV3One::getPluginVersion()

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.166 IPluginV3OneRuntime Class Reference

A plugin capability interface that enables the runtime capability (PluginCapabilityType::kRUNTIME). Exposes methods that allow the expression of the runtime properties and behavior of a plugin.

```
#include <NvInferRuntime.h>
```

9.166.1 Detailed Description

A plugin capability interface that enables the runtime capability (PluginCapabilityType::kRUNTIME). Exposes methods that allow the expression of the runtime properties and behavior of a plugin.

See also

IPluginCapability

PluginCapabilityType

IPluginV3::getCapabilityInterface()

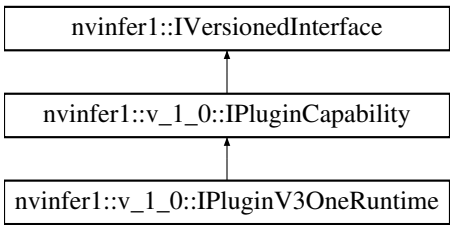
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.167 nvinfer1::v_1_0::IPluginV3OneRuntime Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IPluginV3OneRuntime:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual int32_t setTactic (int32_t) noexcept

  Set the tactic to be used in the subsequent call to enqueue(). If no custom tactics were advertised, this will have a value of 0, which is designated as the default tactic.
- virtual int32_t onShapeChange (PluginTensorDesc const ∗in, int32_t nbInputs, PluginTensorDesc const ∗out, int32_t nbOutputs) noexcept=0

  Called when a plugin is being prepared for execution for specific dimensions. This could happen multiple times in the execution phase, both during creation of an engine by IBuilder and execution of an engine by IExecutionContext.
- virtual int32_t enqueue (PluginTensorDesc const ∗inputDesc, PluginTensorDesc const ∗outputDesc, void const ∗const ∗inputs, void ∗const ∗outputs, void ∗workspace, cudaStream_t stream) noexcept=0

  Execute the layer.
- virtual IPluginV3 ∗ attachToContext (IPluginResourceContext ∗context) noexcept=0

  Clone the plugin, attach the cloned plugin object to a execution context and grant the cloned plugin access to some context resources.
- virtual PluginFieldCollection const ∗ getFieldsToSerialize () noexcept=0

  Get the plugin fields which should be serialized.

Additional Inherited Members

9.167.1 Member Function Documentation

9.167.1.1 attachToContext()

```
virtual IPluginV3 * nvinfer1::v_1_0::IPluginV3OneRuntime::attachToContext (
            IPluginResourceContext * context )  [pure virtual], [noexcept]
```

Clone the plugin, attach the cloned plugin object to a execution context and grant the cloned plugin access to some context resources.

This function is called automatically for each plugin when a new execution context is created. The plugin may use resources provided by the IPluginResourceContext until the plugin is deleted by TensorRT.

If the plugin needs per-context resources, it can be allocated here.

Parameters

context	A resource context that exposes methods to get access to execution context specific resources. A different resource context is guaranteed for each different execution context to which the plugin is attached.

See also

IPluginResourceContext

Note

This method should clone the entire IPluginV3 object, not just the runtime interface

Returns

A clone of the IPluginV3 object whose runtime interface on which this method is invoked, which has attached to the provided resource context.

9.167.1.2 enqueue()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneRuntime::enqueue (
            PluginTensorDesc const * inputDesc,
            PluginTensorDesc const * outputDesc,
            void const *const * inputs,
            void *const * outputs,
            void * workspace,
            cudaStream_t stream )  [pure virtual], [noexcept]
```

Execute the layer.

Parameters

inputDesc	how to interpret the memory for the input tensors.
outputDesc	how to interpret the memory for the output tensors.
inputs	The memory for the input tensors.
outputs	The memory for the output tensors.
workspace	Workspace for execution.
stream	The stream in which to execute the kernels.

Returns

0 for success, else non-zero (which will cause engine termination). The returned code will be reported through the error recorder.

9.167.1.3 getFieldsToSerialize()

```
virtual PluginFieldCollection const * nvinfer1::v_1_0::IPluginV3OneRuntime::getFieldsToSerialize (
)  [pure virtual], [noexcept]
```

Get the plugin fields which should be serialized.

Note

The set of plugin fields returned does not necessarily need to match that advertised through getFieldNames() of the corresponding plugin creator.

To serialize arbitrary plugin data, use a PluginField of PluginFieldType::kUNKNOWN, with the length of the PluginField set to the correct number of bytes.

9.167.1.4 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IPluginV3OneRuntime::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.167.1.5 onShapeChange()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneRuntime::onShapeChange (
            PluginTensorDesc const * in,
            int32_t nbInputs,
            PluginTensorDesc const * out,
            int32_t nbOutputs )  [pure virtual], [noexcept]
```

Called when a plugin is being prepared for execution for specific dimensions. This could happen multiple times in the execution phase, both during creation of an engine by IBuilder and execution of an engine by IExecutionContext.

- IBuilder will call this function once per profile, with `in` resolved to the values specified by the kOPT field of the current profile.
- IExecutionContext will call this during the next subsequent instance of enqueueV3() or executeV2() if:
  - The optimization profile is changed via setOptimizationProfile() or setOptimizationProfileAsync().
  - An input binding is changed via setInputTensorAddress() or setTensorAddress() or setInputShape().

Warning

The execution phase is timing critical during IExecutionContext but is not part of the timing loop when called from IBuilder. Performance bottlenecks of onShapeChange() will not show up during engine building but will be visible during execution if any triggering functions are called.

Parameters

in	The input tensors attributes that are used for configuration.
nbInputs	Number of input tensors.
out	The output tensors attributes that are used for configuration.
nbOutputs	Number of output tensors.

9.167.1.6 setTactic()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneRuntime::setTactic (
            int32_t  )  [inline], [virtual], [noexcept]
```

Set the tactic to be used in the subsequent call to enqueue(). If no custom tactics were advertised, this will have a value of 0, which is designated as the default tactic.

Returns

0 for success, else non-zero (which will cause engine termination). The returned code will be reported through the error recorder.

The documentation for this class was generated from the following file:

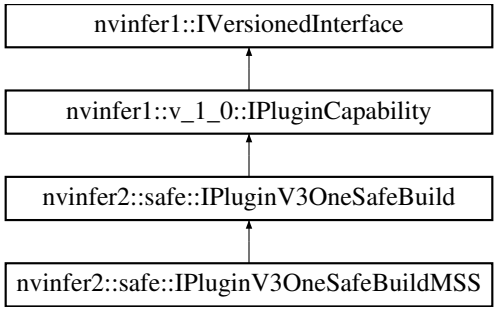
- NvInferRuntime.h

9.168 nvinfer2::safe::IPluginV3OneSafeBuild Class Reference

Interface for plugins used during the TensorRT build phase.

`#include <NvInferSafePlugin.h>`

Inheritance diagram for nvinfer2::safe::IPluginV3OneSafeBuild:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Returns version information associated with this interface.
- IPluginV3OneSafeBuild ()=default

  Default constructor.
- IPluginV3OneSafeBuild (IPluginV3OneSafeBuild &&)=delete

  Deleted move constructor to maintain non-movability.
- IPluginV3OneSafeBuild & operator= (IPluginV3OneSafeBuild &&) &=delete

  Deleted move assignment operator to maintain non-movability.
- ~IPluginV3OneSafeBuild () noexcept override=default

  Destructor for IPluginV3OneSafeBuild.
- virtual int32_t configurePlugin (TensorDescriptor const *in, int32_t nbInputs, TensorDescriptor const *out, int32_t nbOutputs) noexcept=0

  Configure the plugin.
- virtual int32_t getOutputDataTypes (DataType *outputTypes, int32_t nbOutputs, DataType const *inputTypes, int32_t nbInputs) const noexcept=0

  Provide the data types of the plugin outputs if the input tensors have the data types provided.
- virtual int32_t getOutputShapes (Dims const *inputs, int32_t nbInputs, Dims *outputs, int32_t nbOutputs) const noexcept=0

  Provide expressions for computing dimensions of the output tensors from dimensions of the input tensors.
- virtual bool supportsFormatCombination (int32_t pos, TensorDescriptor const *inOut, int32_t nbInputs, int32_t nbOutputs) noexcept=0

  Return true if plugin supports the format and datatype for the input/output indexed by pos.
- virtual int32_t getNbOutputs () const noexcept=0

  Get the number of outputs from the plugin.
- virtual size_t getWorkspaceSize (TensorDescriptor const *, int32_t, TensorDescriptor const *, int32_t) const noexcept

  Find the workspace size required by the layer.
- virtual int32_t getValidTactics (int32_t *, int32_t) noexcept

  Query for any custom tactics that the plugin intends to use.
- virtual int32_t getNbTactics () noexcept

  Query for the number of custom tactics the plugin intends to use.
- virtual char const * getTimingCacheID () noexcept

  Called to query the suffix to use for the timing cache ID. May be called anytime after plugin creation.
- virtual int32_t getFormatCombinationLimit () noexcept

  Return the maximum number of format combinations that will be timed by TensorRT during the build phase.
- virtual char const * getMetadataString () noexcept

  Query for a string representing the configuration of the plugin. May be called anytime after plugin creation.

Static Public Attributes

- static constexpr int32_t kDEFAULT_FORMAT_COMBINATION_LIMIT = 100

  Default maximum number of format combinations timed by TensorRT during the build phase.

Protected Member Functions

- IPluginV3OneSafeBuild (IPluginV3OneSafeBuild const &)=default
- IPluginV3OneSafeBuild & operator= (IPluginV3OneSafeBuild const &) &=default

9.168.1 Detailed Description

Interface for plugins used during the TensorRT build phase.

Extends IPluginCapability to provide build-phase plugin behavior; used when creating an engine by IBuilder. Supports configuration and format combination limits.

9.168.2 Constructor & Destructor Documentation

9.168.2.1 IPluginV3OneSafeBuild() [1/3]

```
nvinfer2::safe::IPluginV3OneSafeBuild::IPluginV3OneSafeBuild ( )  [default]
```

Default constructor.

Precondition

none

Postcondition

Object is in default-constructed state.

9.168.2.2 IPluginV3OneSafeBuild() [2/3]

```
nvinfer2::safe::IPluginV3OneSafeBuild::IPluginV3OneSafeBuild (
            IPluginV3OneSafeBuild &&  )  [delete]
```

Deleted move constructor to maintain non-movability.

Precondition

none

Postcondition

none

9.168.2.3 ~IPluginV3OneSafeBuild()

```
nvinfer2::safe::IPluginV3OneSafeBuild::~IPluginV3OneSafeBuild ( )  [override], [default], [noexcept]
```

Destructor for IPluginV3OneSafeBuild.

Precondition

none

Postcondition

none

9.168.2.4 IPluginV3OneSafeBuild() [3/3]

```
nvinfer2::safe::IPluginV3OneSafeBuild::IPluginV3OneSafeBuild (
            IPluginV3OneSafeBuild const &  )  [protected], [default]
```

9.168.3 Member Function Documentation

9.168.3.1 configurePlugin()

```
virtual int32_t nvinfer2::safe::IPluginV3OneSafeBuild::configurePlugin (
            TensorDescriptor const * in,
            int32_t nbInputs,
            TensorDescriptor const * out,
            int32_t nbOutputs )  [pure virtual], [noexcept]
```

Configure the plugin.

configurePlugin() can be called multiple times in the build phase during creation of an engine by IBuilder.

configurePlugin() is called when a plugin is being prepared for profiling but not for any specific input size. This provides an opportunity for the plugin to make algorithmic choices on the basis of input and output formats, along with the bound of possible dimensions. The min, opt and max value of the TensorDescriptor correspond to the kMIN, kOPT and kMAX value of the current profile that the plugin is being profiled for, with the desc.dims field corresponding to the dimensions of plugin specified at network creation. Wildcard dimensions may exist during this phase in the desc.dims field.

Parameters

in	The input tensors attributes that are used for configuration.
nbInputs	Number of input tensors.
out	The output tensors attributes that are used for configuration.
nbOutputs	Number of output tensors.

9.168.3.2 getFormatCombinationLimit()

```
virtual int32_t nvinfer2::safe::IPluginV3OneSafeBuild::getFormatCombinationLimit ( )  [inline],
[virtual], [noexcept]
```

Return the maximum number of format combinations that will be timed by TensorRT during the build phase.

9.168.3.3 getInterfaceInfo()

```
InterfaceInfo nvinfer2::safe::IPluginV3OneSafeBuild::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Returns version information associated with this interface.

Applications must not override this method.

Precondition

none

Postcondition

none

Returns

InterfaceInfo with kind "PLUGIN_V3ONE_SAFE_BUILD", major 1, minor 0.

Implements nvinfer1::IVersionedInterface.

Reimplemented in nvinfer2::safe::IPluginV3OneSafeBuildMSS.

9.168.3.4 getMetadataString()

```
virtual char const * nvinfer2::safe::IPluginV3OneSafeBuild::getMetadataString ( )  [inline], [virtual],
[noexcept]
```

Query for a string representing the configuration of the plugin. May be called anytime after plugin creation.

Returns

A string representing the plugin's creation state, especially with regard to its attribute values.

9.168.3.5 getNbOutputs()

```
virtual int32_t nvinfer2::safe::IPluginV3OneSafeBuild::getNbOutputs ( ) const  [pure virtual],
[noexcept]
```

Get the number of outputs from the plugin.

Returns

The number of outputs, which must be a positive integer.

9.168.3.6 getNbTactics()

```
virtual int32_t nvinfer2::safe::IPluginV3OneSafeBuild::getNbTactics ( )  [inline], [virtual],
[noexcept]
```

Query for the number of custom tactics the plugin intends to use.

9.168.3.7 getOutputDataTypes()

```
virtual int32_t nvinfer2::safe::IPluginV3OneSafeBuild::getOutputDataTypes (
            DataType * outputTypes,
            int32_t nbOutputs,
            DataType const * inputTypes,
            int32_t nbInputs ) const  [pure virtual], [noexcept]
```

Provide the data types of the plugin outputs if the input tensors have the data types provided.

Parameters

outputTypes	Pre-allocated array to which the output data types should be written.
nbOutputs	The number of output tensors. This matches the value returned from getNbOutputs().
inputTypes	The input data types.
nbInputs	The number of input tensors.

Returns

0 for success, else non-zero (which will cause engine termination). The returned code will be reported through the error recorder.

Note

Provide `DataType::kFLOATs` if the layer has no inputs. The data type for any size tensor outputs must be `DataType::kINT32`. The returned data types must each have a format that is supported by the plugin.

Warning

DataType:kBOOL and DataType::kUINT8 are not supported.

9.168.3.8 getOutputShapes()

```
virtual int32_t nvinfer2::safe::IPluginV3OneSafeBuild::getOutputShapes (
            Dims const * inputs,
            int32_t nbInputs,
            Dims * outputs,
            int32_t nbOutputs ) const  [pure virtual], [noexcept]
```

Provide expressions for computing dimensions of the output tensors from dimensions of the input tensors.

Parameters

inputs	Expressions for dimensions of the input tensors
nbInputs	The number of input tensors
shapeInputs	Expressions for values of the shape tensor inputs
nbShapeInputs	The number of shape tensor inputs
outputs	Pre-allocated array to which the output dimensions must be written
exprBuilder	Object for generating new dimension expressions

Note

Any size tensor outputs must be declared to be 0-D.

The declaration of shapeInputs as DimsExprs is slightly abusive, because the ”dimensions” are actually the values of the shape tensor. For example, if the input shape tensor is a 2x3 matrix, the DimsExprs will have six ”dimensions”: the three values from the first row of the matrix followed by the three values from the second row of the matrix.

Returns

0 for success, else non-zero (which will cause engine termination). Returned code will be reported through the error recorder.

Implemented in nvinfer2::safe::IPluginV3OneSafeBuildMSS.

9.168.3.9 getTimingCacheID()

```
virtual char const * nvinfer2::safe::IPluginV3OneSafeBuild::getTimingCacheID ( )  [inline], [virtual],
[noexcept]
```

Called to query the suffix to use for the timing cache ID. May be called anytime after plugin creation.

Returns

Suffix to use for timing cache ID, considering only the creation state of the plugin. Returning nullptr will disable timing caching for the plugin altogether.

Note

If timing caching is enabled for the plugin (by returning non-null), the I/O shape and format information will be automatically considered to form the prefix of the timing cache ID. Therefore, only other factors determining the creation state of the plugin, such as its attribute values, should be considered to compose the return value.

9.168.3.10 getValidTactics()

```
virtual int32_t nvinfer2::safe::IPluginV3OneSafeBuild::getValidTactics (
            int32_t * ,
            int32_t  )  [inline], [virtual], [noexcept]
```

Query for any custom tactics that the plugin intends to use.

This method queries for the set of tactics T(f) supported by the plugin for the format combination f indicated by the immediately preceding call to configurePlugin(). It is guaranteed to be called after configurePlugin().

For each format combination provided through configurePlugin(), up to a maximum of getFormatCombinationLimit(), the plugin will be timed for each tactic advertised through this method for that format combination. i.e. The plugin will be timed $N = sum_{i=0}^{i<getFormatCombinationLimit()}(T(f[i]))$ times. If $N = 1$, the plugin may not be timed. In peudocode, the timing protocol appears as the following:

counter = 0 for each supported format combination ++counter if counter $>$ getFormatCombinationLimit() goto done configurePlugin(...) for each tactic in getValidTactics(...) time tactic done:

Parameters

tactics	Pre-allocated buffer to which the tactic values should be written
nbTactics	The number of tactics advertised through getNbTactics()

Note

The provided tactic values must be unique and non-zero. The tactic value 0 is reserved for the default tactic attached to each format combination.

Returns

0 for success, else non-zero (which will cause engine termination). The returned code will be reported through the error recorder.

9.168.3.11 getWorkspaceSize()

```
virtual size_t nvinfer2::safe::IPluginV3OneSafeBuild::getWorkspaceSize (
             TensorDescriptor const * ,
             int32_t ,
             TensorDescriptor const * ,
             int32_t  ) const  [inline], [virtual], [noexcept]
```

Find the workspace size required by the layer.

This function is called after the plugin is configured, and possibly during execution. The result should be a sufficient workspace size to deal with inputs and outputs of the given size or any smaller problem.

Returns

The workspace size.

9.168.3.12 operator=() [1/2]

```
IPluginV3OneSafeBuild & nvinfer2::safe::IPluginV3OneSafeBuild::operator= (
             IPluginV3OneSafeBuild &&  ) &  [delete]
```

Deleted move assignment operator to maintain non-movability.

Precondition

none

Postcondition

none

9.168.3.13 operator=() [2/2]

```
IPluginV3OneSafeBuild & nvinfer2::safe::IPluginV3OneSafeBuild::operator= (
            IPluginV3OneSafeBuild const &  ) &  [protected], [default]
```

9.168.3.14 supportsFormatCombination()

```
virtual bool nvinfer2::safe::IPluginV3OneSafeBuild::supportsFormatCombination (
            int32_t pos,
            TensorDescriptor const * inOut,
            int32_t nbInputs,
            int32_t nbOutputs )  [pure virtual], [noexcept]
```

Return true if plugin supports the format and datatype for the input/output indexed by pos.

For this method inputs are numbered 0.. (nbInputs - 1) and outputs are numbered nbInputs.. (nbInputs + nbOutputs

- 1). Using this numbering, pos is an index into InOut, where 0 <= pos < nbInputs + nbOutputs - 1.

TensorRT invokes this method to ask if the input/output indexed by pos supports the format/datatype specified by inOut[pos].format and inOut[pos].type. The override should return true if that format/datatype at inOut[pos] are supported by the plugin. If support is conditional on other input/output formats/datatypes, the plugin can make its result conditional on the formats/datatypes in inOut[0.. pos - 1], which will be set to values that the plugin supports. The override should not inspect inOut[pos1.. nbInputs + nbOutputs - 1], which will have invalid values. In other words, the decision for pos must be based on inOut[0..pos] only.

Some examples:

- A definition for a plugin that supports only FP16 NCHW:

  ```
  return inOut.format[pos] == TensorFormat::kLINEAR && inOut.type[pos] == DataType::kHALF;
  ```

- A definition for a plugin that supports only FP16 NCHW for its two inputs, and FP32 NCHW for its single output:

  ```
  return inOut.format[pos] == TensorFormat::kLINEAR && (inOut.type[pos] == pos < 2 ?  DataType::kHALF :
  DataType::kFLOAT);
  ```

- A definition for a "polymorphic" plugin with two inputs and one output that supports any format or type, but the inputs and output must have the same format and type:

  ```
  return pos == 0 || (inOut.format[pos] == inOut.format[0] && inOut.type[pos] == inOut.type[0]);
  ```

> **Warning**
>
> TensorRT will stop querying once it finds getFormatCombinationLimit() of combinations.

See also

getFormatCombinationLimit

9.168.4 Member Data Documentation

9.168.4.1 kDEFAULT_FORMAT_COMBINATION_LIMIT

```
constexpr int32_t nvinfer2::safe::IPluginV3OneSafeBuild::kDEFAULT_FORMAT_COMBINATION_LIMIT = 100
[static], [constexpr]
```

Default maximum number of format combinations timed by TensorRT during the build phase.

See also

getFormatCombinationLimit

The documentation for this class was generated from the following file:

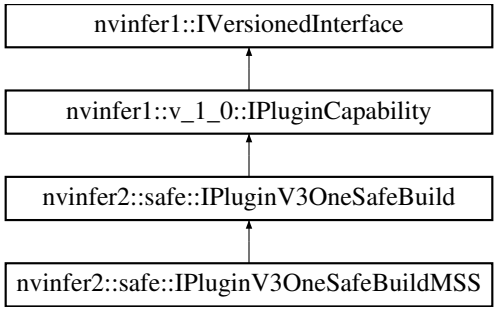
- NvInferSafePlugin.h

9.169 nvinfer2::safe::IPluginV3OneSafeBuildMSS Class Reference

This class provides build capability and let output shapes be calculated by symbolic expression to support multiple static shape.

```
#include <NvInferSafePlugin.h>
```

Inheritance diagram for nvinfer2::safe::IPluginV3OneSafeBuildMSS:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- int32_t getOutputShapes (Dims const *, int32_t, Dims *, int32_t) const noexcept override

  Provide expressions for computing dimensions of the output tensors from dimensions of the input tensors.
- IPluginV3OneSafeBuildMSS (IPluginV3OneSafeBuildMSS const &)=default

  Copy constructor.
- IPluginV3OneSafeBuildMSS (IPluginV3OneSafeBuildMSS &&)=delete

  Move constructor.
- IPluginV3OneSafeBuildMSS & operator= (IPluginV3OneSafeBuildMSS const &) &=default

  Copy assignment operator.
- IPluginV3OneSafeBuildMSS & operator= (IPluginV3OneSafeBuildMSS &&) &=delete

  Move assignment operator.
- ~IPluginV3OneSafeBuildMSS () noexcept override=default

  Destructor for IPluginV3OneSafeBuildMSS.
- virtual int32_t getSymbolicOutputShapes (nvinfer1::DimsExprs const *inputs, int32_t nbInputs, nvinfer1::DimsExprs *outputs, int32_t nbOutputs, nvinfer1::IExprBuilder &exprBuilder) const noexcept=0

  Provide expressions for computing dimensions of the output tensors from dimensions of the input tensors.

Protected Member Functions

- IPluginV3OneSafeBuildMSS ()=default

  Default constructor.

Additional Inherited Members

9.169.1 Detailed Description

This class provides build capability and let output shapes be calculated by symbolic expression to support multiple static shape.

9.169.2 Constructor & Destructor Documentation

9.169.2.1 IPluginV3OneSafeBuildMSS() [1/3]

```
nvinfer2::safe::IPluginV3OneSafeBuildMSS::IPluginV3OneSafeBuildMSS (
            IPluginV3OneSafeBuildMSS const &  )  [default]
```

Copy constructor.

9.169.2.2 IPluginV3OneSafeBuildMSS() [2/3]

```
nvinfer2::safe::IPluginV3OneSafeBuildMSS::IPluginV3OneSafeBuildMSS (
            IPluginV3OneSafeBuildMSS &&  )  [delete]
```

Move constructor.

9.169.2.3 ~IPluginV3OneSafeBuildMSS()

```
nvinfer2::safe::IPluginV3OneSafeBuildMSS::~IPluginV3OneSafeBuildMSS ( )  [override], [default],
[noexcept]
```

Destructor for IPluginV3OneSafeBuildMSS.

9.169.2.4 IPluginV3OneSafeBuildMSS() [3/3]

```
nvinfer2::safe::IPluginV3OneSafeBuildMSS::IPluginV3OneSafeBuildMSS ( )  [protected], [default]
```

Default constructor.

9.169.3 Member Function Documentation

9.169.3.1 getInterfaceInfo()

```
InterfaceInfo nvinfer2::safe::IPluginV3OneSafeBuildMSS::getInterfaceInfo ( ) const  [inline],
[override], [virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Reimplemented from nvinfer2::safe::IPluginV3OneSafeBuild.

9.169.3.2 getOutputShapes()

```
int32_t nvinfer2::safe::IPluginV3OneSafeBuildMSS::getOutputShapes (
            Dims const * inputs,
            int32_t nbInputs,
            Dims * outputs,
            int32_t nbOutputs ) const  [inline], [override], [virtual], [noexcept]
```

Provide expressions for computing dimensions of the output tensors from dimensions of the input tensors.

Parameters

inputs	Expressions for dimensions of the input tensors
nbInputs	The number of input tensors
shapeInputs	Expressions for values of the shape tensor inputs
nbShapeInputs	The number of shape tensor inputs
outputs	Pre-allocated array to which the output dimensions must be written
exprBuilder	Object for generating new dimension expressions

Note

Any size tensor outputs must be declared to be 0-D.

The declaration of shapeInputs as DimsExprs is slightly abusive, because the ”dimensions” are actually the values of the shape tensor. For example, if the input shape tensor is a 2x3 matrix, the DimsExprs will have six ”dimensions”: the three values from the first row of the matrix followed by the three values from the second row of the matrix.

Returns

0 for success, else non-zero (which will cause engine termination). Returned code will be reported through the error recorder.

Implements nvinfer2::safe::IPluginV3OneSafeBuild.

9.169.3.3 getSymbolicOutputShapes()

```
virtual int32_t nvinfer2::safe::IPluginV3OneSafeBuildMSS::getSymbolicOutputShapes (
            nvinfer1::DimsExprs const * inputs,
            int32_t nbInputs,
            nvinfer1::DimsExprs * outputs,
            int32_t nbOutputs,
            nvinfer1::IExprBuilder & exprBuilder ) const  [pure virtual], [noexcept]
```

Provide expressions for computing dimensions of the output tensors from dimensions of the input tensors.

Parameters

inputs	Expressions for dimensions of the input tensors
nbInputs	The number of input tensors
outputs	Pre-allocated array to which the output dimensions must be written
nbOutputs	The number of output tensors
exprBuilder	Object for generating new dimension expressions

Note

Any size tensor outputs must be declared to be 0-D.

Returns

0 for success, else non-zero (which will cause engine termination). Returned code will be reported through the error recorder.

9.169.3.4 operator=() [1/2]

```
IPluginV3OneSafeBuildMSS & nvinfer2::safe::IPluginV3OneSafeBuildMSS::operator= (
            IPluginV3OneSafeBuildMSS &&  ) &  [delete]
```

Move assignment operator.

9.169.3.5 operator=() [2/2]

```
IPluginV3OneSafeBuildMSS & nvinfer2::safe::IPluginV3OneSafeBuildMSS::operator= (
            IPluginV3OneSafeBuildMSS const &  ) &  [default]
```

Copy assignment operator.

The documentation for this class was generated from the following file:

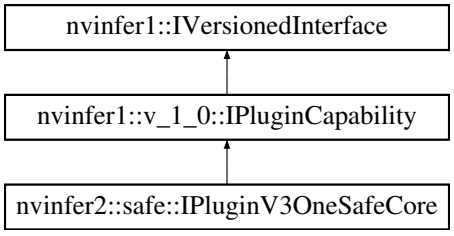
- NvInferSafePlugin.h

9.170 nvinfer2::safe::IPluginV3OneSafeCore Class Reference

Interface specifying the base plugin capability.

`#include <NvInferSafePlugin.h>`

Inheritance diagram for nvinfer2::safe::IPluginV3OneSafeCore:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- IPluginV3OneSafeCore (IPluginV3OneSafeCore const &)=default

  Copy constructor.
- IPluginV3OneSafeCore (IPluginV3OneSafeCore &&)=default

  Move constructor.
- IPluginV3OneSafeCore & operator= (IPluginV3OneSafeCore const &) &=default

  Copy assignment operator.
- IPluginV3OneSafeCore & operator= (IPluginV3OneSafeCore &&) &=default

  Move assignment operator.
- ~IPluginV3OneSafeCore () noexcept override=default

  Destructor for IPluginV3OneSafeCore.
- virtual AsciiChar const * getPluginName () const noexcept=0

  Return the plugin name. Should match the plugin name returned by the corresponding plugin creator.
- virtual AsciiChar const * getPluginVersion () const noexcept=0

  Return the plugin version. Should match the plugin version returned by the corresponding plugin creator.
- virtual AsciiChar const * getPluginNamespace () const noexcept=0

  Return the namespace of the plugin object. Should match the plugin namespace returned by the corresponding plugin creator.

Protected Member Functions

- IPluginV3OneSafeCore ()=default

  Default constructor.

9.170.1 Detailed Description

Interface specifying the base plugin capability.

9.170.2 Constructor & Destructor Documentation

9.170.2.1 IPluginV3OneSafeCore() [1/3]

```
nvinfer2::safe::IPluginV3OneSafeCore::IPluginV3OneSafeCore (
            IPluginV3OneSafeCore const &  )  [default]
```

Copy constructor.

9.170.2.2 IPluginV3OneSafeCore() [2/3]

```
nvinfer2::safe::IPluginV3OneSafeCore::IPluginV3OneSafeCore (
            IPluginV3OneSafeCore &&  )  [default]
```

Move constructor.

9.170.2.3 ~IPluginV3OneSafeCore()

```
nvinfer2::safe::IPluginV3OneSafeCore::~IPluginV3OneSafeCore ( )  [override], [default], [noexcept]
```

Destructor for IPluginV3OneSafeCore.

9.170.2.4 IPluginV3OneSafeCore() [3/3]

```
nvinfer2::safe::IPluginV3OneSafeCore::IPluginV3OneSafeCore ( )  [protected], [default]
```

Default constructor.

9.170.3 Member Function Documentation

9.170.3.1 getInterfaceInfo()

```
InterfaceInfo nvinfer2::safe::IPluginV3OneSafeCore::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Precondition

INIT API

Implements nvinfer1::IVersionedInterface.

9.170.3.2 getPluginName()

```
virtual AsciiChar const * nvinfer2::safe::IPluginV3OneSafeCore::getPluginName ( ) const  [pure
virtual], [noexcept]
```

Return the plugin name. Should match the plugin name returned by the corresponding plugin creator.

Precondition

INIT API

See also

ISafePluginCreatorV3One::getPluginName()

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

9.170.3.3 getPluginNamespace()

```
virtual AsciiChar const * nvinfer2::safe::IPluginV3OneSafeCore::getPluginNamespace ( ) const  [pure virtual], [noexcept]
```

Return the namespace of the plugin object. Should match the plugin namespace returned by the corresponding plugin creator.

Precondition

INIT API

See also

ISafePluginCreatorV3One::getPluginNamespace()

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

9.170.3.4 getPluginVersion()

```
virtual AsciiChar const * nvinfer2::safe::IPluginV3OneSafeCore::getPluginVersion ( ) const  [pure virtual], [noexcept]
```

Return the plugin version. Should match the plugin version returned by the corresponding plugin creator.

Precondition

INIT API

See also

ISafePluginCreatorV3One::getPluginVersion()

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

9.170.3.5 operator=() [1/2]

```
IPluginV3OneSafeCore & nvinfer2::safe::IPluginV3OneSafeCore::operator= (
            IPluginV3OneSafeCore &&  ) &  [default]
```

Move assignment operator.

9.170.3.6 operator=() [2/2]

```
IPluginV3OneSafeCore & nvinfer2::safe::IPluginV3OneSafeCore::operator= (
            IPluginV3OneSafeCore const &  ) &  [default]
```

Copy assignment operator.

The documentation for this class was generated from the following file:

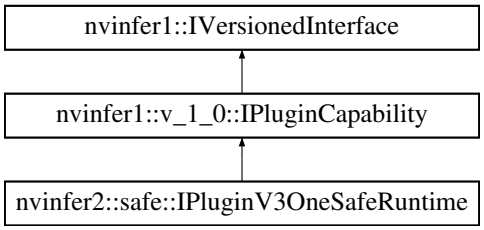
- NvInferSafePlugin.h

9.171 nvinfer2::safe::IPluginV3OneSafeRuntime Class Reference

Runtime component of the pluginV3 system.

```
#include <NvInferSafePlugin.h>
```

Inheritance diagram for nvinfer2::safe::IPluginV3OneSafeRuntime:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- IPluginV3OneSafeRuntime (IPluginV3OneSafeRuntime const &)=default

  Copy constructor.
- IPluginV3OneSafeRuntime (IPluginV3OneSafeRuntime &&)=default

  Move constructor.
- IPluginV3OneSafeRuntime & operator= (IPluginV3OneSafeRuntime const &) &=default

  Copy assignment operator.
- IPluginV3OneSafeRuntime & operator= (IPluginV3OneSafeRuntime &&) &=default

  Move assignment operator.
- ~IPluginV3OneSafeRuntime () noexcept override=default

  Destructor for IPluginV3OneSafeRuntime.
- virtual int32_t setTactic (int32_t) noexcept

  Set the tactic to be used in the subsequent call to enqueue(). If no custom tactics were advertised, this will have a value of 0, which is designated as the default tactic.
- virtual int32_t enqueue (TensorDescriptor const *inputDesc, TensorDescriptor const *outputDesc, void const *const *inputs, void *const *outputs, void *workspace, cudaStream_t stream) noexcept=0

  Execute the layer.
- virtual int32_t initResource (ISafePluginResourceContext const *context) noexcept=0

  This function will be called to initialize a plugin object after it is created (before execution phase), either during graph deserialization or when the graph is cloned. The overriding implementations of this function can use the provided context object to access the message recorder, the memory allocator and the runtime error buffer of the graph context to which this plugin belongs.
- virtual IPluginV3 * clone () noexcept=0

  Clone the plugin object. This copies over internal plugin parameters and returns a new plugin object with these parameters. This call will always be followed by a call to initResource() to complete any initializations that depend on the graph context.
- virtual PluginFieldCollection const * getFieldsToSerialize () noexcept=0

  Get the plugin fields which should be serialized.

Protected Member Functions

- IPluginV3OneSafeRuntime ()=default

  Default constructor.

9.171.1 Detailed Description

Runtime component of the pluginV3 system.

9.171.2 Constructor & Destructor Documentation

9.171.2.1 IPluginV3OneSafeRuntime() [1/3]

```
nvinfer2::safe::IPluginV3OneSafeRuntime::IPluginV3OneSafeRuntime (
            IPluginV3OneSafeRuntime const &  )  [default]
```

Copy constructor.

9.171.2.2 IPluginV3OneSafeRuntime() [2/3]

```
nvinfer2::safe::IPluginV3OneSafeRuntime::IPluginV3OneSafeRuntime (
            IPluginV3OneSafeRuntime &&  )  [default]
```

Move constructor.

9.171.2.3 ~IPluginV3OneSafeRuntime()

```
nvinfer2::safe::IPluginV3OneSafeRuntime::~IPluginV3OneSafeRuntime ( )  [override], [default],
[noexcept]
```

Destructor for IPluginV3OneSafeRuntime.

9.171.2.4 IPluginV3OneSafeRuntime() [3/3]

```
nvinfer2::safe::IPluginV3OneSafeRuntime::IPluginV3OneSafeRuntime ( )  [protected], [default]
```

Default constructor.

9.171.3 Member Function Documentation

9.171.3.1 clone()

```
virtual IPluginV3 * nvinfer2::safe::IPluginV3OneSafeRuntime::clone ( )  [pure virtual], [noexcept]
```

Clone the plugin object. This copies over internal plugin parameters and returns a new plugin object with these parameters. This call will always be followed by a call to initResource() to complete any initializations that depend on the graph context.

Precondition

INIT API

9.171.3.2 enqueue()

```
virtual int32_t nvinfer2::safe::IPluginV3OneSafeRuntime::enqueue (
            TensorDescriptor const * inputDesc,
            TensorDescriptor const * outputDesc,
            void const *const * inputs,
            void *const * outputs,
            void * workspace,
            cudaStream_t stream )  [pure virtual], [noexcept]
```

Execute the layer.

Precondition

RUNTIME API

Parameters

inputDesc	how to interpret the memory for the input tensors.
outputDesc	how to interpret the memory for the output tensors.
inputs	The memory for the input tensors.
outputs	The memory for the output tensors.
workspace	Workspace for execution.
stream	The stream in which to execute the kernels.

Returns

0 for success, else non-zero (which will cause engine termination). The returned code will be reported through the error recorder.

9.171.3.3 getFieldsToSerialize()

```
virtual PluginFieldCollection const * nvinfer2::safe::IPluginV3OneSafeRuntime::getFieldsToSerialize
( )  [pure virtual], [noexcept]
```

Get the plugin fields which should be serialized.

Precondition

INIT API

Note

The set of plugin fields returned does not necessarily need to match that advertised through getFieldNames() of the corresponding plugin creator.

To serialize arbitrary plugin data, use a PluginField of PluginFieldType::kUNKNOWN, with the length of the PluginField set to the correct number of bytes.

9.171.3.4 getInterfaceInfo()

```
InterfaceInfo nvinfer2::safe::IPluginV3OneSafeRuntime::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Precondition

INIT API

Implements nvinfer1::IVersionedInterface.

9.171.3.5 initResource()

```
virtual int32_t nvinfer2::safe::IPluginV3OneSafeRuntime::initResource (
            ISafePluginResourceContext const * context )  [pure virtual], [noexcept]
```

This function will be called to initialize a plugin object after it is created (before execution phase), either during graph deserialization or when the graph is cloned. The overriding implementations of this function can use the provided context object to access the message recorder, the memory allocator and the runtime error buffer of the graph context to which this plugin belongs.

Precondition

INIT API

Parameters

ISafePluginResourceContext	A pointer to the ISafePluginResourceContext which contains the message recorder, the memory allocator, and the runtime error information buffer.

Returns

0 for success, else non-zero (which will cause engine termination). The returned code will be reported through the error recorder.

9.171.3.6 operator=() [1/2]

```
IPluginV3OneSafeRuntime & nvinfer2::safe::IPluginV3OneSafeRuntime::operator= (
            IPluginV3OneSafeRuntime &&  ) &  [default]
```

Move assignment operator.

9.171.3.7 operator=() [2/2]

```
IPluginV3OneSafeRuntime & nvinfer2::safe::IPluginV3OneSafeRuntime::operator= (
            IPluginV3OneSafeRuntime const &  ) &  [default]
```

Copy assignment operator.

9.171.3.8 setTactic()

```
virtual int32_t nvinfer2::safe::IPluginV3OneSafeRuntime::setTactic (
            int32_t  )  [inline], [virtual], [noexcept]
```

Set the tactic to be used in the subsequent call to enqueue(). If no custom tactics were advertised, this will have a value of 0, which is designated as the default tactic.

Precondition

INIT API

Returns

0 for success, else non-zero (which will cause engine termination). The returned code will be reported through the error recorder.

The documentation for this class was generated from the following file:

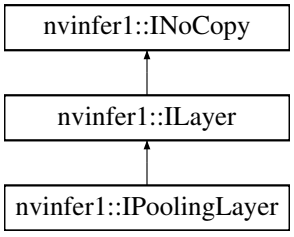
- NvInferSafePlugin.h

9.172 nvinfer1::IPoolingLayer Class Reference

A Pooling layer in a network definition.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IPoolingLayer:



Public Member Functions

- void setPoolingType (PoolingType type) noexcept

  Set the type of activation to be performed.
- PoolingType getPoolingType () const noexcept

  Get the type of activation to be performed.
- void setBlendFactor (float blendFactor) noexcept

  Set the blending factor for the max_average_blend mode: max_average_blendPool = (1-blendFactor)∗maxPool + blend↩ Factor∗avgPool blendFactor is a user value in [0,1] with the default value of 0.0 This value only applies for the kMAX↩ _AVERAGE_BLEND mode.
- float getBlendFactor () const noexcept

  Get the blending factor for the max_average_blend mode: max_average_blendPool = (1-blendFactor)∗maxPool + blendFactor∗avgPool blendFactor is a user value in [0,1] with the default value of 0.0 In modes other than kMAX↩ _AVERAGE_BLEND, blendFactor is ignored.
- void setAverageCountExcludesPadding (bool exclusive) noexcept

  Set whether average pooling uses as a denominator the overlap area between the window and the unpadded input. If this is not set, the denominator is the overlap between the pooling window and the padded input.
- bool getAverageCountExcludesPadding () const noexcept

  Get whether average pooling uses as a denominator the overlap area between the window and the unpadded input.
- void setPrePadding (Dims const &padding) noexcept

  Set the multi-dimension pre-padding for pooling.
- Dims getPrePadding () const noexcept

  Get the pre-padding.
- void setPostPadding (Dims const &padding) noexcept

  Set the multi-dimension post-padding for pooling.
- Dims getPostPadding () const noexcept

  Get the padding.
- void setPaddingMode (PaddingMode paddingMode) noexcept

  Set the padding mode.
- PaddingMode getPaddingMode () const noexcept

  Get the padding mode.
- void setWindowSizeNd (Dims const &windowSize) noexcept

  Set the multi-dimension window size for pooling.
- Dims getWindowSizeNd () const noexcept

  Get the multi-dimension window size for pooling.
- void setStrideNd (Dims const &stride) noexcept

  Set the multi-dimension stride for pooling.
- Dims getStrideNd () const noexcept

  Get the multi-dimension stride for pooling.
- void setPaddingNd (Dims const &padding) noexcept

  Set the multi-dimension padding for pooling.
- Dims getPaddingNd () const noexcept

  Get the multi-dimension padding for pooling.

Protected Member Functions

- virtual ~IPoolingLayer () noexcept=0

Protected Attributes

- apiv::VPoolingLayer * mImpl

9.172.1 Detailed Description

A Pooling layer in a network definition.

The layer applies a reduction operation within a window over the input.

> Warning
>
> When running pooling layer with DeviceType::kDLA in Int8 mode, the dynamic ranges for input and output tensors must be equal.
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.172.2 Constructor & Destructor Documentation

9.172.2.1 ~IPoolingLayer()

```
nvinfer1::IPoolingLayer::~IPoolingLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.172.3 Member Function Documentation

9.172.3.1 getAverageCountExcludesPadding()

```
bool nvinfer1::IPoolingLayer::getAverageCountExcludesPadding ( ) const  [inline], [noexcept]
```

Get whether average pooling uses as a denominator the overlap area between the window and the unpadded input.

See also

setAverageCountExcludesPadding()

9.172.3.2 getBlendFactor()

```
float nvinfer1::IPoolingLayer::getBlendFactor ( ) const  [inline], [noexcept]
```

Get the blending factor for the max_average_blend mode: max_average_blendPool = (1-blendFactor)∗maxPool + blendFactor∗avgPool blendFactor is a user value in [0,1] with the default value of 0.0 In modes other than kMAX_↩ AVERAGE_BLEND, blendFactor is ignored.

See also

setBlendFactor()

9.172.3.3 getPaddingMode()

```
PaddingMode nvinfer1::IPoolingLayer::getPaddingMode ( ) const  [inline], [noexcept]
```

Get the padding mode.

Default: kEXPLICIT_ROUND_DOWN

See also

setPaddingMode()

9.172.3.4 getPaddingNd()

```
Dims nvinfer1::IPoolingLayer::getPaddingNd ( ) const  [inline], [noexcept]
```

Get the multi-dimension padding for pooling.

If the padding is asymmetric, the pre-padding is returned.

See also

setPaddingNd()

9.172.3.5 getPoolingType()

```
PoolingType nvinfer1::IPoolingLayer::getPoolingType ( ) const  [inline], [noexcept]
```

Get the type of activation to be performed.

See also

setPoolingType(), PoolingType

9.172.3.6 getPostPadding()

```
Dims nvinfer1::IPoolingLayer::getPostPadding ( ) const  [inline], [noexcept]
```

Get the padding.

See also

setPostPadding()

9.172.3.7 getPrePadding()

```
Dims nvinfer1::IPoolingLayer::getPrePadding ( ) const  [inline], [noexcept]
```

Get the pre-padding.

See also

setPrePadding()

9.172.3.8 getStrideNd()

```
Dims nvinfer1::IPoolingLayer::getStrideNd ( ) const  [inline], [noexcept]
```

Get the multi-dimension stride for pooling.

See also

setStrideNd()

9.172.3.9 getWindowSizeNd()

```
Dims nvinfer1::IPoolingLayer::getWindowSizeNd ( ) const  [inline], [noexcept]
```

Get the multi-dimension window size for pooling.

See also

setWindowSizeNd()

9.172.3.10 setAverageCountExcludesPadding()

```
void nvinfer1::IPoolingLayer::setAverageCountExcludesPadding (
            bool exclusive )  [inline], [noexcept]
```

Set whether average pooling uses as a denominator the overlap area between the window and the unpadded input. If this is not set, the denominator is the overlap between the pooling window and the padded input.

Default: true

See also

getAverageCountExcludesPadding()

9.172.3.11 setBlendFactor()

```
void nvinfer1::IPoolingLayer::setBlendFactor (
            float blendFactor )  [inline], [noexcept]
```

Set the blending factor for the max_average_blend mode: max_average_blendPool = (1-blendFactor)∗maxPool + blendFactor∗avgPool blendFactor is a user value in [0,1] with the default value of 0.0 This value only applies for the kMAX_AVERAGE_BLEND mode.

Since DLA does not support kMAX_AVERAGE_BLEND, blendFactor is ignored on the DLA.

See also

getBlendFactor()

9.172.3.12 setPaddingMode()

```
void nvinfer1::IPoolingLayer::setPaddingMode (
            PaddingMode paddingMode )  [inline], [noexcept]
```

Set the padding mode.

Padding mode takes precedence if both setPaddingMode and setPre/PostPadding are used.

Default: kEXPLICIT_ROUND_DOWN

See also

getPaddingMode()

9.172.3.13 setPaddingNd()

```
void nvinfer1::IPoolingLayer::setPaddingNd (
            Dims const & padding )  [inline], [noexcept]
```

Set the multi-dimension padding for pooling.

The input will be padded by this number of elements in each dimension. Padding is symmetric. Padding value depends on pooling type, -inf is used for max pooling and zero padding for average pooling.

Default: (0, 0, ..., 0)

If executing this layer on DLA, only support 2D padding, both height and width of padding must be in the range [0,7].

See also

getPaddingNd() setPadding() getPadding()

9.172.3.14 setPoolingType()

```
void nvinfer1::IPoolingLayer::setPoolingType (
            PoolingType type )  [inline], [noexcept]
```

Set the type of activation to be performed.

DLA only supports kMAX and kAVERAGE pooling types.

See also

getPoolingType(), PoolingType

9.172.3.15 setPostPadding()

```
void nvinfer1::IPoolingLayer::setPostPadding (
            Dims const & padding )  [inline], [noexcept]
```

Set the multi-dimension post-padding for pooling.

The end of the input will be padded by this number of elements in each dimension. Padding value depends on pooling type, -inf is used for max pooling and zero padding for average pooling.

Default: (0, 0, ..., 0)

If executing this layer on DLA, only support 2D padding, both height and width of padding must be in the range [0,7].

See also

getPostPadding()

9.172.3.16 setPrePadding()

```
void nvinfer1::IPoolingLayer::setPrePadding (
            Dims const & padding )  [inline], [noexcept]
```

Set the multi-dimension pre-padding for pooling.

The start of the input will be padded by this number of elements in each dimension. Padding value depends on pooling type, -inf is used for max pooling and zero padding for average pooling.

Default: (0, 0, ..., 0)

If executing this layer on DLA, only support 2D padding, both height and width of padding must be in the range [0,7].

See also

getPrePadding()

9.172.3.17 setStrideNd()

```
void nvinfer1::IPoolingLayer::setStrideNd (
            Dims const & stride )  [inline], [noexcept]
```

Set the multi-dimension stride for pooling.

Default: (1, 1, ..., 1)

If executing this layer on DLA, only support 2D stride, both height and width of stride must be in the range [1,16].

See also

getStrideNd()

9.172.3.18 setWindowSizeNd()

```
void nvinfer1::IPoolingLayer::setWindowSizeNd (
            Dims const & windowSize )  [inline], [noexcept]
```

Set the multi-dimension window size for pooling.

If executing this layer on DLA, only support 2D window size, both height and width of window size must be in the range [1,8].

See also

getWindowSizeNd() setWindowSize() getWindowSize()

9.172.4 Member Data Documentation

9.172.4.1 mImpl

```
apiv::VPoolingLayer* nvinfer1::IPoolingLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.173 IProfiler Class Reference

Application-implemented interface for profiling.

```
#include <NvInferRuntime.h>
```

9.173.1 Detailed Description

Application-implemented interface for profiling.

When this class is added to an execution context, the profiler will be called once per layer for each invocation of executeV2()/enqueueV3().

It is not recommended to run inference with profiler enabled when the inference execution time is critical since the profiler may affect execution time negatively.

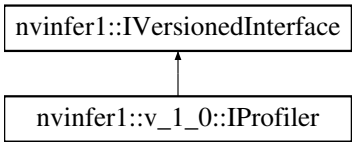
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.174 nvinfer1::v_1_0::IProfiler Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IProfiler:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual void reportLayerTime (char const *layerName, float ms) noexcept=0

  Layer time reporting callback.
- ~IProfiler () override=default

Additional Inherited Members

9.174.1 Constructor & Destructor Documentation

9.174.1.1 ~IProfiler()

`nvinfer1::v_1_0::IProfiler::~IProfiler ( )  [override], [default]`

9.174.2 Member Function Documentation

9.174.2.1 getInterfaceInfo()

`InterfaceInfo nvinfer1::v_1_0::IProfiler::getInterfaceInfo ( ) const  [inline], [override], [virtual], [noexcept]`

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.174.2.2 reportLayerTime()

```
virtual void nvinfer1::v_1_0::IProfiler::reportLayerTime (
            char const * layerName,
            float ms )  [pure virtual], [noexcept]
```

Layer time reporting callback.

Parameters

layerName	The name of the layer, set when constructing the network definition. If the engine is built with profiling verbosity set to kNONE, the layerName is the decimal index of the layer.
ms	The time in milliseconds to execute the layer.

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.175 IProgressMonitor Class Reference

Application-implemented progress reporting interface for TensorRT.

`#include <NvInfer.h>`

9.175.1 Detailed Description

Application-implemented progress reporting interface for TensorRT.

The IProgressMonitor is a user-defined object that TensorRT uses to report back when an internal algorithm has started or finished a phase to help provide feedback on the progress of the optimizer.

The IProgressMonitor will trigger its start function when a phase is entered and will trigger its finish function when that phase is exited. Each phase consists of one or more steps. When each step is completed, the stepComplete function is triggered. This will allow an application using the builder to communicate progress relative to when the optimization step is expected to complete.

The implementation of IProgressMonitor must be thread-safe so that it can be called from multiple internal threads. The lifetime of the IProgressMonitor must exceed the lifetime of all TensorRT objects that use it.

Note

To ensure compatibility of source code with future versions of TensorRT, use IProgressMonitor, not v_1_0::↩ IProgressMonitor

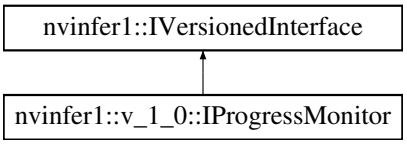
The documentation for this class was generated from the following file:

- NvInfer.h

9.176 nvinfer1::v_1_0::IProgressMonitor Class Reference

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::v_1_0::IProgressMonitor:



Public Member Functions

- IProgressMonitor ()=default
- virtual ~IProgressMonitor () noexcept=default
- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual void phaseStart (char const *phaseName, char const *parentPhase, int32_t nbSteps) noexcept=0

  Signal that a phase of the optimizer has started.
- virtual bool stepComplete (char const *phaseName, int32_t step) noexcept=0

  Signal that a step of an optimizer phase has finished.
- virtual void phaseFinish (char const *phaseName) noexcept=0

  Signal that a phase of the optimizer has finished.

Additional Inherited Members

9.176.1 Constructor & Destructor Documentation

9.176.1.1 IProgressMonitor()

`nvinfer1::v_1_0::IProgressMonitor::IProgressMonitor ( ) [default]`

9.176.1.2 ~IProgressMonitor()

`virtual nvinfer1::v_1_0::IProgressMonitor::~IProgressMonitor ( ) [virtual], [default], [noexcept]`

9.176.2 Member Function Documentation

9.176.2.1 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IProgressMonitor::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.176.2.2 phaseFinish()

```
virtual void nvinfer1::v_1_0::IProgressMonitor::phaseFinish (
            char const * phaseName )  [pure virtual], [noexcept]
```

Signal that a phase of the optimizer has finished.

Parameters

phaseName	The name of the phase that has finished.

The phaseFinish function signals to the application that the phase is complete. This function may be called before all steps in the range [0, nbSteps) have been reported to stepComplete. This scenario can be triggered by error handling, internal optimizations, or when stepComplete returns false to request cancellation of the build.

See also

phaseStart

9.176.2.3 phaseStart()

```
virtual void nvinfer1::v_1_0::IProgressMonitor::phaseStart (
            char const * phaseName,
            char const * parentPhase,
            int32_t nbSteps )  [pure virtual], [noexcept]
```

Signal that a phase of the optimizer has started.

Parameters

phaseName	The name of this phase for tracking purposes.
parentPhase	The parent phase that this phase belongs to, or nullptr if there is no parent.
nbSteps	The number of steps that are involved in this phase.

The phaseStart function signals to the application that the current phase is beginning, and that it has a certain number of steps to perform. If `phaseParent` is nullptr, then the phaseStart is beginning an independent phase, and if `phaseParent` is specified, then the current phase, specified by `phaseName`, is within the scope of the parent phase. `nbSteps` will always be a positive number. The phaseStart function implies that the first step is being executed. TensorRT will signal when each step is complete.

Phase names are human readable English strings which are unique within a single phase hierarchy but which can be reused once the previous instance has completed. Phase names and their hierarchies may change between versions of TensorRT.

See also

phaseFinish

9.176.2.4 stepComplete()

```
virtual bool nvinfer1::v_1_0::IProgressMonitor::stepComplete (
            char const * phaseName,
            int32_t step )  [pure virtual], [noexcept]
```

Signal that a step of an optimizer phase has finished.

Parameters

phaseName	The name of the innermost phase being executed.
step	The step number that was completed.

The stepComplete function signals to the application that TensorRT has finished the current `step` for the phase `phaseName`, and will move onto the next step if there is one. The application can return false for TensorRT to exit the build early. The step value will increase on subsequent calls in the range [0, nbSteps).

Returns

true to continue to the next step or false to stop the build.

The documentation for this class was generated from the following file:

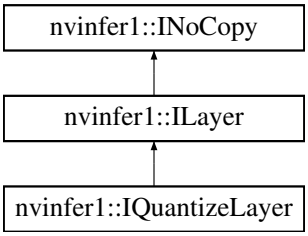
- NvInfer.h

9.177 nvinfer1::IQuantizeLayer Class Reference

A Quantize layer in a network definition.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IQuantizeLayer:



Public Member Functions

- int32_t getAxis () const noexcept

  Get the quantization axis.
- void setAxis (int32_t axis) noexcept

  Set the quantization axis.
- bool setBlockShape (Dims const &blockShape) noexcept

  Set the shape of the quantization block.
- TRT_NODISCARD Dims getBlockShape () const noexcept

  Get the shape of the quantization block.
- void setToType (DataType toType) noexcept

  Set the Quantize layer output type.
- DataType getToType () const noexcept

  Return the Quantize layer output type.

Protected Member Functions

- virtual ~IQuantizeLayer () noexcept=0

Protected Attributes

- apiv::VQuantizeLayer * mImpl

9.177.1 Detailed Description

A Quantize layer in a network definition.

This layer accepts a floating-point data input tensor, and uses the scale and zeroPt inputs to quantize the data according to: `output` = clamp(round(`input` / `scale`) + `zeroPt`)

Rounding type is rounding-to-nearest ties-to-even (https://en.wikipedia.org/wiki/Rounding#Round_half_to_even). Clamping range according to data type:

- FP8: [-448, 448]
- INT4: [-8, 7]
- INT8: [-128, 127]

The first input (index 0) is the tensor to be quantized. The second (index 1) and third (index 2) are the scale and zero point respectively. `scale` and `zeroPt` should have identical dimensions, and rank lower or equal to 2.

The `zeroPt` tensor is optional, and if not set, will be assumed to be zero. Its data type must match the output data type. `zeroPt` must only contain zero-valued coefficients, because only symmetric quantization is supported. The `scale` value must be a scalar for per-tensor quantization, a 1D tensor for per-channel quantization, or the same rank as the input tensor for block quantization. All `scale` coefficients must have strictly positive values. The size of the 1D `scale` tensor must match the size of the quantization axis. For block quantization, the shape of `scale` tensor must match the shape of the input, except for the blocking dimension (the last or second to last dimension). The size of `zeroPt` must match the size of `scale`.

The subgraph which terminates with the `zeroPt` tensor must be a build-time constant containing only zeros. The output type, if constrained, must be constrained to DataType::kINT8, DataType::kFP8, DataType::kINT4 or DataType::kFP4. The input type, if constrained, must be constrained to DataType::kFLOAT, DataType::kHALF, or DataType::kBF16. The output size is the same as the input size. The quantization axis is in reference to the input tensor's dimensions.

IQuantizeLayer supports DataType::kFLOAT, DataType::kHALF, or DataType::kBF16 precision and will default to DataType::kFLOAT precision during instantiation. For strongly typed networks, if the scale data type is DataType::kHALF or DataType::kBF16, it must match the input data type. For MXFP8 quantization, the `scale` data type must be DataType::kE8M0.

IQuantizeLayer supports DataType::kINT8, DataType::kFP8, DataType::kINT4 or DataType::kFP4 output.

As an example of the operation of this layer, imagine a 4D NCHW activation input which can be quantized using a single scale coefficient (referred to as per-tensor quantization):

```
For each n in N:
    For each c in C:
        For each h in H:
            For each w in W:
                output[n,c,h,w] = clamp(round(input[n,c,h,w] / scale) + zeroPt)
```

Per-channel quantization is supported only for weight inputs. Thus, Activations cannot be quantized per-channel. As an example of per-channel operation, imagine a 4D KCRS weights input and K (dimension 0) as the quantization axis. The scale is an array of coefficients, and must have the same size as the quantization axis.

```
For each k in K:
    For each c in C:
        For each r in R:
            For each s in S:
                output[k,c,r,s] = clamp(round(input[k,c,r,s] / scale[k]) + zeroPt[k])
```

Block quantization is supported for input types DataType::kFP4, DataType::kFP8 and DataType::kINT4. As an example of blocked operation, imagine a 2D RS input with R (dimension 0) as the blocking axis and B as the block size. The scale is a 2D array of coefficients, with dimensions (R//B, S).

```
For each r in R:
    For each s in S:
        output[r,s] = clamp(round(input[r,s] / scale[r//B, s]) + zeroPt[r//B, s])
```

Note

Only symmetric quantization is supported.

Currently the only allowed build-time constant `zeroPt` subgraphs are:

1. Constant -> Quantize
2. Constant -> Cast -> Quantize

The input tensor for this layer must not be a scalar.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.177.2 Constructor & Destructor Documentation

9.177.2.1 ~IQuantizeLayer()

`nvinfer1::IQuantizeLayer::~IQuantizeLayer ( ) [inline], [protected], [pure virtual], [default], [noexcept]`

9.177.3 Member Function Documentation

9.177.3.1 getAxis()

`int32_t nvinfer1::IQuantizeLayer::getAxis ( ) const [inline], [noexcept]`

Get the quantization axis.

Returns

axis parameter set by setAxis(). The return value is the index of the quantization axis in the input tensor's dimensions. A value of -1 indicates per-tensor quantization. The default value is -1.

9.177.3.2 getBlockShape()

```
TRT_NODISCARD Dims nvinfer1::IQuantizeLayer::getBlockShape ( ) const  [inline], [noexcept]
```

Get the shape of the quantization block.

The default value is empty Dims.

See also

setBlockShape()

9.177.3.3 getToType()

```
DataType nvinfer1::IQuantizeLayer::getToType ( ) const  [inline], [noexcept]
```

Return the Quantize layer output type.

Returns

toType parameter set during layer creation or by setToType(). The return value is the output type of the quantize layer. The default value is DataType::kINT8.

9.177.3.4 setAxis()

```
void nvinfer1::IQuantizeLayer::setAxis (
            int32_t axis )  [inline], [noexcept]
```

Set the quantization axis.

Set the index of the quantization axis (with reference to the input tensor's dimensions). The axis must be a valid axis if the scale tensor has more than one coefficient. The axis value is used only for per-axis (per-channel) quantization.

9.177.3.5 setBlockShape()

```
bool nvinfer1::IQuantizeLayer::setBlockShape (
            Dims const & blockShape )  [inline], [noexcept]
```

Set the shape of the quantization block.

See also

getBlockShape() Allowed values are positive values and -1 which denotes a fully blocked dimension. Returns true if the block shape was set successfully, false if the block shape is invalid. The default value is empty Dims.

9.177.3.6 setToType()

```
void nvinfer1::IQuantizeLayer::setToType (
            DataType toType )  [inline], [noexcept]
```

Set the Quantize layer output type.

Parameters

toType	The DataType of the output tensor.

Set the output type of the quantize layer. Valid values are DataType::kINT8, DataType::kFP8, DataType::kINT4 and DataType::kFP4. If the network is strongly typed, setToType must be used to set the output type, and use of setOutputType is an error. Otherwise, types passed to setOutputType and setToType must be the same.

See also

NetworkDefinitionCreationFlag::kSTRONGLY_TYPED

9.177.4 Member Data Documentation

9.177.4.1 mImpl

```
apiv::VQuantizeLayer* nvinfer1::IQuantizeLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

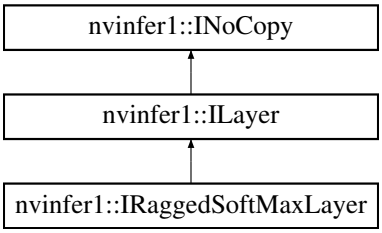
- NvInfer.h

9.178 nvinfer1::IRaggedSoftMaxLayer Class Reference

A RaggedSoftmax layer in a network definition.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IRaggedSoftMaxLayer:



Protected Member Functions

- virtual ~IRaggedSoftMaxLayer () noexcept=0

Protected Attributes

- apiv::VRaggedSoftMaxLayer * mImpl

Additional Inherited Members

9.178.1 Detailed Description

A RaggedSoftmax layer in a network definition.

This layer takes a ZxS input tensor and an additional Zx1 bounds tensor holding the lengths of the Z sequences.

This layer computes a softmax across each of the Z sequences.

The output tensor is of the same size as the input tensor.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.178.2 Constructor & Destructor Documentation

9.178.2.1 ~IRaggedSoftMaxLayer()

```
nvinfer1::IRaggedSoftMaxLayer::~IRaggedSoftMaxLayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.178.3 Member Data Documentation

9.178.3.1 mImpl

```
apiv::VRaggedSoftMaxLayer* nvinfer1::IRaggedSoftMaxLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

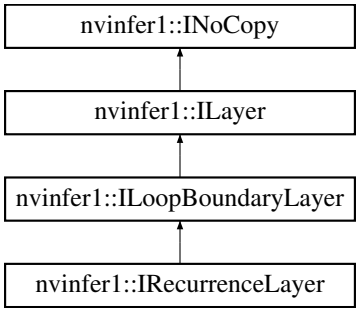
- NvInfer.h

9.179 nvinfer1::IRecurrenceLayer Class Reference

A recurrence layer in a network definition.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IRecurrenceLayer:



Public Member Functions

- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~IRecurrenceLayer () noexcept=0

Protected Attributes

- apiv::VRecurrenceLayer * mImpl

9.179.1 Detailed Description

A recurrence layer in a network definition.

The recurrence layer allows a loop iteration to compute a result from a value computed in the previous iteration.

9.179.2 Constructor & Destructor Documentation

9.179.2.1 ~IRecurrenceLayer()

```
nvinfer1::IRecurrenceLayer::~IRecurrenceLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.179.3 Member Function Documentation

9.179.3.1 setInput()

```
void nvinfer1::ILayer::setInput (
            int32_t index,
            ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor Sets the input tensor for the given index.

For a recurrence layer, the values 0 and 1 are valid. The indices are as follows:

- 0: The initial value of the output tensor. The value must come from outside the loop.
- 1: The next value of the output tensor. The value usually comes from inside the loop, and must have the same dimensions as input 0.

If this function is called with the value 1, then the function getNbInputs() changes from returning 1 to 2.

9.179.4 Member Data Documentation

9.179.4.1 mImpl

```
apiv::VRecurrenceLayer* nvinfer1::IRecurrenceLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

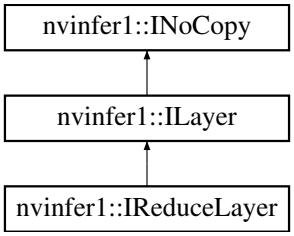
- NvInfer.h

9.180 nvinfer1::IReduceLayer Class Reference

Layer that represents a reduction across a non-bool tensor.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IReduceLayer:



Public Member Functions

- void setOperation (ReduceOperation op) noexcept

  Set the reduce operation for the layer.
- ReduceOperation getOperation () const noexcept

  Get the reduce operation for the layer.
- void setReduceAxes (uint32_t reduceAxes) noexcept

  Set the axes over which to reduce.
- uint32_t getReduceAxes () const noexcept

  Get the axes over which to reduce for the layer.
- void setKeepDimensions (bool keepDimensions) noexcept

  Set the boolean that specifies whether or not to keep the reduced dimensions for the layer.
- bool getKeepDimensions () const noexcept

  Get the boolean that specifies whether or not to keep the reduced dimensions for the layer.

Protected Member Functions

- virtual ~IReduceLayer () noexcept=0

Protected Attributes

- apiv::VReduceLayer * mImpl

9.180.1 Detailed Description

Layer that represents a reduction across a non-bool tensor.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.180.2 Constructor & Destructor Documentation

9.180.2.1 ~IReduceLayer()

```
nvinfer1::IReduceLayer::~IReduceLayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.180.3 Member Function Documentation

9.180.3.1 getKeepDimensions()

```
bool nvinfer1::IReduceLayer::getKeepDimensions ( ) const  [inline], [noexcept]
```

Get the boolean that specifies whether or not to keep the reduced dimensions for the layer.

See also

setKeepDimensions

9.180.3.2 getOperation()

```
ReduceOperation nvinfer1::IReduceLayer::getOperation ( ) const  [inline], [noexcept]
```

Get the reduce operation for the layer.

See also

setOperation(), ReduceOperation

9.180.3.3 getReduceAxes()

```
uint32_t nvinfer1::IReduceLayer::getReduceAxes ( ) const  [inline], [noexcept]
```

Get the axes over which to reduce for the layer.

See also

setReduceAxes

9.180.3.4 setKeepDimensions()

```
void nvinfer1::IReduceLayer::setKeepDimensions (
            bool keepDimensions )  [inline], [noexcept]
```

Set the boolean that specifies whether or not to keep the reduced dimensions for the layer.

See also

getKeepDimensions

9.180.3.5 setOperation()

```
void nvinfer1::IReduceLayer::setOperation (
            ReduceOperation op )  [inline], [noexcept]
```

Set the reduce operation for the layer.

See also

getOperation(), ReduceOperation

9.180.3.6 setReduceAxes()

```
void nvinfer1::IReduceLayer::setReduceAxes (
            uint32_t reduceAxes )  [inline], [noexcept]
```

Set the axes over which to reduce.

See also

getReduceAxes

9.180.4 Member Data Documentation

9.180.4.1 mImpl

```
apiv::VReduceLayer* nvinfer1::IReduceLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

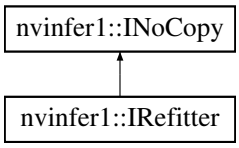
- NvInfer.h

9.181 nvinfer1::IRefitter Class Reference

Updates weights in an engine.

```
#include <NvInferRuntime.h>
```

Inheritance diagram for nvinfer1::IRefitter:



Public Member Functions

- virtual ~IRefitter () noexcept=0
- bool setWeights (char const *layerName, WeightsRole role, Weights weights) noexcept

  Specify new weights for a layer of given name. Returns true on success, or false if new weights are rejected. Possible reasons for rejection are:
- bool refitCudaEngine () noexcept

  Refits associated engine.
- int32_t getMissing (int32_t size, char const **layerNames, WeightsRole *roles) noexcept

  Get description of missing weights.
- int32_t getAll (int32_t size, char const **layerNames, WeightsRole *roles) noexcept

  Get description of all weights that could be refit.
- void setErrorRecorder (IErrorRecorder *recorder) noexcept

  Set the ErrorRecorder for this interface.
- IErrorRecorder * getErrorRecorder () const noexcept

  Get the ErrorRecorder assigned to this interface.
- bool setNamedWeights (char const *name, Weights weights) noexcept

  Specify new weights of given name.
- int32_t getMissingWeights (int32_t size, char const **weightsNames) noexcept

Get names of missing weights.

- int32_t getAllWeights (int32_t size, char const **weightsNames) noexcept

  Get names of all weights that could be refit.

- ILogger * getLogger () const noexcept

  get the logger with which the refitter was created

- bool setMaxThreads (int32_t maxThreads) noexcept

  Set the maximum number of threads.

- int32_t getMaxThreads () const noexcept

  get the maximum number of threads that can be used by the refitter.

- bool setNamedWeights (char const *name, Weights weights, TensorLocation location) noexcept

  Specify new weights on a specified device of given name.

- Weights getNamedWeights (char const *weightsName) const noexcept

  Get weights associated with the given name.

- TensorLocation getWeightsLocation (char const *weightsName) const noexcept

  Get location for the weights associated with the given name.

- bool unsetNamedWeights (char const *weightsName) noexcept

  Unset weights associated with the given name.

- void setWeightsValidation (bool weightsValidation) noexcept

  Set whether to validate weights during refitting.

- bool getWeightsValidation () const noexcept

  Get whether to validate weights values during refitting.

- bool refitCudaEngineAsync (cudaStream_t stream) noexcept

  Enqueue weights refitting of the associated engine on the given stream.

- Weights getWeightsPrototype (char const *weightsName) const noexcept

  Get the Weights prototype associated with the given name.

Protected Attributes

- apiv::VRefitter * mImpl

Additional Inherited Members

9.181.1 Detailed Description

Updates weights in an engine.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.181.2 Constructor & Destructor Documentation

9.181.2.1 ~IRefitter()

```
nvinfer1::IRefitter::~IRefitter ( )  [inline], [pure virtual], [default], [noexcept]
```

9.181.3 Member Function Documentation

9.181.3.1 getAll()

```
int32_t nvinfer1::IRefitter::getAll (
            int32_t size,
            char const ** layerNames,
            WeightsRole * roles )  [inline], [noexcept]
```

Get description of all weights that could be refit.

Parameters

size	The number of items that can be safely written to a non-null layerNames or roles.
layerNames	Where to write the layer names.
roles	Where to write the weights roles.

Returns

The number of Weights that could be refit.

If layerNames!=nullptr, each written pointer points to a string owned by the engine being refit, and becomes invalid when the engine is destroyed.

9.181.3.2 getAllWeights()

```
int32_t nvinfer1::IRefitter::getAllWeights (
            int32_t size,
            char const ** weightsNames )  [inline], [noexcept]
```

Get names of all weights that could be refit.

Parameters

size	The number of weights names that can be safely written to.
weightsNames	The names of the weights to be updated, or nullptr for unnamed weights.

Returns

The number of Weights that could be refit.

If layerNames!=nullptr, each written pointer points to a string owned by the engine being refit, and becomes invalid when the engine is destroyed.

9.181.3.3 getErrorRecorder()

```
IErrorRecorder * nvinfer1::IRefitter::getErrorRecorder ( ) const  [inline], [noexcept]
```

Get the ErrorRecorder assigned to this interface.

Retrieves the assigned error recorder object for the given class. A nullptr will be returned if an error handler has not been set.

Returns

A pointer to the IErrorRecorder object that has been registered.

See also

setErrorRecorder()

9.181.3.4 getLogger()

```
ILogger * nvinfer1::IRefitter::getLogger ( ) const  [inline], [noexcept]
```

get the logger with which the refitter was created

Returns

the logger

9.181.3.5 getMaxThreads()

```
int32_t nvinfer1::IRefitter::getMaxThreads ( ) const  [inline], [noexcept]
```

get the maximum number of threads that can be used by the refitter.

Retrieves the maximum number of threads that can be used by the refitter.

Returns

The maximum number of threads that can be used by the refitter.

See also

setMaxThreads()

9.181.3.6 getMissing()

```
int32_t nvinfer1::IRefitter::getMissing (
            int32_t size,
            char const ** layerNames,
            WeightsRole * roles )  [inline], [noexcept]
```

Get description of missing weights.

For example, if some Weights have been set, but the engine was optimized in a way that combines weights, any unsupplied Weights in the combination are considered missing.

Parameters

size	The number of items that can be safely written to a non-null layerNames or roles.
layerNames	Where to write the layer names.
roles	Where to write the weights roles.

Returns

The number of missing Weights.

If layerNames!=nullptr, each written pointer points to a string owned by the engine being refit, and becomes invalid when the engine is destroyed.

9.181.3.7 getMissingWeights()

```
int32_t nvinfer1::IRefitter::getMissingWeights (
            int32_t size,
            char const ** weightsNames )  [inline], [noexcept]
```

Get names of missing weights.

For example, if some Weights have been set, but the engine was optimized in a way that combines weights, any unsupplied Weights in the combination are considered missing.

Parameters

size	The number of weights names that can be safely written to.
weightsNames	The names of the weights to be updated, or nullptr for unnamed weights.

Returns

The number of missing Weights.

If layerNames!=nullptr, each written pointer points to a string owned by the engine being refit, and becomes invalid when the engine is destroyed.

9.181.3.8 getNamedWeights()

```
Weights nvinfer1::IRefitter::getNamedWeights (
            char const * weightsName ) const  [inline], [noexcept]
```

Get weights associated with the given name.

Parameters

weightsName	The name of the weights to be refitted.

Returns

Weights associated with the given name.

If the weights were never set, returns null weights and reports an error to the refitter errorRecorder.

Warning

The string weightsName must be null-terminated, and be at most 4096 bytes including the terminator.

9.181.3.9 getWeightsLocation()

```
TensorLocation nvinfer1::IRefitter::getWeightsLocation (
            char const * weightsName ) const  [inline], [noexcept]
```

Get location for the weights associated with the given name.

Parameters

weightsName	The name of the weights to be refitted.

Returns

Location for the weights associated with the given name.

If the weights were never set, returns TensorLocation::kHOST and reports an error to the refitter errorRecorder.

Warning

The string weightsName must be null-terminated, and be at most 4096 bytes including the terminator.

9.181.3.10 getWeightsPrototype()

```
Weights nvinfer1::IRefitter::getWeightsPrototype (
            char const * weightsName ) const  [inline], [noexcept]
```

Get the Weights prototype associated with the given name.

Parameters

weightsName	The name of the weights to be refitted.

Returns

Weights prototype associated with the given name.

The type and count of weights prototype is the same as weights used for engine building. The values property is nullptr for weights prototypes. The count of the weights prototype is -1 when the name of the weights is nullptr or does not correspond to any refittable weights.

Warning

The string weightsName must be null-terminated, and be at most 4096 bytes including the terminator.

9.181.3.11 getWeightsValidation()

```
bool nvinfer1::IRefitter::getWeightsValidation ( ) const  [inline], [noexcept]
```

Get whether to validate weights values during refitting.

9.181.3.12 refitCudaEngine()

```
bool nvinfer1::IRefitter::refitCudaEngine ( )  [inline], [noexcept]
```

Refits associated engine.

Returns

True on success, or false if new weights validation fails or getMissingWeights() != 0 before the call. If false is returned, a subset of weights may have been refitted.

The behavior is undefined if the engine has pending enqueued work. Provided weights on CPU or GPU can be unset and released, or updated after refitCudaEngine returns.

IExecutionContexts associated with the engine remain valid for use afterwards. There is no need to set the same weights repeatedly for multiple refit calls as the weights memory can be updated directly instead.

9.181.3.13 refitCudaEngineAsync()

```
bool nvinfer1::IRefitter::refitCudaEngineAsync (
            cudaStream_t stream )  [inline], [noexcept]
```

Enqueue weights refitting of the associated engine on the given stream.

Parameters

stream	The stream to enqueue the weights updating task.

Returns

True on success, or false if new weights validation fails or getMissingWeights() != 0 before the call. If false is returned, a subset of weights may have been refitted.

The behavior is undefined if the engine has pending enqueued work on a different stream from the provided one. Provided weights on CPU can be unset and released, or updated after refitCudaEngineAsync returns. Freeing or updating of the provided weights on GPU can be enqueued on the same stream after refitCudaEngineAsync returns.

IExecutionContexts associated with the engine remain valid for use afterwards. There is no need to set the same weights repeatedly for multiple refit calls as the weights memory can be updated directly instead. The weights updating task should use the same stream as the one used for the refit call.

9.181.3.14 setErrorRecorder()

```
void nvinfer1::IRefitter::setErrorRecorder (
             IErrorRecorder * recorder )  [inline], [noexcept]
```

Set the ErrorRecorder for this interface.

Assigns the ErrorRecorder to this interface. The ErrorRecorder will track all errors during execution. This function will call incRefCount of the registered ErrorRecorder at least once. Setting recorder to nullptr unregisters the recorder with the interface, resulting in a call to decRefCount if a recorder has been registered.

If an error recorder is not set, messages will be sent to the global log stream.

Parameters

recorder	The error recorder to register with this interface.

See also

getErrorRecorder()

9.181.3.15 setMaxThreads()

```
bool nvinfer1::IRefitter::setMaxThreads (
             int32_t maxThreads )  [inline], [noexcept]
```

Set the maximum number of threads.

Parameters

maxThreads	The maximum number of threads that can be used by the refitter.

Returns

True if successful, false otherwise.

The default value is 1 and includes the current thread. A value greater than 1 permits TensorRT to use multi-threaded algorithms. A value less than 1 triggers a kINVALID_ARGUMENT error.

9.181.3.16 setNamedWeights() [1/2]

```
bool nvinfer1::IRefitter::setNamedWeights (
            char const * name,
            Weights weights )  [inline], [noexcept]
```

Specify new weights of given name.

Parameters

name	The name of the weights to be refit.
weights	The new weights to associate with the name.

Returns true on success, or false if new weights are rejected. Possible reasons for rejection are:

- The name of weights is nullptr or does not correspond to any refittable weights.
- The count of the weights is inconsistent with the count returned from calling getWeightsPrototype() with the same name.
- The type of the weights is inconsistent with the type returned from calling getWeightsPrototype() with the same name.

Modifying the weights before method refitCudaEngine or refitCudaEngineAsync returns will result in undefined behavior.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

9.181.3.17 setNamedWeights() [2/2]

```
bool nvinfer1::IRefitter::setNamedWeights (
            char const * name,
            Weights weights,
            TensorLocation location )  [inline], [noexcept]
```

Specify new weights on a specified device of given name.

Parameters

name	The name of the weights to be refitted.
weights	The new weights on the specified device.
location	The location (host vs. device) of the new weights.

Returns

True on success, or false if new weights are rejected. Possible reasons for rejection are:

- The name of the weights is nullptr or does not correspond to any refittable weights.
- The count of the weights is inconsistent with the count returned from calling getWeightsPrototype() with the same name.
- The type of the weights is inconsistent with the type returned from calling getWeightsPrototype() with the same name.

It is allowed to provide some weights on CPU and others on GPU. Modifying the weights before the method refitCudaEngine() or refitCudaEngineAsync() completes will result in undefined behavior.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

9.181.3.18 setWeights()

```
bool nvinfer1::IRefitter::setWeights (
            char const * layerName,
            WeightsRole role,
            Weights weights )  [inline], [noexcept]
```

Specify new weights for a layer of given name. Returns true on success, or false if new weights are rejected. Possible reasons for rejection are:

- There is no such layer by that name.
- The layer does not have weights with the specified role.
- The count of weights is inconsistent with the layer's original specification.
- The type of weights is inconsistent with the layer's original specification.

Modifying the weights before method refitCudaEngine or refitCudaEngineAsync returns will result in undefined behavior.

Warning

The string layerName must be null-terminated, and be at most 4096 bytes including the terminator.

9.181.3.19 setWeightsValidation()

```
void nvinfer1::IRefitter::setWeightsValidation (
            bool weightsValidation )  [inline], [noexcept]
```

Set whether to validate weights during refitting.

Parameters

weightsValidation	Indicate whether to validate weights during refitting.

When set to true, TensorRT will validate weights during FP32 to FP16/BF16 weights conversions or sparsifying weights in the refit call. If provided weights are not proper for some weights transformations, TensorRT will issue a warning and continue the transformation for minor issues (such as overflow during narrowing conversion), or issue an error and stop the refitting process for severe issues (such as sparsifying dense weights). By default the flag is true. Set the flag to false for faster refitting performance.

9.181.3.20 unsetNamedWeights()

```
bool nvinfer1::IRefitter::unsetNamedWeights (
            char const * weightsName )  [inline], [noexcept]
```

Unset weights associated with the given name.

Parameters

weightsName	The name of the weights to be refitted.

Returns

False if the weights were never set, returns true otherwise.

Unset weights before releasing them.

Warning

The string weightsName must be null-terminated, and be at most 4096 bytes including the terminator.

9.181.4 Member Data Documentation

9.181.4.1 mImpl

```
apiv::VRefitter* nvinfer1::IRefitter::mImpl  [protected]
```

The documentation for this class was generated from the following file:

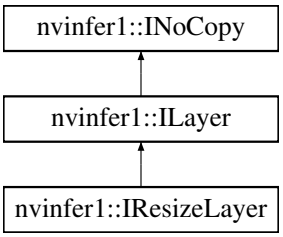
- NvInferRuntime.h

9.182 nvinfer1::IResizeLayer Class Reference

A resize layer in a network definition.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IResizeLayer:



Public Member Functions

- void setOutputDimensions (Dims const &dimensions) noexcept

  Set the output dimensions.
- Dims getOutputDimensions () const noexcept

  Get the output dimensions.
- void setScales (float const *scales, int32_t nbScales) noexcept

  Set the resize scales.
- int32_t getScales (int32_t size, float *scales) const noexcept

  Copies resize scales to scales[0, ..., nbScales-1], where nbScales is the number of scales that were set.
- void setResizeMode (InterpolationMode interpolationMode) noexcept

  Set resize mode for an input tensor.
- InterpolationMode getResizeMode () const noexcept

  Get resize mode for an input tensor.
- void setCoordinateTransformation (ResizeCoordinateTransformation coordTransform) noexcept

  Set coordinate transformation function.
- ResizeCoordinateTransformation getCoordinateTransformation () const noexcept

  Get coordinate transformation function.
- void setSelectorForSinglePixel (ResizeSelector selector) noexcept

  Set coordinate selector function when resized to single pixel.
- ResizeSelector getSelectorForSinglePixel () const noexcept

Get the coordinate selector function when resized to single pixel.

- void setNearestRounding (ResizeRoundMode value) noexcept

  Set rounding mode for nearest neighbor resize.
- ResizeRoundMode getNearestRounding () const noexcept

  Get rounding mode for nearest neighbor resize.
- void setCubicCoeff (float A) noexcept

  Set the coefficient 'A' used in cubic interpolation.
- float getCubicCoeff () const noexcept

  Get the coefficient 'A' used in cubic interpolation.
- void setExcludeOutside (bool excludeFlag) noexcept

  Set the state for excluding outside pixels.
- bool getExcludeOutside () const noexcept

  Get the state for excluding outside pixels.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~IResizeLayer () noexcept=0

Protected Attributes

- apiv::VResizeLayer * mImpl

9.182.1 Detailed Description

A resize layer in a network definition.

Resize layer can be used for resizing a N-D tensor.

Resize layer currently supports the following configurations:

- InterpolationMode::kNEAREST - resizes last `m` dimensions of N-D, where 0 < m <= min(8, N) and N > 0
- InterpolationMode::kLINEAR - resizes last `m` dimensions of N-D, where 0 < m <= min(3, N) and N > 0

Default resize mode is InterpolationMode::kNEAREST.

The coordinates in the output tensor are mapped to coordinates in the input tensor using a function set by calling setCoordinateTransformation(). The default for all InterpolationMode settings (nearest, linear, bilinear, etc.) is ResizeCoordinateTransformation::kASYMMETRIC.

The resize layer provides two ways to resize tensor dimensions.

- Set output dimensions directly. It can be done for static as well as dynamic resize layer. Static resize layer requires output dimensions to be known at build-time. Dynamic resize layer requires output dimensions to be set as one of the input tensors.

- Set scales for resize. Each output dimension is calculated as floor(input dimension ∗ scale). Only static resize layer allows setting scales where the scales are known at build-time.

If executing this layer on DLA, the following combinations of parameters are supported:

- In kNEAREST mode:
  - (ResizeCoordinateTransformation::kASYMMETRIC, ResizeSelector::kFORMULA, ResizeRoundMode::kFLOOR)
  - (ResizeCoordinateTransformation::kHALF_PIXEL, ResizeSelector::kFORMULA, ResizeRoundMode::kHALF_DOWN)
  - (ResizeCoordinateTransformation::kHALF_PIXEL, ResizeSelector::kFORMULA, ResizeRoundMode::kHALF_UP)
- In kLINEAR mode:
  - (ResizeCoordinateTransformation::kHALF_PIXEL, ResizeSelector::kFORMULA)
  - (ResizeCoordinateTransformation::kHALF_PIXEL, ResizeSelector::kUPPER)

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.182.2 Constructor & Destructor Documentation

9.182.2.1 ~IResizeLayer()

```
nvinfer1::IResizeLayer::~IResizeLayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.182.3 Member Function Documentation

9.182.3.1 getCoordinateTransformation()

```
ResizeCoordinateTransformation nvinfer1::IResizeLayer::getCoordinateTransformation ( ) const  [inline],
[noexcept]
```

Get coordinate transformation function.

Returns

The coordinate transformation function.

9.182.3.2 getCubicCoeff()

```
float nvinfer1::IResizeLayer::getCubicCoeff ( ) const  [inline], [noexcept]
```

Get the coefficient 'A' used in cubic interpolation.

See also

setCubicCoeff()

9.182.3.3 getExcludeOutside()

```
bool nvinfer1::IResizeLayer::getExcludeOutside ( ) const  [inline], [noexcept]
```

Get the state for excluding outside pixels.

See also

setExcludeOutside()

9.182.3.4 getNearestRounding()

```
ResizeRoundMode nvinfer1::IResizeLayer::getNearestRounding ( ) const  [inline], [noexcept]
```

Get rounding mode for nearest neighbor resize.

Returns

The rounding mode.

9.182.3.5 getOutputDimensions()

```
Dims nvinfer1::IResizeLayer::getOutputDimensions ( ) const  [inline], [noexcept]
```

Get the output dimensions.

Returns

The output dimensions.

9.182.3.6 getResizeMode()

```
InterpolationMode nvinfer1::IResizeLayer::getResizeMode ( ) const  [inline], [noexcept]
```

Get resize mode for an input tensor.

Returns

The resize mode.

9.182.3.7 getScales()

```
int32_t nvinfer1::IResizeLayer::getScales (
            int32_t size,
            float * scales ) const  [inline], [noexcept]
```

Copies resize scales to scales[0, ..., nbScales-1], where nbScales is the number of scales that were set.

Parameters

size	The number of scales to get. If size != nbScales, no scales will be copied.
scales	Pointer to where to copy the scales. Scales will be copied only if size == nbScales and scales != nullptr.

In case the size is not known consider using size = 0 and scales = nullptr. This method will return the number of resize scales.

Returns

The number of resize scales i.e. nbScales if scales were set. Return -1 in case no scales were set or resize layer is used in dynamic mode.

9.182.3.8 getSelectorForSinglePixel()

```
ResizeSelector nvinfer1::IResizeLayer::getSelectorForSinglePixel ( ) const  [inline], [noexcept]
```

Get the coordinate selector function when resized to single pixel.

Returns

The selector function.

9.182.3.9 setCoordinateTransformation()

```
void nvinfer1::IResizeLayer::setCoordinateTransformation (
            ResizeCoordinateTransformation coordTransform )  [inline], [noexcept]
```

Set coordinate transformation function.

The function maps a coordinate in the output tensor to a coordinate in the input tensor.

Default function is ResizeCoordinateTransformation::kASYMMETRIC.

See also

ResizeCoordinateTransformation

9.182.3.10 setCubicCoeff()

```
void nvinfer1::IResizeLayer::setCubicCoeff (
            float A )  [inline], [noexcept]
```

Set the coefficient 'A' used in cubic interpolation.

Cubic uses the coefficient 'A' to calculate the weight of input pixels:

```
x := The relative distance between the sampled pixels and the input coordinates.

weight(x) := for |x| <= 1, ((A + 2) * x - (A + 3)) * x * x + 1,
             for 1 < |x| < 2, ((A * x - 5 * A) * x + 8 * A) * x - 4 * A,
             others 0;
```

This attribute is valid only if ”resize mode” is ”cubic”.

The default value is -0.75.

9.182.3.11 setExcludeOutside()

```
void nvinfer1::IResizeLayer::setExcludeOutside (
            bool excludeFlag )  [inline], [noexcept]
```

Set the state for excluding outside pixels.

If set to true, the weight of sampling locations outside the input tensor will be set to false, and the weight will be renormalized so that their sum is 1.0.

The default value is false.

9.182.3.12 setInput()

```
void nvinfer1::ILayer::setInput (
            int32_t index,
            ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor.

Sets the input tensor for the given index. The index must be 0 for a static resize layer. A static resize layer is converted to a dynamic resize layer by calling setInput with an index 1. A dynamic resize layer cannot be converted back to a static resize layer.

For a dynamic resize layer, the values 0 and 1 are valid. The indices in the dynamic case are as follows:

- 0: Execution tensor to be resized.
- 1: The output dimensions, as a 1D tensor of type Int32 or Int64.

If this function is called with the value 1, then the function getNbInputs() changes from returning 1 to 2.

9.182.3.13 setNearestRounding()

```
void nvinfer1::IResizeLayer::setNearestRounding (
             ResizeRoundMode value )  [inline], [noexcept]
```

Set rounding mode for nearest neighbor resize.

This value is used for nearest neighbor interpolation rounding. It is applied after coordinate transformation.

Default is kFLOOR.

See also

ResizeRoundMode

9.182.3.14 setOutputDimensions()

```
void nvinfer1::IResizeLayer::setOutputDimensions (
             Dims const & dimensions )  [inline], [noexcept]
```

Set the output dimensions.

Parameters

dimensions	The output dimensions. Number of output dimensions must be the same as the number of input dimensions.

If executing this layer on DLA, setOutputDimensions() is not supported.

If there is a second input, i.e. resize layer is dynamic, calling setOutputDimensions() is an error and does not update the dimensions.

Output dimensions can be specified directly, or via scale factors relative to input dimensions. Scales for resize can be provided using setScales().

See also

setScales

getOutputDimensions

9.182.3.15 setResizeMode()

```
void nvinfer1::IResizeLayer::setResizeMode (
            InterpolationMode interpolationMode )  [inline], [noexcept]
```

Set resize mode for an input tensor.

Supported resize modes are Nearest Neighbor and Linear.

See also

InterpolationMode

9.182.3.16 setScales()

```
void nvinfer1::IResizeLayer::setScales (
            float const * scales,
            int32_t nbScales )  [inline], [noexcept]
```

Set the resize scales.

Parameters

scales	An array of resize scales.
nbScales	Number of scales. Number of scales must be equal to the number of input dimensions.

If executing this layer on DLA, there are three restrictions: 1) nbScales has to be exactly 4. 2) the first two elements in scales need to be exactly 1 (for unchanged batch and channel dimensions). 3) The last two elements in scales, representing the scale values along height and width dimensions, respectively, need to be integer values in the range of [1, 32] for kNEAREST mode and [1, 4] for kLINEAR. Example of DLA-supported scales: {1, 1, 2, 2}.

If there is a second input, i.e. resize layer is dynamic, calling setScales() is an error and does not update the scales.

Output dimensions are calculated as follows: outputDims[i] = floor(inputDims[i] ∗ scales[i])

Output dimensions can be specified directly, or via scale factors relative to input dimensions. Output dimensions can be provided directly using setOutputDimensions().

See also

setOutputDimensions

getScales

9.182.3.17 setSelectorForSinglePixel()

```
void nvinfer1::IResizeLayer::setSelectorForSinglePixel (
            ResizeSelector selector )  [inline], [noexcept]
```

Set coordinate selector function when resized to single pixel.

When resize to single pixel image, use this function to decide how to map the coordinate in the original image.

Default is ResizeSelector::kFORMULA.

See also

ResizeSelector

9.182.4 Member Data Documentation

9.182.4.1 mImpl

```
apiv::VResizeLayer* nvinfer1::IResizeLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

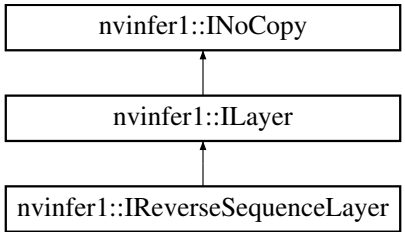
- NvInfer.h

9.183 nvinfer1::IReverseSequenceLayer Class Reference

A ReverseSequence layer in a network definition.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IReverseSequenceLayer:



Public Member Functions

- void setBatchAxis (int32_t batchAxis) noexcept

  Set the batch axis. Default is 1.
- int32_t getBatchAxis () const noexcept

  Return the batch axis. Return 1 if no batch axis was set.
- void setSequenceAxis (int32_t sequenceAxis) noexcept

  Set the sequence axis. Default is 0.
- int32_t getSequenceAxis () const noexcept

  Return the sequence axis. Return 0 if no sequence axis was set.

Protected Member Functions

- virtual ~IReverseSequenceLayer () noexcept=0

Protected Attributes

- apiv::VReverseSequenceLayer * mImpl

9.183.1 Detailed Description

A ReverseSequence layer in a network definition.

This layer performs batch-wise reversal, which slices the input tensor along the axis batchAxis. For the i-th slice, the operation reverses the first N elements, specified by the corresponding i-th value in sequenceLens, along sequenceAxis and keeps the remaining elements unchanged. The output tensor will have the same shape as the input tensor.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.183.2 Constructor & Destructor Documentation

9.183.2.1 ~IReverseSequenceLayer()

```
nvinfer1::IReverseSequenceLayer::~IReverseSequenceLayer ( )  [inline], [protected], [pure virtual],
[default], [noexcept]
```

9.183.3 Member Function Documentation

9.183.3.1 getBatchAxis()

```
int32_t nvinfer1::IReverseSequenceLayer::getBatchAxis ( ) const  [inline], [noexcept]
```

Return the batch axis. Return 1 if no batch axis was set.

See also

getBatchAxis()

9.183.3.2 getSequenceAxis()

```
int32_t nvinfer1::IReverseSequenceLayer::getSequenceAxis ( ) const  [inline], [noexcept]
```

Return the sequence axis. Return 0 if no sequence axis was set.

See also

getSequenceAxis()

9.183.3.3 setBatchAxis()

```
void nvinfer1::IReverseSequenceLayer::setBatchAxis (
            int32_t batchAxis )  [inline], [noexcept]
```

Set the batch axis. Default is 1.

batchAxis should be between zero (inclusive) and the rank of input (exclusive), and different from sequenceAxis. Otherwise, ErrorCode::kINVALID_ARGUMENT will be triggered.

See also

setBatchAxis()

9.183.3.4 setSequenceAxis()

```
void nvinfer1::IReverseSequenceLayer::setSequenceAxis (
            int32_t sequenceAxis )  [inline], [noexcept]
```

Set the sequence axis. Default is 0.

sequenceAxis should be between zero (inclusive) and the rank of input (exclusive), and different from batchAxis. Otherwise, ErrorCode::kINVALID_ARGUMENT will be triggered.

See also

setSequenceAxis()

9.183.4 Member Data Documentation

9.183.4.1 mImpl

```
apiv::VReverseSequenceLayer* nvinfer1::IReverseSequenceLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

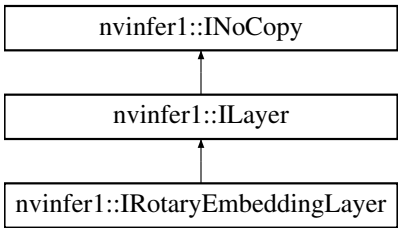
- NvInfer.h

9.184 nvinfer1::IRotaryEmbeddingLayer Class Reference

Layer that implements Rotary Position Embedding (RoPE) (https://arxiv.org/abs/2104.09864).

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IRotaryEmbeddingLayer:



Public Member Functions

- void setInterleaved (bool interleaved) noexcept

  Set whether the input is in interleaved format, i.e., whether the 2-d vectors rotated are taken from adjacent 2 elements in the hidden dimension. The default value is false.
- TRT_NODISCARD bool getInterleaved () const noexcept

  Get whether the input is in interleaved format. The default value is false.
- TRT_NODISCARD bool setRotaryEmbeddingDim (int32_t rotaryEmbeddingDim) noexcept

  Set the number of hidden dimensions participating in RoPE. The default value is 0, representing H, i.e., all hidden dimensions in each head. Must be non-negative and even.
- TRT_NODISCARD int32_t getRotaryEmbeddingDim () const noexcept

  Get the number of hidden dimensions participating in RoPE. The default value is 0, representing H, i.e., all hidden dimensions in each head.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~IRotaryEmbeddingLayer () noexcept=0

Protected Attributes

- apiv::VRotaryEmbeddingLayer * mImpl

9.184.1 Detailed Description

Layer that implements Rotary Position Embedding (RoPE) (https://arxiv.org/abs/2104.09864).

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.184.2 Constructor & Destructor Documentation

9.184.2.1 ~IRotaryEmbeddingLayer()

```
nvinfer1::IRotaryEmbeddingLayer::~IRotaryEmbeddingLayer ( )  [inline], [protected], [pure virtual],
[default], [noexcept]
```

9.184.3 Member Function Documentation

9.184.3.1 getInterleaved()

```
TRT_NODISCARD bool nvinfer1::IRotaryEmbeddingLayer::getInterleaved ( ) const  [inline], [noexcept]
```

Get whether the input is in interleaved format. The default value is false.

See also

setInterleaved

9.184.3.2 getRotaryEmbeddingDim()

```
TRT_NODISCARD int32_t nvinfer1::IRotaryEmbeddingLayer::getRotaryEmbeddingDim ( ) const  [inline],
[noexcept]
```

Get the number of hidden dimensions participating in RoPE. The default value is 0, representing H, i.e., all hidden dimensions in each head.

See also

setRotaryEmbeddingDim

9.184.3.3 setInput()

```
void nvinfer1::ILayer::setInput (
            int32_t index,
            ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor

The indices are as follows:

Input 0 is the input activation tensor. Input 1 is the cosine cache tensor. Input 2 is the sine cache tensor. Input 3 (optional) is the positionIds tensor, which is used for indexing into the cosine and sine caches.

9.184.3.4 setInterleaved()

```
void nvinfer1::IRotaryEmbeddingLayer::setInterleaved (
            bool interleaved )  [inline], [noexcept]
```

Set whether the input is in interleaved format, i.e., whether the 2-d vectors rotated are taken from adjacent 2 elements in the hidden dimension. The default value is false.

See also

getInterleaved

9.184.3.5 setRotaryEmbeddingDim()

```
TRT_NODISCARD bool nvinfer1::IRotaryEmbeddingLayer::setRotaryEmbeddingDim (
            int32_t rotaryEmbeddingDim )  [inline], [noexcept]
```

Set the number of hidden dimensions participating in RoPE. The default value is 0, representing H, i.e., all hidden dimensions in each head. Must be non-negative and even.

See also

getRotaryEmbeddingDim

9.184.4 Member Data Documentation

9.184.4.1 mImpl

```
apiv::VRotaryEmbeddingLayer* nvinfer1::IRotaryEmbeddingLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

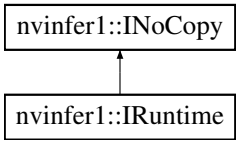
- NvInfer.h

9.185 nvinfer1::IRuntime Class Reference

Allows a serialized functionally unsafe engine to be deserialized.

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::IRuntime:



Public Member Functions

- virtual ~IRuntime () noexcept=0
- void setDLACore (int32_t dlaCore) noexcept

  Sets the DLA core used by the network. Defaults to -1.
- int32_t getDLACore () const noexcept

  Get the DLA core that the engine executes on.
- int32_t getNbDLACores () const noexcept

  Returns number of DLA hardware cores accessible or 0 if DLA is unavailable.
- void setGpuAllocator (IGpuAllocator *allocator) noexcept

  Set the GPU allocator.
- void setErrorRecorder (IErrorRecorder *recorder) noexcept

  Set the ErrorRecorder for this interface.
- IErrorRecorder * getErrorRecorder () const noexcept

  get the ErrorRecorder assigned to this interface.
- ICudaEngine * deserializeCudaEngine (void const *blob, std::size_t size) noexcept

  Deserialize an engine from host memory.
- ICudaEngine * deserializeCudaEngine (IStreamReaderV2 &streamReader)

  Deserialize an engine from a stream. IStreamReaderV2 is expected to support reading to both host and device pointers.
- ILogger * getLogger () const noexcept

  get the logger with which the runtime was created
- bool setMaxThreads (int32_t maxThreads) noexcept

  Set the maximum number of threads.
- int32_t getMaxThreads () const noexcept

  Get the maximum number of threads that can be used by the runtime.
- void setTemporaryDirectory (char const *path) noexcept

  Set the directory that will be used by this runtime for temporary files.
- char const * getTemporaryDirectory () const noexcept

  Get the directory that will be used by this runtime for temporary files.
- void setTempfileControlFlags (TempfileControlFlags flags) noexcept

  Set the tempfile control flags for this runtime.
- TempfileControlFlags getTempfileControlFlags () const noexcept

  Get the tempfile control flags for this runtime.

- IPluginRegistry & getPluginRegistry () noexcept

  Get the local plugin registry that can be used by the runtime.
- IRuntime * loadRuntime (char const *path) noexcept

  Load IRuntime from the file.
- void setEngineHostCodeAllowed (bool allowed) noexcept

  Set whether the runtime is allowed to deserialize engines with host executable code.
- bool getEngineHostCodeAllowed () const noexcept

  Get whether the runtime is allowed to deserialize engines with host executable code.

Protected Attributes

- apiv::VRuntime * mImpl {}

Additional Inherited Members

9.185.1 Detailed Description

Allows a serialized functionally unsafe engine to be deserialized.

> **Warning**
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.185.2 Constructor & Destructor Documentation

9.185.2.1 ~IRuntime()

```
nvinfer1::IRuntime::~IRuntime ( )  [inline], [pure virtual], [default], [noexcept]
```

9.185.3 Member Function Documentation

9.185.3.1 deserializeCudaEngine() [1/2]

```
ICudaEngine * nvinfer1::IRuntime::deserializeCudaEngine (
            IStreamReaderV2 & streamReader )  [inline]
```

Deserialize an engine from a stream. IStreamReaderV2 is expected to support reading to both host and device pointers.

If an error recorder has been set for the runtime, it will also be passed to the engine.

This deserialization path will reduce engine load time when applied with GDS (GPU Direct storage), or when weight streaming is enabled.

Warning

Destroying the IRuntime before destroying all associated ICudaEngine instances results in undefined behavior.

Parameters

streamReader	a read-only stream from which TensorRT will deserialize a previously serialized engine.

Returns

The engine, or nullptr if it could not be deserialized. The pointer may not be valid immediately after the function returns.

9.185.3.2 deserializeCudaEngine() [2/2]

```
ICudaEngine * nvinfer1::IRuntime::deserializeCudaEngine (
            void const * blob,
            std::size_t size )  [inline], [noexcept]
```

Deserialize an engine from host memory.

If an error recorder has been set for the runtime, it will also be passed to the engine.

Warning

Destroying the IRuntime before destroying all associated ICudaEngine instances results in undefined behavior.

Parameters

blob	The memory that holds the serialized engine.
size	The size of the memory.

Returns

The engine, or nullptr if it could not be deserialized.

9.185.3.3 getDLACore()

```
int32_t nvinfer1::IRuntime::getDLACore ( ) const  [inline], [noexcept]
```

Get the DLA core that the engine executes on.

Returns

assigned DLA core or -1 for DLA not present or unset.

9.185.3.4 getEngineHostCodeAllowed()

```
bool nvinfer1::IRuntime::getEngineHostCodeAllowed ( ) const  [inline], [noexcept]
```

Get whether the runtime is allowed to deserialize engines with host executable code.

Returns

Whether the runtime is allowed to deserialize engines with host executable code.

9.185.3.5 getErrorRecorder()

```
IErrorRecorder * nvinfer1::IRuntime::getErrorRecorder ( ) const  [inline], [noexcept]
```

get the ErrorRecorder assigned to this interface.

Retrieves the assigned error recorder object for the given class. A nullptr will be returned if an error handler has not been set.

Returns

A pointer to the IErrorRecorder object that has been registered.

See also

setErrorRecorder()

9.185.3.6 getLogger()

```
ILogger * nvinfer1::IRuntime::getLogger ( ) const  [inline], [noexcept]
```

get the logger with which the runtime was created

Returns

the logger

9.185.3.7 getMaxThreads()

```
int32_t nvinfer1::IRuntime::getMaxThreads ( ) const  [inline], [noexcept]
```

Get the maximum number of threads that can be used by the runtime.

Retrieves the maximum number of threads that can be used by the runtime.

Returns

The maximum number of threads that can be used by the runtime.

See also

setMaxThreads()

9.185.3.8 getNbDLACores()

```
int32_t nvinfer1::IRuntime::getNbDLACores ( ) const  [inline], [noexcept]
```

Returns number of DLA hardware cores accessible or 0 if DLA is unavailable.

9.185.3.9 getPluginRegistry()

```
IPluginRegistry & nvinfer1::IRuntime::getPluginRegistry ( )  [inline], [noexcept]
```

Get the local plugin registry that can be used by the runtime.

Returns

The local plugin registry that can be used by the runtime.

9.185.3.10 getTempfileControlFlags()

```
TempfileControlFlags nvinfer1::IRuntime::getTempfileControlFlags ( ) const  [inline], [noexcept]
```

Get the tempfile control flags for this runtime.

Returns

The flags currently set.

See also

TempfileControlFlag, TempfileControlFlags, setTempfileControlFlags()

9.185.3.11 getTemporaryDirectory()

```
char const * nvinfer1::IRuntime::getTemporaryDirectory ( ) const  [inline], [noexcept]
```

Get the directory that will be used by this runtime for temporary files.

Returns

A path to the temporary directory in use, or nullptr if no path is specified.

See also

setTemporaryDirectory()

9.185.3.12 loadRuntime()

```
IRuntime * nvinfer1::IRuntime::loadRuntime (
            char const * path )  [inline], [noexcept]
```

Load IRuntime from the file.

This method loads a runtime library from a shared library file. The runtime can then be used to execute a plan file built with BuilderFlag::kVERSION_COMPATIBLE and BuilderFlag::kEXCLUDE_LEAN_RUNTIME both set and built with the same version of TensorRT as the loaded runtime library.

Parameters

path	Path to the runtime lean library.

Returns

the runtime library, or nullptr if it could not be loaded

Warning

The path string must be null-terminated, and be at most 4096 bytes including the terminator.

9.185.3.13 setDLACore()

```
void nvinfer1::IRuntime::setDLACore (
            int32_t dlaCore )  [inline], [noexcept]
```

Sets the DLA core used by the network. Defaults to -1.

Parameters

dlaCore	The DLA core to execute the engine on, in the range [0,getNbDlaCores()).

This function is used to specify which DLA core to use via indexing, if multiple DLA cores are available.

Warning

if getNbDLACores() returns 0, then this function does nothing.

See also

getDLACore()

9.185.3.14 setEngineHostCodeAllowed()

```
void nvinfer1::IRuntime::setEngineHostCodeAllowed (
            bool allowed )  [inline], [noexcept]
```

Set whether the runtime is allowed to deserialize engines with host executable code.

Parameters

allowed	Whether the runtime is allowed to deserialize engines with host executable code.

The default value is false.

9.185.3.15 setErrorRecorder()

```
void nvinfer1::IRuntime::setErrorRecorder (
            IErrorRecorder * recorder )  [inline], [noexcept]
```

Set the ErrorRecorder for this interface.

Assigns the ErrorRecorder to this interface. The ErrorRecorder will track all errors during execution. This function will call incRefCount of the registered ErrorRecorder at least once. Setting recorder to nullptr unregisters the recorder with the interface, resulting in a call to decRefCount if a recorder has been registered.

If an error recorder is not set, messages will be sent to the global log stream.

Parameters

recorder	The error recorder to register with this interface.

See also

getErrorRecorder()

9.185.3.16 setGpuAllocator()

```
void nvinfer1::IRuntime::setGpuAllocator (
            IGpuAllocator * allocator )  [inline], [noexcept]
```

Set the GPU allocator.

Parameters

allocator	Set the GPU allocator to be used by the runtime. All GPU memory acquired will use this allocator. If NULL is passed, the default allocator will be used.

Default: allocateAsync uses cudaMallocAsync if cudaDevAttrMemoryPoolsSupported returns true, otherwise falls back to cudaMalloc. allocate always uses cudaMalloc.

If nullptr is passed, the default allocator will be used.

9.185.3.17 setMaxThreads()

```
bool nvinfer1::IRuntime::setMaxThreads (
            int32_t maxThreads )  [inline], [noexcept]
```

Set the maximum number of threads.

Parameters

maxThreads	The maximum number of threads that can be used by the runtime.

Returns

True if successful, false otherwise.

The default value is 1 and includes the current thread. A value greater than 1 permits TensorRT to use multi-threaded algorithms. A value less than 1 triggers a kINVALID_ARGUMENT error.

9.185.3.18 setTempfileControlFlags()

```
void nvinfer1::IRuntime::setTempfileControlFlags (
            TempfileControlFlags flags )  [inline], [noexcept]
```

Set the tempfile control flags for this runtime.

Parameters

flags	The flags to set.

The default value is all flags set, i.e.

(1U << static_cast<uint32_t>(kALLOW_IN_MEMORY_FILES)) | (1U << static_cast<uint32_t>(kALLOW_↩ TEMPORARY_FILES))

See also

TempfileControlFlag, TempfileControlFlags, getTempfileControlFlags()

9.185.3.19 setTemporaryDirectory()

```
void nvinfer1::IRuntime::setTemporaryDirectory (
             char const * path )  [inline], [noexcept]
```

Set the directory that will be used by this runtime for temporary files.

On some platforms the TensorRT runtime may need to create and use temporary files with read/write/execute permissions to implement runtime functionality.

Parameters

path	Path to the temporary directory for use, or nullptr.

If path is nullptr, then TensorRT will use platform-specific heuristics to pick a default temporary directory if required:

- On UNIX/Linux platforms, TensorRT will first try the TMPDIR environment variable, then fall back to /tmp
- On Windows, TensorRT will try the TEMP environment variable.

See the TensorRT Developer Guide for more information.

The default value is nullptr.

Warning

If path is not nullptr, it must be a non-empty string representing a relative or absolute path in the format expected by the host operating system.

The string path must be null-terminated, and be at most 4096 bytes including the terminator. Note that the operating system may have stricter path length requirements.

The process using TensorRT must have rwx permissions for the temporary directory, and the directory shall be configured to disallow other users from modifying created files (e.g. on Linux, if the directory is shared

with other users, the sticky bit must be set).

See also

getTemporaryDirectory()

9.185.4 Member Data Documentation

9.185.4.1 mImpl

```
apiv::VRuntime* nvinfer1::IRuntime::mImpl {}  [protected]
```

The documentation for this class was generated from the following file:

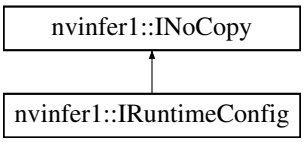
- NvInferRuntime.h

9.186 nvinfer1::IRuntimeConfig Class Reference

A class for runtime configuration. This class is used during execution context creation.

```
#include <NvInferRuntime.h>
```

Inheritance diagram for nvinfer1::IRuntimeConfig:



Public Member Functions

- virtual ~IRuntimeConfig () noexcept=0
- void setExecutionContextAllocationStrategy (ExecutionContextAllocationStrategy strategy) noexcept

  Set the execution context allocation strategy. Default value is kSTATIC.
- ExecutionContextAllocationStrategy getExecutionContextAllocationStrategy () const noexcept

  Get the execution context allocation strategy.

Protected Attributes

- apiv::VRuntimeConfig * mImpl

Additional Inherited Members

9.186.1 Detailed Description

A class for runtime configuration. This class is used during execution context creation.

See also

IRuntime, IBuilderConfig

9.186.2 Constructor & Destructor Documentation

9.186.2.1 ~IRuntimeConfig()

```
nvinfer1::IRuntimeConfig::~IRuntimeConfig ( )  [inline], [pure virtual], [default], [noexcept]
```

9.186.3 Member Function Documentation

9.186.3.1 getExecutionContextAllocationStrategy()

```
ExecutionContextAllocationStrategy nvinfer1::IRuntimeConfig::getExecutionContextAllocationStrategy
( ) const  [inline], [noexcept]
```

Get the execution context allocation strategy.

Returns

The execution context allocation strategy.

9.186.3.2 setExecutionContextAllocationStrategy()

```
void nvinfer1::IRuntimeConfig::setExecutionContextAllocationStrategy (
            ExecutionContextAllocationStrategy strategy )  [inline], [noexcept]
```

Set the execution context allocation strategy. Default value is kSTATIC.

Parameters

strategy	The execution context allocation strategy.

9.186.4 Member Data Documentation

9.186.4.1 mImpl

```
apiv::VRuntimeConfig* nvinfer1::IRuntimeConfig::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.187 nvinfer2::safe::ISafeMemAllocator Class Reference

Application-implemented class for controlling memory allocation on the GPU/CPU.

```
#include <NvInferSafeMemAllocator.h>
```

Public Member Functions

- virtual void * allocate (uint64_t const size, uint64_t const alignment, MemoryPlacement const flags, MemoryUsage const usage, ISafeRecorder &recorder) noexcept=0

  A thread-safe callback implemented by the application to handle acquisition of GPU/CPU memory.
- virtual ~ISafeMemAllocator ()=default

  Destructor for ISafeMemAllocator.
- ISafeMemAllocator ()=default

  Default constructor for ISafeMemAllocator.
- virtual bool deallocate (void *const memory, MemoryPlacement const flags, ISafeRecorder &recorder) noexcept=0

  A thread-safe callback implemented by the application to handle release of GPU/CPU memory.

Protected Member Functions

- ISafeMemAllocator (ISafeMemAllocator const &)=default

  Copy constructor (defaulted).
- ISafeMemAllocator (ISafeMemAllocator &&)=default

  Move constructor (defaulted).
- ISafeMemAllocator & operator= (ISafeMemAllocator const &) &=default

  Copy assignment operator (defaulted).
- ISafeMemAllocator & operator= (ISafeMemAllocator &&) &=default

  Move assignment operator (defaulted).

9.187.1 Detailed Description

Application-implemented class for controlling memory allocation on the GPU/CPU.

Warning

The lifetime of an ISafeMemAllocator object must exceed that of all objects that use it.

9.187.2 Constructor & Destructor Documentation

9.187.2.1 ~ISafeMemAllocator()

```
virtual nvinfer2::safe::ISafeMemAllocator::~ISafeMemAllocator ( )  [virtual], [default]
```

Destructor for ISafeMemAllocator.

Precondition

none

Postcondition

Object is destroyed; allocator is no longer valid for use.

9.187.2.2 ISafeMemAllocator() [1/3]

```
nvinfer2::safe::ISafeMemAllocator::ISafeMemAllocator ( )  [default]
```

Default constructor for ISafeMemAllocator.

Precondition

none

Postcondition

Object is in default-constructed state; ready to handle allocate/deallocate requests.

9.187.2.3 ISafeMemAllocator() [2/3]

```
nvinfer2::safe::ISafeMemAllocator::ISafeMemAllocator (
            ISafeMemAllocator const &  )  [protected], [default]
```

Copy constructor (defaulted).

Precondition

none

Postcondition

This object is a copy of the source.

9.187.2.4 ISafeMemAllocator() [3/3]

```
nvinfer2::safe::ISafeMemAllocator::ISafeMemAllocator (
            ISafeMemAllocator &&  )  [protected], [default]
```

Move constructor (defaulted).

Precondition

none

Postcondition

This object has the state that the source had; source is in valid but unspecified state.

9.187.3 Member Function Documentation

9.187.3.1 allocate()

```
virtual void * nvinfer2::safe::ISafeMemAllocator::allocate (
            uint64_t const size,
            uint64_t const alignment,
            MemoryPlacement const flags,
            MemoryUsage const usage,
            ISafeRecorder & recorder )  [pure virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle acquisition of GPU/CPU memory.

Precondition

INIT API

Parameters

size	The size of the memory block required (in bytes).
alignment	The required alignment of memory. Alignment will be zero or a power of 2 not exceeding the alignment guaranteed by cudaMalloc. Thus this allocator can be safely implemented with cudaMalloc/cudaFree. An alignment value of zero indicates any alignment is acceptable.
flags	The type of memory to be allocated: one of MemoryPlacement::kGPU, MemoryPlacement::kCPU, MemoryPlacement::kCPU_PINNED.
usage	The intended usage of memory to be allocated: one of MemoryUsage::kIMMUTABLE, MemoryUsage::kSCRATCH, MemoryUsage::kIOTENSOR.
recorder	A safe message recorder

Returns

If the allocation was successful, the start address of a device memory block of the requested size, nullptr otherwise.

- If an allocation request of size 0 is made, nullptr must be returned.
- If an allocation request cannot be satisfied for any reason, nullptr must be returned.
- If a non-null address is returned, it is guaranteed to have the specified alignment.

Postcondition

If non-null returned, a block of at least `size` bytes with the requested alignment is allocated and owned by the caller until deallocate() is called. If nullptr returned, no allocation was made.

Warning

The implementation must guarantee thread safety for concurrent allocate/deallocate requests.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads.

9.187.3.2 deallocate()

```
virtual bool nvinfer2::safe::ISafeMemAllocator::deallocate (
            void *const memory,
            MemoryPlacement const flags,
            ISafeRecorder & recorder )  [pure virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle release of GPU/CPU memory.

Precondition

INIT API

TensorRT may pass a nullptr to this function if it was previously returned by allocate().

Parameters

memory	A memory address that was previously returned by an allocate() call of the same allocator object.
flags	The type of memory to be deallocated: one of MemoryPlacement::kGPU, MemoryPlacement::kCPU, MemoryPlacement::kCPU_PINNED.
recorder	A safe message recorder

Returns

True if the acquired memory is released successfully, false otherwise.

Postcondition

If true, the memory at `memory` has been released and must not be used. If false, release failed or `memory` was nullptr.

Warning

The implementation must guarantee thread safety for concurrent allocate/deallocate requests.

Note

Calling deallocate() multiple times with the same pointer results in undefined behavior. Implementations MUST detect and reject double-free attempts.

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads.

9.187.3.3 operator=() [1/2]

```
ISafeMemAllocator & nvinfer2::safe::ISafeMemAllocator::operator= (
            ISafeMemAllocator &&  ) &  [protected], [default]
```

Move assignment operator (defaulted).

Precondition

none

Postcondition

This object has the state that the source had; source is in valid but unspecified state.

9.187.3.4 operator=() [2/2]

```
ISafeMemAllocator & nvinfer2::safe::ISafeMemAllocator::operator= (
            ISafeMemAllocator const &  ) &  [protected], [default]
```

Copy assignment operator (defaulted).

Precondition

none

Postcondition

This object is a copy of the source.

The documentation for this class was generated from the following file:

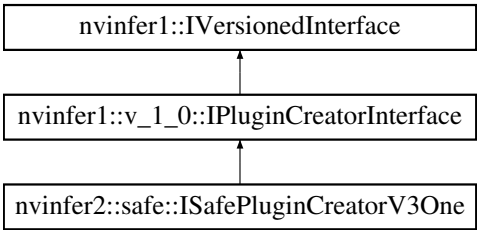
- NvInferSafeMemAllocator.h

9.188 nvinfer2::safe::ISafePluginCreatorV3One Class Reference

The main interface to be implemented by all plugin creator classes.

`#include <NvInferSafePlugin.h>`

Inheritance diagram for nvinfer2::safe::ISafePluginCreatorV3One:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual IPluginV3 * createPlugin (AsciiChar const *name, PluginFieldCollection const *fc, TensorRTPhase phase) noexcept=0

  Return a plugin object. Return nullptr in case of error.
- virtual PluginFieldCollection const * getFieldNames () noexcept=0

  Return a list of fields that need to be passed to createPlugin() when creating a plugin for use in the TensorRT build phase.
- virtual AsciiChar const * getPluginName () const noexcept=0

  Return the plugin name.

- virtual AsciiChar const * getPluginVersion () const noexcept=0

  Return the plugin version.
- virtual AsciiChar const * getPluginNamespace () const noexcept=0

  Return the plugin namespace.
- virtual ISafeRecorder * getSafeRecorder () const noexcept=0

  Return the safe recorder.
- virtual void setSafeRecorder (ISafeRecorder &) noexcept=0

  Set the safe recorder for this interface.
- ISafePluginCreatorV3One ()=default
- ~ISafePluginCreatorV3One () noexcept override=default

Protected Member Functions

- ISafePluginCreatorV3One (ISafePluginCreatorV3One const &)=default
- ISafePluginCreatorV3One (ISafePluginCreatorV3One &&)=default
- ISafePluginCreatorV3One & operator= (ISafePluginCreatorV3One const &) &=default
- ISafePluginCreatorV3One & operator= (ISafePluginCreatorV3One &&) &=default

9.188.1 Detailed Description

The main interface to be implemented by all plugin creator classes.

9.188.2 Constructor & Destructor Documentation

9.188.2.1 ISafePluginCreatorV3One() [1/3]

```
nvinfer2::safe::ISafePluginCreatorV3One::ISafePluginCreatorV3One ( )  [default]
```

9.188.2.2 ~ISafePluginCreatorV3One()

```
nvinfer2::safe::ISafePluginCreatorV3One::~ISafePluginCreatorV3One ( )  [override], [default],
[noexcept]
```

9.188.2.3 ISafePluginCreatorV3One() [2/3]

```
nvinfer2::safe::ISafePluginCreatorV3One::ISafePluginCreatorV3One (
            ISafePluginCreatorV3One const &  )  [protected], [default]
```

9.188.2.4 ISafePluginCreatorV3One() [3/3]

```
nvinfer2::safe::ISafePluginCreatorV3One::ISafePluginCreatorV3One (
            ISafePluginCreatorV3One &&  )  [protected], [default]
```

9.188.3 Member Function Documentation

9.188.3.1 createPlugin()

```
virtual IPluginV3 * nvinfer2::safe::ISafePluginCreatorV3One::createPlugin (
            AsciiChar const * name,
            PluginFieldCollection const * fc,
            TensorRTPhase phase )  [pure virtual], [noexcept]
```

Return a plugin object. Return nullptr in case of error.

Precondition

INIT API

Parameters

name	A NULL-terminated name string of length 1024 or less, including the NULL terminator.
fc	A pointer to a collection of fields needed for constructing the plugin.
phase	The TensorRT phase in which the plugin is being created

When the phase is TensorRTPhase::kRUNTIME, the PluginFieldCollection provided for serialization by the plugin's runtime interface will be passed as fc.

Note

The returned plugin object must be in an initialized state

9.188.3.2 getFieldNames()

```
virtual PluginFieldCollection const * nvinfer2::safe::ISafePluginCreatorV3One::getFieldNames ( )
[pure virtual], [noexcept]
```

Return a list of fields that need to be passed to createPlugin() when creating a plugin for use in the TensorRT build phase.

Precondition

INIT API

See also

PluginFieldCollection

9.188.3.3 getInterfaceInfo()

```
InterfaceInfo nvinfer2::safe::ISafePluginCreatorV3One::getInterfaceInfo ( ) const  [inline], [override], [virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Precondition

INIT API

Implements nvinfer1::IVersionedInterface.

9.188.3.4 getPluginName()

```
virtual AsciiChar const * nvinfer2::safe::ISafePluginCreatorV3One::getPluginName ( ) const  [pure virtual], [noexcept]
```

Return the plugin name.

Precondition

INIT API

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

9.188.3.5 getPluginNamespace()

```
virtual AsciiChar const * nvinfer2::safe::ISafePluginCreatorV3One::getPluginNamespace ( ) const [pure virtual], [noexcept]
```

Return the plugin namespace.

Precondition

INIT API

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

9.188.3.6 getPluginVersion()

```
virtual AsciiChar const * nvinfer2::safe::ISafePluginCreatorV3One::getPluginVersion ( ) const
[pure virtual], [noexcept]
```

Return the plugin version.

Precondition

INIT API

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

9.188.3.7 getSafeRecorder()

```
virtual ISafeRecorder * nvinfer2::safe::ISafePluginCreatorV3One::getSafeRecorder ( ) const  [pure
virtual], [noexcept]
```

Return the safe recorder.

Precondition

INIT API

9.188.3.8 operator=() [1/2]

```
ISafePluginCreatorV3One & nvinfer2::safe::ISafePluginCreatorV3One::operator= (
            ISafePluginCreatorV3One &&  ) &  [protected], [default]
```

9.188.3.9 operator=() [2/2]

```
ISafePluginCreatorV3One & nvinfer2::safe::ISafePluginCreatorV3One::operator= (
            ISafePluginCreatorV3One const &  ) &  [protected], [default]
```

9.188.3.10 setSafeRecorder()

```
virtual void nvinfer2::safe::ISafePluginCreatorV3One::setSafeRecorder (
            ISafeRecorder &  )  [pure virtual], [noexcept]
```

Set the safe recorder for this interface.

Precondition

INIT API

The documentation for this class was generated from the following file:

- NvInferSafePlugin.h

9.189 nvinfer2::safe::ISafePluginRegistry Class Reference

PluginRegistry interface class that contains all the methods user can use to interface with the registry object.

`#include <NvInferSafePlugin.h>`

Public Member Functions

- virtual ErrorCode registerCreator (IPluginCreatorInterface &creator, AsciiChar const ∗const pluginNamespace, ISafeRecorder &recorder) noexcept=0

  Register a plugin creator.
- virtual ErrorCode getAllCreators (IPluginCreatorInterface ∗const ∗&creators, int32_t &numCreators) const noexcept=0

  Return all the registered plugin creators and the number of registered plugin creators.
- virtual ErrorCode getCreator (IPluginCreatorInterface ∗&creator, AsciiChar const ∗const pluginName, AsciiChar const ∗const pluginVersion, AsciiChar const ∗const pluginNamespace="") noexcept=0

  Return plugin creator based on plugin name, version, and namespace associated with plugin during network creation.
- virtual ErrorCode setSafeRecorder (ISafeRecorder &recorder) noexcept=0

  Set the message recorder for this interface.
- virtual ErrorCode getSafeRecorder (ISafeRecorder ∗&recorder) const noexcept=0

  Get the message recorder assigned to this interface.
- virtual ErrorCode deregisterCreator (IPluginCreatorInterface const &creator) noexcept=0

  Deregister a previously registered plugin creator.

- ISafePluginRegistry ()=default

  Default constructor for ISafePluginRegistry.
- ISafePluginRegistry (ISafePluginRegistry const &)=delete

  Deleted copy constructor to maintain non-copyability.
- ISafePluginRegistry (ISafePluginRegistry &&)=delete

  Deleted move constructor to maintain non-movability.
- ISafePluginRegistry & operator= (ISafePluginRegistry const &) &=delete

  Deleted copy assignment operator (lvalue) to maintain non-copyability.
- ISafePluginRegistry & operator= (ISafePluginRegistry &&) &=delete

  Deleted move assignment operator (lvalue) to maintain non-movability.

Protected Member Functions

- virtual ~ISafePluginRegistry () noexcept=default

  Destructor for ISafePluginRegistry.

9.189.1 Detailed Description

PluginRegistry interface class that contains all the methods user can use to interface with the registry object.

9.189.2 Constructor & Destructor Documentation

9.189.2.1 ISafePluginRegistry() [1/3]

```
nvinfer2::safe::ISafePluginRegistry::ISafePluginRegistry ( )  [default]
```

Default constructor for ISafePluginRegistry.

Constructs an object in a default state. Only factory or derived types create instances.

Precondition

none

Postcondition

Object is in default-constructed state.

9.189.2.2 ISafePluginRegistry() [2/3]

```
nvinfer2::safe::ISafePluginRegistry::ISafePluginRegistry (
            ISafePluginRegistry const &  )  [delete]
```

Deleted copy constructor to maintain non-copyability.

Precondition

none

Postcondition

none

9.189.2.3 ISafePluginRegistry() [3/3]

```
nvinfer2::safe::ISafePluginRegistry::ISafePluginRegistry (
            ISafePluginRegistry &&  )  [delete]
```

Deleted move constructor to maintain non-movability.

Precondition

none

Postcondition

none

9.189.2.4 ~ISafePluginRegistry()

```
virtual nvinfer2::safe::ISafePluginRegistry::~ISafePluginRegistry ( )  [protected], [virtual],
[default], [noexcept]
```

Destructor for ISafePluginRegistry.

9.189.3 Member Function Documentation

9.189.3.1 deregisterCreator()

```
virtual ErrorCode nvinfer2::safe::ISafePluginRegistry::deregisterCreator (
            IPluginCreatorInterface const & creator )  [pure virtual], [noexcept]
```

Deregister a previously registered plugin creator.

Precondition

INIT API

Since there may be a desire to limit the number of plugins, this function provides a mechanism for removing plugin creators registered in TensorRT. The plugin creator that is specified by `creator` is removed from TensorRT and no longer tracked.

Parameters

creator	The plugin creator to deregister.

Returns

Returns kSUCCESS if the plugin creator was successfully deregistered. Returns kINVALID_CONFIG if the plugin creator was not found in the registry or otherwise could not be deregistered.

- Allowed context for the API call
  - Thread-safe: Yes

9.189.3.2 getAllCreators()

```
virtual ErrorCode nvinfer2::safe::ISafePluginRegistry::getAllCreators (
            IPluginCreatorInterface *const *& creators,
            int32_t & numCreators ) const  [pure virtual], [noexcept]
```

Return all the registered plugin creators and the number of registered plugin creators.

Precondition

INIT API

Parameters

out	*creators*	The start address of an IPluginCreatorInterface* array of length numCreators if at least one plugin creator has been registered, or nullptr if there are no registered plugin creators.
out	*numCreators*	If the call completes successfully, the number of registered plugin creators (which will be an integer between 0 and 100 inclusive).

Returns

Returns kSUCCESS if the call completes successfully and at least one plugin creator is registered. Returns kINVALID_ARGUMENT if numCreators or creators is a nullptr. Returns kUNSPECIFIED_ERROR if an unexpected error occurs.

- Allowed context for the API call
  - Thread-safe: No

9.189.3.3 getCreator()

```
virtual ErrorCode nvinfer2::safe::ISafePluginRegistry::getCreator (
            IPluginCreatorInterface *& creator,
            AsciiChar const *const pluginName,
            AsciiChar const *const pluginVersion,
            AsciiChar const *const pluginNamespace = "" )  [pure virtual], [noexcept]
```

Return plugin creator based on plugin name, version, and namespace associated with plugin during network creation.

Precondition

INIT API

Warning

The strings pluginName, pluginVersion, and pluginNamespace must be NULL terminated and have a length of 1024 bytes or less including the NULL terminator.

Parameters

out	*creator*	Pointer to the plugin creator pointer to be returned.
	pluginName	The plugin name string.
	pluginVersion	The plugin version string.
	pluginNamespace	The plugin namespace (by default empty string).

Returns

Returns kSUCCESS if a plugin creator corresponding to the passed name, version, and namespace can be found in the registry. Returns kINVALID_ARGUMENT if any of the input arguments is nullptr or exceeds the string length limit. Returns kINVALID_CONFIG if no plugin creator corresponding to the input arguments can be found in the registry or if a plugin creator can be found but its stored namespace attribute does not match the pluginNamespace.

- Allowed context for the API call
  - Thread-safe: Yes

9.189.3.4 getSafeRecorder()

```
virtual ErrorCode nvinfer2::safe::ISafePluginRegistry::getSafeRecorder (
            ISafeRecorder *& recorder ) const  [pure virtual], [noexcept]
```

Get the message recorder assigned to this interface.

Precondition

INIT API

Retrieves the assigned error recorder object for the given class. A default error recorder does not exist, so recorder parameter will be set to nullptr (pointer to nullptr) if setSafeRecorder has not been called, or an message recorder has not been inherited.

Parameters

out	*recorder*	Pointer to the ISafeRecorder object to be returned.

Returns

Returns kSUCCESS if the message recorder is successfully retrieved. Returns kINVALID_ARGUMENT if the recorder parameter is nullptr.

See also

setSafeRecorder()

- Allowed context for the API call
  - Thread-safe: Yes

9.189.3.5 operator=() [1/2]

```
ISafePluginRegistry & nvinfer2::safe::ISafePluginRegistry::operator= (
            ISafePluginRegistry &&  ) &  [delete]
```

Deleted move assignment operator (lvalue) to maintain non-movability.

Precondition

none

Postcondition

none

9.189.3.6 operator=() [2/2]

```
ISafePluginRegistry & nvinfer2::safe::ISafePluginRegistry::operator= (
            ISafePluginRegistry const &  ) &  [delete]
```

Deleted copy assignment operator (lvalue) to maintain non-copyability.

Precondition

none

Postcondition

none

9.189.3.7 registerCreator()

```
virtual ErrorCode nvinfer2::safe::ISafePluginRegistry::registerCreator (
            IPluginCreatorInterface & creator,
            AsciiChar const *const pluginNamespace,
            ISafeRecorder & recorder )  [pure virtual], [noexcept]
```

Register a plugin creator.

Precondition

INIT API

Parameters

creator	The plugin creator to be registered.
pluginNamespace	A NULL-terminated namespace string, which must be 1024 bytes or less including the NULL terminator. It must be identical with the result of calling ISafePluginCreatorV3One::getPluginNamespace() on the creator object.
recorder	The error recorder to register with this interface

Returns

Returns kSUCCESS if the plugin creator is successfully registered. Returns kINVALID_ARGUMENT if the pluginNamespace string is a nullptr, exceeds the maximum length, or does not match the result of creator.↩ getPluginNamespace(). Returns kINVALID_CONFIG if there have already been 100 plugin creators registered or another plugin creator with the same combination of plugin name, version, and namespace has already been registered.

- Allowed context for the API call

- Thread-safe: Yes; calls to this method will be synchronized by a mutex.

9.189.3.8 setSafeRecorder()

```
virtual ErrorCode nvinfer2::safe::ISafePluginRegistry::setSafeRecorder (
            ISafeRecorder & recorder ) [pure virtual], [noexcept]
```

Set the message recorder for this interface.

Precondition

INIT API

Assigns the message recorder to this interface. The message recorder will track all errors during execution. This function will call incRefCount of the registered message recorder at least once.

Note

Since the pluginRegistry is a global static object, the recorder used for the registry must have an equal or longer life time than the registry. I.e., when the process terminates the registry needs to hold a global static error recorder.

Parameters

recorder	The error recorder to register with this interface, or nullptr to deregister the current recorder.

Returns

Returns kSUCCESS if the message recorder is successfully registered or deregistered. Returns kINVALID_↩ ARGUMENT if the recorder is set to nullptr and the recorder has been registered.

See also

getSafeRecorder()

- Allowed context for the API call
  - Thread-safe: No

The documentation for this class was generated from the following file:

- NvInferSafePlugin.h

9.190 nvinfer2::safe::ISafePluginResourceContext Class Reference

Interface for plugins to access per context resources provided by TensorRT.

`#include <NvInferSafePlugin.h>`

Public Member Functions

- virtual ISafeMemAllocator * getSafeMemAllocator () const noexcept=0

  Returns the GPU memory allocator associated with this resource context.
- virtual ISafeRecorder * getSafeRecorder () const noexcept=0

  Returns the error recorder associated with this resource context.
- virtual RuntimeErrorInformation * getRuntimeErrorInformation () const noexcept=0

  Returns the RuntimeErrorInformation buffer associated with this resource context.
- virtual ~ISafePluginResourceContext () noexcept=default

  Destructor for ISafePluginResourceContext.

Protected Member Functions

- ISafePluginResourceContext ()=default

  Default constructor; access limited to derived types and factory.
- ISafePluginResourceContext (ISafePluginResourceContext const &)=default

  Copy constructor (defaulted).
- ISafePluginResourceContext (ISafePluginResourceContext &&)=default

  Move constructor (defaulted).
- ISafePluginResourceContext & operator= (ISafePluginResourceContext const &) &=default

  Copy assignment operator (defaulted).
- ISafePluginResourceContext & operator= (ISafePluginResourceContext &&) &=default

  Move assignment operator (defaulted).

9.190.1 Detailed Description

Interface for plugins to access per context resources provided by TensorRT.

There is no public way to construct an ISafePluginResourceContext (i.e. User should not construct this object). It appears as an argument to IPluginV3OneSafeRuntime::initResource(). Overrides of that method can use the ISafePluginResourceContext object to access the ISafeMemAllocator and ISafeRecorder object asscociate with the specific TRTGraph that the plugin is part of

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

See also

IPluginV3OneSafeRuntime::initResource()

9.190.2 Constructor & Destructor Documentation

9.190.2.1 ~ISafePluginResourceContext()

```
virtual nvinfer2::safe::ISafePluginResourceContext::~ISafePluginResourceContext ( )  [virtual],
[default], [noexcept]
```

Destructor for ISafePluginResourceContext.

Destroys the object and releases any resources owned by the implementation.

Precondition

none

Postcondition

none

9.190.2.2 ISafePluginResourceContext() [1/3]

```
nvinfer2::safe::ISafePluginResourceContext::ISafePluginResourceContext ( )  [protected], [default]
```

Default constructor; access limited to derived types and factory.

Constructs an object in a default state. Not intended for direct use by API consumers.

Precondition

none

Postcondition

none

9.190.2.3 ISafePluginResourceContext() [2/3]

```
nvinfer2::safe::ISafePluginResourceContext::ISafePluginResourceContext (
            ISafePluginResourceContext const &  )  [protected], [default]
```

Copy constructor (defaulted).

Creates a copy of the source context; semantics are implementation-defined.

Precondition

none

Postcondition

This object is a copy of the source.

9.190.2.4 ISafePluginResourceContext() [3/3]

```
nvinfer2::safe::ISafePluginResourceContext::ISafePluginResourceContext (
            ISafePluginResourceContext &&  )  [protected], [default]
```

Move constructor (defaulted).

Moves state from the source; source is left in valid but unspecified state.

Precondition

none

Postcondition

This object has the state that the source had; source is in valid but unspecified state.

9.190.3 Member Function Documentation

9.190.3.1 getRuntimeErrorInformation()

```
virtual RuntimeErrorInformation * nvinfer2::safe::ISafePluginResourceContext::getRuntimeError↩
Information ( ) const  [pure virtual], [noexcept]
```

Returns the RuntimeErrorInformation buffer associated with this resource context.

The buffer holds runtime error details for the execution context; may be written by the runtime.

Precondition

RUNTIME API

Postcondition

none

Returns

Pointer to the runtime error information, or nullptr if not set.

Return values

non-null	When error information has been associated with this context.
nullptr	When no error information is associated.

See also

IPluginV3OneSafeRuntime::initResource()

9.190.3.2 getSafeMemAllocator()

```
virtual ISafeMemAllocator * nvinfer2::safe::ISafePluginResourceContext::getSafeMemAllocator ( )
const  [pure virtual], [noexcept]
```

Returns the GPU memory allocator associated with this resource context.

The allocator is used by the plugin for GPU allocations during initResource() and execution.

Precondition

INIT API (e.g. can be called from IPluginV3OneSafeRuntime::initResource()).

Postcondition

none

Returns

Pointer to the allocator, or nullptr if not set.

Return values

non-null	When an allocator has been associated with this context.
nullptr	When no allocator is associated.

See also

IPluginV3OneSafeRuntime::initResource()

9.190.3.3 getSafeRecorder()

```
virtual ISafeRecorder * nvinfer2::safe::ISafePluginResourceContext::getSafeRecorder ( ) const
[pure virtual], [noexcept]
```

Returns the error recorder associated with this resource context.

The recorder receives error, warning, and diagnostic messages from the runtime for this context.

Precondition

RUNTIME API

Postcondition

none

Returns

Pointer to the recorder, or nullptr if not set.

Return values

non-null	When a recorder has been associated with this context.
nullptr	When no recorder is associated.

See also

IPluginV3OneSafeRuntime::initResource()

9.190.3.4 operator=() [1/2]

```
ISafePluginResourceContext & nvinfer2::safe::ISafePluginResourceContext::operator= (
            ISafePluginResourceContext &&  ) &  [protected], [default]
```

Move assignment operator (defaulted).

Moves state from the source; source is left in valid but unspecified state.

Precondition

none

Postcondition

This object has the state that the source had; source is in valid but unspecified state.

9.190.3.5 operator=() [2/2]

```
ISafePluginResourceContext & nvinfer2::safe::ISafePluginResourceContext::operator= (
            ISafePluginResourceContext const &  ) &  [protected], [default]
```

Copy assignment operator (defaulted).

Assigns a copy of the source to this object.

Precondition

none

Postcondition

This object is a copy of the source.

The documentation for this class was generated from the following file:

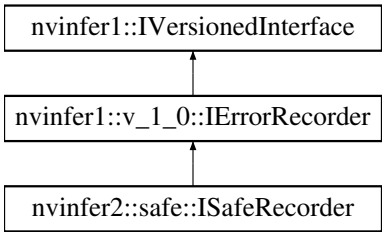
- NvInferSafePlugin.h

9.191 nvinfer2::safe::ISafeRecorder Class Reference

Interface for extended recorder which allows error, warn, debug, or info messages to be recorded.

```
#include <NvInferSafeRecorder.h>
```

Inheritance diagram for nvinfer2::safe::ISafeRecorder:



Public Member Functions

- ISafeRecorder (Severity severity_=Severity::kERROR, int32_t id_=-1)

  Constructor for ISafeRecorder.
- ISafeRecorder (ISafeRecorder const &)=delete

  Deleted copy constructor to maintain non-copyability.
- ISafeRecorder (ISafeRecorder &&)=delete

  Deleted move constructor to maintain non-movability.
- ISafeRecorder & operator= (ISafeRecorder const &) &=delete

  Deleted copy assignment operator to maintain non-copyability.
- ISafeRecorder & operator= (ISafeRecorder &&) &=delete

  Deleted move assignment operator to maintain non-movability.
- ~ISafeRecorder () noexcept override=default

  Destructor for ISafeRecorder.
- virtual void setSeverity (Severity severity_) noexcept

  Sets the severity level for the recorder.
- virtual Severity getSeverity () const noexcept

  Get the current severity level of the recorder.
- virtual int32_t getId () const noexcept

  Gets the identifier of the recorder.
- virtual bool reportWarn (ErrorDesc) noexcept

  Reports a warning message.
- virtual bool reportDebug (ErrorDesc) noexcept

  Reports a debug message.
- virtual bool reportInfo (ErrorDesc) noexcept

  Reports an informational message.
- virtual bool reportVerbose (ErrorDesc) noexcept

  Reports a verbose informational message.

Static Public Attributes

- static constexpr size_t kMAX_SAFE_DESC_LENGTH {4095U}

  The length limit for an error description in bytes, excluding the '\0' string terminator.

Protected Attributes

- Severity mSeverity

  Severity level of the recorder; messages above this level are filtered.
- int32_t const mId

  Identifier of the recorder instance; used to distinguish recorders in multi-threaded use.

Additional Inherited Members

9.191.1 Detailed Description

Interface for extended recorder which allows error, warn, debug, or info messages to be recorded.

Inherits from IErrorRecorder and extends functionality to include various levels of message severity.

Note

Any user implementation of this interface must ensure that error messages are always reported and not suppressed due to lower reporting severity.

9.191.2 Constructor & Destructor Documentation

9.191.2.1 ISafeRecorder() [1/3]

```
nvinfer2::safe::ISafeRecorder::ISafeRecorder (
            Severity severity_ = Severity::kERROR,
            int32_t id_ = -1 )  [inline]
```

Constructor for ISafeRecorder.

Precondition

INIT API

Parameters

in	*severity_*	Initial severity level for recording messages. Only messages with severity less than or equal to this level will be reported.
in	*id_*	Optional identifier for the recorder instance; used to identify messages issued by different recorders in multiple threads.

Postcondition

Object is constructed with the given severity and id; ready to receive messages.

9.191.2.2 ISafeRecorder() [2/3]

```
nvinfer2::safe::ISafeRecorder::ISafeRecorder (
            ISafeRecorder const &  )  [delete]
```

Deleted copy constructor to maintain non-copyability.

Precondition

none

Postcondition

none

9.191.2.3 ISafeRecorder() [3/3]

```
nvinfer2::safe::ISafeRecorder::ISafeRecorder (
            ISafeRecorder &&  )  [delete]
```

Deleted move constructor to maintain non-movability.

Precondition

none

Postcondition

none

9.191.2.4 ~ISafeRecorder()

```
nvinfer2::safe::ISafeRecorder::~ISafeRecorder ( )  [override], [default], [noexcept]
```

Destructor for ISafeRecorder.

Precondition

none

Postcondition

Object is destroyed and resources released.

9.191.3 Member Function Documentation

9.191.3.1 getId()

```
virtual int32_t nvinfer2::safe::ISafeRecorder::getId ( ) const  [inline], [virtual], [noexcept]
```

Gets the identifier of the recorder.

Precondition

RUNTIME API

Postcondition

none

Returns

The identifier of the recorder.

Remarks

This method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.191.3.2 getSeverity()

```
virtual Severity nvinfer2::safe::ISafeRecorder::getSeverity ( ) const  [inline], [virtual], [noexcept]
```

Get the current severity level of the recorder.

Precondition

RUNTIME API

Postcondition

none

Returns

The severity level of the recorder.

Remarks

This method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.191.3.3 operator=() [1/2]

```
ISafeRecorder & nvinfer2::safe::ISafeRecorder::operator= (
            ISafeRecorder &&  ) &  [delete]
```

Deleted move assignment operator to maintain non-movability.

Precondition

none

Postcondition

none

9.191.3.4 operator=() [2/2]

```
ISafeRecorder & nvinfer2::safe::ISafeRecorder::operator= (
            ISafeRecorder const &  ) &  [delete]
```

Deleted copy assignment operator to maintain non-copyability.

Precondition

none

Postcondition

none

9.191.3.5 reportDebug()

```
virtual bool nvinfer2::safe::ISafeRecorder::reportDebug (
            ErrorDesc  )  [inline], [virtual], [noexcept]
```

Reports a debug message.

Parameters

in	*desc*	Description of the debug information.

Precondition

RUNTIME API

Postcondition

If true, message was recorded; if false, recording failed or was suppressed.

Returns

True if the message was reported successfully, false otherwise.

Remarks

This method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.191.3.6 reportInfo()

```
virtual bool nvinfer2::safe::ISafeRecorder::reportInfo (
            ErrorDesc ) [inline], [virtual], [noexcept]
```

Reports an informational message.

Parameters

in	*desc*	Description of the information.

Precondition

RUNTIME API

Postcondition

If true, message was recorded; if false, recording failed or was suppressed.

Returns

True if the message was reported successfully, false otherwise.

Remarks

This method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.191.3.7 reportVerbose()

```
virtual bool nvinfer2::safe::ISafeRecorder::reportVerbose (
            ErrorDesc  )  [inline], [virtual], [noexcept]
```

Reports a verbose informational message.

Parameters

in	*desc*	Description of the information.

Precondition

RUNTIME API

Postcondition

If true, message was recorded; if false, recording failed or was suppressed.

Returns

True if the message was reported successfully, false otherwise.

Remarks

This method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.191.3.8 reportWarn()

```
virtual bool nvinfer2::safe::ISafeRecorder::reportWarn (
            ErrorDesc  )  [inline], [virtual], [noexcept]
```

Reports a warning message.

Parameters

in	*desc*	Description of the warning.

Precondition

RUNTIME API

Postcondition

If true, message was recorded; if false, recording failed or was suppressed.

Returns

True if the message was reported successfully, false otherwise.

Remarks

This method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.191.3.9 setSeverity()

```
virtual void nvinfer2::safe::ISafeRecorder::setSeverity (
            Severity severity_ ) [inline], [virtual], [noexcept]
```

Sets the severity level for the recorder.

Parameters

in	*severity*_	New severity level for recording messages.

Precondition

RUNTIME API

Postcondition

Recorder severity is set to `severity_`; subsequent messages filtered by this level.

Remarks

Not thread-safe

9.191.4 Member Data Documentation

9.191.4.1 kMAX_SAFE_DESC_LENGTH

```
constexpr size_t nvinfer2::safe::ISafeRecorder::kMAX_SAFE_DESC_LENGTH {4095U}  [static], [constexpr]
```

The length limit for an error description in bytes, excluding the '\0' string terminator.

Note

Only applicable to messages issued by the TensorRT safe runtime. User recorder implementation can use any size appropriate for the use case.

9.191.4.2 mId

```
int32_t const nvinfer2::safe::ISafeRecorder::mId  [protected]
```

Identifier of the recorder instance; used to distinguish recorders in multi-threaded use.

9.191.4.3 mSeverity

```
Severity nvinfer2::safe::ISafeRecorder::mSeverity  [protected]
```

Severity level of the recorder; messages above this level are filtered.

The documentation for this class was generated from the following file:

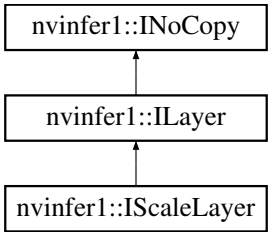
- NvInferSafeRecorder.h

9.192 nvinfer1::IScaleLayer Class Reference

A Scale layer in a network definition.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IScaleLayer:



Public Member Functions

- void setMode (ScaleMode mode) noexcept

  Set the scale mode.
- ScaleMode getMode () const noexcept

  Get the scale mode.
- void setShift (Weights shift) noexcept

  Set the shift value.
- Weights getShift () const noexcept

  Get the shift value.
- void setScale (Weights scale) noexcept

  Set the scale value.
- Weights getScale () const noexcept

  Get the scale value.
- void setPower (Weights power) noexcept

  Set the power value.
- Weights getPower () const noexcept

  Get the power value.
- int32_t getChannelAxis () const noexcept

  Get the channel axis.
- void setChannelAxis (int32_t channelAxis) noexcept

  Set the channel axis.

Protected Member Functions

- virtual ~IScaleLayer () noexcept=0

Protected Attributes

- apiv::VScaleLayer * mImpl

9.192.1 Detailed Description

A Scale layer in a network definition.

This layer applies a per-element computation to its input:

`output = (input* scale + shift)^ power`

The coefficients can be applied on a per-tensor, per-channel, or per-element basis.

Note

If the number of weights is 0, then a default value is used for shift, power, and scale. The default shift is 0, the default power is 1, and the default scale is 1.

The output size is the same as the input size.

Note

The input tensor is required to have at least 4 dimensions.

A scale layer may be used as an INT8 quantization node in a graph, if the output is constrained to INT8 and the input to FP32. Quantization rounds ties to even, and clamps to [-128, 127].

See also

ScaleMode

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.192.2 Constructor & Destructor Documentation

9.192.2.1 ~IScaleLayer()

```
nvinfer1::IScaleLayer::~IScaleLayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.192.3 Member Function Documentation

9.192.3.1 getChannelAxis()

```
int32_t nvinfer1::IScaleLayer::getChannelAxis ( ) const  [inline], [noexcept]
```

Get the channel axis.

Returns

channelAxis parameter passed to addScaleNd() or set by setChannelAxis()

The value is the index of the channel axis in the input tensor's dimensions. Scaling happens along the channel axis when ScaleMode::kCHANNEL is enabled.

See also

addScaleNd()

9.192.3.2 getMode()

```
ScaleMode nvinfer1::IScaleLayer::getMode ( ) const  [inline], [noexcept]
```

Get the scale mode.

See also

setMode()

9.192.3.3 getPower()

```
Weights nvinfer1::IScaleLayer::getPower ( ) const  [inline], [noexcept]
```

Get the power value.

See also

setPower()

9.192.3.4 getScale()

```
Weights nvinfer1::IScaleLayer::getScale ( ) const  [inline], [noexcept]
```

Get the scale value.

See also

setScale()

9.192.3.5 getShift()

```
Weights nvinfer1::IScaleLayer::getShift ( ) const  [inline], [noexcept]
```

Get the shift value.

See also

setShift()

9.192.3.6 setChannelAxis()

```
void nvinfer1::IScaleLayer::setChannelAxis (
            int32_t channelAxis )  [inline], [noexcept]
```

Set the channel axis.

The value is the index of the channel axis in the input tensor's dimensions.

For ScaleMode::kCHANNEL, there can be distinct scale, shift, and power weights for each channel coordinate. For ScaleMode::kELEMENTWISE, there can be distinct scale, shift, and power weights for each combination of coordinates from the channel axis and axes after it.

For example, suppose the input tensor has dimensions [10,20,30,40] and the channel axis is 1. Let [n,c,h,w] denote an input coordinate. For ScaleMode::kCHANNEL, the scale, shift, and power weights are indexed by c. For ScaleMode::kELEMENTWISE, the scale, shift, and power weights are indexed by [c,h,w].

See also

addScaleNd()

9.192.3.7 setMode()

```
void nvinfer1::IScaleLayer::setMode (
            ScaleMode mode )  [inline], [noexcept]
```

Set the scale mode.

See also

getMode()

9.192.3.8 setPower()

```
void nvinfer1::IScaleLayer::setPower (
            Weights power )  [inline], [noexcept]
```

Set the power value.

See also

getPower()

9.192.3.9 setScale()

```
void nvinfer1::IScaleLayer::setScale (
            Weights scale )  [inline], [noexcept]
```

Set the scale value.

See also

getScale()

9.192.3.10 setShift()

```
void nvinfer1::IScaleLayer::setShift (
            Weights shift )  [inline], [noexcept]
```

Set the shift value.

See also

getShift()

9.192.4 Member Data Documentation

9.192.4.1 mImpl

```
apiv::VScaleLayer* nvinfer1::IScaleLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

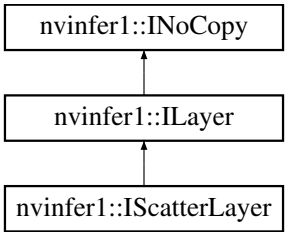
- NvInfer.h

9.193 nvinfer1::IScatterLayer Class Reference

A scatter layer in a network definition. Supports several kinds of scattering.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IScatterLayer:



Public Member Functions

- void setMode (ScatterMode mode) noexcept

  Set the scatter mode.
- ScatterMode getMode () const noexcept

  Get the scatter mode.
- void setAxis (int32_t axis) noexcept

  Set the axis used by ScatterMode::kELEMENTS.
- int32_t getAxis () const noexcept

  Get the axis.

Protected Member Functions

- virtual ~IScatterLayer () noexcept=0

Protected Attributes

- apiv::VScatterLayer * mImpl

9.193.1 Detailed Description

A scatter layer in a network definition. Supports several kinds of scattering.

The Scatter layer has three input tensors: Data, Indices, and Updates, one output tensor Output, and a scatter mode. When kELEMENT mode is used an optional axis parameter is available.

- Data is a tensor of rank r >= 1 that stores the values to be duplicated in Output.
- Indices is a tensor of rank q that determines which locations in Output to write new values to. Constraints on the rank q depend on the mode: ScatterMode::kND: q >= 1 ScatterMode::kELEMENT: q must be the same as r
- Updates is a tensor of rank s >= 1 that provides the data to write to Output specified by its corresponding location in Indices. Constraints on the rank of Updates depend on the mode: ScatterMode::kND: s = r + q - shape(Indices)[-1] - 1 Scattermode::kELEMENT: s = q = r
- Output is a tensor with the same dimensions as Data that stores the resulting values of the transformation. It must not be a shape tensor. The types of Data, Update, and Output shall be the same, and Indices shall be of type DataType::kINT32 or DataType::kINT64.

The output is computed by copying the data, and then updating elements of it based on indices. How Indices are interpreted depends upon the ScatterMode.

ScatterMode::kND

```
The indices are interpreted as a tensor of rank q-1 of indexing tuples.
The axis parameter is ignored.

Given that data dims are {d_0,...,d_{r-1}} and indices dims are {i_0,...,i_{q-1}},
define k = indices[q-1], it follows that updates dims are {i_0,...,i_{q-2},d_k,...,d_{r-1}}
The updating can be computed by:
    foreach slice in indices[i_0,...,i_{q-2}]
        output[indices[slice]] = updates[slice]
```

ScatterMode::kELEMENT

```
Here "axis" denotes the result of getAxis().

For each element X of indices:
    Let J denote a sequence for the subscripts of X
    Let K = sequence J with element [axis] replaced by X
    output[K] = updates[J]

For example, if indices has dimensions [N,C,H,W] and axis is 2, then the updates happen as:

    for n in [0,n)
        for c in [0,n)
            for h in [0,n)
                for w in [0,n)
                    output[n,c,indices[n,c,h,w],w] = updates[n,c,h,w]
```

Writes to the same output element cause undefined behavior.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.193.2 Constructor & Destructor Documentation

9.193.2.1 ~IScatterLayer()

```
nvinfer1::IScatterLayer::~IScatterLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.193.3 Member Function Documentation

9.193.3.1 getAxis()

```
int32_t nvinfer1::IScatterLayer::getAxis ( ) const  [inline], [noexcept]
```

Get the axis.

9.193.3.2 getMode()

```
ScatterMode nvinfer1::IScatterLayer::getMode ( ) const  [inline], [noexcept]
```

Get the scatter mode.

See also

setMode()

9.193.3.3 setAxis()

```
void nvinfer1::IScatterLayer::setAxis (
            int32_t axis )  [inline], [noexcept]
```

Set the axis used by ScatterMode::kELEMENTS.

The axis defaults to 0.

9.193.3.4 setMode()

```
void nvinfer1::IScatterLayer::setMode (
            ScatterMode mode )  [inline], [noexcept]
```

Set the scatter mode.

See also

getMode()

9.193.4 Member Data Documentation

9.193.4.1 mImpl

```
apiv::VScatterLayer* nvinfer1::IScatterLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

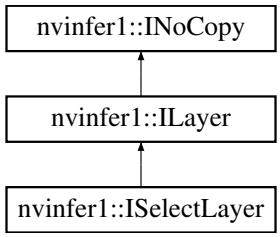
- NvInfer.h

9.194 nvinfer1::ISelectLayer Class Reference

Select elements from two data tensors based on a condition tensor.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::ISelectLayer:



Protected Member Functions

- virtual ~ISelectLayer () noexcept=0

Protected Attributes

- apiv::VSelectLayer * mImpl

Additional Inherited Members

9.194.1 Detailed Description

Select elements from two data tensors based on a condition tensor.

The select layer makes elementwise selections from two data tensors based on a condition tensor, behaving similarly to the `numpy.where` function with three parameters. The three input tensors must share the same rank. Multidirectional broadcasting is supported. The output tensor has the dimensions of the inputs AFTER applying the broadcast rule.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.194.2 Constructor & Destructor Documentation

9.194.2.1 ~ISelectLayer()

`nvinfer1::ISelectLayer::~ISelectLayer ( ) [inline], [protected], [pure virtual], [default], [noexcept]`

9.194.3 Member Data Documentation

9.194.3.1 mImpl

```
apiv::VSelectLayer* nvinfer1::ISelectLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

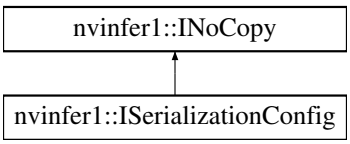
- NvInfer.h

9.195 nvinfer1::ISerializationConfig Class Reference

Holds properties for configuring an engine to serialize the binary.

```
#include <NvInferRuntime.h>
```

Inheritance diagram for nvinfer1::ISerializationConfig:



Public Member Functions

- virtual ~ISerializationConfig () noexcept=0
- bool setFlags (SerializationFlags serializationFlags) noexcept

  Set the serialization flags to turn on for this config.
- SerializationFlags getFlags () const noexcept

  Get the serialization flags for this config.
- bool clearFlag (SerializationFlag serializationFlag) noexcept

  clear a serialization flag.
- bool setFlag (SerializationFlag serializationFlag) noexcept

  Set a serialization flag.
- bool getFlag (SerializationFlag serializationFlag) const noexcept

  Returns true if the serialization flag is set.

Protected Attributes

- apiv::VSerializationConfig * mImpl

Additional Inherited Members

9.195.1 Detailed Description

Holds properties for configuring an engine to serialize the binary.

See also

SerializationFlag

9.195.2 Constructor & Destructor Documentation

9.195.2.1 ~ISerializationConfig()

```
nvinfer1::ISerializationConfig::~ISerializationConfig ( )  [inline], [pure virtual], [default],
[noexcept]
```

9.195.3 Member Function Documentation

9.195.3.1 clearFlag()

```
bool nvinfer1::ISerializationConfig::clearFlag (
            SerializationFlag serializationFlag )  [inline], [noexcept]
```

clear a serialization flag.

clears the serialization flag from the config.

See also

setFlags()

9.195.3.2 getFlag()

```
bool nvinfer1::ISerializationConfig::getFlag (
            SerializationFlag serializationFlag ) const  [inline], [noexcept]
```

Returns true if the serialization flag is set.

See also

getFlags()

Returns

True if flag is set, false if unset.

9.195.3.3 getFlags()

```
SerializationFlags nvinfer1::ISerializationConfig::getFlags ( ) const  [inline], [noexcept]
```

Get the serialization flags for this config.

Returns

The serialization flags as a bitmask.

See also

setFlags()

9.195.3.4 setFlag()

```
bool nvinfer1::ISerializationConfig::setFlag (
            SerializationFlag serializationFlag )  [inline], [noexcept]
```

Set a serialization flag.

Add the input serialization flag to the already enabled flags.

See also

setFlags()

9.195.3.5 setFlags()

```
bool nvinfer1::ISerializationConfig::setFlags (
            SerializationFlags serializationFlags )  [inline], [noexcept]
```

Set the serialization flags to turn on for this config.

The flags are listed in the SerializationFlag enum.

Parameters

serializationFlags	The serialization flags for an engine.

Note

This function will override the previous set flags, rather than bitwise ORing the new flag.

See also

getFlags()

9.195.4 Member Data Documentation

9.195.4.1 mImpl

```
apiv::VSerializationConfig* nvinfer1::ISerializationConfig::mImpl  [protected]
```

The documentation for this class was generated from the following file:

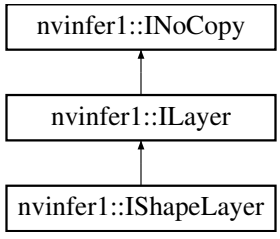
- NvInferRuntime.h

9.196 nvinfer1::IShapeLayer Class Reference

Layer type for getting shape of a tensor.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IShapeLayer:



Protected Member Functions

- virtual ~IShapeLayer () noexcept=0

Protected Attributes

- apiv::VShapeLayer * mImpl

Additional Inherited Members

9.196.1 Detailed Description

Layer type for getting shape of a tensor.

This layer sets the output to a 1D tensor of type Int64 with the dimensions of the input tensor.

For example, if the input is a four-dimensional tensor (of any type) with dimensions [2,3,5,7], the output tensor is a one-dimensional Int64 tensor of length 4 containing the sequence 2, 3, 5, 7.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.196.2 Constructor & Destructor Documentation

9.196.2.1 ~IShapeLayer()

```
nvinfer1::IShapeLayer::~IShapeLayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.196.3 Member Data Documentation

9.196.3.1 mImpl

```
apiv::VShapeLayer* nvinfer1::IShapeLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

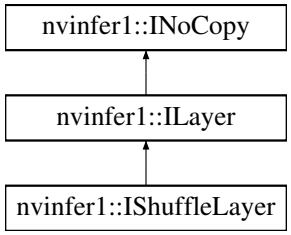
- NvInfer.h

9.197 nvinfer1::IShuffleLayer Class Reference

Layer type for shuffling data.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IShuffleLayer:



Public Member Functions

- void setFirstTranspose (Permutation permutation) noexcept

  Set the permutation applied by the first transpose operation.
- Permutation getFirstTranspose () const noexcept

  Get the permutation applied by the first transpose operation.
- void setReshapeDimensions (Dims const &dimensions) noexcept

  Set the reshaped dimensions.
- Dims getReshapeDimensions () const noexcept

  Get the reshaped dimensions.
- void setSecondTranspose (Permutation permutation) noexcept

  Set the permutation applied by the second transpose operation.
- Permutation getSecondTranspose () const noexcept

  Get the permutation applied by the second transpose operation.
- void setZeroIsPlaceholder (bool zeroIsPlaceholder) noexcept

  Set meaning of 0 in reshape dimensions.
- bool getZeroIsPlaceholder () const noexcept

  Get meaning of 0 in reshape dimensions.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~IShuffleLayer () noexcept=0

Protected Attributes

- apiv::VShuffleLayer * mImpl

9.197.1 Detailed Description

Layer type for shuffling data.

This layer shuffles data by applying in sequence: a transpose operation, a reshape operation and a second transpose operation. The dimension types of the output are those of the reshape dimension.

The layer has an optional second input. If present, it must be a 1D tensor of type Int32 or Int64, and the reshape dimensions are taken from it.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.197.2 Constructor & Destructor Documentation

9.197.2.1 ~IShuffleLayer()

```
nvinfer1::IShuffleLayer::~IShuffleLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.197.3 Member Function Documentation

9.197.3.1 getFirstTranspose()

```
Permutation nvinfer1::IShuffleLayer::getFirstTranspose ( ) const  [inline], [noexcept]
```

Get the permutation applied by the first transpose operation.

Returns

The dimension permutation applied before the reshape.

See also

setFirstTranspose

9.197.3.2 getReshapeDimensions()

```
Dims nvinfer1::IShuffleLayer::getReshapeDimensions ( ) const  [inline], [noexcept]
```

Get the reshaped dimensions.

Returns

The reshaped dimensions.

If a second input is present and non-null, or setReshapeDimensions has not yet been called, this function returns Dims with nbDims == -1.

9.197.3.3 getSecondTranspose()

```
Permutation nvinfer1::IShuffleLayer::getSecondTranspose ( ) const  [inline], [noexcept]
```

Get the permutation applied by the second transpose operation.

Returns

The dimension permutation applied after the reshape.

See also

setSecondTranspose

9.197.3.4 getZeroIsPlaceholder()

```
bool nvinfer1::IShuffleLayer::getZeroIsPlaceholder ( ) const  [inline], [noexcept]
```

Get meaning of 0 in reshape dimensions.

Returns

true if 0 is placeholder for corresponding input dimension, false if 0 denotes a zero-length dimension.

See also

setZeroIsPlaceholder

9.197.3.5 setFirstTranspose()

```
void nvinfer1::IShuffleLayer::setFirstTranspose (
            Permutation permutation )  [inline], [noexcept]
```

Set the permutation applied by the first transpose operation.

Parameters

permutation	The dimension permutation applied before the reshape.

The default is the identity permutation.

See also

getFirstTranspose

9.197.3.6 setInput()

```
void nvinfer1::ILayer::setInput (
             int32_t index,
             ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor Sets the input tensor for the given index. The index must be 0 for a static shuffle layer. A static shuffle layer is converted to a dynamic shuffle layer by calling setInput with an index 1. A dynamic shuffle layer cannot be converted back to a static shuffle layer.

For a dynamic shuffle layer, the values 0 and 1 are valid. The indices in the dynamic case are as follows:

- 0: Data or Shape tensor to be shuffled.
- 1: The dimensions for the reshape operation, as a 1D tensor of type Int32 or Int64.

If this function is called with the value 1, then the function getNbInputs() changes from returning 1 to 2.

The reshape dimensions are treated identically to how they are treated if set statically via setReshapeDimensions. In particular, a -1 is treated as a wildcard even if dynamically supplied at runtime, and a 0 is treated as a placeholder if getZeroIsPlaceholder() = true, which is the default. If the placeholder interpretation of 0 is unwanted because the runtime dimension should be 0 when the reshape dimension is 0, be sure to call setZeroIsPlacholder(false) on the IShuffleLayer.

See also

setReshapeDimensions.

9.197.3.7 setReshapeDimensions()

```
void nvinfer1::IShuffleLayer::setReshapeDimensions (
            Dims const & dimensions )  [inline], [noexcept]
```

Set the reshaped dimensions.

Parameters

dimensions	The reshaped dimensions.

Two special values can be used as dimensions.

Value 0 copies the corresponding dimension from input. This special value can be used more than once in the dimensions. If number of reshape dimensions is less than input, 0s are resolved by aligning the most significant dimensions of input.

Value -1 infers that particular dimension by looking at input and rest of the reshape dimensions. Note that only a maximum of one dimension is permitted to be specified as -1. Avoid using -1 if the input can have zero volume and any of the other reshape dimensions can be zero (after resolving special treatment of 0), because the solution for the -1 becomes indeterminate and TensorRT will report an error.

The product of the new dimensions must be equal to the product of the old.

If a second input had been used to create this layer, that input is reset to null by this method.

9.197.3.8 setSecondTranspose()

```
void nvinfer1::IShuffleLayer::setSecondTranspose (
            Permutation permutation )  [inline], [noexcept]
```

Set the permutation applied by the second transpose operation.

Parameters

permutation	The dimension permutation applied after the reshape.

The default is the identity permutation.

The permutation is applied as outputDimensionIndex = permutation.order[inputDimensionIndex], so to permute from CHW order to HWC order, the required permutation is [1, 2, 0].

See also

getSecondTranspose

9.197.3.9 setZeroIsPlaceholder()

```
void nvinfer1::IShuffleLayer::setZeroIsPlaceholder (
            bool zeroIsPlaceholder )  [inline], [noexcept]
```

Set meaning of 0 in reshape dimensions.

If true, then a 0 in the reshape dimensions denotes copying the corresponding dimension from the first input tensor. If false, then a 0 in the reshape dimensions denotes a zero-length dimension.

Default: true

See also

getZeroIsPlaceholder();

9.197.4 Member Data Documentation

9.197.4.1 mImpl

```
apiv::VShuffleLayer* nvinfer1::IShuffleLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

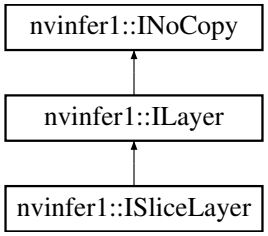
- NvInfer.h

9.198 nvinfer1::ISliceLayer Class Reference

Slices an input tensor into an output tensor based on the offset and strides.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::ISliceLayer:



Public Member Functions

- void setStart (Dims const &start) noexcept

  Set the start offset that the slice layer uses to create the output slice.
- Dims getStart () const noexcept

  Get the start offset for the slice layer.
- void setSize (Dims const &size) noexcept

  Set the dimensions of the output slice.
- Dims getSize () const noexcept

  Get dimensions of the output slice.
- void setStride (Dims const &stride) noexcept

  Set the stride for computing the output slice data.
- Dims getStride () const noexcept

  Get the stride for the output slice.
- void setMode (SampleMode mode) noexcept

  Set the slice mode.
- SampleMode getMode () const noexcept

  Get the slice mode.
- void setAxes (Dims const &axes) noexcept

  Set the axes for this ISliceLayer.
- Dims getAxes () const noexcept

  Get the axes for this ISliceLayer.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~ISliceLayer () noexcept=0

Protected Attributes

- apiv::VSliceLayer * mImpl

9.198.1 Detailed Description

Slices an input tensor into an output tensor based on the offset and strides.

The slice layer has two variants, static and dynamic. Static slice specifies the start, size, and stride dimensions at layer creation time via Dims and can use the get/set accessor functions of the ISliceLayer. Static slice layers can also optionally specify axes through the get/set accessor functions of the ISliceLayer. Dynamic slice specifies one or more of start, size, stride, or axes as ITensors, by using ILayer::setInput to add a second, third, fourth, or sixth input respectively. The corresponding Dims are used if an input is missing or null.

An application can determine if the ISliceLayer has a dynamic output shape based on whether the size or axes input is present and non-null.

The slice layer selects for each dimension a start location from within the input tensor, and copies elements to the output tensor using the specified stride across the input tensor. Start, size, and stride tensors must be 1D tensors of type Int32 or Int64 if not specified via Dims.

An example of using slice on a tensor: input = {{0, 2, 4}, {1, 3, 5}} start = {1, 0} size = {1, 2} stride = {1, 2} output = {{1, 5}}

If axes are provided then starts, ends, and strides must have the same length as axes and specifies a subset of dimensions to slice. If axes are not provided, starts, ends, and strides must be of the same length as the rank of the input tensor.

An example of using slice on a tensor with axes specified: input = {{0, 2, 4}, {1, 3, 5}} start = {1} size = {2} stride = {1} axes = {1} output = {{2, 4}, {3, 5}}

When the sampleMode is kCLAMP or kREFLECT, for each input dimension, if its size is 0 then the corresponding output dimension must be 0 too.

When the sampleMode is kFILL, the fifth input to the slice layer is used to determine the value to fill in out-of-bound indices. It is an error to specify the fifth input in any other sampleMode.

A slice layer can produce a shape tensor if the following conditions are met:

- start, size, and stride are build time constants, either as static Dims or as constant input tensors.
- axes, if provided, are build time constants, either as static Dims or as a constant input tensor.
- The number of elements in the output tensor does not exceed 2 ∗ Dims::MAX_DIMS.

The input tensor is a shape tensor if the output is a shape tensor.

The following constraints must be satisfied to execute this layer on DLA:

- start, size, and stride are build time constants, either as static Dims or as constant input tensors.
- axes, if provided, are build time constants, either as static Dims or as a constant input tensor.
- sampleMode is kDEFAULT, kWRAP, or kFILL.
- Strides are 1 for all dimensions.
- Slicing is not performed on the first dimension.
- The input tensor has four dimensions.
- For kFILL sliceMode, the fill value input is a scalar output of an IConstantLayer with value 0 that is not consumed by any other layer.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.198.2 Constructor & Destructor Documentation

9.198.2.1 ~ISliceLayer()

```
nvinfer1::ISliceLayer::~ISliceLayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.198.3 Member Function Documentation

9.198.3.1 getAxes()

```
Dims nvinfer1::ISliceLayer::getAxes ( ) const  [inline], [noexcept]
```

Get the axes for this ISliceLayer.

Returns

The axes on which the starts, ends, and strides parameters of this slice apply to.

If the sixth input is present and non-null, this function returns a Dims with nbDims = -1.

See also

setAxes

9.198.3.2 getMode()

```
SampleMode nvinfer1::ISliceLayer::getMode ( ) const  [inline], [noexcept]
```

Get the slice mode.

See also

setMode()

9.198.3.3 getSize()

```
Dims nvinfer1::ISliceLayer::getSize ( ) const  [inline], [noexcept]
```

Get dimensions of the output slice.

Returns

The output dimension, or an invalid Dims structure.

If the third input is present and non-null, this function returns a Dims with nbDims = -1.

See also

setSize

9.198.3.4 getStart()

```
Dims nvinfer1::ISliceLayer::getStart ( ) const  [inline], [noexcept]
```

Get the start offset for the slice layer.

Returns

The start offset, or an invalid Dims structure.

If the second input is present and non-null, this function returns a Dims with nbDims = -1.

See also

setStart

9.198.3.5 getStride()

```
Dims nvinfer1::ISliceLayer::getStride ( ) const  [inline], [noexcept]
```

Get the stride for the output slice.

Returns

The slicing stride, or an invalid Dims structure.

If the fourth input is present and non-null, this function returns a Dims with nbDims = -1.

See also

setStride

9.198.3.6 setAxes()

```
void nvinfer1::ISliceLayer::setAxes (
            Dims const & axes )  [inline], [noexcept]
```

Set the axes for this ISliceLayer.

Parameters

axes	The axes on which the starts, ends, and strides parameters of the slice apply to.

If a sixth input had been used to create this layer, that input is reset to null by this method.

See also

getAxes

9.198.3.7 setInput()

```
void nvinfer1::ILayer::setInput (
             int32_t index,
             ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor

For a slice layer, the values 0-5 are valid. The indices are as follows:

- 0: Tensor to be sliced.
- 1: The start tensor to begin slicing, as a 1D tensor of type Int32 or Int64.
- 2: The size tensor of the resulting slice, as a 1D tensor of type Int32 or Int64.
- 3: The stride of the slicing operation, as a 1D tensor of type Int32 or Int64.
- 4: Value for the kFILL slice mode. The fill value data type should either be the same or be implicitly convertible to the input data type. Implicit data type conversion is supported among kFLOAT, kHALF, kINT8, and kFP8 data types. This input is disallowed for other modes.
- 5: The axes tensor indicating the corresponding axes that start, size, and stride should apply to, as a 1D tensor or type Int32 or Int64. Negative values for axes indicate indexing from the back of the input tensor. Values must be unique and be within the interval of [-rank(input), rank(input)-1].

Using the corresponding setter resets the input to null.

If this function is called with a value greater than 0, then the function getNbInputs() changes from returning 1 to index + 1.

9.198.3.8 setMode()

```
void nvinfer1::ISliceLayer::setMode (
            SampleMode mode )  [inline], [noexcept]
```

Set the slice mode.

See also

getMode()

9.198.3.9 setSize()

```
void nvinfer1::ISliceLayer::setSize (
            Dims const & size )  [inline], [noexcept]
```

Set the dimensions of the output slice.

Parameters

size	The dimensions of the output slice.

If a third input had been used to create this layer, that input is reset to null by this method.

See also

getSize

9.198.3.10 setStart()

```
void nvinfer1::ISliceLayer::setStart (
            Dims const & start )  [inline], [noexcept]
```

Set the start offset that the slice layer uses to create the output slice.

Parameters

start	The start offset to read data from the input tensor.

If a second input had been used to create this layer, that input is reset to null by this method.

See also

getStart

9.198.3.11 setStride()

```
void nvinfer1::ISliceLayer::setStride (
            Dims const & stride )  [inline], [noexcept]
```

Set the stride for computing the output slice data.

Parameters

stride	The dimensions of the stride to compute the values to store in the output slice.

If a fourth input had been used to create this layer, that input is reset to null by this method.

See also

getStride

9.198.4 Member Data Documentation

9.198.4.1 mImpl

```
apiv::VSliceLayer* nvinfer1::ISliceLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

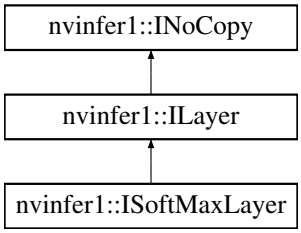
- NvInfer.h

9.199 nvinfer1::ISoftMaxLayer Class Reference

A Softmax layer in a network definition.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::ISoftMaxLayer:



Public Member Functions

- void setAxes (uint32_t axes) noexcept

  Set the axis along which softmax is computed. Currently, only one axis can be set.
- uint32_t getAxes () const noexcept

  Get the axis along which softmax occurs.

Protected Member Functions

- virtual ~ISoftMaxLayer () noexcept=0

Protected Attributes

- apiv::VSoftMaxLayer * mImpl

9.199.1 Detailed Description

A Softmax layer in a network definition.

This layer applies a per-channel softmax to its input.

The output size is the same as the input size.

The following constraints must be satisfied to execute this layer on DLA:

- Axis must be one of the channel or spatial dimensions.
- There are two classes of supported input sizes:
  1. Non-axis, non-batch dimensions are all 1 and the axis dimension is at most 8192. This is the recommended case for using softmax since it is the most accurate.
  2. At least one non-axis, non-batch dimension greater than 1 and the axis dimension is at most 1024. Note that in this case, there may be some approximation error as the axis dimension size approaches the upper bound. See the TensorRT Developer Guide for more details on the approximation error.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.199.2 Constructor & Destructor Documentation

9.199.2.1 ~ISoftMaxLayer()

```
nvinfer1::ISoftMaxLayer::~ISoftMaxLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.199.3 Member Function Documentation

9.199.3.1 getAxes()

```
uint32_t nvinfer1::ISoftMaxLayer::getAxes ( ) const  [inline], [noexcept]
```

Get the axis along which softmax occurs.

See also

setAxes()

9.199.3.2 setAxes()

```
void nvinfer1::ISoftMaxLayer::setAxes (
            uint32_t axes )  [inline], [noexcept]
```

Set the axis along which softmax is computed. Currently, only one axis can be set.

The axis is specified by setting the bit corresponding to the axis to 1. For example, consider an NCHW tensor as input.

Bit 0 corresponds to the N dimension boolean. Bit 1 corresponds to the C dimension boolean. Bit 2 corresponds to the H dimension boolean. Bit 3 corresponds to the W dimension boolean. By default, softmax is performed on the axis which is the number of axes minus three. It is 0 if there are fewer than 3 axes. For example, if the input is NCHW, the default axis is C. If the input is NHW, then the default axis is N.

For example, to perform softmax on axis R of a NPQRCHW input, set bit 3.

Parameters

axes	The axis along which softmax is computed. Here axes is a bitmap. For example, when doing softmax along axis 0, bit 0 is set to 1, axes = 1 << axis = 1.

9.199.4 Member Data Documentation

9.199.4.1 mImpl

```
apiv::VSoftMaxLayer* nvinfer1::ISoftMaxLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

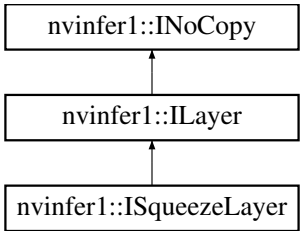
- NvInfer.h

9.200 nvinfer1::ISqueezeLayer Class Reference

Layer that represents a squeeze operation, removing unit dimensions of the first input tensor on a set of axes specified by the second input tensor.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::ISqueezeLayer:



Public Member Functions

- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~ISqueezeLayer () noexcept=0

Protected Attributes

- apiv::VSqueezeLayer * mImpl

9.200.1 Detailed Description

Layer that represents a squeeze operation, removing unit dimensions of the first input tensor on a set of axes specified by the second input tensor.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.200.2 Constructor & Destructor Documentation

9.200.2.1 ~ISqueezeLayer()

```
nvinfer1::ISqueezeLayer::~ISqueezeLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.200.3 Member Function Documentation

9.200.3.1 setInput()

```
void nvinfer1::ILayer::setInput (
            int32_t index,
            ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	The index of the input to modify.
tensor	The new input tensor.

For a Squeeze layer, the values 0-1 are valid for index. The indices are as follows:

- 0: Input data tensor.
- 1: The axes to remove. Must resolve to a constant Int32 or Int64 1D shape tensor.

9.200.4 Member Data Documentation

9.200.4.1 mImpl

```
apiv::VSqueezeLayer* nvinfer1::ISqueezeLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.201 IStreamReader Class Reference

Application-implemented class for reading data in a stream-based manner.

`#include <NvInferRuntime.h>`

9.201.1 Detailed Description

Application-implemented class for reading data in a stream-based manner.

Note

To ensure compatibility of source code with future versions of TensorRT, use IStreamReader, not v_1_0::↩ IStreamReader

Deprecated Deprecated in TensorRT 11.0. Superseded by IStreamReaderV2.

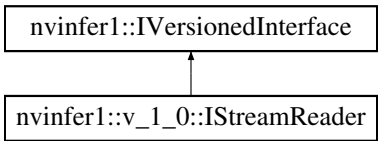
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.202 nvinfer1::v_1_0::IStreamReader Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IStreamReader:



Public Member Functions

- ~IStreamReader () override=default
- IStreamReader ()=default
- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual int64_t read (void *destination, int64_t nbBytes)=0

  Read the next number of bytes in the stream.

Protected Member Functions

- IStreamReader (IStreamReader const &)=default
- IStreamReader (IStreamReader &&)=default
- IStreamReader & operator= (IStreamReader const &) &=default
- IStreamReader & operator= (IStreamReader &&) &=default

9.202.1 Detailed Description

Deprecated Deprecated in TensorRT 11.0. Superseded by IStreamReaderV2.

9.202.2 Constructor & Destructor Documentation

9.202.2.1 ~IStreamReader()

```
nvinfer1::v_1_0::IStreamReader::~IStreamReader ( )  [override], [default]
```

TensorRT never calls the destructor for an IStreamReader defined by the application.

9.202.2.2 IStreamReader() [1/3]

```
nvinfer1::v_1_0::IStreamReader::IStreamReader ( )  [default]
```

9.202.2.3 IStreamReader() [2/3]

```
nvinfer1::v_1_0::IStreamReader::IStreamReader (
            IStreamReader const &  )  [protected], [default]
```

9.202.2.4 IStreamReader() [3/3]

```
nvinfer1::v_1_0::IStreamReader::IStreamReader (
            IStreamReader &&  )  [protected], [default]
```

9.202.3 Member Function Documentation

9.202.3.1 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IStreamReader::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.202.3.2 operator=() [1/2]

```
IStreamReader & nvinfer1::v_1_0::IStreamReader::operator= (
            IStreamReader &&  ) &  [protected], [default]
```

9.202.3.3 operator=() [2/2]

```
IStreamReader & nvinfer1::v_1_0::IStreamReader::operator= (
            IStreamReader const &  ) &  [protected], [default]
```

9.202.3.4 read()

```
virtual int64_t nvinfer1::v_1_0::IStreamReader::read (
            void * destination,
            int64_t nbBytes )  [pure virtual]
```

Read the next number of bytes in the stream.

Parameters

destination	The memory to write to
nbBytes	The number of bytes to read

Returns

The number of bytes read. Negative values will be considered an automatic error.

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.203 IStreamReaderV2 Class Reference

Application-implemented class for reading data in a stream-based manner asynchronously. Intended for use with the GDS API for optimizing load times.

`#include <NvInferRuntime.h>`

9.203.1 Detailed Description

Application-implemented class for reading data in a stream-based manner asynchronously. Intended for use with the GDS API for optimizing load times.

Note

To ensure compatibility of source code with future versions of TensorRT, use IStreamReaderV2, not v_1_0::IStreamReaderV2

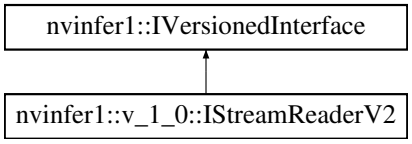
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.204 nvinfer1::v_1_0::IStreamReaderV2 Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IStreamReaderV2:



Public Member Functions

- ~IStreamReaderV2 () override=default
- IStreamReaderV2 ()=default
- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual int64_t read (void *destination, int64_t nbBytes, cudaStream_t stream) noexcept=0

  Read the next number of bytes in the stream asynchronously.
- virtual bool seek (int64_t offset, SeekPosition where) noexcept=0

  Sets the position of the stream to the given offset.

Protected Member Functions

- IStreamReaderV2 (IStreamReaderV2 const &)=default
- IStreamReaderV2 (IStreamReaderV2 &&)=default
- IStreamReaderV2 & operator= (IStreamReaderV2 const &) &=default
- IStreamReaderV2 & operator= (IStreamReaderV2 &&) &=default

9.204.1 Constructor & Destructor Documentation

9.204.1.1 ~IStreamReaderV2()

```
nvinfer1::v_1_0::IStreamReaderV2::~IStreamReaderV2 ( )  [override], [default]
```

TensorRT never calls the destructor for an IStreamReaderV2 defined by the application.

9.204.1.2 IStreamReaderV2() [1/3]

```
nvinfer1::v_1_0::IStreamReaderV2::IStreamReaderV2 ( )  [default]
```

9.204.1.3 IStreamReaderV2() [2/3]

```
nvinfer1::v_1_0::IStreamReaderV2::IStreamReaderV2 (
            IStreamReaderV2 const &  )  [protected], [default]
```

9.204.1.4 IStreamReaderV2() [3/3]

```
nvinfer1::v_1_0::IStreamReaderV2::IStreamReaderV2 (
            IStreamReaderV2 &&  )  [protected], [default]
```

9.204.2 Member Function Documentation

9.204.2.1 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IStreamReaderV2::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.204.2.2 operator=() [1/2]

```
IStreamReaderV2 & nvinfer1::v_1_0::IStreamReaderV2::operator= (
            IStreamReaderV2 &&  ) &  [protected], [default]
```

9.204.2.3 operator=() [2/2]

```
IStreamReaderV2 & nvinfer1::v_1_0::IStreamReaderV2::operator= (
            IStreamReaderV2 const &  ) &  [protected], [default]
```

9.204.2.4 read()

```
virtual int64_t nvinfer1::v_1_0::IStreamReaderV2::read (
            void * destination,
            int64_t nbBytes,
            cudaStream_t stream )  [pure virtual], [noexcept]
```

Read the next number of bytes in the stream asynchronously.

Parameters

destination	The memory to write to, call cudaPointerGetAttributes to get the memory location
nbBytes	The number of bytes to read
stream	The CUDA stream used to do the copy

Returns

The number of bytes read. Negative values indicate an unrecoverable error. A zero indicates that the end of the stream has been reached.

9.204.2.5 seek()

```
virtual bool nvinfer1::v_1_0::IStreamReaderV2::seek (
            int64_t offset,
            SeekPosition where )  [pure virtual], [noexcept]
```

Sets the position of the stream to the given offset.

Parameters

offset	The number of bytes to offset from where.
where	The position from where the offset is added.

See also

SeekPosition

Returns

True if the position is updated successfully.

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.205 IStreamWriter Class Reference

Application-implemented class for writing data in a stream-based manner.

```
#include <NvInferRuntime.h>
```

9.205.1 Detailed Description

Application-implemented class for writing data in a stream-based manner.

Note

To ensure compatibility of source code with future versions of TensorRT, use IStreamWriter, not v_1_0::IStream↩ Writer

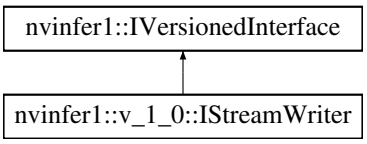
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.206 nvinfer1::v_1_0::IStreamWriter Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IStreamWriter:



Public Member Functions

- ~IStreamWriter () override=default
- IStreamWriter ()=default
- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual int64_t write (void const *data, int64_t nbBytes)=0

  write nbBytes of data into the stream.

Protected Member Functions

- IStreamWriter (IStreamWriter const &)=default
- IStreamWriter (IStreamWriter &&)=default
- IStreamWriter & operator= (IStreamWriter const &) &=default
- IStreamWriter & operator= (IStreamWriter &&) &=default

9.206.1 Constructor & Destructor Documentation

9.206.1.1 ~IStreamWriter()

`nvinfer1::v_1_0::IStreamWriter::~IStreamWriter ( ) [override], [default]`

TensorRT never calls the destructor for an IStreamWriter defined by the application.

9.206.1.2 IStreamWriter() [1/3]

`nvinfer1::v_1_0::IStreamWriter::IStreamWriter ( ) [default]`

9.206.1.3 IStreamWriter() [2/3]

```
nvinfer1::v_1_0::IStreamWriter::IStreamWriter (
            IStreamWriter const &  )  [protected], [default]
```

9.206.1.4 IStreamWriter() [3/3]

```
nvinfer1::v_1_0::IStreamWriter::IStreamWriter (
            IStreamWriter &&  )  [protected], [default]
```

9.206.2 Member Function Documentation

9.206.2.1 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IStreamWriter::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.206.2.2 operator=() [1/2]

```
IStreamWriter & nvinfer1::v_1_0::IStreamWriter::operator= (
            IStreamWriter &&  ) &  [protected], [default]
```

9.206.2.3 operator=() [2/2]

```
IStreamWriter & nvinfer1::v_1_0::IStreamWriter::operator= (
            IStreamWriter const &  ) &  [protected], [default]
```

9.206.2.4 write()

```
virtual int64_t nvinfer1::v_1_0::IStreamWriter::write (
            void const * data,
            int64_t nbBytes )  [pure virtual]
```

write nbBytes of data into the stream.

Parameters

data	The data to be written to stream
nbBytes	The number of bytes to write

Returns

The number of bytes written. A value that is negative or less than nBytes indicates that an error occurred and TensorRT will give up on writing to the stream.

The documentation for this class was generated from the following file:

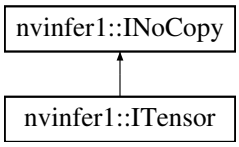
- NvInferRuntime.h

9.207 nvinfer1::ITensor Class Reference

A tensor in a network definition.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::ITensor:



Public Member Functions

- void setName (char const *name) noexcept

  Set the tensor name.
- char const * getName () const noexcept

  Get the tensor name.
- void setDimensions (Dims const &dimensions) noexcept

  Set the dimensions of a tensor.
- Dims getDimensions () const noexcept

  Get the dimensions of a tensor.
- DataType getType () const noexcept

  Get the data type of a tensor.
- bool isNetworkInput () const noexcept

  Whether the tensor is a network input.
- bool isNetworkOutput () const noexcept

  Whether the tensor is a network output.
- TRT_DEPRECATED void setBroadcastAcrossBatch (bool broadcastAcrossBatch) noexcept

Set whether to enable broadcast of tensor across the implicit batch dimension.

- TRT_DEPRECATED bool getBroadcastAcrossBatch () const noexcept

  Check if tensor is broadcast across the implicit batch dimension.

- TensorLocation getLocation () const noexcept

  Get the storage location of a tensor.

- TRT_DEPRECATED void setLocation (TensorLocation location) noexcept

  Set the storage location of a tensor.

- void setAllowedFormats (TensorFormats formats) noexcept

  Set allowed formats for an input or output tensor. By default all formats are allowed. Shape tensors (for which isShapeTensor() returns true) may only have row-major linear format.

- TensorFormats getAllowedFormats () const noexcept

  Get a bitmask of TensorFormat values that the tensor supports. For a shape tensor, only row-major linear format is allowed.

- bool isShapeTensor () const noexcept

  Whether the tensor is a shape tensor.

- bool isExecutionTensor () const noexcept

  Whether the tensor is an execution tensor.

- void setDimensionName (int32_t index, char const *name) noexcept

  Name a dimension of an input tensor.

- char const * getDimensionName (int32_t index) const noexcept

  Get the name of an input dimension.

Protected Member Functions

- virtual ~ITensor () noexcept=0

Protected Attributes

- apiv::VTensor * mImpl

9.207.1 Detailed Description

A tensor in a network definition.

To remove a tensor from a network definition, use INetworkDefinition::removeTensor().

When using the DLA, the cumulative size of all Tensors that are not marked as Network Input or Output tensors, must be less than 1GB in size to fit into a single subgraph. If the build option kGPU_FALLBACK is specified, then multiple subgraphs can be created, with each subgraph limited to less than 1GB of internal tensors data.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.207.2 Constructor & Destructor Documentation

9.207.2.1 ~ITensor()

```
nvinfer1::ITensor::~ITensor ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.207.3 Member Function Documentation

9.207.3.1 getAllowedFormats()

```
TensorFormats nvinfer1::ITensor::getAllowedFormats ( ) const  [inline], [noexcept]
```

Get a bitmask of TensorFormat values that the tensor supports. For a shape tensor, only row-major linear format is allowed.

Returns

The value specified by setAllowedFormats or all possible formats.

See also

ITensor::setAllowedFormats()

9.207.3.2 getBroadcastAcrossBatch()

```
TRT_DEPRECATED bool nvinfer1::ITensor::getBroadcastAcrossBatch ( ) const  [inline], [noexcept]
```

Check if tensor is broadcast across the implicit batch dimension.

Returns

Always false since TensorRT 10.0 does not support an implicit batch dimension.

See also

setBroadcastAcrossBatch()

Deprecated Deprecated in TensorRT 10.0. Implicit batch is not supported since TensorRT 10.0.

9.207.3.3 getDimensionName()

```
char const * nvinfer1::ITensor::getDimensionName (
            int32_t index ) const  [inline], [noexcept]
```

Get the name of an input dimension.

Parameters

index	index of the dimension

Returns

The name of the input dimension, or nullptr if the dimension has no name. The name is a pointer to a null-terminated character sequence.

See also

setDimensionName()

9.207.3.4 getDimensions()

```
Dims nvinfer1::ITensor::getDimensions ( ) const  [inline], [noexcept]
```

Get the dimensions of a tensor.

Returns

The dimensions of the tensor.

Warning

getDimensions() returns a -1 for dimensions that are derived from a wildcard dimension.

See also

setDimensions()

9.207.3.5 getLocation()

```
TensorLocation nvinfer1::ITensor::getLocation ( ) const  [inline], [noexcept]
```

Get the storage location of a tensor.

Returns

The location of tensor data.

See also

setLocation()

9.207.3.6 getName()

```
char const * nvinfer1::ITensor::getName ( ) const  [inline], [noexcept]
```

Get the tensor name.

Returns

The name as a null-terminated C-style string.

See also

setName()

9.207.3.7 getType()

```
DataType nvinfer1::ITensor::getType ( ) const  [inline], [noexcept]
```

Get the data type of a tensor.

Returns

The data type of the tensor.

The type is the inferred type.

See also

setType()

9.207.3.8 isExecutionTensor()

```
bool nvinfer1::ITensor::isExecutionTensor ( ) const  [inline], [noexcept]
```

Whether the tensor is an execution tensor.

Tensors are usually execution tensors. The exceptions are tensors used solely for shape calculations or whose contents are not needed to compute the outputs.

The result of isExecutionTensor() is reliable only when network construction is complete. For example, if a partially built network has no path from a tensor to a network output, isExecutionTensor() returns false. Completing the path would cause it to become true.

A tensor with isShapeTensor() == false and isExecutionTensor() == false can still show up as an input to the engine if its dimensions are required. In that case, only its dimensions need to be set at runtime and a nullptr can be passed instead of a pointer to its contents.

9.207.3.9 isNetworkInput()

```
bool nvinfer1::ITensor::isNetworkInput ( ) const  [inline], [noexcept]
```

Whether the tensor is a network input.

9.207.3.10 isNetworkOutput()

```
bool nvinfer1::ITensor::isNetworkOutput ( ) const  [inline], [noexcept]
```

Whether the tensor is a network output.

9.207.3.11 isShapeTensor()

```
bool nvinfer1::ITensor::isShapeTensor ( ) const  [inline], [noexcept]
```

Whether the tensor is a shape tensor.

A shape tensor is a tensor that is related to shape calculations. It must have type Int32, Int64, Bool, or Float, and its shape must be determinable at build time. Furthermore, it must be needed as a shape tensor, either marked as a network shape output via markOutputForShapes(), or as a layer input that is required to be a shape tensor, such as the second input to IShuffleLayer. Some layers are ”polymorphic” in this respect. For example, the inputs to IElementWiseLayer must be shape tensors if the output is a shape tensor.

The TensorRT Developer Guide gives the formal rules for what tensors are shape tensors.

The result of isShapeTensor() is reliable only when network construction is complete. For example, if a partially built network sums two tensors T1 and T2 to create tensor T3, and none are yet needed as shape tensors, isShapeTensor() returns false for all three tensors. Setting the second input of IShuffleLayer to be T3 would cause all three tensors to be shape tensors, because IShuffleLayer requires that its second optional input be a shape tensor, and IElementWiseLayer is ”polymorphic”.

It is possible for a tensor to be both a shape tensor and an execution tensor.

Returns

True if tensor is a shape tensor, false otherwise.

See also

INetworkDefinition::markOutputForShapes()

9.207.3.12 setAllowedFormats()

```
void nvinfer1::ITensor::setAllowedFormats (
            TensorFormats formats )  [inline], [noexcept]
```

Set allowed formats for an input or output tensor. By default all formats are allowed. Shape tensors (for which isShapeTensor() returns true) may only have row-major linear format.

When running network on DLA and the build option kGPU_FALLBACK is not specified, if DLA format(kCHW4 with Int8, kCHW4 with FP16, kCHW16 with FP16, kCHW32 with Int8) is set, the input format is treated as native DLA format with line stride requirement. Input/output binding with these format should have correct layout during inference.

Tensor formats are determined at build time by TensorRT for tensors not marked as input or output.

Parameters

formats	A bitmask of TensorFormat values that are supported for this tensor.

See also

ITensor::getAllowedFormats()

TensorFormats

9.207.3.13 setBroadcastAcrossBatch()

```
TRT_DEPRECATED void nvinfer1::ITensor::setBroadcastAcrossBatch (
            bool broadcastAcrossBatch )  [inline], [noexcept]
```

Set whether to enable broadcast of tensor across the implicit batch dimension.

Warning

This method has no effect other than issuing a warning.

Parameters

broadcastAcrossBatch	Whether to broadcast the tensor across the implicit batch dimension that was a feature of TensorRT 9.x and prior.

See also

getBroadcastAcrossBatch()

Deprecated Deprecated in TensorRT 10.0. Implicit batch is not supported since TensorRT 10.0.

9.207.3.14 setDimensionName()

```
void nvinfer1::ITensor::setDimensionName (
            int32_t index,
            char const * name )  [inline], [noexcept]
```

Name a dimension of an input tensor.

Associate a runtime dimension of an input tensor with a symbolic name. Dimensions with the same non-empty name must be equal at runtime. Knowing this equality for runtime dimensions may help the TensorRT optimizer. Both runtime and build-time dimensions can be named.

For example, setDimensionName(0, ”n”) associates the symbolic name ”n” with the leading dimension.

This method copies the name string. If the function is called again, with the same index, it will overwrite the previous name. If nullptr is passed as name, it will clear the name of the dimension.

Parameters

index	index of the dimension
name	of the dimension, as a pointer to a null-terminated character sequence.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

See also

getDimensionName()

9.207.3.15 setDimensions()

```
void nvinfer1::ITensor::setDimensions (
            Dims const & dimensions )  [inline], [noexcept]
```

Set the dimensions of a tensor.

For a network input, the dimensions are assigned by the application. For a network output, the dimensions are computed based on the layer parameters and the inputs to the layer. If a tensor size or a parameter is modified in the network, the dimensions of all dependent tensors will be recomputed.

This call is only legal for network input tensors, since the dimensions of layer output tensors are inferred based on layer inputs and parameters.

Parameters

dimensions	The dimensions of the tensor.

See also

getDimensions()

9.207.3.16 setLocation()

```
TRT_DEPRECATED void nvinfer1::ITensor::setLocation (
            TensorLocation location )  [inline], [noexcept]
```

Set the storage location of a tensor.

Parameters

location	the location of tensor data

Only network input tensors for storing sequence lengths for RNNv2 are supported. Using host storage for layers that do not support it will generate errors at build time.

See also

getLocation()

Deprecated Deprecated in TensorRT 10.0. RNNv2 is not supported and the location must always be TensorLocation::kDEVICE since TensorRT 10.0.

9.207.3.17 setName()

```
void nvinfer1::ITensor::setName (
            char const * name )  [inline], [noexcept]
```

Set the tensor name.

For a network input, the name is assigned by the application. For tensors which are layer outputs, a default name is assigned consisting of the layer name followed by the index of the output in brackets. Each input and output tensor must have a unique name.

This method copies the name string.

Parameters

name	The name.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

See also

getName()

9.207.4 Member Data Documentation

9.207.4.1 mImpl

```
apiv::VTensor* nvinfer1::ITensor::mImpl  [protected]
```

The documentation for this class was generated from the following file:

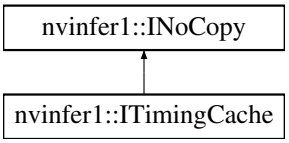
- NvInfer.h

9.208 nvinfer1::ITimingCache Class Reference

Class to handle tactic timing info collected from builder.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::ITimingCache:



Public Member Functions

- virtual ~ITimingCache () noexcept=0
- nvinfer1::IHostMemory * serialize () const noexcept

  Serialize a timing cache to IHostMemory object.
- bool combine (ITimingCache const &inputCache, bool ignoreMismatch) noexcept

  Combine input timing cache into local instance.
- bool reset () noexcept

  Empty the timing cache.
- int64_t queryKeys (TimingCacheKey *keyBuffer, int64_t capacity) const noexcept

  Query cache keys from Timing Cache.
- TimingCacheValue query (TimingCacheKey const &key) const noexcept

  Query value in a cache entry.
- bool update (TimingCacheKey const &key, TimingCacheValue const &value) noexcept

  Update values in a cache entry.

Protected Attributes

- apiv::VTimingCache * mImpl

Additional Inherited Members

9.208.1 Detailed Description

Class to handle tactic timing info collected from builder.

The timing cache is created or initialized by IBuilderConfig. It can be shared across builder instances to reduce the builder wallclock time.

Warning

Rebuilding the same engine multiple times using the same timing cache will always yield a correct engine but the selected tactics and formats may vary between generated engine instances, if weak typing is used.

See also

IBuilderConfig

9.208.2 Constructor & Destructor Documentation

9.208.2.1 ~ITimingCache()

```
nvinfer1::ITimingCache::~ITimingCache ( )  [inline], [pure virtual], [default], [noexcept]
```

9.208.3 Member Function Documentation

9.208.3.1 combine()

```
bool nvinfer1::ITimingCache::combine (
            ITimingCache const & inputCache,
            bool ignoreMismatch )  [inline], [noexcept]
```

Combine input timing cache into local instance.

This function allows combining entries in the input timing cache to local cache object.

Parameters

inputCache	The input timing cache.
ignoreMismatch	Whether or not to allow cache verification header mismatch.

Returns

True if combined successfully, false otherwise.

Append entries in input cache to local cache. Conflicting entries will be skipped The input cache must be generated by a TensorRT build of exact same version, otherwise combine will be skipped and return false. ignoreMismatch must be set to true if combining a timing cache created from a different device.

Warning

Combining caches generated from devices with different device properties may lead to functional/performance bugs!

9.208.3.2 query()

```
TimingCacheValue nvinfer1::ITimingCache::query (
            TimingCacheKey const & key ) const  [inline], [noexcept]
```

Query value in a cache entry.

The function queries the value in a specific cache entry.

Parameters

key	The query key.

Returns

Cache value if the key exists, otherwise an invalid value.

Query the value of the given cache key. If the key exists, write the value out, otherwise return an invalid value.

9.208.3.3 queryKeys()

```
int64_t nvinfer1::ITimingCache::queryKeys (
            TimingCacheKey * keyBuffer,
            int64_t capacity ) const  [inline], [noexcept]
```

Query cache keys from Timing Cache.

This function queries the entry count and writes the keys out.

Parameters

keyBuffer	The buffer to store keys.
capacity	The capacity of the buffer.

Returns

The count of entries in the cache and fill keys if keyBuffer is non-null. If an error occurs, -1 will be returned.

Query the count of entries in the cache and write out cache keys if keyBuffer is provided. Any key entries exceeding the capacity of the keyBuffer will not be copied.

9.208.3.4 reset()

```
bool nvinfer1::ITimingCache::reset ( )  [inline], [noexcept]
```

Empty the timing cache.

Returns

True if reset successfully, false otherwise.

9.208.3.5 serialize()

```
nvinfer1::IHostMemory * nvinfer1::ITimingCache::serialize ( ) const  [inline], [noexcept]
```

Serialize a timing cache to IHostMemory object.

This function allows serialization of current timing cache.

Returns

A pointer to a IHostMemory object that contains a serialized timing cache.

See also

IHostMemory

9.208.3.6 update()

```
bool nvinfer1::ITimingCache::update (
            TimingCacheKey const & key,
            TimingCacheValue const & value )  [inline], [noexcept]
```

Update values in a cache entry.

The function updates the value in a specific cache entry.

Parameters

key	The key to the entry to be updated.
value	New cache value.

Returns

True if update succeeds, otherwise false.

Update the value of the given cache key. If the key does not exist, return false. If the key exists and the new tactic timing is NaN, delete the cache entry and return true. If tactic timing is not NaN and the new value is valid, override the cache value and return true. False is returned when the new value is invalid. If this layer cannot use the new tactic, build errors will be reported when building the next engine.

9.208.4 Member Data Documentation

9.208.4.1 mImpl

```
apiv::VTimingCache* nvinfer1::ITimingCache::mImpl  [protected]
```

The documentation for this class was generated from the following file:

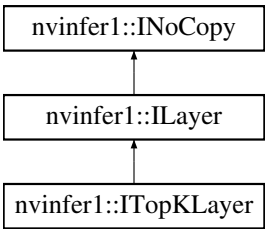
- NvInfer.h

9.209 nvinfer1::ITopKLayer Class Reference

Layer that represents a TopK reduction.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::ITopKLayer:



Public Member Functions

- void setOperation (TopKOperation op) noexcept

  Set the operation for the layer.
- TopKOperation getOperation () const noexcept

  Get the operation for the layer.
- void setK (int32_t k) noexcept

  Set the static k value for the layer.
- int32_t getK () const noexcept

  Get the k value for the layer.
- void setReduceAxes (uint32_t reduceAxes) noexcept

  Set which axes to reduce for the layer.
- uint32_t getReduceAxes () const noexcept

  Get the axes to reduce for the layer.
- bool setIndicesType (DataType type) noexcept

  Set the indices type for the layer.
- DataType getIndicesType () const noexcept

  Return the TopK layer indices type.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~ITopKLayer () noexcept=0

Protected Attributes

- apiv::VTopKLayer * mImpl

9.209.1 Detailed Description

Layer that represents a TopK reduction.

This layer can accept both static and dynamic k. Static k can be set through the addTopK() API function, or accessed using the getK() and setK() functions after layer creation. For dynamic k, use the setInput() method to pass in k as a tensor with index 1, which overrides the static k value in calculations.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.209.2 Constructor & Destructor Documentation

9.209.2.1 ~ITopKLayer()

```
nvinfer1::ITopKLayer::~ITopKLayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.209.3 Member Function Documentation

9.209.3.1 getIndicesType()

```
DataType nvinfer1::ITopKLayer::getIndicesType ( ) const  [inline], [noexcept]
```

Return the TopK layer indices type.

Returns

indices type set during layer creation or by setIndicesType(). The return value is the indices type of the TopK layer. The default value is DataType::kINT32.

9.209.3.2 getK()

```
int32_t nvinfer1::ITopKLayer::getK ( ) const  [inline], [noexcept]
```

Get the k value for the layer.

This function will return the static k value passed into addTopK(), or the value passed into setK().

If a second layer input is present and non-null, this function returns -1.

See also

setK()

9.209.3.3 getOperation()

```
TopKOperation nvinfer1::ITopKLayer::getOperation ( ) const  [inline], [noexcept]
```

Get the operation for the layer.

See also

setOperation(), TopKOperation

9.209.3.4 getReduceAxes()

```
uint32_t nvinfer1::ITopKLayer::getReduceAxes ( ) const  [inline], [noexcept]
```

Get the axes to reduce for the layer.

See also

setReduceAxes()

9.209.3.5 setIndicesType()

```
bool nvinfer1::ITopKLayer::setIndicesType (
            DataType type )  [inline], [noexcept]
```

Set the indices type for the layer.

Parameters

type	The DataType of the indices tensor.

Returns

true if set successfully, false otherwise.

Set the indices (the second output) type of the TopK layer. Valid values are DataType::kINT32 and DataType::kINT64, otherwise an error occurs and the type is not updated.

9.209.3.6 setInput()

```
void nvinfer1::ILayer::setInput (
            int32_t index,
            ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	The index of the input to modify.
tensor	The new input tensor.

For a TopK layer, the values 0-1 are valid. The indices are as follows:

- 0: Input data tensor.
- 1: A scalar Int32 tensor containing a positive value corresponding to the number of top elements to retrieve. Values larger than 3840 will result in a runtime error. If provided, this will override the static k value in calculations.

9.209.3.7 setK()

```
void nvinfer1::ITopKLayer::setK (
             int32_t k )  [inline], [noexcept]
```

Set the static k value for the layer.

Currently only values up to 3840 are supported.

If a second input to this layer has been set, it will be reset to null by this method.

See also

getK()

9.209.3.8 setOperation()

```
void nvinfer1::ITopKLayer::setOperation (
             TopKOperation op )  [inline], [noexcept]
```

Set the operation for the layer.

See also

getOperation(), TopKOperation

9.209.3.9 setReduceAxes()

```
void nvinfer1::ITopKLayer::setReduceAxes (
             uint32_t reduceAxes )  [inline], [noexcept]
```

Set which axes to reduce for the layer.

See also

getReduceAxes()

9.209.4 Member Data Documentation

9.209.4.1 mImpl

```
apiv::VTopKLayer* nvinfer1::ITopKLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

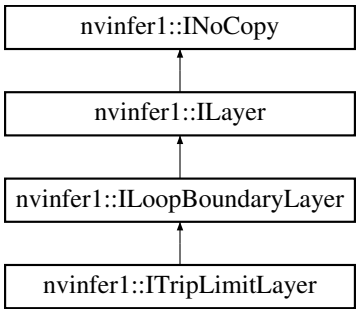
- NvInfer.h

9.210 nvinfer1::ITripLimitLayer Class Reference

A layer that represents a trip-count limiter.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::ITripLimitLayer:



Public Member Functions

- TripLimit getTripLimit () const noexcept
  Get a trip limiter type.

Protected Member Functions

- virtual ~ITripLimitLayer () noexcept=0

Protected Attributes

- apiv::VTripLimitLayer * mImpl

9.210.1 Detailed Description

A layer that represents a trip-count limiter.

The trip limit layer sets the execution condition for loops, using kCOUNT to define the number of iterations or k↩ WHILE for a conditional loop. A loop can have one of each kind of limit, in which case the loop exits when the trip count is reached or the condition becomes false.

See INetworkDefinition::addTripLimit().

9.210.2 Constructor & Destructor Documentation

9.210.2.1 ~ITripLimitLayer()

```
nvinfer1::ITripLimitLayer::~ITripLimitLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.210.3 Member Function Documentation

9.210.3.1 getTripLimit()

```
TripLimit nvinfer1::ITripLimitLayer::getTripLimit ( ) const  [inline], [noexcept]
```

Get a trip limiter type.

9.210.4 Member Data Documentation

9.210.4.1 mImpl

```
apiv::VTripLimitLayer* nvinfer1::ITripLimitLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.211 nvinfer2::safe::ITRTGraph Class Reference

Abstract Interface for a functionally safe graph for executing inference on a built network.

`#include <NvInferSafeRuntime.h>`

Public Member Functions

- ITRTGraph (ITRTGraph const &)=delete
- ITRTGraph (ITRTGraph &&)=delete
- ITRTGraph & operator= (ITRTGraph const &) &=delete
- ITRTGraph & operator= (ITRTGraph &&) &=delete
- virtual ~ITRTGraph () noexcept=default

  A shallow destructor of ITRTGraph.
- virtual ErrorCode clone (ITRTGraph *&graph, ISafeRecorder &recorder) noexcept=0

  Specialized Graph shallow copy.
- virtual ErrorCode getScratchMemorySize (size_t &size) const noexcept=0

  This function returns the scratch memory size (in bytes) needed to store all the intermediate tensors for inference. The user could allocate a scratch memory of this size and pass it to TRT via setScratchMemory if not using trtManagedScratch.
- virtual ErrorCode getTRTManagedScratch (bool &flag) const noexcept=0

  This function returns the trtManagedScratch flag provided in createTRTGraph call.
- virtual ErrorCode setScratchMemory (void *memory) noexcept=0

  This function sets the scratch memory for the graph. This should only be called if scratch memory is not TRT managed (trtManagedScratch is false). An error will be recorded if called on a graph where trtManagedScratch is true.
- virtual ErrorCode getScratchMemory (void *&memory) noexcept=0

  This function gets the scratch memory for the graph. This should only be called if scratch memory is not TRT managed (trtManagedScratch is false). An error will be recorded if called on a graph where trtManagedScratch is true.
- virtual ErrorCode getNbIOTensors (int64_t &nb) const noexcept=0

  This function returns the total number of input and output tensor for the current graph.
- virtual ErrorCode getIOTensorName (AsciiChar const *&name, size_t const index) const noexcept=0

  This function returns the name of a tensor for a given index.
- virtual ErrorCode getIOTensorDescriptor (TensorDescriptor &desc, AsciiChar const *const tensorName) const noexcept=0

  This function should return a TensorDescriptor which contains all the information about the tensor based on the name.
- virtual ErrorCode getIOTensorDescriptor (TensorDescriptor &desc, int32_t const index) const noexcept=0

  This function should return a TensorDescriptor which contains all the information about the tensor based on the index.
- virtual ErrorCode setIOTensorAddress (AsciiChar const *const tensorName, TypedArray const &tensor) noexcept=0

  This function assigns a user allocated device memory block for an input tensor to the graph based on its name.
- virtual ErrorCode setIOTensorAddress (int32_t const index, TypedArray const &tensor) noexcept=0

  This function assigns a user allocated device memory block for an input tensor to the graph based on its index.
- virtual ErrorCode getIOTensorAddress (AsciiChar const *const tensorName, TypedArray &tensor) noexcept=0

  This function gets the memory address for an user provided input tensor to the graph based on its name.
- virtual ErrorCode getIOTensorAddress (int32_t const index, TypedArray &tensor) noexcept=0

  This function gets the memory address for an user provided input tensor to the graph based on its index.
- virtual ErrorCode setInputConsumedEvent (cudaEvent_t event) noexcept=0

  This function sets a cudaEvent on the current graph that triggers when the input is consumed. At that point, the input memory can be recycled, i.e. new input for next inference can be loaded.

- virtual ErrorCode getInputConsumedEvent (cudaEvent_t &event) const noexcept=0

  This function retrieves the cudaEvent on the current graph that triggers when the input is fully consumed. At that point, the input memory can be recycled, i.e. new input for next inference can be loaded.
- virtual ErrorCode getErrorBuffer (RuntimeErrorInformation *&buffer) const noexcept=0

  This function retrieves the RuntimeErrorInformation (for async error) buffer for the current graph. This buffer includes all the runtime error types such as gather out of bound, silently consumed NaN value etc.
- virtual ErrorCode getSafeRecorder (ISafeRecorder *&recorder) const noexcept=0

  This function retrieves the ISafeRecorder for the current graph.
- virtual ErrorCode getNbIOProfiles (int64_t &nb) const noexcept=0

  This function returns the total number of IO tensor profiles for the current graph.
- virtual ErrorCode setIOProfile (int64_t profileIndex) noexcept=0

  This function selects the active IOProfile for the graph. If this function is not called, the TRTGraph will default to profile 0. Each IOProfile on the graph is mutually exclusive, meaning only one IOProfile can be active at a time.
- virtual ErrorCode getIOProfile (int64_t &profileIndex) const noexcept=0

  This function retrieves the index of the current active IOProfile for the graph.
- virtual ErrorCode getNbAuxStreams (int32_t &nbStreams) const noexcept=0

  Return the number of auxiliary streams used by this graph.
- virtual ErrorCode setAuxStreams (cudaStream_t *auxStreams, int32_t nbStreams) noexcept=0

  Set the auxiliary streams that TensorRT should use to run kernels on.
- virtual ErrorCode executeAsync (cudaStream_t stream) noexcept=0

  execute one inference of this graph.
- virtual ErrorCode sync () noexcept=0

  synchronize one inference of this graph.

Protected Member Functions

- ITRTGraph ()=default

9.211.1 Detailed Description

Abstract Interface for a functionally safe graph for executing inference on a built network.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

APIs are not thread safe: Users must guarantee that each concurrently running thread uses a unique instance (clone) of the API object.

Note

The set APIs are not thread safe and get APIs are thread-safe.

9.211.2 Constructor & Destructor Documentation

9.211.2.1 ITRTGraph() [1/3]

```
nvinfer2::safe::ITRTGraph::ITRTGraph (
            ITRTGraph const &  )  [delete]
```

9.211.2.2 ITRTGraph() [2/3]

```
nvinfer2::safe::ITRTGraph::ITRTGraph (
            ITRTGraph &&  )  [delete]
```

9.211.2.3 ~ITRTGraph()

```
virtual nvinfer2::safe::ITRTGraph::~ITRTGraph ( )  [virtual], [default], [noexcept]
```

A shallow destructor of ITRTGraph.

Precondition

DEINIT API

9.211.2.4 ITRTGraph() [3/3]

```
nvinfer2::safe::ITRTGraph::ITRTGraph ( )  [protected], [default]
```

9.211.3 Member Function Documentation

9.211.3.1 clone()

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::clone (
            ITRTGraph *& graph,
            ISafeRecorder & recorder )  [pure virtual], [noexcept]
```

Specialized Graph shallow copy.

This function constructs a new ITRTGraph which uses a shared pointer to the persistent part of the graph (pointing to the same set of weights) so we do not duplicate the weights anywhere. Clone will automatically allocate a new scratch memory if the scratch memory is trtManaged (trtManagedScratch set to true in createTRTGraph) otherwise user has to call setScratchMemory before calling execute. The cloned graph will be using the same memory allocator as the original graph.

Precondition

INIT API

Parameters

graph	A reference to an ITRTGraph pointer, that will be initialized after this call.
recorder	An ISafeRecorder that records the errors happening during graph initialization and inference.

Note

Each cloned graph may be given it's own recorder so there is an independent source of truth for each graph.

Returns

ErrorCode

9.211.3.2 executeAsync()

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::executeAsync (
            cudaStream_t stream )  [pure virtual], [noexcept]
```

execute one inference of this graph.

Precondition

RUNTIME API

Parameters

stream	A CUDA main stream on which the inference kernels will be enqueued. Must be a valid CUDA stream.

Returns

ErrorCode kSUCCESS on success, if any execution error occurred other error code might be returned Errors may include but not be limited to:

- Internal errors during executing one engine layer (host side)
- CUDA errors
- Some input or output tensor addresses have not been set.

9.211.3.3 getErrorBuffer()

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::getErrorBuffer (
            RuntimeErrorInformation *& buffer ) const  [pure virtual], [noexcept]
```

This function retrieves the RuntimeErrorInformation (for async error) buffer for the current graph. This buffer includes all the runtime error types such as gather out of bound, silently consumed NaN value etc.

Precondition

RUNTIME API

Parameters

buffer	retrieved error buffer.

See also

RuntimeErrorInformation

Returns

ErrorCode

9.211.3.4 getInputConsumedEvent()

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::getInputConsumedEvent (
            cudaEvent_t & event ) const  [pure virtual], [noexcept]
```

This function retrieves the cudaEvent on the current graph that triggers when the input is fully consumed. At that point, the input memory can be recycled, i.e. new input for next inference can be loaded.

Precondition

RUNTIME API

Parameters

event	retrieved cuda event

Returns

ErrorCode

9.211.3.5 getIOProfile()

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::getIOProfile (
            int64_t & profileIndex ) const  [pure virtual], [noexcept]
```

This function retrieves the index of the current active IOProfile for the graph.

Precondition

RUNTIME API

Parameters

profileIndex	retrieved profile index.

Returns

ErrorCode

9.211.3.6 getIOTensorAddress() [1/2]

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::getIOTensorAddress (
            AsciiChar const *const tensorName,
            TypedArray & tensor )  [pure virtual], [noexcept]
```

This function gets the memory address for an user provided input tensor to the graph based on its name.

Precondition

INIT API

Parameters

tensorName	the tensor name that the user would like to obtain memory for.
tensor	set to a device memory pointer allocated by user for the tensor.

Returns

ErrorCode

9.211.3.7 getIOTensorAddress() [2/2]

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::getIOTensorAddress (
            int32_t const index,
            TypedArray & tensor )  [pure virtual], [noexcept]
```

This function gets the memory address for an user provided input tensor to the graph based on its index.

Precondition

INIT API

Parameters

index	the tensor index that the user would like to obtain memory for.
tensor	set to a device memory pointer allocated by user for the tensor.

Returns

ErrorCode

9.211.3.8 getIOTensorDescriptor() [1/2]

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::getIOTensorDescriptor (
            TensorDescriptor & desc,
            AsciiChar const *const tensorName ) const  [pure virtual], [noexcept]
```

This function should return a TensorDescriptor which contains all the information about the tensor based on the name.

Precondition

INIT API

Parameters

desc	corresponding TensorDescriptor that will be filled out after the call.
name	name of the tensor we are interested in.

Returns

ErrorCode

Warning

If the name does not correspond to a valid IO tensor, the function will fail with an error code of ErrorCode::kINVALID_ARGUMENT

9.211.3.9 getIOTensorDescriptor() [2/2]

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::getIOTensorDescriptor (
            TensorDescriptor & desc,
            int32_t const index ) const  [pure virtual], [noexcept]
```

This function should return a TensorDescriptor which contains all the information about the tensor based on the index.

Precondition

INIT API

Parameters

desc	corresponding TensorDescriptor that will be filled out after the call.
index	index of the tensor we are interested in (starting from 0).

Returns

ErrorCode

Warning

If the index does not fall between 0 and getNbIOTensors()-1, the function will fail with an error code of ErrorCode::kINVALID_ARGUMENT

9.211.3.10 getIOTensorName()

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::getIOTensorName (
            AsciiChar const *& name,
            size_t const index ) const  [pure virtual], [noexcept]
```

This function returns the name of a tensor for a given index.

Precondition

INIT API

Parameters

name	The name string of the tensor that will be filled out after the call.
index	The index of the tensor whose name is to be returned.

Returns

ErrorCode

9.211.3.11 getNbAuxStreams()

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::getNbAuxStreams (
            int32_t & nbStreams ) const  [pure virtual], [noexcept]
```

Return the number of auxiliary streams used by this graph.

Precondition

INIT API

Parameters

nbStreams	The return value for the number of auxiliary streams.

Returns

ErrorCode

See also

setAuxStreams()

9.211.3.12 getNbIOProfiles()

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::getNbIOProfiles (
            int64_t & nb ) const  [pure virtual], [noexcept]
```

This function returns the total number of IO tensor profiles for the current graph.

Precondition

RUNTIME API

Parameters

nb	The return value for the total number of IO tensor profiles for the current graph. A graph will at least have 1 IOProfile

Returns

ErrorCode

9.211.3.13 getNbIOTensors()

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::getNbIOTensors (
            int64_t & nb ) const  [pure virtual], [noexcept]
```

This function returns the total number of input and output tensor for the current graph.

Precondition

INIT API

Parameters

nb	The return value for total number of input and output tensor for the current graph.

Returns

ErrorCode

9.211.3.14 getSafeRecorder()

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::getSafeRecorder (
            ISafeRecorder *& recorder ) const  [pure virtual], [noexcept]
```

This function retrieves the ISafeRecorder for the current graph.

Precondition

RUNTIME API

Parameters

recorder	retrieved ISafeRecorder.

Returns

ErrorCode

9.211.3.15 getScratchMemory()

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::getScratchMemory (
            void *& memory )  [pure virtual], [noexcept]
```

This function gets the scratch memory for the graph. This should only be called if scratch memory is not TRT managed (trtManagedScratch is false). An error will be recorded if called on a graph where trtManagedScratch is true.

Precondition

INIT API

Parameters

memory	set to a device memory pointer allocated by user or nullptr when the scratch memory is TRT managed.

See also

getScratchMemorySize

Returns

ErrorCode

9.211.3.16 getScratchMemorySize()

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::getScratchMemorySize (
            size_t & size ) const  [pure virtual], [noexcept]
```

This function returns the scratch memory size (in bytes) needed to store all the intermediate tensors for inference. The user could allocate a scratch memory of this size and pass it to TRT via setScratchMemory if not using trtManaged↩ Scratch.

Precondition

INIT API

See also

setScratchMemory

Parameters

size	The return value for the scratch memory size (in bytes) needed to store all the intermediate tensors for inference.

Returns

ErrorCode

9.211.3.17 getTRTManagedScratch()

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::getTRTManagedScratch (
            bool & flag ) const  [pure virtual], [noexcept]
```

This function returns the trtManagedScratch flag provided in createTRTGraph call.

Precondition

INIT API

See also

createTRTGraph

Parameters

flag	A boolean flag that determines if scratch memory is managed by TRT

Returns

ErrorCode

9.211.3.18 operator=() [1/2]

```
ITRTGraph & nvinfer2::safe::ITRTGraph::operator= (
            ITRTGraph &&  ) &  [delete]
```

9.211.3.19 operator=() [2/2]

```
ITRTGraph & nvinfer2::safe::ITRTGraph::operator= (
            ITRTGraph const &  ) &  [delete]
```

9.211.3.20 setAuxStreams()

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::setAuxStreams (
            cudaStream_t * auxStreams,
            int32_t nbStreams )  [pure virtual], [noexcept]
```

Set the auxiliary streams that TensorRT should use to run kernels on.

TRT will launch the kernels that are supposed to run on the auxiliary streams using the streams provided by the user via this API. The user is responsible for allocating and deallocating these streams.

If getNbAuxStreams() returns a value greater than 0, this API must be called before executeAsync() to provide the required auxiliary streams.

If getNbAuxStreams() returns 0, setAuxStreams() can only be called with an array of size 0.

Precondition

INIT API

Parameters

auxStreams	The pointer to an array of cudaStream_t with the array length equal to nbStreams. All streams in the array must be valid CUDA streams.
nbStreams	The number of auxiliary streams provided. Must be equal to the value returned by getNbAuxStreams(). If nbStreams does not match, kINVALID_ARGUMENT will be returned.

Returns

ErrorCode kSUCCESS on success, kINVALID_ARGUMENT if auxStreams contains nullptr, or if nbStreams does not match getNbAuxStreams().

See also

getNbAuxStreams()

9.211.3.21 setInputConsumedEvent()

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::setInputConsumedEvent (
            cudaEvent_t event ) [pure virtual], [noexcept]
```

This function sets a cudaEvent on the current graph that triggers when the input is consumed. At that point, the input memory can be recycled, i.e. new input for next inference can be loaded.

Precondition

INIT API

Parameters

event	cuda event to be set

Returns

ErrorCode

9.211.3.22 setIOProfile()

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::setIOProfile (
            int64_t profileIndex ) [pure virtual], [noexcept]
```

This function selects the active IOProfile for the graph. If this function is not called, the TRTGraph will default to profile 0. Each IOProfile on the graph is mutually exclusive, meaning only one IOProfile can be active at a time.

Precondition

RUNTIME API

Parameters

profileIndex	The index of the profile to select.

Returns

ErrorCode

Note

Other graph APIs are executed in the context of the current IOProfile id, so IOProfile id should be changed only on synchronized inference boundaries.

9.211.3.23 setIOTensorAddress() [1/2]

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::setIOTensorAddress (
            AsciiChar const *const tensorName,
            TypedArray const & tensor )  [pure virtual], [noexcept]
```

This function assigns a user allocated device memory block for an input tensor to the graph based on its name.

Precondition

INIT API

Parameters

tensorName	the tensor name that the user would like to provide memory for.
tensor	a device memory block allocated by user for the tensor.

Returns

ErrorCode

9.211.3.24 setIOTensorAddress() [2/2]

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::setIOTensorAddress (
            int32_t const index,
            TypedArray const & tensor )  [pure virtual], [noexcept]
```

This function assigns a user allocated device memory block for an input tensor to the graph based on its index.

Precondition

INIT API

Parameters

index	the tensor index that the user would like to provide memory for.
tensor	a device memory block allocated by user for the tensor.

Returns

ErrorCode

9.211.3.25 setScratchMemory()

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::setScratchMemory (
            void * memory )  [pure virtual], [noexcept]
```

This function sets the scratch memory for the graph. This should only be called if scratch memory is not TRT managed (trtManagedScratch is false). An error will be recorded if called on a graph where trtManagedScratch is true.

Precondition

INIT API

Parameters

memory	pointer to a device memory block allocated by user that is at least as large as getScratchMemorySize, or nullptr which will reset the internal pointer.

See also

getScratchMemorySize

Returns

ErrorCode

Warning

User must guarantee that the allocated scratch memory is large enough.

9.211.3.26 sync()

```
virtual ErrorCode nvinfer2::safe::ITRTGraph::sync ( )  [pure virtual], [noexcept]
```

synchronize one inference of this graph.

Precondition

RUNTIME API

Returns

ErrorCode kSUCCESS on success, if any execution error occurred other error code might be returned. Errors may include but not be limited to:

- Internal errors during executing one engine layer (host side)
- CUDA errors
- Some input or output tensor addresses have not been set.

The documentation for this class was generated from the following file:

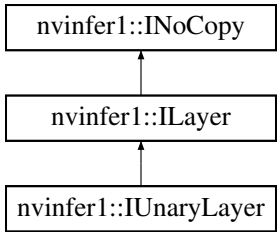
- NvInferSafeRuntime.h

9.212 nvinfer1::IUnaryLayer Class Reference

Layer that represents an unary operation.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IUnaryLayer:



Public Member Functions

- void setOperation (UnaryOperation op) noexcept

  Set the unary operation for the layer.
- UnaryOperation getOperation () const noexcept

  Get the unary operation for the layer.

Protected Member Functions

- virtual ~IUnaryLayer () noexcept=0

Protected Attributes

- apiv::VUnaryLayer * mImpl

9.212.1 Detailed Description

Layer that represents an unary operation.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.212.2 Constructor & Destructor Documentation

9.212.2.1 ~IUnaryLayer()

```
nvinfer1::IUnaryLayer::~IUnaryLayer ( )  [inline], [protected], [pure virtual], [default], [noexcept]
```

9.212.3 Member Function Documentation

9.212.3.1 getOperation()

```
UnaryOperation nvinfer1::IUnaryLayer::getOperation ( ) const  [inline], [noexcept]
```

Get the unary operation for the layer.

See also

setOperation(), UnaryOperation

9.212.3.2 setOperation()

```
void nvinfer1::IUnaryLayer::setOperation (
            UnaryOperation op )  [inline], [noexcept]
```

Set the unary operation for the layer.

When running this layer on DLA, only UnaryOperation::kABS is supported.

See also

getOperation(), UnaryOperation

9.212.4 Member Data Documentation

9.212.4.1 mImpl

```
apiv::VUnaryLayer* nvinfer1::IUnaryLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

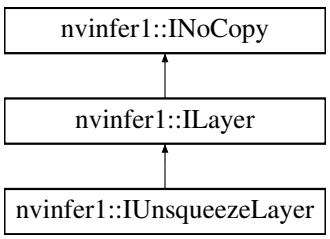
- NvInfer.h

9.213 nvinfer1::IUnsqueezeLayer Class Reference

Layer that represents an unsqueeze operation, which reshapes the first input tensor by inserting unit-length dimensions to the output at the axes specified by the second input tensor.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IUnsqueezeLayer:



Public Member Functions

- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~IUnsqueezeLayer () noexcept=0

Protected Attributes

- apiv::VUnsqueezeLayer * mImpl

9.213.1 Detailed Description

Layer that represents an unsqueeze operation, which reshapes the first input tensor by inserting unit-length dimensions to the output at the axes specified by the second input tensor.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.213.2 Constructor & Destructor Documentation

9.213.2.1 ~IUnsqueezeLayer()

```
nvinfer1::IUnsqueezeLayer::~IUnsqueezeLayer ( )  [inline], [protected], [pure virtual], [default],
[noexcept]
```

9.213.3 Member Function Documentation

9.213.3.1 setInput()

```
void nvinfer1::ILayer::setInput (
            int32_t index,
            ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	The index of the input to modify.
tensor	The new input tensor.

For an Unsqueeze layer, the values 0-1 are valid for index. The indices are as follows:

- 0: Input data tensor.
- 1: The output axes at which unit-length dimensions are inserted. Must resolve to a constant Int32 or Int64 1D shape tensor.

9.213.4 Member Data Documentation

9.213.4.1 mImpl

```
apiv::VUnsqueezeLayer* nvinfer1::IUnsqueezeLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

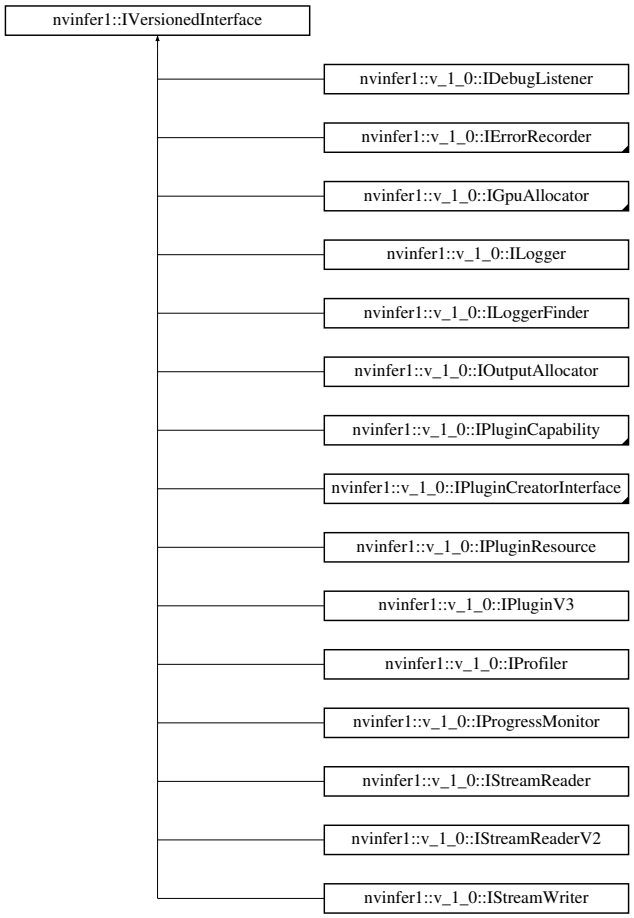
- NvInfer.h

9.214 nvinfer1::IVersionedInterface Class Reference

An Interface class for version control.

```
#include <NvInferRuntimeBase.h>
```

Inheritance diagram for nvinfer1::IVersionedInterface:



Public Member Functions

- virtual APILanguage getAPILanguage () const noexcept

  The language used to build the implementation of this Interface.
- virtual InterfaceInfo getInterfaceInfo () const noexcept=0

  Return version information associated with this interface. Applications must not override this method.
- virtual ~IVersionedInterface () noexcept=default

Protected Member Functions

- IVersionedInterface ()=default
- IVersionedInterface (IVersionedInterface const &)=default
- IVersionedInterface (IVersionedInterface &&)=default
- IVersionedInterface & operator= (IVersionedInterface const &) &=default
- IVersionedInterface & operator= (IVersionedInterface &&) &=default

9.214.1 Detailed Description

An Interface class for version control.

9.214.2 Constructor & Destructor Documentation

9.214.2.1 ~IVersionedInterface()

```
virtual nvinfer1::IVersionedInterface::~IVersionedInterface ( )  [virtual], [default], [noexcept]
```

9.214.2.2 IVersionedInterface() [1/3]

```
nvinfer1::IVersionedInterface::IVersionedInterface ( )  [protected], [default]
```

9.214.2.3 IVersionedInterface() [2/3]

```
nvinfer1::IVersionedInterface::IVersionedInterface (
            IVersionedInterface const &  )  [protected], [default]
```

9.214.2.4 IVersionedInterface() [3/3]

```
nvinfer1::IVersionedInterface::IVersionedInterface (
            IVersionedInterface &&  )  [protected], [default]
```

9.214.3 Member Function Documentation

9.214.3.1 getAPILanguage()

```
virtual APILanguage nvinfer1::IVersionedInterface::getAPILanguage ( ) const  [inline], [virtual],
[noexcept]
```

The language used to build the implementation of this Interface.

Applications must not override this method.

9.214.3.2 getInterfaceInfo()

```
virtual InterfaceInfo nvinfer1::IVersionedInterface::getInterfaceInfo ( ) const  [pure virtual],
[noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implemented in nvinfer1::v_1_0::IProgressMonitor, nvinfer1::v_1_0::IPluginResource, nvinfer1::v_1_0::IPluginV3, nvinfer1::v_1_0::IStreamReader, nvinfer1::v_1_0::IStreamWriter, nvinfer1::v_1_0::IStreamReaderV2, nvinfer1::v_1_0::IPluginV3OneCor nvinfer1::v_1_0::IPluginV3OneBuild, nvinfer1::v_1_0::IPluginV3OneRuntime, nvinfer1::v_2_0::IPluginV3OneBuild, nvinfer1::v_1_0::IProfiler, nvinfer1::v_1_0::ILogger, nvinfer1::v_1_0::IGpuAllocator, nvinfer1::v_1_0::IOutputAllocator, nvinfer1::v_1_0::IDebugListener, nvinfer1::v_1_0::ILoggerFinder, nvinfer1::v_1_0::IGpuAsyncAllocator, nvinfer1::v_1_0::IPluginCreator nvinfer1::v_1_0::IErrorRecorder, nvinfer1::v_1_0::IPluginCreator, nvinfer2::safe::IPluginV3OneSafeCore, nvinfer2::safe::IPluginV3One nvinfer2::safe::IPluginV3OneSafeBuildMSS, nvinfer2::safe::IPluginV3OneSafeRuntime, and nvinfer2::safe::ISafePluginCreatorV3One

9.214.3.3 operator=() [1/2]

```
IVersionedInterface & nvinfer1::IVersionedInterface::operator= (
            IVersionedInterface &&  ) &  [protected], [default]
```

9.214.3.4 operator=() [2/2]

```
IVersionedInterface & nvinfer1::IVersionedInterface::operator= (
            IVersionedInterface const &  ) &  [protected], [default]
```

The documentation for this class was generated from the following file:

- NvInferRuntimeBase.h

9.215 nvinfer1::Permutation Struct Reference

Represents a permutation of dimensions.

```
#include <NvInfer.h>
```

Public Attributes

- int32_t order [Dims::MAX_DIMS]

9.215.1 Detailed Description

Represents a permutation of dimensions.

9.215.2 Member Data Documentation

9.215.2.1 order

```
int32_t nvinfer1::Permutation::order[Dims::MAX_DIMS]
```

The elements of the permutation. The permutation is applied as outputDimensionIndex = permutation.order[input↩ DimensionIndex], so to permute from CHW order to HWC order, the required permutation is [1, 2, 0], and to permute from HWC to CHW, the required permutation is [2, 0, 1].

The documentation for this struct was generated from the following file:

- NvInfer.h

9.216 nvinfer2::safe::PhysicalDims Class Reference

Structure to define the physical dimensions of a tensor with support for up to 9 dimensions.

```
#include <NvInferSafeRuntime.h>
```

Public Attributes

- int32_t nbDims

  The rank (number of dimensions).
- int64_t d [MAX_DIMS]

  The extent of each dimension.

Static Public Attributes

- static constexpr int32_t MAX_DIMS {9}

9.216.1 Detailed Description

Structure to define the physical dimensions of a tensor with support for up to 9 dimensions.

This structure is specifically designed for describing physical tensors layout in memory which can have up to 9 dimensions after vectorization. Unlike the standard Dims structure which is limited to 8 dimensions, PhysicalDims supports the additional dimension needed when tensors are vectorized.

When a tensor is vectorized by size T along some dimension with size K, the vectorized dimension is split into two dimensions of sizes ceil(K/T) and T, potentially resulting in 9 physical dimensions for an originally 8 logical dimensional tensor.

9.216.2 Member Data Documentation

9.216.2.1 d

```
int64_t nvinfer2::safe::PhysicalDims::d[MAX_DIMS]
```

The extent of each dimension.

9.216.2.2 MAX_DIMS

```
constexpr int32_t nvinfer2::safe::PhysicalDims::MAX_DIMS {9}  [static], [constexpr]
```

The maximum rank (number of dimensions) supported for physical tensor layout. This is set to 9 to accommodate vectorized tensors that split one dimension into two.

9.216.2.3 nbDims

```
int32_t nvinfer2::safe::PhysicalDims::nbDims
```

The rank (number of dimensions).

The documentation for this class was generated from the following file:

- NvInferSafeRuntime.h

9.217 nvinfer1::PluginField Class Reference

Structure containing plugin attribute field names and associated data This information can be parsed to decode necessary plugin metadata.

```
#include <NvInferPluginBase.h>
```

Public Member Functions

- PluginField (AsciiChar const *const name_=nullptr, void const *const data_=nullptr, PluginFieldType const type_=PluginFieldType::kUNKNOWN, int32_t const length_=0) noexcept

Public Attributes

- AsciiChar const * name

  Plugin field attribute name.
- void const * data

  Plugin field attribute data.
- PluginFieldType type

  Plugin field attribute type.
- int32_t length

  Number of data entries in the Plugin attribute.

9.217.1 Detailed Description

Structure containing plugin attribute field names and associated data This information can be parsed to decode necessary plugin metadata.

9.217.2 Constructor & Destructor Documentation

9.217.2.1 PluginField()

```
nvinfer1::PluginField::PluginField (
            AsciiChar const *const name_ = nullptr,
            void const *const data_ = nullptr,
            PluginFieldType const type_ = PluginFieldType::kUNKNOWN,
            int32_t const length_ = 0 )  [inline], [noexcept]
```

9.217.3 Member Data Documentation

9.217.3.1 data

```
void const* nvinfer1::PluginField::data
```

Plugin field attribute data.

9.217.3.2 length

```
int32_t nvinfer1::PluginField::length
```

Number of data entries in the Plugin attribute.

9.217.3.3 name

```
AsciiChar const* nvinfer1::PluginField::name
```

Plugin field attribute name.

9.217.3.4 type

```
PluginFieldType nvinfer1::PluginField::type
```

Plugin field attribute type.

The documentation for this class was generated from the following file:

- NvInferPluginBase.h

9.218 nvinfer1::PluginFieldCollection Struct Reference

Plugin field collection struct.

```
#include <NvInferPluginBase.h>
```

Public Attributes

- int32_t nbFields {}
  Number of PluginField entries.
- PluginField const * fields {}
  Pointer to PluginField entries.

9.218.1 Detailed Description

Plugin field collection struct.

9.218.2 Member Data Documentation

9.218.2.1 fields

```
PluginField const* nvinfer1::PluginFieldCollection::fields {}
```

Pointer to PluginField entries.

9.218.2.2 nbFields

```
int32_t nvinfer1::PluginFieldCollection::nbFields {}
```

Number of PluginField entries.

The documentation for this struct was generated from the following file:

- NvInferPluginBase.h

9.219 nvinfer1::PluginRegistrar< T > Class Template Reference

Register the plugin creator to the registry The static registry object will be instantiated when the plugin library is loaded. This static object will register all creators available in the library to the registry.

```
#include <NvInferRuntime.h>
```

Public Member Functions

- PluginRegistrar ()

9.219.1 Detailed Description

template<typename T>
class nvinfer1::PluginRegistrar< T >

Register the plugin creator to the registry The static registry object will be instantiated when the plugin library is loaded. This static object will register all creators available in the library to the registry.

Warning

Statically registering plugins should be avoided in the automotive safety context as the application developer should first register an error recorder with the plugin registry via IPluginRegistry::setErrorRecorder() before using IPluginRegistry::registerCreator() or other methods.

9.219.2 Constructor & Destructor Documentation

9.219.2.1 PluginRegistrar()

```
template<typename T >
nvinfer1::PluginRegistrar< T >::PluginRegistrar ( )  [inline]
```

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.220 nvinfer1::PluginTensorDesc Struct Reference

Fields that a plugin might see for an input or output.

```
#include <NvInferRuntimePlugin.h>
```

Public Attributes

- Dims dims

  Dimensions.
- DataType type
- TensorFormat format

  Tensor format.
- float scale

  Scale for INT8 data type.

9.220.1 Detailed Description

Fields that a plugin might see for an input or output.

Scale is only valid when data type is DataType::kINT8. TensorRT will set the value to -1.0F if it is invalid.

See also

IPluginV2IOExt::supportsFormatCombination

IPluginV2IOExt::configurePlugin

9.220.2 Member Data Documentation

9.220.2.1 dims

```
Dims nvinfer1::PluginTensorDesc::dims
```

Dimensions.

9.220.2.2 format

```
TensorFormat nvinfer1::PluginTensorDesc::format
```

Tensor format.

9.220.2.3 scale

```
float nvinfer1::PluginTensorDesc::scale
```

Scale for INT8 data type.

9.220.2.4 type

```
DataType nvinfer1::PluginTensorDesc::type
```

> **Warning**
>
> DataType:kBOOL and DataType::kUINT8 are not supported.

The documentation for this struct was generated from the following file:

- NvInferRuntimePlugin.h

9.221 PluginVersion Struct Reference

Definition of plugin versions.

```
#include <NvInferRuntimePlugin.h>
```

9.221.1 Detailed Description

Definition of plugin versions.

Tag for plug-in versions. Used in upper byte of getTensorRTVersion().

Deprecated Deprecated in TensorRT 10.10. PluginVersion is used only in relation to IPluginV2-descendent plugin interfaces, which are all deprecated.

The documentation for this struct was generated from the following file:

- NvInferRuntimePlugin.h

9.222 nvinfer1::plugin::PriorBoxParameters Struct Reference

The PriorBox plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions (H x W).

`#include <NvInferPluginUtils.h>`

Public Attributes

- float * minSize

  Minimum box size in pixels. Can not be nullptr.
- float * maxSize

  Maximum box size in pixels. Can be nullptr.
- float * aspectRatios

  Aspect ratios of the boxes. Can be nullptr.
- int32_t numMinSize

  Number of elements in minSize. Must be larger than 0.
- int32_t numMaxSize

  Number of elements in maxSize. Can be 0 or same as numMinSize.
- int32_t numAspectRatios

  Number of elements in aspectRatios. Can be 0.
- bool flip
- bool clip

  If true, will clip the prior so that it is within [0,1].
- float variance [4]

  Variance for adjusting the prior boxes.
- int32_t imgH

  Image height. If 0, then the H dimension of the data tensor will be used.
- int32_t imgW

  Image width. If 0, then the W dimension of the data tensor will be used.
- float stepH

  Step in H. If 0, then (float)imgH/h will be used where h is the H dimension of the 1st input tensor.
- float stepW

  Step in W. If 0, then (float)imgW/w will be used where w is the W dimension of the 1st input tensor.
- float offset

  Offset to the top left corner of each cell.

9.222.1 Detailed Description

The PriorBox plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions (H x W).

PriorBoxParameters defines a set of parameters for creating the PriorBox plugin layer.

9.222.2 Member Data Documentation

9.222.2.1 aspectRatios

```
float* nvinfer1::plugin::PriorBoxParameters::aspectRatios
```

Aspect ratios of the boxes. Can be nullptr.

9.222.2.2 clip

```
bool nvinfer1::plugin::PriorBoxParameters::clip
```

If true, will clip the prior so that it is within [0,1].

9.222.2.3 flip

```
bool nvinfer1::plugin::PriorBoxParameters::flip
```

If true, will flip each aspect ratio. For example, if there is an aspect ratio ”r”, the aspect ratio ”1.0/r” will be generated as well.

9.222.2.4 imgH

```
int32_t nvinfer1::plugin::PriorBoxParameters::imgH
```

Image height. If 0, then the H dimension of the data tensor will be used.

9.222.2.5 imgW

```
int32_t nvinfer1::plugin::PriorBoxParameters::imgW
```

Image width. If 0, then the W dimension of the data tensor will be used.

9.222.2.6 maxSize

```
float* nvinfer1::plugin::PriorBoxParameters::maxSize
```

Maximum box size in pixels. Can be nullptr.

9.222.2.7 minSize

```
float* nvinfer1::plugin::PriorBoxParameters::minSize
```

Minimum box size in pixels. Can not be nullptr.

9.222.2.8 numAspectRatios

```
int32_t nvinfer1::plugin::PriorBoxParameters::numAspectRatios
```

Number of elements in aspectRatios. Can be 0.

9.222.2.9 numMaxSize

```
int32_t nvinfer1::plugin::PriorBoxParameters::numMaxSize
```

Number of elements in maxSize. Can be 0 or same as numMinSize.

9.222.2.10 numMinSize

```
int32_t nvinfer1::plugin::PriorBoxParameters::numMinSize
```

Number of elements in minSize. Must be larger than 0.

9.222.2.11 offset

```
float nvinfer1::plugin::PriorBoxParameters::offset
```

Offset to the top left corner of each cell.

9.222.2.12 stepH

```
float nvinfer1::plugin::PriorBoxParameters::stepH
```

Step in H. If 0, then (float)imgH/h will be used where h is the H dimension of the 1st input tensor.

9.222.2.13 stepW

```
float nvinfer1::plugin::PriorBoxParameters::stepW
```

Step in W. If 0, then (float)imgW/w will be used where w is the W dimension of the 1st input tensor.

9.222.2.14 variance

```
float nvinfer1::plugin::PriorBoxParameters::variance[4]
```

Variance for adjusting the prior boxes.

The documentation for this struct was generated from the following file:

- NvInferPluginUtils.h

9.223 nvinfer1::plugin::RegionParameters Struct Reference

The Region plugin layer performs region proposal calculation.

```
#include <NvInferPluginUtils.h>
```

Public Attributes

- int32_t num

  Number of predicted bounding box for each grid cell.
- int32_t coords

  Number of coordinates for a bounding box.
- int32_t classes

  Number of classifications to be predicted.
- softmaxTree * smTree

  Helping structure to do softmax on confidence scores.

9.223.1 Detailed Description

The Region plugin layer performs region proposal calculation.

Generate 5 bounding boxes per cell (for yolo9000, generate 3 bounding boxes per cell). For each box, calculating its probabilities of objects detections from 80 pre-defined classifications (yolo9000 has 9418 pre-defined classifications, and these 9418 items are organized as work-tree structure). RegionParameters defines a set of parameters for creating the Region plugin layer.

9.223.2 Member Data Documentation

9.223.2.1 classes

```
int32_t nvinfer1::plugin::RegionParameters::classes
```

Number of classifications to be predicted.

9.223.2.2 coords

```
int32_t nvinfer1::plugin::RegionParameters::coords
```

Number of coordinates for a bounding box.

9.223.2.3 num

```
int32_t nvinfer1::plugin::RegionParameters::num
```

Number of predicted bounding box for each grid cell.

9.223.2.4 smTree

```
softmaxTree* nvinfer1::plugin::RegionParameters::smTree
```

Helping structure to do softmax on confidence scores.

The documentation for this struct was generated from the following file:

- NvInferPluginUtils.h

9.224 nvinfer1::plugin::RPROIParams Struct Reference

RPROIParams is used to create the RPROIPlugin instance.

```
#include <NvInferPluginUtils.h>
```

Public Attributes

- int32_t poolingH

  Height of the output in pixels after ROI pooling on feature map.
- int32_t poolingW

  Width of the output in pixels after ROI pooling on feature map.
- int32_t featureStride
- int32_t preNmsTop

  Number of proposals to keep before applying NMS.
- int32_t nmsMaxOut

  Number of remaining proposals after applying NMS.
- int32_t anchorsRatioCount

  Number of anchor box ratios.
- int32_t anchorsScaleCount

  Number of anchor box scales.
- float iouThreshold

  IoU (Intersection over Union) threshold used for the NMS step.
- float minBoxSize

  Minimum allowed bounding box size before scaling, used for anchor box calculation.
- float spatialScale

  Spatial scale between the input image and the last feature map.

9.224.1 Detailed Description

RPROIParams is used to create the RPROIPlugin instance.

9.224.2 Member Data Documentation

9.224.2.1 anchorsRatioCount

```
int32_t nvinfer1::plugin::RPROIParams::anchorsRatioCount
```

Number of anchor box ratios.

9.224.2.2 anchorsScaleCount

```
int32_t nvinfer1::plugin::RPROIParams::anchorsScaleCount
```

Number of anchor box scales.

9.224.2.3 featureStride

```
int32_t nvinfer1::plugin::RPROIParams::featureStride
```

Feature stride; ratio of input image size to feature map size. Assuming that max pooling layers in the neural network use square filters.

9.224.2.4 iouThreshold

```
float nvinfer1::plugin::RPROIParams::iouThreshold
```

IoU (Intersection over Union) threshold used for the NMS step.

9.224.2.5 minBoxSize

```
float nvinfer1::plugin::RPROIParams::minBoxSize
```

Minimum allowed bounding box size before scaling, used for anchor box calculation.

9.224.2.6 nmsMaxOut

```
int32_t nvinfer1::plugin::RPROIParams::nmsMaxOut
```

Number of remaining proposals after applying NMS.

9.224.2.7 poolingH

```
int32_t nvinfer1::plugin::RPROIParams::poolingH
```

Height of the output in pixels after ROI pooling on feature map.

9.224.2.8 poolingW

```
int32_t nvinfer1::plugin::RPROIParams::poolingW
```

Width of the output in pixels after ROI pooling on feature map.

9.224.2.9 preNmsTop

```
int32_t nvinfer1::plugin::RPROIParams::preNmsTop
```

Number of proposals to keep before applying NMS.

9.224.2.10 spatialScale

```
float nvinfer1::plugin::RPROIParams::spatialScale
```

Spatial scale between the input image and the last feature map.

The documentation for this struct was generated from the following file:

- NvInferPluginUtils.h

9.225 nvinfer2::safe::RuntimeErrorInformation Struct Reference

Holds information about runtime errors that occur during asynchronous kernel execution.

`#include <NvInferSafeRecorder.h>`

Public Attributes

- uint64_t bitMask {0ULL}

  Bitmask of errors; see RuntimeErrorType for bit positions.

9.225.1 Detailed Description

Holds information about runtime errors that occur during asynchronous kernel execution.

Stores a bitmask where each bit represents a different type of runtime error.

- Each bit corresponds to a value in RuntimeErrorType.
- Set a bit: bitMask |= static_cast<uint64_t>(RuntimeErrorType::<ERROR_NAME>).
- Check a bit: (bitMask & static_cast<uint64_t>(RuntimeErrorType::<ERROR_NAME>)) != 0ULL.
- Clear a bit: bitMask &= ~static_cast<uint64_t>(RuntimeErrorType::<ERROR_NAME>).
- Combine bits: use bitwise OR of multiple RuntimeErrorType values.

9.225.2 Member Data Documentation

9.225.2.1 bitMask

`uint64_t nvinfer2::safe::RuntimeErrorInformation::bitMask {0ULL}`

Bitmask of errors; see RuntimeErrorType for bit positions.

The documentation for this struct was generated from the following file:

- NvInferSafeRecorder.h

9.226 nvinfer1::plugin::softmaxTree Struct Reference

When performing yolo9000, softmaxTree is helping to do softmax on confidence scores, for element to get the precise classification through word-tree structured classification definition.

`#include <NvInferPluginUtils.h>`

Public Attributes

- int32_t * leaf
- int32_t n
- int32_t * parent
- int32_t * child
- int32_t * group
- char ** name
- int32_t groups
- int32_t * groupSize
- int32_t * groupOffset

9.226.1 Detailed Description

When performing yolo9000, softmaxTree is helping to do softmax on confidence scores, for element to get the precise classification through word-tree structured classification definition.

9.226.2 Member Data Documentation

9.226.2.1 child

```
int32_t* nvinfer1::plugin::softmaxTree::child
```

9.226.2.2 group

```
int32_t* nvinfer1::plugin::softmaxTree::group
```

9.226.2.3 groupOffset

```
int32_t* nvinfer1::plugin::softmaxTree::groupOffset
```

9.226.2.4 groups

```
int32_t nvinfer1::plugin::softmaxTree::groups
```

9.226.2.5 groupSize

```
int32_t* nvinfer1::plugin::softmaxTree::groupSize
```

9.226.2.6 leaf

```
int32_t* nvinfer1::plugin::softmaxTree::leaf
```

9.226.2.7 n

```
int32_t nvinfer1::plugin::softmaxTree::n
```

9.226.2.8 name

```
char** nvinfer1::plugin::softmaxTree::name
```

9.226.2.9 parent

```
int32_t* nvinfer1::plugin::softmaxTree::parent
```

The documentation for this struct was generated from the following file:

- NvInferPluginUtils.h

9.227 nvinfer2::safe::TensorDescriptor Struct Reference

A simple record summarizing various properties of a network IO Tensor.

```
#include <NvInferSafeRuntime.h>
```

Public Attributes

- AsciiChar const * tensorName {nullptr}

  Name of the IO Tensor.
- PhysicalDims shape {-1, {}}
- PhysicalDims stride {-1, {}}

  Stride vector for each element of the IO Tensor.
- DataType dataType {DataType::kFLOAT}
- uint64_t bytesPerComponent {0U}

  The size of the tensor data type in bytes (4 for float and int32, 2 for half, 1 for int8)
- uint64_t componentsPerVector {1U}

  The vector length (in scalars) for a vectorized tensor, 1 if the tensor is not vectorized.
- int64_t vectorizedDim {-1}

  The dimension index along which the tensor is vectorized, -1 if the tensor is not vectorized.
- uint64_t sizeInBytes {0U}

  Total size in bytes for the IO Tensor.
- TensorIOMode ioMode {TensorIOMode::kNONE}

  Enum to denote whether the Tensor is for input or output.
- MemoryPlacement memPlacement {MemoryPlacement::kNONE}

  Enum to denote whether the Tensor memory is allocated on the GPU, CPU, or CPU_PINNED.
- PhysicalDims strideOrder {-1, {}}

  The order in which the dimensions are laid out in memory.
- Dims userShape {-1, {}}

9.227.1 Detailed Description

A simple record summarizing various properties of a network IO Tensor.

All properties needed to query, allocate, read, and write IO Tensors are recorded here. The user can determine the physical tensor shape and layout used by the compiler, which may be different from the original user-specified shape due to padding or tiling for vectorized layouts. For example, a user-specified shape [1,10,3,4] with a TensorFormat specification of NHWC8 would lead to tiling by 8 elements on the C dimension with the following descriptor fields:

User-specified properties (NHWC8): userShape : [1,10,3,4] (in NCHW order) componentsPerVector : 8 vectorized↩ Dim : 1 (C)

Compiler-determined layout: shape : [1,2,8,3,4] (C dimension is split into two dims) stride : [192,8,1,64,16] stride↩ Order : [4,1,0,3,2] (Tiled C dimensions are changing fastest)

9.227.2 Member Data Documentation

9.227.2.1 bytesPerComponent

```
uint64_t nvinfer2::safe::TensorDescriptor::bytesPerComponent {0U}
```

The size of the tensor data type in bytes (4 for float and int32, 2 for half, 1 for int8)

9.227.2.2 componentsPerVector

```
uint64_t nvinfer2::safe::TensorDescriptor::componentsPerVector {1U}
```

The vector length (in scalars) for a vectorized tensor, 1 if the tensor is not vectorized.

9.227.2.3 dataType

```
DataType nvinfer2::safe::TensorDescriptor::dataType {DataType::kFLOAT}
```

DataType enum for the IO Tensor.

> Warning
>
> The actual type of the tensor data must correspond to this DataType.

9.227.2.4 ioMode

```
TensorIOMode nvinfer2::safe::TensorDescriptor::ioMode {TensorIOMode::kNONE}
```

Enum to denote whether the Tensor is for input or output.

9.227.2.5 memPlacement

```
MemoryPlacement nvinfer2::safe::TensorDescriptor::memPlacement {MemoryPlacement::kNONE}
```

Enum to denote whether the Tensor memory is allocated on the GPU, CPU, or CPU_PINNED.

9.227.2.6 shape

```
PhysicalDims nvinfer2::safe::TensorDescriptor::shape {-1, {}}
```

Static shape of the IO Tensor.

> Note
>
> The static shape will depend on the IO Profile selected.

9.227.2.7 sizeInBytes

```
uint64_t nvinfer2::safe::TensorDescriptor::sizeInBytes {0U}
```

Total size in bytes for the IO Tensor.

9.227.2.8 stride

```
PhysicalDims nvinfer2::safe::TensorDescriptor::stride {-1, {}}
```

Stride vector for each element of the IO Tensor.

9.227.2.9 strideOrder

```
PhysicalDims nvinfer2::safe::TensorDescriptor::strideOrder {-1, {}}
```

The order in which the dimensions are laid out in memory.

9.227.2.10 tensorName

```
AsciiChar const* nvinfer2::safe::TensorDescriptor::tensorName {nullptr}
```

Name of the IO Tensor.

9.227.2.11 userShape

```
Dims nvinfer2::safe::TensorDescriptor::userShape {-1, {}}
```

The original user-specified shape of the tensor, provided as a reference if the tensor is vectorized. When a tensor is vectorized by size T along some dimension with size K, the vectorized dimension is split into two dimensions of sizes ceil(K/T) and T.

9.227.2.12 vectorizedDim

```
int64_t nvinfer2::safe::TensorDescriptor::vectorizedDim {-1}
```

The dimension index along which the tensor is vectorized, -1 if the tensor is not vectorized.

The documentation for this struct was generated from the following file:

- NvInferSafeRuntime.h

9.228 nvinfer1::v_1_0::TimingCacheKey Struct Reference

The key to retrieve timing cache entries.

`#include <NvInfer.h>`

Public Attributes

- uint8_t data [16]

9.228.1 Detailed Description

The key to retrieve timing cache entries.

TimingCacheKey has two types of representation: binary and string. The conversion rule from binary to string is: 1) Convert each uint8_t element in binary key into two hexadecimal ascii chars, e.g. 0xab -> ”ab” 2) Concat the ascii chars of all elements in sequence. The result should have exact 32 chars 3) Add prefix ”0x” to the string produced in step 2.

See also

ITimingCache::query(), ITimingCache::update()

9.228.2 Member Data Documentation

9.228.2.1 data

`uint8_t nvinfer1::v_1_0::TimingCacheKey::data[16]`

The documentation for this struct was generated from the following file:

- NvInfer.h

9.229 nvinfer1::v_1_0::TimingCacheValue Struct Reference

`#include <NvInfer.h>`

Public Attributes

- uint64_t tacticHash

  Hash of the selected tactic.
- float timingMSec

  Timing of this tactic in milliseconds. Negative numbers and NaN are invalid values.

Static Public Attributes

- static constexpr uint64_t kINVALID_TACTIC_HASH = UINT64_MAX

  UINT64_MAX represents the invalid tactic hash.

9.229.1 Member Data Documentation

9.229.1.1 kINVALID_TACTIC_HASH

```
constexpr uint64_t nvinfer1::v_1_0::TimingCacheValue::kINVALID_TACTIC_HASH = UINT64_MAX  [static],
[constexpr]
```

UINT64_MAX represents the invalid tactic hash.

9.229.1.2 tacticHash

```
uint64_t nvinfer1::v_1_0::TimingCacheValue::tacticHash
```

Hash of the selected tactic.

9.229.1.3 timingMSec

```
float nvinfer1::v_1_0::TimingCacheValue::timingMSec
```

Timing of this tactic in milliseconds. Negative numbers and NaN are invalid values.

The documentation for this struct was generated from the following file:

- NvInfer.h

9.230 nvinfer2::safe::TypedArray Class Reference

A standard_layout and trivially_copyable typed array that knows the data type and size it is holding.

`#include <NvInferSafeRuntime.h>`

Public Member Functions

- TypedArray () noexcept
- TypedArray (TypedArray const &)=default
- TypedArray (TypedArray &&)=default
- TypedArray & operator= (TypedArray const &) &=default
- TypedArray & operator= (TypedArray &&) &=default
- ~TypedArray () noexcept=default
- TypedArray (float *ptr, uint64_t const bufferSize) noexcept

  sets the data to a float ptr. It also sets the current type to kFLOAT
- TypedArray (half_t *ptr, uint64_t const bufferSize) noexcept

  sets the data to a half_t ptr. It also sets the current type to kHALF
- TypedArray (int64_t *ptr, uint64_t const bufferSize) noexcept

  sets the data to a int64_t ptr. It also sets the current type to kINT64
- TypedArray (int32_t *ptr, uint64_t const bufferSize) noexcept

  sets the data to a int32_t ptr. It also sets the current type to kINT32
- TypedArray (int8_t *ptr, uint64_t const bufferSize) noexcept

  sets the data to a int8_t ptr. It also sets the current type to kINT8
- TypedArray (uint8_t *ptr, uint64_t const bufferSize) noexcept

  sets the data to a uint8_t ptr. It also sets the current type to kUINT8
- TypedArray (bool *ptr, uint64_t const bufferSize) noexcept

  sets the data to a bool ptr. It also sets the current type to kBOOL
- DataType getType () const noexcept

  This method returns the current type of the data.
- float * getFloat () const noexcept

  Retrieves the data as float. It should only be called when the current type is kFLOAT. Returns nullptr otherwise.*
- half_t * getHalf () const noexcept

  Retrieves the data as half_t. It should only be called when the current type is kHALF. Returns nullptr otherwise.*
- int64_t * getInt64 () const noexcept

  Retrieves the data as int64_t. It should only be called when the current type is kINT64. Returns nullptr otherwise.*
- int32_t * getInt32 () const noexcept

  Retrieves the data as int32_t. It should only be called when the current type is kINT32. Returns nullptr otherwise.*
- int8_t * getInt8 () const noexcept

  Retrieves the data as int8_t. It should only be called when the current type is kINT8. Returns nullptr otherwise.*
- uint8_t * getUint8 () const noexcept

  Retrieves the data as uint8_t. It should only be called when the current type is kUINT8. Returns nullptr otherwise.*
- bool * getBool () const noexcept

  Retrieves the data as bool. It should only be called when the current type is kBOOL. Returns nullptr otherwise.*
- uint64_t getSize () const noexcept

  Retrieves the size of the array in bytes.
- void * getData () const noexcept

  Retrieves the data regardless of the type.

9.230.1 Detailed Description

A standard_layout and trivially_copyable typed array that knows the data type and size it is holding.

Warning

The user of this class is responsible for making sure the data pointer is set to a valid piece of memory

Precondition

user provides a valid data ptr for this object before passing it to TRT's API

9.230.2 Constructor & Destructor Documentation

9.230.2.1 TypedArray() [1/10]

```
nvinfer2::safe::TypedArray::TypedArray ( )  [inline], [noexcept]
```

9.230.2.2 TypedArray() [2/10]

```
nvinfer2::safe::TypedArray::TypedArray (
            TypedArray const &  )  [default]
```

9.230.2.3 TypedArray() [3/10]

```
nvinfer2::safe::TypedArray::TypedArray (
            TypedArray &&  )  [default]
```

9.230.2.4 ~TypedArray()

```
nvinfer2::safe::TypedArray::~TypedArray ( )  [default], [noexcept]
```

9.230.2.5 TypedArray() [4/10]

```
nvinfer2::safe::TypedArray::TypedArray (
            float * ptr,
            uint64_t const bufferSize )  [inline], [noexcept]
```

sets the data to a float ptr. It also sets the current type to kFLOAT

9.230.2.6 TypedArray() [5/10]

```
nvinfer2::safe::TypedArray::TypedArray (
            half_t * ptr,
            uint64_t const bufferSize )  [inline], [noexcept]
```

sets the data to a half_t ptr. It also sets the current type to kHALF

9.230.2.7 TypedArray() [6/10]

```
nvinfer2::safe::TypedArray::TypedArray (
            int64_t * ptr,
            uint64_t const bufferSize )  [inline], [noexcept]
```

sets the data to a int64_t ptr. It also sets the current type to kINT64

9.230.2.8 TypedArray() [7/10]

```
nvinfer2::safe::TypedArray::TypedArray (
            int32_t * ptr,
            uint64_t const bufferSize )  [inline], [noexcept]
```

sets the data to a int32_t ptr. It also sets the current type to kINT32

9.230.2.9 TypedArray() [8/10]

```
nvinfer2::safe::TypedArray::TypedArray (
            int8_t * ptr,
            uint64_t const bufferSize )  [inline], [noexcept]
```

sets the data to a int8_t ptr. It also sets the current type to kINT8

9.230.2.10 TypedArray() [9/10]

```
nvinfer2::safe::TypedArray::TypedArray (
            uint8_t * ptr,
            uint64_t const bufferSize )  [inline], [noexcept]
```

sets the data to a uint8_t ptr. It also sets the current type to kUINT8

9.230.2.11 TypedArray() [10/10]

```
nvinfer2::safe::TypedArray::TypedArray (
            bool * ptr,
            uint64_t const bufferSize )  [inline], [noexcept]
```

sets the data to a bool ptr. It also sets the current type to kBOOL

9.230.3 Member Function Documentation

9.230.3.1 getBool()

```
bool * nvinfer2::safe::TypedArray::getBool ( ) const  [inline], [noexcept]
```

Retrieves the data as bool∗. It should only be called when the current type is kBOOL. Returns nullptr otherwise.

9.230.3.2 getData()

```
void * nvinfer2::safe::TypedArray::getData ( ) const  [inline], [noexcept]
```

Retrieves the data regardless of the type.

Note

This is an internal method, not to be used outside of TensorRT safe runtime APIs.

9.230.3.3 getFloat()

```
float * nvinfer2::safe::TypedArray::getFloat ( ) const  [inline], [noexcept]
```

Retrieves the data as float∗. It should only be called when the current type is kFLOAT. Returns nullptr otherwise.

9.230.3.4 getHalf()

```
half_t * nvinfer2::safe::TypedArray::getHalf ( ) const  [inline], [noexcept]
```

Retrieves the data as half_t∗. It should only be called when the current type is kHALF. Returns nullptr otherwise.

9.230.3.5 getInt32()

```
int32_t * nvinfer2::safe::TypedArray::getInt32 ( ) const  [inline], [noexcept]
```

Retrieves the data as int32_t∗. It should only be called when the current type is kINT32. Returns nullptr otherwise.

9.230.3.6 getInt64()

```
int64_t * nvinfer2::safe::TypedArray::getInt64 ( ) const  [inline], [noexcept]
```

Retrieves the data as int64_t∗. It should only be called when the current type is kINT64. Returns nullptr otherwise.

9.230.3.7 getInt8()

```
int8_t * nvinfer2::safe::TypedArray::getInt8 ( ) const  [inline], [noexcept]
```

Retrieves the data as int8_t∗. It should only be called when the current type is kINT8. Returns nullptr otherwise.

9.230.3.8 getSize()

```
uint64_t nvinfer2::safe::TypedArray::getSize ( ) const  [inline], [noexcept]
```

Retrieves the size of the array in bytes.

9.230.3.9 getType()

```
DataType nvinfer2::safe::TypedArray::getType ( ) const  [inline], [noexcept]
```

This method returns the current type of the data.

9.230.3.10 getUint8()

```
uint8_t * nvinfer2::safe::TypedArray::getUint8 ( ) const  [inline], [noexcept]
```

Retrieves the data as uint8_t*. It should only be called when the current type is kUINT8. Returns nullptr otherwise.

9.230.3.11 operator=() [1/2]

```
TypedArray & nvinfer2::safe::TypedArray::operator= (
            TypedArray &&  ) &  [default]
```

9.230.3.12 operator=() [2/2]

```
TypedArray & nvinfer2::safe::TypedArray::operator= (
            TypedArray const &  ) &  [default]
```

The documentation for this class was generated from the following file:

- NvInferSafeRuntime.h

9.231 Value Struct Reference

The values in the cache entry.

```
#include <NvInfer.h>
```

9.231.1 Detailed Description

The values in the cache entry.

See also

ITimingCache::query(), ITimingCache::update()

The documentation for this struct was generated from the following file:

- NvInfer.h

9.232 nvinfer1::Weights Class Reference

An array of weights used as a layer parameter.

`#include <NvInferRuntime.h>`

Public Attributes

- DataType type
  The type of the weights.
- void const * values
  The weight values, in a contiguous array.
- int64_t count
  The number of weights in the array.

9.232.1 Detailed Description

An array of weights used as a layer parameter.

When using the DLA, the cumulative size of all Weights used in a network must be less than 512MB in size. If the build option kGPU_FALLBACK is specified, then multiple DLA sub-networks may be generated from the single original network.

The weights are held by reference until the engine has been built. Therefore the data referenced by `values` field should be preserved until the build is complete.

The term ”empty weights” refers to Weights with weight coefficients (`count` == 0 and `values` == nullptr).

9.232.2 Member Data Documentation

9.232.2.1 count

```
int64_t nvinfer1::Weights::count
```

The number of weights in the array.

9.232.2.2 type

```
DataType nvinfer1::Weights::type
```

The type of the weights.

9.232.2.3 values

```
void const* nvinfer1::Weights::values
```

The weight values, in a contiguous array.

The documentation for this class was generated from the following file:

- NvInferRuntime.h

Chapter 10

File Documentation

10.1 NvInfer.h File Reference

```
#include "NvInferLegacyDims.h"
#include "NvInferRuntime.h"
```

Classes

- struct nvinfer1::impl::EnumMaxImpl< LayerType >
- struct nvinfer1::impl::EnumMaxImpl< ActivationType >
- class nvinfer1::ITensor

  A tensor in a network definition.
- class nvinfer1::ILayer

  Base class for all layer classes in a network definition.
- struct nvinfer1::impl::EnumMaxImpl< PaddingMode >
- class nvinfer1::IConvolutionLayer

  A convolution layer in a network definition.
- class nvinfer1::IActivationLayer

  An Activation layer in a network definition.
- struct nvinfer1::impl::EnumMaxImpl< PoolingType >
- class nvinfer1::IPoolingLayer

  A Pooling layer in a network definition.
- class nvinfer1::ILRNLayer

  A LRN layer in a network definition.
- struct nvinfer1::impl::EnumMaxImpl< ScaleMode >
- class nvinfer1::IScaleLayer

  A Scale layer in a network definition.
- class nvinfer1::ISoftMaxLayer

  A Softmax layer in a network definition.
- class nvinfer1::IConcatenationLayer

  A concatenation layer in a network definition.

- class nvinfer1::IDeconvolutionLayer

  A deconvolution layer in a network definition.
- struct nvinfer1::impl::EnumMaxImpl< ElementWiseOperation >
- class nvinfer1::IElementWiseLayer

  A elementwise layer in a network definition.
- struct nvinfer1::impl::EnumMaxImpl< GatherMode >
- class nvinfer1::IGatherLayer

  A Gather layer in a network definition. Supports several kinds of gathering.
- class nvinfer1::IPluginV2Layer

  Layer type for pluginV2.
- class nvinfer1::IPluginV3Layer

  Layer type for V3 plugins.
- struct nvinfer1::impl::EnumMaxImpl< UnaryOperation >
- class nvinfer1::IUnaryLayer

  Layer that represents an unary operation.
- struct nvinfer1::impl::EnumMaxImpl< ReduceOperation >
- struct nvinfer1::impl::EnumMaxImpl< CollectiveOperation >
- class nvinfer1::IReduceLayer

  Layer that represents a reduction across a non-bool tensor.
- class nvinfer1::IPaddingLayer

  Layer that represents a padding operation.
- struct nvinfer1::Permutation

  Represents a permutation of dimensions.
- class nvinfer1::IShuffleLayer

  Layer type for shuffling data.
- struct nvinfer1::impl::EnumMaxImpl< SampleMode >
- class nvinfer1::ISliceLayer

  Slices an input tensor into an output tensor based on the offset and strides.
- class nvinfer1::IShapeLayer

  Layer type for getting shape of a tensor.
- struct nvinfer1::impl::EnumMaxImpl< TopKOperation >
- class nvinfer1::ITopKLayer

  Layer that represents a TopK reduction.
- struct nvinfer1::impl::EnumMaxImpl< MatrixOperation >
- class nvinfer1::IMatrixMultiplyLayer

  Layer that represents a Matrix Multiplication.
- class nvinfer1::INonZeroLayer
- class nvinfer1::IRaggedSoftMaxLayer

  A RaggedSoftmax layer in a network definition.
- class nvinfer1::IIdentityLayer

  A layer that represents the identity function.
- class nvinfer1::ICastLayer

  A cast layer in a network.
- class nvinfer1::IConstantLayer

  Layer that represents a constant value.
- class nvinfer1::IParametricReLULayer

  Layer that represents a parametric ReLU operation.
- struct nvinfer1::impl::EnumMaxImpl< InterpolationMode >

- struct nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation >
- struct nvinfer1::impl::EnumMaxImpl< ResizeSelector >
- struct nvinfer1::impl::EnumMaxImpl< ResizeRoundMode >
- class nvinfer1::IResizeLayer

  A resize layer in a network definition.
- struct nvinfer1::impl::EnumMaxImpl< LoopOutput >
- struct nvinfer1::impl::EnumMaxImpl< TripLimit >
- class nvinfer1::ILoopBoundaryLayer

  This is a base class for Loop boundary layers.
- class nvinfer1::IIfConditionalBoundaryLayer

  This is a base class for Conditional boundary layers.
- class nvinfer1::IConditionLayer

  This layer represents a condition input to an IIfConditional.
- class nvinfer1::IIfConditionalOutputLayer

  This layer represents an output of an IIfConditional.
- class nvinfer1::IIfConditionalInputLayer

  This layer represents an input to an IIfConditional.
- class nvinfer1::IIfConditional

  Helper for constructing conditionally-executed subgraphs.
- class nvinfer1::IRecurrenceLayer

  A recurrence layer in a network definition.
- class nvinfer1::ILoopOutputLayer

  An ILoopOutputLayer is the sole way to get output from a loop.
- class nvinfer1::ITripLimitLayer

  A layer that represents a trip-count limiter.
- class nvinfer1::IIteratorLayer

  A layer to do iterations.
- class nvinfer1::ILoop

  Helper for creating a recurrent subgraph.
- class nvinfer1::ISelectLayer

  Select elements from two data tensors based on a condition tensor.
- class nvinfer1::IAssertionLayer

  An assertion layer in a network.
- struct nvinfer1::impl::EnumMaxImpl< FillOperation >
- class nvinfer1::IFillLayer

  Generate a tensor according to a specified mode.
- class nvinfer1::IQuantizeLayer

  A Quantize layer in a network definition.
- class nvinfer1::IDequantizeLayer

  A Dequantize layer in a network definition.
- class nvinfer1::IDynamicQuantizeLayer

  A network layer to perform dynamic quantization.
- class nvinfer1::IEinsumLayer

  An Einsum layer in a network.
- struct nvinfer1::impl::EnumMaxImpl< ScatterMode >
- class nvinfer1::IScatterLayer

  A scatter layer in a network definition. Supports several kinds of scattering.

- class nvinfer1::IOneHotLayer

  A OneHot layer in a network definition.
- class nvinfer1::IGridSampleLayer

  A GridSample layer in a network definition.
- struct nvinfer1::impl::EnumMaxImpl< BoundingBoxFormat >
- class nvinfer1::INMSLayer

  A non-maximum suppression layer in a network definition.
- class nvinfer1::IReverseSequenceLayer

  A ReverseSequence layer in a network definition.
- class nvinfer1::INormalizationLayer

  A normalization layer in a network definition.
- class nvinfer1::ISqueezeLayer

  Layer that represents a squeeze operation, removing unit dimensions of the first input tensor on a set of axes specified by the second input tensor.
- class nvinfer1::IUnsqueezeLayer

  Layer that represents an unsqueeze operation, which reshapes the first input tensor by inserting unit-length dimensions to the output at the axes specified by the second input tensor.
- struct nvinfer1::impl::EnumMaxImpl< CumulativeOperation >

  Maximum number of elements in CumulativeOperation enum.
- class nvinfer1::ICumulativeLayer

  Layer that represents a cumulative operation across a tensor.
- struct nvinfer1::impl::EnumMaxImpl< AttentionNormalizationOp >
- struct nvinfer1::impl::EnumMaxImpl< CausalMaskKind >
- struct nvinfer1::impl::EnumMaxImpl< AttentionIOForm >
- class nvinfer1::IAttentionBoundaryLayer

  This is a base class for Attention boundary layers.
- class nvinfer1::IAttentionInputLayer

  This layer represents an input to an attention subgraph.
- class nvinfer1::IAttentionOutputLayer

  This layer represents an output of an IAttention.
- class nvinfer1::IAttention

  Helper for constructing an attention that consumes query, key and value tensors.
- class nvinfer1::IRotaryEmbeddingLayer

  Layer that implements Rotary Position Embedding (RoPE) (https://arxiv.org/abs/2104.09864).
- struct nvinfer1::impl::EnumMaxImpl< KVCacheMode >
- class nvinfer1::IKVCacheUpdateLayer

  Layer that represents a KVCacheUpdate operation.
- struct nvinfer1::impl::EnumMaxImpl< MoEActType >
- class nvinfer1::IMoELayer

  A MoE layer in a network definition. Mixture of Experts (MoE) is a collection of experts with each expert specializing in processing different subsets of input data. The key innovation lies in using a Router that selectively activates only the specific experts needed for a given input, rather than engaging the entire neural network for every task.
- class nvinfer1::IDistCollectiveLayer
- class nvinfer1::INetworkDefinition

  A network definition for input to the builder.
- struct nvinfer1::impl::EnumMaxImpl< RuntimePlatform >
- struct nvinfer1::impl::EnumMaxImpl< BuilderFlag >
- struct nvinfer1::v_1_0::TimingCacheKey

The key to retrieve timing cache entries.

- struct nvinfer1::v_1_0::TimingCacheValue
- class nvinfer1::ITimingCache

  Class to handle tactic timing info collected from builder.
- struct nvinfer1::impl::EnumMaxImpl< MemoryPoolType >
- struct nvinfer1::impl::EnumMaxImpl< PreviewFeature >
- struct nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel >
- struct nvinfer1::impl::EnumMaxImpl< TilingOptimizationLevel >
- class nvinfer1::v_1_0::IProgressMonitor
- class nvinfer1::IBuilderConfig

  Holds properties for configuring a builder to produce an engine.
- struct nvinfer1::impl::EnumMaxImpl< NetworkDefinitionCreationFlag >
- class nvinfer1::IBuilder

  Builds an engine from a network definition.

Namespaces

- namespace nvinfer1

  The TensorRT API version 1 namespace.
- namespace nvinfer1::v_1_0

  Forward declare IErrorRecorder and ILogger for use in other interfaces.
- namespace nvinfer1::anonymous_namespace{NvInfer.h}
- namespace nvinfer1::safe

Typedefs

- using nvinfer1::TensorFormats = uint32_t

  It is capable of representing one or more TensorFormat by binary OR operations, e.g., 1U << TensorFormat::kCHW4 | 1U << TensorFormat::kCHW32.
- using nvinfer1::BuilderFlags = uint32_t

  Represents one or more BuilderFlag values using binary OR operations, e.g., 1U << BuilderFlag::kDEBUG.
- using nvinfer1::IProgressMonitor = v_1_0::IProgressMonitor
- using nvinfer1::NetworkDefinitionCreationFlags = uint32_t

  Represents one or more NetworkDefinitionCreationFlag flags using binary OR operations. e.g., 1U << NetworkDefinitionCreationFlag::kPREFER_JIT_PYTHON_PLUGINS.

Enumerations

- enum class nvinfer1::LayerType : int32_t {
  nvinfer1::kCONVOLUTION = 0 , nvinfer1::kCAST = 1 , nvinfer1::kACTIVATION = 2 , nvinfer1::kPOOLING = 3 ,
  nvinfer1::kLRN = 4 , nvinfer1::kSCALE = 5 , nvinfer1::kSOFTMAX = 6 , nvinfer1::kDECONVOLUTION = 7 ,
  nvinfer1::kCONCATENATION = 8 , nvinfer1::kELEMENTWISE = 9 , nvinfer1::kPLUGIN = 10 , nvinfer1::kUNARY = 11 ,
  nvinfer1::kPADDING = 12 , nvinfer1::kSHUFFLE = 13 , nvinfer1::kREDUCE = 14 , nvinfer1::kTOPK = 15 ,
  nvinfer1::kGATHER = 16 , nvinfer1::kMATRIX_MULTIPLY = 17 , nvinfer1::kRAGGED_SOFTMAX = 18 ,

nvinfer1::kCONSTANT = 19 ,
nvinfer1::kIDENTITY = 20 , nvinfer1::kPLUGIN_V2 = 21 , nvinfer1::kSLICE = 22 , nvinfer1::kSHAPE = 23 ,
nvinfer1::kPARAMETRIC_RELU = 24 , nvinfer1::kRESIZE = 25 , nvinfer1::kTRIP_LIMIT = 26 ,
nvinfer1::kRECURRENCE = 27 ,
nvinfer1::kITERATOR = 28 , nvinfer1::kLOOP_OUTPUT = 29 , nvinfer1::kSELECT = 30 , nvinfer1::kFILL =
31 ,
nvinfer1::kQUANTIZE = 32 , nvinfer1::kDEQUANTIZE = 33 , nvinfer1::kCONDITION = 34 , nvinfer1::kCONDITIONAL_INPU
= 35 ,
nvinfer1::kCONDITIONAL_OUTPUT = 36 , nvinfer1::kSCATTER = 37 , nvinfer1::kEINSUM = 38 ,
nvinfer1::kASSERTION = 39 ,
nvinfer1::kONE_HOT = 40 , nvinfer1::kNON_ZERO = 41 , nvinfer1::kGRID_SAMPLE = 42 , nvinfer1::kNMS
= 43 ,
nvinfer1::kREVERSE_SEQUENCE = 44 , nvinfer1::kNORMALIZATION = 45 , nvinfer1::kPLUGIN_V3 = 46
, nvinfer1::kSQUEEZE = 47 ,
nvinfer1::kUNSQUEEZE = 48 , nvinfer1::kCUMULATIVE = 49 , nvinfer1::kDYNAMIC_QUANTIZE = 50 ,
nvinfer1::kATTENTION_INPUT = 51 ,
nvinfer1::kATTENTION_OUTPUT = 52 , nvinfer1::kROTARY_EMBEDDING = 53 , nvinfer1::kKVCACHE_UPDATE
= 54 , nvinfer1::kMOE = 55 ,
nvinfer1::kDIST_COLLECTIVE = 56 }

The type values of layer classes.

- enum class nvinfer1::ActivationType : int32_t {
nvinfer1::kRELU = 0 , nvinfer1::kSIGMOID = 1 , nvinfer1::kTANH = 2 , nvinfer1::kLEAKY_RELU = 3 ,
nvinfer1::kELU = 4 , nvinfer1::kSELU = 5 , nvinfer1::kSOFTSIGN = 6 , nvinfer1::kSOFTPLUS = 7 ,
nvinfer1::kCLIP = 8 , nvinfer1::kHARD_SIGMOID = 9 , nvinfer1::kSCALED_TANH = 10 , nvinfer1::kTHRESHOLDED_RELU
= 11 ,
nvinfer1::kGELU_ERF = 12 , nvinfer1::kGELU_TANH = 13 }

  Enumerates the types of activation to perform in an activation layer.

- enum class nvinfer1::PaddingMode : int32_t { nvinfer1::kEXPLICIT_ROUND_DOWN = 0 , nvinfer1::kEXPLICIT_ROUND_UP
= 1 , nvinfer1::kSAME_UPPER = 2 , nvinfer1::kSAME_LOWER = 3 }

  Enumerates the modes of padding to perform in convolution, deconvolution and pooling layer, padding mode takes precedence if setPaddingMode() and setPrePadding() are also used.

- enum class nvinfer1::PoolingType : int32_t { nvinfer1::kMAX = 0 , nvinfer1::kAVERAGE = 1 ,
nvinfer1::kMAX_AVERAGE_BLEND = 2 }

  The type of pooling to perform in a pooling layer.

- enum class nvinfer1::ScaleMode : int32_t { nvinfer1::kUNIFORM = 0 , nvinfer1::kCHANNEL = 1 ,
nvinfer1::kELEMENTWISE = 2 }

  Controls how shift, scale and power are applied in a Scale layer.

- enum class nvinfer1::ElementWiseOperation : int32_t {
nvinfer1::kSUM = 0 , nvinfer1::kPROD = 1 , nvinfer1::kMAX = 2 , nvinfer1::kMIN = 3 ,
nvinfer1::kSUB = 4 , nvinfer1::kDIV = 5 , nvinfer1::kPOW = 6 , nvinfer1::kFLOOR_DIV = 7 ,
nvinfer1::kAND = 8 , nvinfer1::kOR = 9 , nvinfer1::kXOR = 10 , nvinfer1::kEQUAL = 11 ,
nvinfer1::kGREATER = 12 , nvinfer1::kLESS = 13 }

  Enumerates the binary operations that may be performed by an ElementWise layer.

- enum class nvinfer1::GatherMode : int32_t { nvinfer1::kDEFAULT = 0 , nvinfer1::kELEMENT = 1 ,
nvinfer1::kND = 2 }

  Control form of IGatherLayer.

- enum class nvinfer1::UnaryOperation : int32_t {
nvinfer1::kEXP = 0 , nvinfer1::kLOG = 1 , nvinfer1::kSQRT = 2 , nvinfer1::kRECIP = 3 ,
nvinfer1::kABS = 4 , nvinfer1::kNEG = 5 , nvinfer1::kSIN = 6 , nvinfer1::kCOS = 7 ,
nvinfer1::kTAN = 8 , nvinfer1::kSINH = 9 , nvinfer1::kCOSH = 10 , nvinfer1::kASIN = 11 ,
nvinfer1::kACOS = 12 , nvinfer1::kATAN = 13 , nvinfer1::kASINH = 14 , nvinfer1::kACOSH = 15 ,
nvinfer1::kATANH = 16 , nvinfer1::kCEIL = 17 , nvinfer1::kFLOOR = 18 , nvinfer1::kERF = 19 ,

nvinfer1::kNOT = 20 , nvinfer1::kSIGN = 21 , nvinfer1::kROUND = 22 , nvinfer1::kISINF = 23 , nvinfer1::kISNAN = 24 }

Enumerates the unary operations that may be performed by a Unary layer.

- enum class nvinfer1::ReduceOperation : int32_t { nvinfer1::kSUM = 0 , nvinfer1::kPROD = 1 , nvinfer1::kMAX = 2 , nvinfer1::kMIN = 3 , nvinfer1::kAVG = 4 , nvinfer1::kNONE = 5 }

  Enumerates the reduce operations that may be performed by a Reduce layer.
- enum class nvinfer1::CollectiveOperation : int32_t { nvinfer1::kALL_REDUCE = 0 , nvinfer1::kALL_GATHER = 1 , nvinfer1::kBROADCAST = 2 , nvinfer1::kREDUCE = 3 , nvinfer1::kREDUCE_SCATTER = 4 , nvinfer1::kALL_TO_ALL = 5 , nvinfer1::kGATHER = 6 , nvinfer1::kSCATTER = 7 }

  Enumerates the collective operations that may be performed by a DistCollective layer.
- enum class nvinfer1::SampleMode : int32_t { nvinfer1::kSTRICT_BOUNDS = 0 , nvinfer1::kWRAP = 1 , nvinfer1::kCLAMP = 2 , nvinfer1::kFILL = 3 , nvinfer1::kREFLECT = 4 }

  Controls how ISliceLayer and IGridSample handle out-of-bounds coordinates.
- enum class nvinfer1::TopKOperation : int32_t { nvinfer1::kMAX = 0 , nvinfer1::kMIN = 1 }

  Enumerates the operations that may be performed by a TopK layer.
- enum class nvinfer1::MatrixOperation : int32_t { nvinfer1::kNONE = 0 , nvinfer1::kTRANSPOSE = 1 , nvinfer1::kVECTOR = 2 }

  Enumerates the operations that may be performed on a tensor by IMatrixMultiplyLayer before multiplication.
- enum class nvinfer1::InterpolationMode : int32_t { nvinfer1::kNEAREST = 0 , nvinfer1::kLINEAR = 1 , nvinfer1::kCUBIC = 2 }

  Enumerates various modes of interpolation.
- enum class nvinfer1::ResizeCoordinateTransformation : int32_t { nvinfer1::kALIGN_CORNERS = 0 , nvinfer1::kASYMMETRIC = 1 , nvinfer1::kHALF_PIXEL = 2 }

  The resize coordinate transformation function.
- enum class nvinfer1::ResizeSelector : int32_t { nvinfer1::kFORMULA = 0 , nvinfer1::kUPPER = 1 }

  The coordinate selector when resize to single pixel output.
- enum class nvinfer1::ResizeRoundMode : int32_t { nvinfer1::kHALF_UP = 0 , nvinfer1::kHALF_DOWN = 1 , nvinfer1::kFLOOR = 2 , nvinfer1::kCEIL = 3 }

  The rounding mode for nearest neighbor resize.
- enum class nvinfer1::LoopOutput : int32_t { nvinfer1::kLAST_VALUE = 0 , nvinfer1::kCONCATENATE = 1 , nvinfer1::kREVERSE = 2 }

  Enum that describes kinds of loop outputs.
- enum class nvinfer1::TripLimit : int32_t { nvinfer1::kCOUNT = 0 , nvinfer1::kWHILE = 1 }

  Enum that describes kinds of trip limits.
- enum class nvinfer1::FillOperation : int32_t { nvinfer1::kLINSPACE = 0 , nvinfer1::kRANDOM_UNIFORM = 1 , nvinfer1::kRANDOM_NORMAL = 2 }

  Enumerates the tensor fill operations that may performed by a fill layer.
- enum class nvinfer1::ScatterMode : int32_t { nvinfer1::kELEMENT = 0 , nvinfer1::kND = 1 }

  Control form of IScatterLayer.
- enum class nvinfer1::BoundingBoxFormat : int32_t { nvinfer1::kCORNER_PAIRS = 0 , nvinfer1::kCENTER_SIZES = 1 }

  Representation of bounding box data used for the Boxes input tensor in INMSLayer.
- enum class nvinfer1::CumulativeOperation : int32_t { nvinfer1::kSUM = 0 }

  Enumerates the cumulative operations that may be performed by a Cumulative layer.
- enum class nvinfer1::AttentionNormalizationOp : int32_t { nvinfer1::kNONE = 0 , nvinfer1::kSOFTMAX = 1 }

Enumerates the operations that may be performed by the normalization in the attention subgraph.

- enum class nvinfer1::CausalMaskKind : int32_t { nvinfer1::kNONE = 0 , nvinfer1::kUPPER_LEFT = 1 , nvinfer1::kLOWER_RIGHT = 2 }

  Enumerates the causal mask alignment orientation for the attention.

- enum class nvinfer1::AttentionIOForm : int32_t { nvinfer1::kPADDED_BHND = 0 , nvinfer1::kPACKED_NHD = 1 }

  Enumerates the layout of the input/output tensors in an Attention layer.

- enum class nvinfer1::KVCacheMode : int32_t { nvinfer1::kLINEAR = 0 }

  Enumerates the KVCache modes that may be performed by a KVCacheUpdate layer.

- enum class nvinfer1::MoEActType : int32_t { nvinfer1::kNONE = 0 , nvinfer1::kSILU = 1 }

  Enumerates the activation type for the MoE layer.

- enum class nvinfer1::RuntimePlatform : int32_t { nvinfer1::kSAME_AS_BUILD = 0 , nvinfer1::kWINDOWS_AMD64 = 1 }

  Describes the intended runtime platform (operating system and CPU architecture) for the execution of the TensorRT engine. TensorRT provides support for cross-platform engine compatibility when the target runtime platform is different from the build platform.

- enum class nvinfer1::BuilderFlag : int32_t {
  nvinfer1::kDEBUG = 2 , nvinfer1::kGPU_FALLBACK = 3 , nvinfer1::kREFIT = 4 , nvinfer1::kDISABLE_TIMING_CACHE = 5 ,
  nvinfer1::kTF32 = 6 , nvinfer1::kSPARSE_WEIGHTS = 7 , nvinfer1::kSAFETY_SCOPE = 8 , nvinfer1::kDIRECT_IO = 11 ,
  nvinfer1::kVERSION_COMPATIBLE = 12 , nvinfer1::kEXCLUDE_LEAN_RUNTIME = 13 , nvinfer1::kERROR_ON_TIMING_C = 15 , nvinfer1::kDISABLE_COMPILATION_CACHE = 17 ,
  nvinfer1::kSTRIP_PLAN = 18 , nvinfer1::kREFIT_IDENTICAL = 19 , nvinfer1::kWEIGHT_STREAMING = 20 , nvinfer1::kREFIT_INDIVIDUAL = 22 ,
  nvinfer1::kSTRICT_NANS = 23 , nvinfer1::kMONITOR_MEMORY = 24 , nvinfer1::kEDITABLE_TIMING_CACHE = 26 , nvinfer1::kDISTRIBUTIVE_INDEPENDENCE = 27 }

  List of valid modes that the builder can enable when creating an engine from a network definition.

- enum class nvinfer1::MemoryPoolType : int32_t {
  nvinfer1::kWORKSPACE = 0 , nvinfer1::kDLA_MANAGED_SRAM = 1 , nvinfer1::kDLA_LOCAL_DRAM = 2 , nvinfer1::kDLA_GLOBAL_DRAM = 3 ,
  nvinfer1::kTACTIC_DRAM = 4 , nvinfer1::kTACTIC_SHARED_MEMORY = 5 }

  The type for memory pools used by TensorRT.

- enum class nvinfer1::PreviewFeature : int32_t { nvinfer1::kALIASED_PLUGIN_IO_10_03 = 0 , nvinfer1::kRUNTIME_ACTIVATI = 1 }

  Define preview features.

- enum class nvinfer1::HardwareCompatibilityLevel : int32_t { nvinfer1::kNONE = 0 , nvinfer1::kAMPERE_PLUS = 1 , nvinfer1::kSAME_COMPUTE_CAPABILITY = 2 }

  Describes requirements of compatibility with GPU architectures other than that of the GPU on which the engine was built.

- enum class nvinfer1::TilingOptimizationLevel : int32_t { nvinfer1::kNONE = 0 , nvinfer1::kFAST = 1 , nvinfer1::kMODERATE = 2 , nvinfer1::kFULL = 3 }

  Define the optimization levels for Tiling.

- enum class nvinfer1::NetworkDefinitionCreationFlag : int32_t { nvinfer1::kSTRONGLY_TYPED = 0 , nvinfer1::kPREFER_JIT_PYTHON_PLUGINS = 1 , nvinfer1::kPREFER_AOT_PYTHON_PLUGINS = 2 }

  List of immutable network properties expressed at network creation time. NetworkDefinitionCreationFlag is used with createNetworkV2() to specify immutable properties of the network.

Functions

- IBuilder * nvinfer1::anonymous_namespace{NvInfer.h}::createInferBuilder (ILogger &logger) noexcept

  Create an instance of an IBuilder class.
- TENSORRTAPI nvinfer1::IPluginRegistry * nvinfer1::getBuilderPluginRegistry (nvinfer1::EngineCapability capability) noexcept

  Return the plugin registry for building a Standard engine, or nullptr if no registry exists.
- TENSORRTAPI bool nvinfer1::setInternalLibraryPath (AsciiChar const *path) noexcept

  Set a custom directory path for loading internal TensorRT libraries when building engines.
- TRT_DEPRECATED_API nvinfer1::safe::IPluginRegistry * nvinfer1::getBuilderSafePluginRegistry (nvinfer1::EngineCapability capability) noexcept

  Return the plugin registry for building a Safety engine, or nullptr if no registry exists.

10.1.1 Detailed Description

This is the top-level API file for TensorRT.

10.2 NvInfer.h

Go to the documentation of this file.

```
1 /*
2 * SPDX-FileCopyrightText:  Copyright (c) 1993-2026 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3 * SPDX-License-Identifier:  Apache-2.0
4 *
5 * Licensed under the Apache License, Version 2.0 (the "License");
6 * you may not use this file except in compliance with the License.
7 * You may obtain a copy of the License at
8 *
9 * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17
18 #ifndef NV_INFER_H
19 #define NV_INFER_H
20
21 #include "NvInferLegacyDims.h"
22 #include "NvInferRuntime.h" // IWYU pragma:  export
23
35
41
47 namespace nvinfer1
48 {
49
57 enum class LayerType :  int32_t
58 {
59     kCONVOLUTION = 0,
60     kCAST = 1,
61     kACTIVATION = 2,
62     kPOOLING = 3,
63     kLRN = 4,
64     kSCALE = 5,
65     kSOFTMAX = 6,
66     kDECONVOLUTION = 7,
67     kCONCATENATION = 8,
68     kELEMENTWISE = 9,
69     kPLUGIN = 10,
70     kUNARY = 11,
71     kPADDING = 12,
```

```
72      kSHUFFLE = 13,
73      kREDUCE = 14,
74      kTOPK = 15,
75      kGATHER = 16,
76      kMATRIX_MULTIPLY = 17,
77      kRAGGED_SOFTMAX = 18,
78      kCONSTANT = 19,
79      kIDENTITY = 20,
80      kPLUGIN_V2 = 21,
81      kSLICE = 22,
82      kSHAPE = 23,
83      kPARAMETRIC_RELU = 24,
84      kRESIZE = 25,
85      kTRIP_LIMIT = 26,
86      kRECURRENCE = 27,
87      kITERATOR = 28,
88      kLOOP_OUTPUT = 29,
89      kSELECT = 30,
90      kFILL = 31,
91      kQUANTIZE = 32,
92      kDEQUANTIZE = 33,
93      kCONDITION = 34,
94      kCONDITIONAL_INPUT = 35,
95      kCONDITIONAL_OUTPUT = 36,
96      kSCATTER = 37,
97      kEINSUM = 38,
98      kASSERTION = 39,
99      kONE_HOT = 40,
100      kNON_ZERO = 41,
101      kGRID_SAMPLE = 42,
102      kNMS = 43,
103      kREVERSE_SEQUENCE = 44,
104      kNORMALIZATION = 45,
105      kPLUGIN_V3 = 46,
106      kSQUEEZE = 47,
107      kUNSQUEEZE = 48,
108      kCUMULATIVE = 49,
109      kDYNAMIC_QUANTIZE = 50,
110      kATTENTION_INPUT = 51,
111      kATTENTION_OUTPUT = 52,
112      kROTARY_EMBEDDING = 53,
113      kKVCACHE_UPDATE = 54,
114      kMOE = 55,
115      kDIST_COLLECTIVE = 56,
116  };
117
123  template <>
124  struct impl::EnumMaxImpl<LayerType>
125  {
126      static constexpr int32_t kVALUE = 57;
127  };
128
135  using TensorFormats = uint32_t;
136
142  enum class ActivationType :  int32_t
143  {
144      kRELU = 0,
145      kSIGMOID = 1,
146      kTANH = 2,
147      kLEAKY_RELU = 3,
148      kELU = 4,
149      kSELU = 5,
150      kSOFTSIGN = 6,
151      kSOFTPLUS = 7,
152      kCLIP = 8,
153      kHARD_SIGMOID = 9,
154      kSCALED_TANH = 10,
155      kTHRESHOLDED_RELU = 11,
156      kGELU_ERF = 12,
157      kGELU_TANH = 13
158  };
159
165  template <>
166  struct impl::EnumMaxImpl<ActivationType>
167  {
168      static constexpr int32_t kVALUE = 14;
169  };
170
185  class ITensor :  public INoCopy
186  {
187  public:
```

```
203     void setName(char const* name) noexcept
204     {
205         mImpl->setName(name);
206     }
207
215     char const* getName() const noexcept
216     {
217         return mImpl->getName();
218     }
219
234     void setDimensions(Dims const& dimensions) noexcept
235     {
236         mImpl->setDimensions(dimensions);
237     }
238
248     Dims getDimensions() const noexcept
249     {
250         return mImpl->getDimensions();
251     }
252
253
263     DataType getType() const noexcept
264     {
265         return mImpl->getType();
266     }
267
271     bool isNetworkInput() const noexcept
272     {
273         return mImpl->isNetworkInput();
274     }
275
279     bool isNetworkOutput() const noexcept
280     {
281         return mImpl->isNetworkOutput();
282     }
283
296     TRT_DEPRECATED void setBroadcastAcrossBatch(bool broadcastAcrossBatch) noexcept
297     {
298         mImpl->setBroadcastAcrossBatch(broadcastAcrossBatch);
299     }
300
310     TRT_DEPRECATED bool getBroadcastAcrossBatch() const noexcept
311     {
312         return mImpl->getBroadcastAcrossBatch();
313     }
314
322     TensorLocation getLocation() const noexcept
323     {
324         return mImpl->getLocation();
325     }
326
341     TRT_DEPRECATED void setLocation(TensorLocation location) noexcept
342     {
343         mImpl->setLocation(location);
344     }
345
346
364     void setAllowedFormats(TensorFormats formats) noexcept
365     {
366         mImpl->setAllowedFormats(formats);
367     }
368
377     TensorFormats getAllowedFormats() const noexcept
378     {
379         return mImpl->getAllowedFormats();
380     }
381
408     bool isShapeTensor() const noexcept
409     {
410         return mImpl->isShapeTensor();
411     }
412
429     bool isExecutionTensor() const noexcept
430     {
431         return mImpl->isExecutionTensor();
432     }
433
455     void setDimensionName(int32_t index, char const* name) noexcept
456     {
457         mImpl->setDimensionName(index, name);
458     }
```

```
459
470     char const* getDimensionName(int32_t index) const noexcept
471     {
472         return mImpl->getDimensionName(index);
473     }
474
475 protected:
476     apiv::VTensor* mImpl;
477     virtual ~ITensor() noexcept = 0;
478 };
479
480 inline ITensor::~ITensor() noexcept = default;
481
489 class ILayer :  public INoCopy
490 {
491 public:
497     LayerType getType() const noexcept
498     {
499         return mLayer->getType();
500     }
501
511     void setName(char const* name) noexcept
512     {
513         mLayer->setName(name);
514     }
515
521     char const* getName() const noexcept
522     {
523         return mLayer->getName();
524     }
525
529     int32_t getNbInputs() const noexcept
530     {
531         return mLayer->getNbInputs();
532     }
533
542     ITensor* getInput(int32_t index) const noexcept
543     {
544         return mLayer->getInput(index);
545     }
546
550     int32_t getNbOutputs() const noexcept
551     {
552         return mLayer->getNbOutputs();
553     }
554
560     ITensor* getOutput(int32_t index) const noexcept
561     {
562         return mLayer->getOutput(index);
563     }
564
577     void setInput(int32_t index, ITensor& tensor) noexcept
578     {
579         return mLayer->setInput(index, tensor);
580     }
581
582
592     DataType getOutputType(int32_t index) const noexcept
593     {
594         return mLayer->getOutputType(index);
595     }
596
597
611     void setMetadata(char const* metadata) noexcept
612     {
613         mLayer->setMetadata(metadata);
614     }
615
624     char const* getMetadata() const noexcept
625     {
626         return mLayer->getMetadata();
627     }
628
645     bool setNbRanks(int32_t nbRanks) noexcept
646     {
647         return mLayer->setNbRanks(nbRanks);
648     }
649
657     int32_t getNbRanks() const noexcept
658     {
659         return mLayer->getNbRanks();
```

```
660     }
661
662 protected:
663     virtual ~ILayer() noexcept = 0;
664     apiv::VLayer* mLayer;
665 };
666
667 inline ILayer::~ILayer() noexcept = default;
668
825 enum class PaddingMode :  int32_t
826 {
827     kEXPLICIT_ROUND_DOWN = 0,
828     kEXPLICIT_ROUND_UP = 1,
829     kSAME_UPPER = 2,
830     kSAME_LOWER = 3,
831 };
832
838 template <>
839 struct impl::EnumMaxImpl<PaddingMode>
840 {
841     static constexpr int32_t kVALUE = 4;
842 };
843
856 class IConvolutionLayer :  public ILayer
857 {
858 public:
866     void setNbOutputMaps(int64_t nbOutputMaps) noexcept
867     {
868         mImpl->setNbOutputMaps(nbOutputMaps);
869     }
870
876     int64_t getNbOutputMaps() const noexcept
877     {
878         return mImpl->getNbOutputMaps();
879     }
880
896     void setNbGroups(int64_t nbGroups) noexcept
897     {
898         mImpl->setNbGroups(nbGroups);
899     }
900
906     int64_t getNbGroups() const noexcept
907     {
908         return mImpl->getNbGroups();
909     }
910
920     void setKernelWeights(Weights weights) noexcept
921     {
922         mImpl->setKernelWeights(weights);
923     }
924
930     Weights getKernelWeights() const noexcept
931     {
932         return mImpl->getKernelWeights();
933     }
934
945     void setBiasWeights(Weights weights) noexcept
946     {
947         mImpl->setBiasWeights(weights);
948     }
949
955     Weights getBiasWeights() const noexcept
956     {
957         return mImpl->getBiasWeights();
958     }
959
972     void setPrePadding(Dims const& padding) noexcept
973     {
974         mImpl->setPrePadding(padding);
975     }
976
982     Dims getPrePadding() const noexcept
983     {
984         return mImpl->getPrePadding();
985     }
986
999     void setPostPadding(Dims const& padding) noexcept
1000     {
1001         mImpl->setPostPadding(padding);
1002     }
1003
```

```
1009     Dims getPostPadding() const noexcept
1010     {
1011         return mImpl->getPostPadding();
1012     }
1013
1023     void setPaddingMode(PaddingMode paddingMode) noexcept
1024     {
1025         mImpl->setPaddingMode(paddingMode);
1026     }
1027
1035     PaddingMode getPaddingMode() const noexcept
1036     {
1037         return mImpl->getPaddingMode();
1038     }
1039
1048     void setKernelSizeNd(Dims const& kernelSize) noexcept
1049     {
1050         mImpl->setKernelSizeNd(kernelSize);
1051     }
1052
1058     Dims getKernelSizeNd() const noexcept
1059     {
1060         return mImpl->getKernelSizeNd();
1061     }
1062
1073     void setStrideNd(Dims const& stride) noexcept
1074     {
1075         mImpl->setStrideNd(stride);
1076     }
1077
1083     Dims getStrideNd() const noexcept
1084     {
1085         return mImpl->getStrideNd();
1086     }
1087
1101     void setPaddingNd(Dims const& padding) noexcept
1102     {
1103         mImpl->setPaddingNd(padding);
1104     }
1105
1113     Dims getPaddingNd() const noexcept
1114     {
1115         return mImpl->getPaddingNd();
1116     }
1117
1127     void setDilationNd(Dims const& dilation) noexcept
1128     {
1129         mImpl->setDilationNd(dilation);
1130     }
1131
1137     Dims getDilationNd() const noexcept
1138     {
1139         return mImpl->getDilationNd();
1140     }
1141
1156     using ILayer::setInput;
1157
1158 protected:
1159     virtual ~IConvolutionLayer() noexcept = 0;
1160     apiv::VConvolutionLayer* mImpl;
1161 };
1162
1163 inline IConvolutionLayer::~IConvolutionLayer() noexcept = default;
1164
1178 class IActivationLayer :  public ILayer
1179 {
1180 public:
1188     void setActivationType(ActivationType type) noexcept
1189     {
1190         mImpl->setActivationType(type);
1191     }
1192
1198     ActivationType getActivationType() const noexcept
1199     {
1200         return mImpl->getActivationType();
1201     }
1202
1213     void setAlpha(float alpha) noexcept
1214     {
1215         mImpl->setAlpha(alpha);
1216     }
```

```
1217
1227     void setBeta(float beta) noexcept
1228     {
1229         mImpl->setBeta(beta);
1230     }
1231
1236     float getAlpha() const noexcept
1237     {
1238         return mImpl->getAlpha();
1239     }
1240
1245     float getBeta() const noexcept
1246     {
1247         return mImpl->getBeta();
1248     }
1249
1250 protected:
1251     virtual ~IActivationLayer() noexcept = 0;
1252     apiv::VActivationLayer* mImpl;
1253 };
1254
1255 inline IActivationLayer::~IActivationLayer() noexcept = default;
1256
1262 enum class PoolingType :  int32_t
1263 {
1264     kMAX = 0,
1265     kAVERAGE = 1,
1266     kMAX_AVERAGE_BLEND = 2
1267 };
1268
1274 template <>
1275 struct impl::EnumMaxImpl<PoolingType>
1276 {
1277     static constexpr int32_t kVALUE = 3;
1278 };
1279
1291 class IPoolingLayer :  public ILayer
1292 {
1293 public:
1301     void setPoolingType(PoolingType type) noexcept
1302     {
1303         mImpl->setPoolingType(type);
1304     }
1305
1311     PoolingType getPoolingType() const noexcept
1312     {
1313         return mImpl->getPoolingType();
1314     }
1315
1326     void setBlendFactor(float blendFactor) noexcept
1327     {
1328         mImpl->setBlendFactor(blendFactor);
1329     }
1330
1339     float getBlendFactor() const noexcept
1340     {
1341         return mImpl->getBlendFactor();
1342     }
1343
1353     void setAverageCountExcludesPadding(bool exclusive) noexcept
1354     {
1355         mImpl->setAverageCountExcludesPadding(exclusive);
1356     }
1357
1364     bool getAverageCountExcludesPadding() const noexcept
1365     {
1366         return mImpl->getAverageCountExcludesPadding();
1367     }
1368
1382     void setPrePadding(Dims const& padding) noexcept
1383     {
1384         mImpl->setPrePadding(padding);
1385     }
1386
1392     Dims getPrePadding() const noexcept
1393     {
1394         return mImpl->getPrePadding();
1395     }
1396
1410     void setPostPadding(Dims const& padding) noexcept
1411     {
```

```
1412         mImpl->setPostPadding(padding);
1413     }
1414
1420     Dims getPostPadding() const noexcept
1421     {
1422         return mImpl->getPostPadding();
1423     }
1424
1433     void setPaddingMode(PaddingMode paddingMode) noexcept
1434     {
1435         mImpl->setPaddingMode(paddingMode);
1436     }
1437
1444     PaddingMode getPaddingMode() const noexcept
1445     {
1446         return mImpl->getPaddingMode();
1447     }
1448
1457     void setWindowSizeNd(Dims const& windowSize) noexcept
1458     {
1459         mImpl->setWindowSizeNd(windowSize);
1460     }
1461
1467     Dims getWindowSizeNd() const noexcept
1468     {
1469         return mImpl->getWindowSizeNd();
1470     }
1471
1482     void setStrideNd(Dims const& stride) noexcept
1483     {
1484         mImpl->setStrideNd(stride);
1485     }
1486
1492     Dims getStrideNd() const noexcept
1493     {
1494         return mImpl->getStrideNd();
1495     }
1496
1511     void setPaddingNd(Dims const& padding) noexcept
1512     {
1513         mImpl->setPaddingNd(padding);
1514     }
1515
1523     Dims getPaddingNd() const noexcept
1524     {
1525         return mImpl->getPaddingNd();
1526     }
1527
1528 protected:
1529     virtual ~IPoolingLayer() noexcept = 0;
1530     apiv::VPoolingLayer* mImpl;
1531 };
1532
1533 inline IPoolingLayer::~IPoolingLayer() noexcept = default;
1534
1544 class ILRNLayer :  public ILayer
1545 {
1546 public:
1556     void setWindowSize(int64_t windowSize) noexcept
1557     {
1558         mImpl->setWindowSize(windowSize);
1559     }
1560
1566     int64_t getWindowSize() const noexcept
1567     {
1568         return mImpl->getWindowSize();
1569     }
1570
1578     void setAlpha(float alpha) noexcept
1579     {
1580         mImpl->setAlpha(alpha);
1581     }
1582
1588     float getAlpha() const noexcept
1589     {
1590         return mImpl->getAlpha();
1591     }
1592
1600     void setBeta(float beta) noexcept
1601     {
1602         mImpl->setBeta(beta);
```

```
1603     }
1604
1610     float getBeta() const noexcept
1611     {
1612         return mImpl->getBeta();
1613     }
1614
1622     void setK(float k) noexcept
1623     {
1624         mImpl->setK(k);
1625     }
1626
1632     float getK() const noexcept
1633     {
1634         return mImpl->getK();
1635     }
1636
1637 protected:
1638     virtual ~ILRNLayer() noexcept = 0;
1639     apiv::VLRNLayer* mImpl;
1640 };
1641
1642 inline ILRNLayer::~ILRNLayer() noexcept = default;
1643
1649 enum class ScaleMode :  int32_t
1650 {
1651     kUNIFORM = 0,
1652     kCHANNEL = 1,
1653     kELEMENTWISE = 2
1654 };
1655
1661 template <>
1662 struct impl::EnumMaxImpl<ScaleMode>
1663 {
1664     static constexpr int32_t kVALUE = 3;
1665 };
1666
1692 class IScaleLayer :  public ILayer
1693 {
1694 public:
1700     void setMode(ScaleMode mode) noexcept
1701     {
1702         mImpl->setMode(mode);
1703     }
1704
1710     ScaleMode getMode() const noexcept
1711     {
1712         return mImpl->getMode();
1713     }
1714
1720     void setShift(Weights shift) noexcept
1721     {
1722         mImpl->setShift(shift);
1723     }
1724
1730     Weights getShift() const noexcept
1731     {
1732         return mImpl->getShift();
1733     }
1734
1740     void setScale(Weights scale) noexcept
1741     {
1742         mImpl->setScale(scale);
1743     }
1744
1750     Weights getScale() const noexcept
1751     {
1752         return mImpl->getScale();
1753     }
1754
1760     void setPower(Weights power) noexcept
1761     {
1762         mImpl->setPower(power);
1763     }
1764
1770     Weights getPower() const noexcept
1771     {
1772         return mImpl->getPower();
1773     }
1774
1785     int32_t getChannelAxis() const noexcept
```

```
1786     {
1787         return mImpl->getChannelAxis();
1788     }
1789
1806     void setChannelAxis(int32_t channelAxis) noexcept
1807     {
1808         mImpl->setChannelAxis(channelAxis);
1809     }
1810
1811 protected:
1812     virtual ~IScaleLayer() noexcept = 0;
1813     apiv::VScaleLayer* mImpl;
1814 };
1815
1816 inline IScaleLayer::~IScaleLayer() noexcept = default;
1817
1838 class ISoftMaxLayer :  public ILayer
1839 {
1840 public:
1861     void setAxes(uint32_t axes) noexcept
1862     {
1863         mImpl->setAxes(axes);
1864     }
1865
1871     uint32_t getAxes() const noexcept
1872     {
1873         return mImpl->getAxes();
1874     }
1875
1876 protected:
1877     virtual ~ISoftMaxLayer() noexcept = 0;
1878     apiv::VSoftMaxLayer* mImpl;
1879 };
1880
1881 inline ISoftMaxLayer::~ISoftMaxLayer() noexcept = default;
1882
1895 class IConcatenationLayer :  public ILayer
1896 {
1897 public:
1909     void setAxis(int32_t axis) noexcept
1910     {
1911         mImpl->setAxis(axis);
1912     }
1913
1919     int32_t getAxis() const noexcept
1920     {
1921         return mImpl->getAxis();
1922     }
1923
1924 protected:
1925     virtual ~IConcatenationLayer() noexcept = 0;
1926     apiv::VConcatenationLayer* mImpl;
1927 };
1928
1929 inline IConcatenationLayer::~IConcatenationLayer() noexcept = default;
1930
1938 class IDeconvolutionLayer :  public ILayer
1939 {
1940 public:
1948     void setNbOutputMaps(int64_t nbOutputMaps) noexcept
1949     {
1950         mImpl->setNbOutputMaps(nbOutputMaps);
1951     }
1952
1958     int64_t getNbOutputMaps() const noexcept
1959     {
1960         return mImpl->getNbOutputMaps();
1961     }
1962
1978     void setNbGroups(int64_t nbGroups) noexcept
1979     {
1980         mImpl->setNbGroups(nbGroups);
1981     }
1982
1988     int64_t getNbGroups() const noexcept
1989     {
1990         return mImpl->getNbGroups();
1991     }
1992
2002     void setKernelWeights(Weights weights) noexcept
2003     {
```

```
2004         mImpl->setKernelWeights(weights);
2005     }
2006
2012     Weights getKernelWeights() const noexcept
2013     {
2014         return mImpl->getKernelWeights();
2015     }
2016
2027     void setBiasWeights(Weights weights) noexcept
2028     {
2029         mImpl->setBiasWeights(weights);
2030     }
2031
2037     Weights getBiasWeights() const noexcept
2038     {
2039         return mImpl->getBiasWeights();
2040     }
2041
2054     void setPrePadding(Dims const& padding) noexcept
2055     {
2056         mImpl->setPrePadding(padding);
2057     }
2058
2064     Dims getPrePadding() const noexcept
2065     {
2066         return mImpl->getPrePadding();
2067     }
2068
2081     void setPostPadding(Dims const& padding) noexcept
2082     {
2083         mImpl->setPostPadding(padding);
2084     }
2085
2091     Dims getPostPadding() const noexcept
2092     {
2093         return mImpl->getPostPadding();
2094     }
2095
2105     void setPaddingMode(PaddingMode paddingMode) noexcept
2106     {
2107         mImpl->setPaddingMode(paddingMode);
2108     }
2109
2117     PaddingMode getPaddingMode() const noexcept
2118     {
2119         return mImpl->getPaddingMode();
2120     }
2121
2132     void setKernelSizeNd(Dims const& kernelSize) noexcept
2133     {
2134         mImpl->setKernelSizeNd(kernelSize);
2135     }
2136
2142     Dims getKernelSizeNd() const noexcept
2143     {
2144         return mImpl->getKernelSizeNd();
2145     }
2146
2159     void setStrideNd(Dims const& stride) noexcept
2160     {
2161         mImpl->setStrideNd(stride);
2162     }
2163
2169     Dims getStrideNd() const noexcept
2170     {
2171         return mImpl->getStrideNd();
2172     }
2173
2187     void setPaddingNd(Dims const& padding) noexcept
2188     {
2189         mImpl->setPaddingNd(padding);
2190     }
2191
2199     Dims getPaddingNd() const noexcept
2200     {
2201         return mImpl->getPaddingNd();
2202     }
2203
2216     using ILayer::setInput;
2217
2225     void setDilationNd(Dims const& dilation) noexcept
```

```
2226     {
2227         mImpl->setDilationNd(dilation);
2228     }
2229
2235     Dims getDilationNd() const noexcept
2236     {
2237         return mImpl->getDilationNd();
2238     }
2239
2240 protected:
2241     virtual ~IDeconvolutionLayer() noexcept = 0;
2242     apiv::VDeconvolutionLayer* mImpl;
2243 };
2244
2245 inline IDeconvolutionLayer::~IDeconvolutionLayer() noexcept = default;
2246
2259 enum class ElementWiseOperation :  int32_t
2260 {
2261     kSUM = 0,
2262     kPROD = 1,
2263     kMAX = 2,
2264     kMIN = 3,
2265     kSUB = 4,
2266     kDIV = 5,
2267     kPOW = 6,
2268     kFLOOR_DIV = 7,
2269     kAND = 8,
2270     kOR = 9,
2271     kXOR = 10,
2272     kEQUAL = 11,
2273     kGREATER = 12,
2274     kLESS = 13
2275 };
2276
2282 template <>
2283 struct impl::EnumMaxImpl<ElementWiseOperation>
2284 {
2285     static constexpr int32_t kVALUE = 14;
2286 };
2287
2307 class IElementWiseLayer :  public ILayer
2308 {
2309 public:
2319     void setOperation(ElementWiseOperation op) noexcept
2320     {
2321         return mImpl->setOperation(op);
2322     }
2323
2331     ElementWiseOperation getOperation() const noexcept
2332     {
2333         return mImpl->getOperation();
2334     }
2335
2336 protected:
2337     apiv::VElementWiseLayer* mImpl;
2338     virtual ~IElementWiseLayer() noexcept = 0;
2339 };
2340
2341 inline IElementWiseLayer::~IElementWiseLayer() noexcept = default;
2342
2348 enum class GatherMode :  int32_t
2349 {
2350     kDEFAULT = 0,
2351     kELEMENT = 1,
2352     kND = 2
2353 };
2354
2360 template <>
2361 struct impl::EnumMaxImpl<GatherMode>
2362 {
2363     static constexpr int32_t kVALUE = 3;
2364 };
2365
2442 class IGatherLayer :  public ILayer
2443 {
2444 public:
2454     void setGatherAxis(int32_t axis) noexcept
2455     {
2456         mImpl->setGatherAxis(axis);
2457     }
2458
```

```
2466     int32_t getGatherAxis() const noexcept
2467     {
2468         return mImpl->getGatherAxis();
2469     }
2470 
2489     void setNbElementWiseDims(int32_t elementWiseDims) noexcept
2490     {
2491         mImpl->setNbElementWiseDims(elementWiseDims);
2492     }
2493 
2499     int32_t getNbElementWiseDims() const noexcept
2500     {
2501         return mImpl->getNbElementWiseDims();
2502     }
2503 
2509     void setMode(GatherMode mode) noexcept
2510     {
2511         mImpl->setMode(mode);
2512     }
2513 
2519     GatherMode getMode() const noexcept
2520     {
2521         return mImpl->getMode();
2522     }
2523 
2524 protected:
2525     apiv::VGatherLayer* mImpl;
2526     virtual ~IGatherLayer() noexcept = 0;
2527 };
2528 
2529 inline IGatherLayer::~IGatherLayer() noexcept = default;
2530 
2542 class TRT_DEPRECATED IPluginV2Layer :  public ILayer
2543 {
2544 public:
2550     IPluginV2& getPlugin() noexcept
2551     {
2552         return mImpl->getPlugin();
2553     }
2554 
2555 protected:
2556     apiv::VPluginV2Layer* mImpl;
2557     virtual ~IPluginV2Layer() noexcept = 0;
2558 };
2559 
2560 inline IPluginV2Layer::~IPluginV2Layer() noexcept = default;
2561 
2571 class IPluginV3Layer :  public ILayer
2572 {
2573 public:
2579     IPluginV3& getPlugin() noexcept
2580     {
2581         return mImpl->getPlugin();
2582     }
2583 
2584 protected:
2585     apiv::VPluginV3Layer* mImpl;
2586     virtual ~IPluginV3Layer() noexcept = 0;
2587 };
2588 
2589 inline IPluginV3Layer::~IPluginV3Layer() noexcept = default;
2590 
2607 enum class UnaryOperation :  int32_t
2608 {
2609     kEXP = 0,
2610     kLOG = 1,
2611     kSQRT = 2,
2612     kRECIP = 3,
2613     kABS = 4,
2614     kNEG = 5,
2615     kSIN = 6,
2616     kCOS = 7,
2617     kTAN = 8,
2618     kSINH = 9,
2619     kCOSH = 10,
2620     kASIN = 11,
2621     kACOS = 12,
2622     kATAN = 13,
2623     kASINH = 14,
2624     kACOSH = 15,
2625     kATANH = 16,
```

```
2626     kCEIL = 17,
2627     kFLOOR = 18,
2628     kERF = 19,
2629     kNOT = 20,
2630     kSIGN = 21,
2631     kROUND = 22,
2632     kISINF = 23,
2633     kISNAN = 24,
2634 };
2635
2641 template <>
2642 struct impl::EnumMaxImpl<UnaryOperation>
2643 {
2644     static constexpr int32_t kVALUE = 25;
2645 };
2646
2654 class IUnaryLayer :  public ILayer
2655 {
2656 public:
2664     void setOperation(UnaryOperation op) noexcept
2665     {
2666         mImpl->setOperation(op);
2667     }
2668
2674     UnaryOperation getOperation() const noexcept
2675     {
2676         return mImpl->getOperation();
2677     }
2678
2679 protected:
2680     apiv::VUnaryLayer* mImpl;
2681     virtual ~IUnaryLayer() noexcept = 0;
2682 };
2683
2684 inline IUnaryLayer::~IUnaryLayer() noexcept = default;
2685
2709 enum class ReduceOperation :  int32_t
2710 {
2711     kSUM = 0,
2712     kPROD = 1,
2713     kMAX = 2,
2714     kMIN = 3,
2715     kAVG = 4,
2716     kNONE = 5,
2717 };
2718
2724 template <>
2725 struct impl::EnumMaxImpl<ReduceOperation>
2726 {
2727     static constexpr int32_t kVALUE = 6;
2728 };
2729
2737 enum class CollectiveOperation :  int32_t
2738 {
2739     kALL_REDUCE = 0,
2740     kALL_GATHER = 1,
2741     kBROADCAST = 2,
2742     kREDUCE = 3,
2743     kREDUCE_SCATTER = 4,
2744     kALL_TO_ALL = 5,
2745     kGATHER = 6,
2746     kSCATTER = 7,
2747 };
2748
2754 template <>
2755 struct impl::EnumMaxImpl<CollectiveOperation>
2756 {
2757     static constexpr int32_t kVALUE = 8;
2758 };
2759
2767 class IReduceLayer :  public ILayer
2768 {
2769 public:
2775     void setOperation(ReduceOperation op) noexcept
2776     {
2777         mImpl->setOperation(op);
2778     }
2779
2785     ReduceOperation getOperation() const noexcept
2786     {
2787         return mImpl->getOperation();
```

```
2788     }
2789
2795     void setReduceAxes(uint32_t reduceAxes) noexcept
2796     {
2797         mImpl->setReduceAxes(reduceAxes);
2798     }
2799
2805     uint32_t getReduceAxes() const noexcept
2806     {
2807         return mImpl->getReduceAxes();
2808     }
2809
2815     void setKeepDimensions(bool keepDimensions) noexcept
2816     {
2817         mImpl->setKeepDimensions(keepDimensions);
2818     }
2819
2825     bool getKeepDimensions() const noexcept
2826     {
2827         return mImpl->getKeepDimensions();
2828     }
2829
2830 protected:
2831     apiv::VReduceLayer* mImpl;
2832     virtual ~IReduceLayer() noexcept = 0;
2833 };
2834
2835 inline IReduceLayer::~IReduceLayer() noexcept = default;
2836
2849 class IPaddingLayer :  public ILayer
2850 {
2851 public:
2861     void setPrePaddingNd(Dims const& padding) noexcept
2862     {
2863         mImpl->setPrePaddingNd(padding);
2864     }
2865
2873     Dims getPrePaddingNd() const noexcept
2874     {
2875         return mImpl->getPrePaddingNd();
2876     }
2877
2887     void setPostPaddingNd(Dims const& padding) noexcept
2888     {
2889         mImpl->setPostPaddingNd(padding);
2890     }
2891
2899     Dims getPostPaddingNd() const noexcept
2900     {
2901         return mImpl->getPostPaddingNd();
2902     }
2903
2904 protected:
2905     apiv::VPaddingLayer* mImpl;
2906     virtual ~IPaddingLayer() noexcept = 0;
2907 };
2908
2909 inline IPaddingLayer::~IPaddingLayer() noexcept = default;
2910
2916 struct Permutation
2917 {
2924     int32_t order[Dims::MAX_DIMS];
2925 };
2926
2939 class IShuffleLayer :  public ILayer
2940 {
2941 public:
2951     void setFirstTranspose(Permutation permutation) noexcept
2952     {
2953         mImpl->setFirstTranspose(permutation);
2954     }
2955
2963     Permutation getFirstTranspose() const noexcept
2964     {
2965         return mImpl->getFirstTranspose();
2966     }
2967
2991     void setReshapeDimensions(Dims const& dimensions) noexcept
2992     {
2993         mImpl->setReshapeDimensions(dimensions);
2994     }
```

```
2995
3004     Dims getReshapeDimensions() const noexcept
3005     {
3006         return mImpl->getReshapeDimensions();
3007     }
3008
3014     //
3037     using ILayer::setInput;
3038
3051     void setSecondTranspose(Permutation permutation) noexcept
3052     {
3053         mImpl->setSecondTranspose(permutation);
3054     }
3055
3063     Permutation getSecondTranspose() const noexcept
3064     {
3065         return mImpl->getSecondTranspose();
3066     }
3067
3079     void setZeroIsPlaceholder(bool zeroIsPlaceholder) noexcept
3080     {
3081         return mImpl->setZeroIsPlaceholder(zeroIsPlaceholder);
3082     }
3083
3092     bool getZeroIsPlaceholder() const noexcept
3093     {
3094         return mImpl->getZeroIsPlaceholder();
3095     }
3096
3097 protected:
3098     apiv::VShuffleLayer* mImpl;
3099     virtual ~IShuffleLayer() noexcept = 0;
3100 };
3101
3102 inline IShuffleLayer::~IShuffleLayer() noexcept = default;
3103
3109 enum class SampleMode :  int32_t
3110 {
3111     kSTRICT_BOUNDS = 0,
3112     kWRAP = 1,
3113     kCLAMP = 2,
3114     kFILL = 3,
3115     kREFLECT = 4,
3118 };
3119
3125 template <>
3126 struct impl::EnumMaxImpl<SampleMode>
3127 {
3128     static constexpr int32_t kVALUE = 5;
3129 };
3130
3193 class ISliceLayer :  public ILayer
3194 {
3195 public:
3205     void setStart(Dims const& start) noexcept
3206     {
3207         mImpl->setStart(start);
3208     }
3209
3220     Dims getStart() const noexcept
3221     {
3222         return mImpl->getStart();
3223     }
3224
3234     void setSize(Dims const& size) noexcept
3235     {
3236         return mImpl->setSize(size);
3237     }
3238
3249     Dims getSize() const noexcept
3250     {
3251         return mImpl->getSize();
3252     }
3253
3263     void setStride(Dims const& stride) noexcept
3264     {
3265         mImpl->setStride(stride);
3266     }
3267
3278     Dims getStride() const noexcept
3279     {
```

```
3280         return mImpl->getStride();
3281     }
3282
3288     void setMode(SampleMode mode) noexcept
3289     {
3290         mImpl->setMode(mode);
3291     }
3292
3298     SampleMode getMode() const noexcept
3299     {
3300         return mImpl->getMode();
3301     }
3302
3330     using ILayer::setInput;
3331
3341     void setAxes(Dims const& axes) noexcept
3342     {
3343         mImpl->setAxes(axes);
3344     }
3345
3356     Dims getAxes() const noexcept
3357     {
3358         return mImpl->getAxes();
3359     }
3360
3361 protected:
3362     apiv::VSliceLayer* mImpl;
3363     virtual ~ISliceLayer() noexcept = 0;
3364 };
3365
3366 inline ISliceLayer::~ISliceLayer() noexcept = default;
3367
3380 class IShapeLayer :  public ILayer
3381 {
3382 protected:
3383     apiv::VShapeLayer* mImpl;
3384     virtual ~IShapeLayer() noexcept = 0;
3385 };
3386
3387 inline IShapeLayer::~IShapeLayer() noexcept = default;
3388
3394 enum class TopKOperation :  int32_t
3395 {
3396     kMAX = 0,
3397     kMIN = 1,
3398 };
3399
3405 template <>
3406 struct impl::EnumMaxImpl<TopKOperation>
3407 {
3408     static constexpr int32_t kVALUE = 2;
3409 };
3410
3422 class ITopKLayer :  public ILayer
3423 {
3424 public:
3430     void setOperation(TopKOperation op) noexcept
3431     {
3432         mImpl->setOperation(op);
3433     }
3434
3440     TopKOperation getOperation() const noexcept
3441     {
3442         return mImpl->getOperation();
3443     }
3444
3454     void setK(int32_t k) noexcept
3455     {
3456         mImpl->setK(k);
3457     }
3458
3468     int32_t getK() const noexcept
3469     {
3470         return mImpl->getK();
3471     }
3472
3478     void setReduceAxes(uint32_t reduceAxes) noexcept
3479     {
3480         mImpl->setReduceAxes(reduceAxes);
3481     }
3482
```

```
3488     uint32_t getReduceAxes() const noexcept
3489     {
3490         return mImpl->getReduceAxes();
3491     }
3492
3507     using ILayer::setInput;
3508
3519     bool setIndicesType(DataType type) noexcept
3520     {
3521         return mImpl->setIndicesType(type);
3522     }
3523
3531     DataType getIndicesType() const noexcept
3532     {
3533         return mImpl->getIndicesType();
3534     }
3535
3536 protected:
3537     apiv::VTopKLayer* mImpl;
3538     virtual ~ITopKLayer() noexcept = 0;
3539 };
3540
3541 inline ITopKLayer::~ITopKLayer() noexcept = default;
3542
3549 enum class MatrixOperation :  int32_t
3550 {
3554     kNONE = 0,
3555
3557     kTRANSPOSE = 1,
3558
3569     kVECTOR = 2,
3570 };
3571
3577 template <>
3578 struct impl::EnumMaxImpl<MatrixOperation>
3579 {
3580     static constexpr int32_t kVALUE = 3;
3581 };
3582
3608 class IMatrixMultiplyLayer :  public ILayer
3609 {
3610 public:
3619     void setOperation(int32_t index, MatrixOperation op) noexcept
3620     {
3621         mImpl->setOperation(index, op);
3622     }
3623
3631     MatrixOperation getOperation(int32_t index) const noexcept
3632     {
3633         return mImpl->getOperation(index);
3634     }
3635
3636 protected:
3637     apiv::VMatrixMultiplyLayer* mImpl;
3638     virtual ~IMatrixMultiplyLayer() noexcept = 0;
3639 };
3640
3641 inline IMatrixMultiplyLayer::~IMatrixMultiplyLayer() noexcept = default;
3642
3664 class INonZeroLayer :  public ILayer
3665 {
3666 public:
3677     bool setIndicesType(DataType type) noexcept
3678     {
3679         return mImpl->setIndicesType(type);
3680     }
3681
3689     DataType getIndicesType() const noexcept
3690     {
3691         return mImpl->getIndicesType();
3692     }
3693
3694 protected:
3695     virtual ~INonZeroLayer() noexcept = 0;
3696     apiv::VNonZeroLayer* mImpl;
3697 };
3698
3699 inline INonZeroLayer::~INonZeroLayer() noexcept = default;
3700
3715 class IRaggedSoftMaxLayer :  public ILayer
3716 {
```

```
3717 protected:
3718     apiv::VRaggedSoftMaxLayer* mImpl;
3719     virtual ~IRaggedSoftMaxLayer() noexcept = 0;
3720 };
3721
3722 inline IRaggedSoftMaxLayer::~IRaggedSoftMaxLayer() noexcept = default;
3723
3767 class IIdentityLayer :  public ILayer
3768 {
3769 protected:
3770     apiv::VIdentityLayer* mImpl;
3771     virtual ~IIdentityLayer() noexcept = 0;
3772 };
3773
3774 inline IIdentityLayer::~IIdentityLayer() noexcept = default;
3775
3782 class ICastLayer :  public ILayer
3783 {
3784 public:
3792     void setToType(DataType toType) noexcept
3793     {
3794         mImpl->setToType(toType);
3795     }
3796
3803     DataType getToType() const noexcept
3804     {
3805         return mImpl->getToType();
3806     }
3807
3808 protected:
3809     apiv::VCastLayer* mImpl;
3810     virtual ~ICastLayer() noexcept = 0;
3811 };
3812
3813 inline ICastLayer::~ICastLayer() noexcept = default;
3814
3823 class IConstantLayer :  public ILayer
3824 {
3825 public:
3834     void setWeights(Weights weights) noexcept
3835     {
3836         mImpl->setWeights(weights);
3837     }
3838
3844     Weights getWeights() const noexcept
3845     {
3846         return mImpl->getWeights();
3847     }
3848
3856     void setDimensions(Dims const& dimensions) noexcept
3857     {
3858         mImpl->setDimensions(dimensions);
3859     }
3860
3868     Dims getDimensions() const noexcept
3869     {
3870         return mImpl->getDimensions();
3871     }
3872
3873 protected:
3874     apiv::VConstantLayer* mImpl;
3875     virtual ~IConstantLayer() noexcept = 0;
3876 };
3877
3878 inline IConstantLayer::~IConstantLayer() noexcept = default;
3879
3889 class IParametricReLULayer :  public ILayer
3890 {
3891 protected:
3892     apiv::VParametricReLULayer* mImpl;
3893     virtual ~IParametricReLULayer() noexcept = 0;
3894 };
3895
3896 inline IParametricReLULayer::~IParametricReLULayer() noexcept = default;
3897
3903 enum class InterpolationMode :  int32_t
3904 {
3905     kNEAREST = 0,
3906     kLINEAR = 1,
3907     kCUBIC = 2
3908 };
```

```
3909
3915 template <>
3916 struct impl::EnumMaxImpl<InterpolationMode>
3917 {
3918     static constexpr int32_t kVALUE = 3;
3919 };
3920
3928 enum class ResizeCoordinateTransformation :  int32_t
3929 {
3944     kALIGN_CORNERS = 0,
3945
3954     kASYMMETRIC = 1,
3955
3964     kHALF_PIXEL = 2,
3965 };
3966
3972 template <>
3973 struct impl::EnumMaxImpl<ResizeCoordinateTransformation>
3974 {
3975     static constexpr int32_t kVALUE = 3;
3976 };
3977
3985 enum class ResizeSelector :  int32_t
3986 {
3988     kFORMULA = 0,
3989
3991     kUPPER = 1,
3992 };
3993
3999 template <>
4000 struct impl::EnumMaxImpl<ResizeSelector>
4001 {
4002     static constexpr int32_t kVALUE = 2;
4003 };
4004
4012 enum class ResizeRoundMode :  int32_t
4013 {
4015     kHALF_UP = 0,
4016
4018     kHALF_DOWN = 1,
4019
4021     kFLOOR = 2,
4022
4024     kCEIL = 3,
4025 };
4026
4032 template <>
4033 struct impl::EnumMaxImpl<ResizeRoundMode>
4034 {
4035     static constexpr int32_t kVALUE = 4;
4036 };
4037
4074 class IResizeLayer :  public ILayer
4075 {
4076 public:
4095     void setOutputDimensions(Dims const& dimensions) noexcept
4096     {
4097         return mImpl->setOutputDimensions(dimensions);
4098     }
4099
4105     Dims getOutputDimensions() const noexcept
4106     {
4107         return mImpl->getOutputDimensions();
4108     }
4109
4135     void setScales(float const* scales, int32_t nbScales) noexcept
4136     {
4137         mImpl->setScales(scales, nbScales);
4138     }
4139
4154     int32_t getScales(int32_t size, float* scales) const noexcept
4155     {
4156         return mImpl->getScales(size, scales);
4157     }
4158
4166     void setResizeMode(InterpolationMode interpolationMode) noexcept
4167     {
4168         mImpl->setResizeMode(interpolationMode);
4169     }
4170
4176     InterpolationMode getResizeMode() const noexcept
```

```
4177     {
4178         return mImpl->getResizeMode();
4179     }
4180 
4200     using ILayer::setInput;
4201 
4211     void setCoordinateTransformation(ResizeCoordinateTransformation coordTransform) noexcept
4212     {
4213         mImpl->setCoordinateTransformation(coordTransform);
4214     }
4215 
4221     ResizeCoordinateTransformation getCoordinateTransformation() const noexcept
4222     {
4223         return mImpl->getCoordinateTransformation();
4224     }
4225 
4236     void setSelectorForSinglePixel(ResizeSelector selector) noexcept
4237     {
4238         mImpl->setSelectorForSinglePixel(selector);
4239     }
4240 
4246     ResizeSelector getSelectorForSinglePixel() const noexcept
4247     {
4248         return mImpl->getSelectorForSinglePixel();
4249     }
4250 
4260     void setNearestRounding(ResizeRoundMode value) noexcept
4261     {
4262         mImpl->setNearestRounding(value);
4263     }
4264 
4270     ResizeRoundMode getNearestRounding() const noexcept
4271     {
4272         return mImpl->getNearestRounding();
4273     }
4274 
4292     void setCubicCoeff(float A) noexcept
4293     {
4294         mImpl->setCubicCoeff(A);
4295     }
4296 
4302     float getCubicCoeff() const noexcept
4303     {
4304         return mImpl->getCubicCoeff();
4305     }
4306 
4315     void setExcludeOutside(bool excludeFlag) noexcept
4316     {
4317         mImpl->setExcludeOutside(excludeFlag);
4318     }
4319 
4325     bool getExcludeOutside() const noexcept
4326     {
4327         return mImpl->getExcludeOutside();
4328     }
4329 
4330 protected:
4331     virtual ~IResizeLayer() noexcept = 0;
4332     apiv::VResizeLayer* mImpl;
4333 };
4334 
4335 inline IResizeLayer::~IResizeLayer() noexcept = default;
4336 
4342 enum class LoopOutput :  int32_t
4343 {
4345     kLAST_VALUE = 0,
4346 
4348     kCONCATENATE = 1,
4349 
4351     kREVERSE = 2
4352 };
4353 
4359 template <>
4360 struct impl::EnumMaxImpl<LoopOutput>
4361 {
4362     static constexpr int32_t kVALUE = 3;
4363 };
4364 
4370 enum class TripLimit :  int32_t
4371 {
4372 
```

```
4373     kCOUNT = 0,
4374     kWHILE = 1
4375 };
4376
4382 template <>
4383 struct impl::EnumMaxImpl<TripLimit>
4384 {
4385     static constexpr int32_t kVALUE = 2;
4386 };
4387
4388 class ILoop;
4389
4403 class ILoopBoundaryLayer :  public ILayer
4404 {
4405 public:
4409     ILoop* getLoop() const noexcept
4410     {
4411         return mBoundary->getLoop();
4412     }
4413
4414 protected:
4415     virtual ~ILoopBoundaryLayer() noexcept = 0;
4416     apiv::VLoopBoundaryLayer* mBoundary;
4417 };
4418
4419 inline ILoopBoundaryLayer::~ILoopBoundaryLayer() noexcept = default;
4420
4428 class IIfConditionalBoundaryLayer :  public ILayer
4429 {
4430 public:
4434     IIfConditional* getConditional() const noexcept
4435     {
4436         return mBoundary->getConditional();
4437     }
4438
4439 protected:
4440     virtual ~IIfConditionalBoundaryLayer() noexcept = 0;
4441     apiv::VConditionalBoundaryLayer* mBoundary;
4442 };
4443
4444 inline IIfConditionalBoundaryLayer::~IIfConditionalBoundaryLayer() noexcept = default;
4445
4451 class IConditionLayer :  public IIfConditionalBoundaryLayer
4452 {
4453 public:
4454 protected:
4455     virtual ~IConditionLayer() noexcept = 0;
4456     apiv::VConditionLayer* mImpl;
4457 };
4458
4459 inline IConditionLayer::~IConditionLayer() noexcept = default;
4460
4470 class IIfConditionalOutputLayer :  public IIfConditionalBoundaryLayer
4471 {
4472 public:
4473 protected:
4474     virtual ~IIfConditionalOutputLayer() noexcept = 0;
4475     apiv::VConditionalOutputLayer* mImpl;
4476 };
4477
4478 inline IIfConditionalOutputLayer::~IIfConditionalOutputLayer() noexcept = default;
4479
4485 class IIfConditionalInputLayer :  public IIfConditionalBoundaryLayer
4486 {
4487 public:
4488 protected:
4489     virtual ~IIfConditionalInputLayer() noexcept = 0;
4490     apiv::VConditionalInputLayer* mImpl;
4491 };
4492
4493 inline IIfConditionalInputLayer::~IIfConditionalInputLayer() noexcept = default;
4494
4519 class IIfConditional :  public INoCopy
4520 {
4521 public:
4531     IConditionLayer* setCondition(ITensor& condition) noexcept
4532     {
4533         return mImpl->setCondition(condition);
4534     }
4535
4549     IIfConditionalOutputLayer* addOutput(ITensor& trueSubgraphOutput, ITensor& falseSubgraphOutput)
```

```
    noexcept
4550     {
4551         return mImpl->addOutput(trueSubgraphOutput, falseSubgraphOutput);
4552     }
4553
4561     IIfConditionalInputLayer* addInput(ITensor& input) noexcept
4562     {
4563         return mImpl->addInput(input);
4564     }
4565
4576     void setName(char const* name) noexcept
4577     {
4578         mImpl->setName(name);
4579     }
4580
4586     char const* getName() const noexcept
4587     {
4588         return mImpl->getName();
4589     }
4590
4591 protected:
4592     virtual ~IIfConditional() noexcept = 0;
4593     apiv::VIfConditional* mImpl;
4594 };
4595
4596 inline IIfConditional::~IIfConditional() noexcept = default;
4597
4605 class IRecurrenceLayer :  public ILoopBoundaryLayer
4606 {
4607 public:
4613     //
4626     using ILayer::setInput;
4627
4628 protected:
4629     virtual ~IRecurrenceLayer() noexcept = 0;
4630     apiv::VRecurrenceLayer* mImpl;
4631 };
4632
4633 inline IRecurrenceLayer::~IRecurrenceLayer() noexcept = default;
4634
4654 class ILoopOutputLayer :  public ILoopBoundaryLayer
4655 {
4656 public:
4660     LoopOutput getLoopOutput() const noexcept
4661     {
4662         return mImpl->getLoopOutput();
4663     }
4664
4677     void setAxis(int32_t axis) noexcept
4678     {
4679         mImpl->setAxis(axis);
4680     }
4681
4685     int32_t getAxis() const noexcept
4686     {
4687         return mImpl->getAxis();
4688     }
4689
4695     //
4710     using ILayer::setInput;
4711
4712 protected:
4713     virtual ~ILoopOutputLayer() noexcept = 0;
4714     apiv::VLoopOutputLayer* mImpl;
4715 };
4716
4717 inline ILoopOutputLayer::~ILoopOutputLayer() noexcept = default;
4718
4730 class ITripLimitLayer :  public ILoopBoundaryLayer
4731 {
4732 public:
4736     TripLimit getTripLimit() const noexcept
4737     {
4738         return mImpl->getTripLimit();
4739     }
4740
4741 protected:
4742     virtual ~ITripLimitLayer() noexcept = 0;
4743     apiv::VTripLimitLayer* mImpl;
4744 };
4745
```

```
4746 inline ITripLimitLayer::~ITripLimitLayer() noexcept = default;
4747
4758 class IIteratorLayer :  public ILoopBoundaryLayer
4759 {
4760 public:
4764     void setAxis(int32_t axis) noexcept
4765     {
4766         mImpl->setAxis(axis);
4767     }
4768
4772     int32_t getAxis() const noexcept
4773     {
4774         return mImpl->getAxis();
4775     }
4776
4786     void setReverse(bool reverse) noexcept
4787     {
4788         mImpl->setReverse(reverse);
4789     }
4790
4796     bool getReverse() const noexcept
4797     {
4798         return mImpl->getReverse();
4799     }
4800
4801 protected:
4802     virtual ~IIteratorLayer() noexcept = 0;
4803     apiv::VIteratorLayer* mImpl;
4804 };
4805
4806 inline IIteratorLayer::~IIteratorLayer() noexcept = default;
4807
4818 class ILoop :  public INoCopy
4819 {
4820 public:
4827     IRecurrenceLayer* addRecurrence(ITensor& initialValue) noexcept
4828     {
4829         return mImpl->addRecurrence(initialValue);
4830     }
4831
4848     ITripLimitLayer* addTripLimit(ITensor& tensor, TripLimit limit) noexcept
4849     {
4850         return mImpl->addTripLimit(tensor, limit);
4851     }
4852
4861     IIteratorLayer* addIterator(ITensor& tensor, int32_t axis = 0, bool reverse = false) noexcept
4862     {
4863         return mImpl->addIterator(tensor, axis, reverse);
4864     }
4865
4874     ILoopOutputLayer* addLoopOutput(ITensor& tensor, LoopOutput outputKind, int32_t axis = 0) noexcept
4875     {
4876         return mImpl->addLoopOutput(tensor, outputKind, axis);
4877     }
4878
4889     void setName(char const* name) noexcept
4890     {
4891         mImpl->setName(name);
4892     }
4893
4899     char const* getName() const noexcept
4900     {
4901         return mImpl->getName();
4902     }
4903
4904 protected:
4905     virtual ~ILoop() noexcept = 0;
4906     apiv::VLoop* mImpl;
4907 };
4908
4909 inline ILoop::~ILoop() noexcept = default;
4910
4923 class ISelectLayer :  public ILayer
4924 {
4925 protected:
4926     virtual ~ISelectLayer() noexcept = 0;
4927     apiv::VSelectLayer* mImpl;
4928 };
4929
4930 inline ISelectLayer::~ISelectLayer() noexcept = default;
4931
```

```
4947 class IAssertionLayer :  public ILayer
4948 {
4949 public:
4958     void setMessage(char const* message) noexcept
4959     {
4960         mImpl->setMessage(message);
4961     }
4962
4968     char const* getMessage() const noexcept
4969     {
4970         return mImpl->getMessage();
4971     }
4972
4973 protected:
4974     virtual ~IAssertionLayer() noexcept = 0;
4975
4976     apiv::VAssertionLayer* mImpl;
4977 };
4978
4979 inline IAssertionLayer::~IAssertionLayer() noexcept = default;
4980
4988 enum class FillOperation :  int32_t
4989 {
5006     kLINSPACE = 0,
5007
5010     kRANDOM_UNIFORM = 1,
5011
5014     kRANDOM_NORMAL = 2
5015 };
5016
5022 template <>
5023 struct impl::EnumMaxImpl<FillOperation>
5024 {
5025     static constexpr int32_t kVALUE = 3;
5026 };
5027
5063 class IFillLayer :  public ILayer
5064 {
5065 public:
5074     //
5075     void setDimensions(Dims const& dimensions) noexcept
5076     {
5077         mImpl->setDimensions(dimensions);
5078     }
5079
5090     Dims getDimensions() const noexcept
5091     {
5092         return mImpl->getDimensions();
5093     }
5094
5100     void setOperation(FillOperation op) noexcept
5101     {
5102         mImpl->setOperation(op);
5103     }
5104
5110     FillOperation getOperation() const noexcept
5111     {
5112         return mImpl->getOperation();
5113     }
5114
5129     //
5130     void setAlpha(double alpha) noexcept
5131     {
5132         mImpl->setAlpha(alpha);
5133     }
5134
5145     double getAlpha() const noexcept
5146     {
5147         return mImpl->getAlpha();
5148     }
5149
5165     void setBeta(double beta) noexcept
5166     {
5167         mImpl->setBeta(beta);
5168     }
5169
5180     double getBeta() const noexcept
5181     {
5182         return mImpl->getBeta();
5183     }
5184
```

```
5225     using ILayer::setInput;
5226
5241     //
5242     void setAlphaInt64(int64_t alpha) noexcept
5243     {
5244         mImpl->setAlphaInt64(alpha);
5245     }
5246
5257     int64_t getAlphaInt64() const noexcept
5258     {
5259         return mImpl->getAlphaInt64();
5260     }
5261
5277     void setBetaInt64(int64_t beta) noexcept
5278     {
5279         mImpl->setBetaInt64(beta);
5280     }
5281
5292     int64_t getBetaInt64() const noexcept
5293     {
5294         return mImpl->getBetaInt64();
5295     }
5296
5300     bool isAlphaBetaInt64() const noexcept
5301     {
5302         return mImpl->isAlphaBetaInt64();
5303     }
5304
5318     void setToType(DataType toType) noexcept
5319     {
5320         mImpl->setToType(toType);
5321     }
5322
5330     DataType getToType() const noexcept
5331     {
5332         return mImpl->getToType();
5333     }
5334
5335 protected:
5336     virtual ~IFillLayer() noexcept = 0;
5337     apiv::VFillLayer* mImpl;
5338 };
5339
5340 inline IFillLayer::~IFillLayer() noexcept = default;
5341
5419 class IQuantizeLayer :  public ILayer
5420 {
5421 public:
5430     int32_t getAxis() const noexcept
5431     {
5432         return mImpl->getAxis();
5433     }
5441     void setAxis(int32_t axis) noexcept
5442     {
5443         mImpl->setAxis(axis);
5444     }
5445
5454     bool setBlockShape(Dims const& blockShape) noexcept
5455     {
5456         return mImpl->setBlockShape(blockShape);
5457     }
5458
5465     TRT_NODISCARD Dims getBlockShape() const noexcept
5466     {
5467         return mImpl->getBlockShape();
5468     }
5469
5481     void setToType(DataType toType) noexcept
5482     {
5483         mImpl->setToType(toType);
5484     }
5485
5493     DataType getToType() const noexcept
5494     {
5495         return mImpl->getToType();
5496     }
5497
5498 protected:
5499     virtual ~IQuantizeLayer() noexcept = 0;
5500     apiv::VQuantizeLayer* mImpl;
5501 };
```

```
5502
5503 inline IQuantizeLayer::~IQuantizeLayer() noexcept = default;
5504
5576 class IDequantizeLayer :  public ILayer
5577 {
5578 public:
5587     int32_t getAxis() const noexcept
5588     {
5589         return mImpl->getAxis();
5590     }
5598     void setAxis(int32_t axis) noexcept
5599     {
5600         mImpl->setAxis(axis);
5601     }
5602
5615     bool setBlockShape(Dims const& blockShape) noexcept
5616     {
5617         return mImpl->setBlockShape(blockShape);
5618     }
5619
5626     TRT_NODISCARD Dims getBlockShape() const noexcept
5627     {
5628         return mImpl->getBlockShape();
5629     }
5630
5642     void setToType(DataType toType) noexcept
5643     {
5644         mImpl->setToType(toType);
5645     }
5646
5654     DataType getToType() const noexcept
5655     {
5656         return mImpl->getToType();
5657     }
5658
5659 protected:
5660     virtual ~IDequantizeLayer() noexcept = 0;
5661     apiv::VDequantizeLayer* mImpl;
5662 };
5663
5664 inline IDequantizeLayer::~IDequantizeLayer() noexcept = default;
5665
5683 class IDynamicQuantizeLayer :  public ILayer
5684 {
5685 public:
5697     using ILayer::setInput;
5698
5711     void setToType(DataType toType) noexcept
5712     {
5713         mImpl->setToType(toType);
5714     }
5715
5724     DataType getToType() const noexcept
5725     {
5726         return mImpl->getToType();
5727     }
5728
5737     void setScaleType(DataType scaleType) noexcept
5738     {
5739         mImpl->setScaleType(scaleType);
5740     }
5741
5750     DataType getScaleType() const noexcept
5751     {
5752         return mImpl->getScaleType();
5753     }
5754
5763     TRT_DEPRECATED void setAxis(int32_t axis) noexcept
5764     {
5765         mImpl->setAxis(axis);
5766     }
5767
5773     TRT_DEPRECATED int32_t getAxis() const noexcept
5774     {
5775         return mImpl->getAxis();
5776     }
5777
5786     TRT_DEPRECATED void setBlockSize(int32_t size) noexcept
5787     {
5788         mImpl->setBlockSize(size);
5789     }
```

```
5790
5796     TRT_DEPRECATED int32_t getBlockSize() const noexcept
5797     {
5798         return mImpl->getBlockSize();
5799     }
5800
5809     void setBlockShape(Dims const& blockShape) noexcept
5810     {
5811         mImpl->setBlockShape(blockShape);
5812     }
5813
5821     Dims getBlockShape() const noexcept
5822     {
5823         return mImpl->getBlockShape();
5824     }
5825
5826 protected:
5827     virtual ~IDynamicQuantizeLayer() noexcept = 0;
5828     apiv::VDynamicQuantizeLayer* mImpl;
5829 };
5830
5831 inline IDynamicQuantizeLayer::~IDynamicQuantizeLayer() noexcept = default;
5832
5867 class IEinsumLayer :  public ILayer
5868 {
5869 public:
5879     bool setEquation(char const* equation) noexcept
5880     {
5881         return mImpl->setEquation(equation);
5882     }
5883
5889     char const* getEquation() const noexcept
5890     {
5891         return mImpl->getEquation();
5892     }
5893
5894 protected:
5895     virtual ~IEinsumLayer() noexcept = 0;
5896     apiv::VEinsumLayer* mImpl;
5897 };
5898
5899 inline IEinsumLayer::~IEinsumLayer() noexcept = default;
5900
5908 enum class ScatterMode :  int32_t
5909 {
5910     kELEMENT = 0,
5911     kND = 1,
5912 };
5913
5919 template <>
5920 struct impl::EnumMaxImpl<ScatterMode>
5921 {
5922     static constexpr int32_t kVALUE = 2;
5923 };
5924
5982 class IScatterLayer :  public ILayer
5983 {
5984 public:
5990     void setMode(ScatterMode mode) noexcept
5991     {
5992         mImpl->setMode(mode);
5993     }
5994
6000     ScatterMode getMode() const noexcept
6001     {
6002         return mImpl->getMode();
6003     }
6004
6010     void setAxis(int32_t axis) noexcept
6011     {
6012         mImpl->setAxis(axis);
6013     }
6014
6018     int32_t getAxis() const noexcept
6019     {
6020         return mImpl->getAxis();
6021     }
6022
6023 protected:
6024     apiv::VScatterLayer* mImpl;
6025     virtual ~IScatterLayer() noexcept = 0;
```

```
6026 }; // class IScatterLayer
6027
6028 inline IScatterLayer::~IScatterLayer() noexcept = default;
6029
6057 class IOneHotLayer :  public ILayer
6058 {
6059 public:
6065     void setAxis(int32_t axis) noexcept
6066     {
6067         mImpl->setAxis(axis);
6068     }
6069
6073     int32_t getAxis() const noexcept
6074     {
6075         return mImpl->getAxis();
6076     }
6077
6078 protected:
6079     apiv::VOneHotLayer* mImpl;
6080     virtual ~IOneHotLayer() noexcept = 0;
6081 };
6082
6083 inline IOneHotLayer::~IOneHotLayer() noexcept = default;
6084
6096 class IGridSampleLayer :  public ILayer
6097 {
6098 public:
6104     void setInterpolationMode(InterpolationMode mode) noexcept
6105     {
6106         mImpl->setInterpolationMode(mode);
6107     }
6108
6116     InterpolationMode getInterpolationMode() const noexcept
6117     {
6118         return mImpl->getInterpolationMode();
6119     }
6120
6126     void setAlignCorners(bool alignCorners) noexcept
6127     {
6128         mImpl->setAlignCorners(alignCorners);
6129     }
6130
6138     bool getAlignCorners() const noexcept
6139     {
6140         return mImpl->getAlignCorners();
6141     }
6142
6150     bool setSampleMode(SampleMode mode) noexcept
6151     {
6152         return mImpl->setSampleMode(mode);
6153     }
6154
6162     SampleMode getSampleMode() const noexcept
6163     {
6164         return mImpl->getSampleMode();
6165     }
6166
6167 protected:
6168     apiv::VGridSampleLayer* mImpl;
6169     virtual ~IGridSampleLayer() noexcept = 0;
6170 }; // class IGridSampleLayer
6171
6172 inline IGridSampleLayer::~IGridSampleLayer() noexcept = default;
6173
6181 enum class BoundingBoxFormat :  int32_t
6182 {
6184     kCORNER_PAIRS = 0,
6186     kCENTER_SIZES = 1
6187 };
6188
6194 template <>
6195 struct impl::EnumMaxImpl<BoundingBoxFormat>
6196 {
6197     static constexpr int32_t kVALUE = 2;
6198 };
6199
6250 class INMSLayer :  public ILayer
6251 {
6252 public:
6262     void setBoundingBoxFormat(BoundingBoxFormat fmt) noexcept
6263     {
```

```
6264         mImpl->setBoundingBoxFormat(fmt);
6265     }
6266
6274     BoundingBoxFormat getBoundingBoxFormat() const noexcept
6275     {
6276         return mImpl->getBoundingBoxFormat();
6277     }
6278
6288     void setTopKBoxLimit(int32_t limit) noexcept
6289     {
6290         mImpl->setTopKBoxLimit(limit);
6291     }
6292
6298     int32_t getTopKBoxLimit() const noexcept
6299     {
6300         return mImpl->getTopKBoxLimit();
6301     }
6302
6321     using ILayer::setInput;
6322
6333     bool setIndicesType(DataType type) noexcept
6334     {
6335         return mImpl->setIndicesType(type);
6336     }
6337
6345     DataType getIndicesType() const noexcept
6346     {
6347         return mImpl->getIndicesType();
6348     }
6349
6350 protected:
6351     apiv::VNMSLayer* mImpl;
6352     virtual ~INMSLayer() noexcept = 0;
6353 }; // class INMSLayer
6354
6355 inline INMSLayer::~INMSLayer() noexcept = default;
6356
6369 class IReverseSequenceLayer :  public ILayer
6370 {
6371 public:
6380     void setBatchAxis(int32_t batchAxis) noexcept
6381     {
6382         mImpl->setBatchAxis(batchAxis);
6383     }
6384
6390     int32_t getBatchAxis() const noexcept
6391     {
6392         return mImpl->getBatchAxis();
6393     }
6394
6403     void setSequenceAxis(int32_t sequenceAxis) noexcept
6404     {
6405         mImpl->setSequenceAxis(sequenceAxis);
6406     }
6407
6413     int32_t getSequenceAxis() const noexcept
6414     {
6415         return mImpl->getSequenceAxis();
6416     }
6417
6418 protected:
6419     apiv::VReverseSequenceLayer* mImpl;
6420     virtual ~IReverseSequenceLayer() noexcept = 0;
6421 }; // class IReverseSequenceLayer
6422
6423 inline IReverseSequenceLayer::~IReverseSequenceLayer() noexcept = default;
6424
6443 class INormalizationLayer :  public ILayer
6444 {
6445 public:
6453     void setEpsilon(float eps) noexcept
6454     {
6455         return mImpl->setEpsilon(eps);
6456     }
6457
6463     float getEpsilon() const noexcept
6464     {
6465         return mImpl->getEpsilon();
6466     }
6467
6473     void setAxes(uint32_t axesMask) noexcept
```

```
6474     {
6475         return mImpl->setAxes(axesMask);
6476     }
6477
6483     uint32_t getAxes() const noexcept
6484     {
6485         return mImpl->getAxes();
6486     }
6487
6504     void setNbGroups(int64_t nbGroups) noexcept
6505     {
6506         return mImpl->setNbGroups(nbGroups);
6507     }
6508
6514     int64_t getNbGroups() const noexcept
6515     {
6516         return mImpl->getNbGroups();
6517     }
6518
6519
6525     TRT_NODISCARD bool isV2() const noexcept
6526     {
6527         return mImpl->isV2();
6528     }
6529
6530 protected:
6531     apiv::VNormalizationLayer* mImpl;
6532     virtual ~INormalizationLayer() noexcept = 0;
6533 };
6534
6535 inline INormalizationLayer::~INormalizationLayer() noexcept = default;
6536
6537
6546 class ISqueezeLayer :  public ILayer
6547 {
6548 public:
6561     using ILayer::setInput;
6562
6563 protected:
6564     apiv::VSqueezeLayer* mImpl;
6565     virtual ~ISqueezeLayer() noexcept = 0;
6566 };
6567
6568 inline ISqueezeLayer::~ISqueezeLayer() noexcept = default;
6569
6578 class IUnsqueezeLayer :  public ILayer
6579 {
6580 public:
6594     using ILayer::setInput;
6595
6596 protected:
6597     apiv::VUnsqueezeLayer* mImpl;
6598     virtual ~IUnsqueezeLayer() noexcept = 0;
6599 };
6600
6601 inline IUnsqueezeLayer::~IUnsqueezeLayer() noexcept = default;
6602
6614 enum class CumulativeOperation :  int32_t
6615 {
6616     kSUM = 0,
6617 };
6618
6624 template <>
6625 struct impl::EnumMaxImpl<CumulativeOperation>
6626 {
6627     static constexpr int32_t kVALUE = 1;
6628 };
6629
6657 class ICumulativeLayer :  public ILayer
6658 {
6659 public:
6669     bool setOperation(CumulativeOperation op) noexcept
6670     {
6671         return mImpl->setOperation(op);
6672     }
6673
6681     CumulativeOperation getOperation() const noexcept
6682     {
6683         return mImpl->getOperation();
6684     }
6685
```

```
6693     void setExclusive(bool exclusive) noexcept
6694     {
6695         mImpl->setExclusive(exclusive);
6696     }
6697
6705     bool getExclusive() const noexcept
6706     {
6707         return mImpl->getExclusive();
6708     }
6709
6717     void setReverse(bool reverse) noexcept
6718     {
6719         mImpl->setReverse(reverse);
6720     }
6721
6729     bool getReverse() const noexcept
6730     {
6731         return mImpl->getReverse();
6732     }
6733
6734 protected:
6735     apiv::VCumulativeLayer* mImpl;
6736     virtual ~ICumulativeLayer() noexcept = 0;
6737 };
6738
6739 inline ICumulativeLayer::~ICumulativeLayer() noexcept = default;
6740
6746 enum class AttentionNormalizationOp :  int32_t
6747 {
6748     kNONE
6749     = 0,
6750     kSOFTMAX = 1,
6751 };
6752
6758 template <>
6759 struct impl::EnumMaxImpl<AttentionNormalizationOp>
6760 {
6761     static constexpr int32_t kVALUE = 2;
6762 };
6763
6778 enum class CausalMaskKind :  int32_t
6779 {
6781     kNONE = 0,
6782
6784     kUPPER_LEFT = 1,
6785
6787     kLOWER_RIGHT = 2,
6788 };
6789
6795 template <>
6796 struct impl::EnumMaxImpl<CausalMaskKind>
6797 {
6798     static constexpr int32_t kVALUE = 3;
6799 };
6800
6806 enum class AttentionIOForm :  int32_t
6807 {
6810     kPADDED_BHND = 0,
6813     kPACKED_NHD = 1,
6814 };
6815
6821 template <>
6822 struct impl::EnumMaxImpl<AttentionIOForm>
6823 {
6824     static constexpr int32_t kVALUE = 2;
6825 };
6826
6836 class IAttentionBoundaryLayer :  public ILayer
6837 {
6838 public:
6842     IAttention* getAttention() const noexcept
6843     {
6844         return mBoundary->getAttention();
6845     }
6846
6847 protected:
6848     virtual ~IAttentionBoundaryLayer() noexcept = 0;
6849     apiv::VAttentionBoundaryLayer* mBoundary;
6850 };
6851
6852 inline IAttentionBoundaryLayer::~IAttentionBoundaryLayer() noexcept = default;
```

```
6853
6864 class IAttentionInputLayer :  public IAttentionBoundaryLayer
6865 {
6866 public:
6884     using ILayer::setInput;
6885
6886 protected:
6887     virtual ~IAttentionInputLayer() noexcept = 0;
6888     apiv::VAttentionInputLayer* mImpl;
6889 };
6890
6891 inline IAttentionInputLayer::~IAttentionInputLayer() noexcept = default;
6892
6903 class IAttentionOutputLayer :  public IAttentionBoundaryLayer
6904 {
6905 public:
6906 protected:
6907     virtual ~IAttentionOutputLayer() noexcept = 0;
6908     apiv::VAttentionOutputLayer* mImpl;
6909 };
6910
6911 inline IAttentionOutputLayer::~IAttentionOutputLayer() noexcept = default;
6912
6973 class IAttention :  public INoCopy
6974 {
6975 public:
6983     bool setNormalizationOperation(AttentionNormalizationOp op) noexcept
6984     {
6985         return mImpl->setNormalizationOperation(op);
6986     }
6987
6995     AttentionNormalizationOp getNormalizationOperation() const noexcept
6996     {
6997         return mImpl->getNormalizationOperation();
6998     }
6999
7012     bool setMask(ITensor& mask) noexcept
7013     {
7014         return mImpl->setMask(mask);
7015     }
7016
7024     ITensor* getMask() noexcept
7025     {
7026         return mImpl->getMask();
7027     }
7028
7042     TRT_DEPRECATED bool setCausal(bool isCausal) noexcept
7043     {
7044         return mImpl->setCausal(isCausal);
7045     }
7046
7056     TRT_DEPRECATED bool getCausal() const noexcept
7057     {
7058         return mImpl->getCausal();
7059     }
7060
7076     bool setCausalKind(CausalMaskKind kind) noexcept
7077     {
7078         return mImpl->setCausalKind(kind);
7079     }
7080
7088     CausalMaskKind getCausalKind() const noexcept
7089     {
7090         return mImpl->getCausalKind();
7091     }
7092
7100     bool setDecomposable(bool decomposable) noexcept
7101     {
7102         return mImpl->setDecomposable(decomposable);
7103     }
7104
7113     bool getDecomposable() const noexcept
7114     {
7115         return mImpl->getDecomposable();
7116     }
7117
7132     bool setInput(int32_t index, ITensor& input) noexcept
7133     {
7134         return mImpl->setInput(index, input);
7135     }
7136
```

```
7141     int32_t getNbInputs() const noexcept
7142     {
7143         return mImpl->getNbInputs();
7144     }
7145
7153     ITensor* getInput(int32_t index) const noexcept
7154     {
7155         return mImpl->getInput(index);
7156     }
7157
7161     int32_t getNbOutputs() const noexcept
7162     {
7163         return mImpl->getNbOutputs();
7164     }
7165
7173     ITensor* getOutput(int32_t index) const noexcept
7174     {
7175         return mImpl->getOutput(index);
7176     }
7177
7190     bool setName(char const* name) noexcept
7191     {
7192         return mImpl->setName(name);
7193     }
7194
7202     char const* getName() const noexcept
7203     {
7204         return mImpl->getName();
7205     }
7206
7218     bool setNormalizationQuantizeScale(ITensor& tensor) noexcept
7219     {
7220         return mImpl->setNormalizationQuantizeScale(tensor);
7221     }
7222
7229     ITensor* getNormalizationQuantizeScale() const noexcept
7230     {
7231         return mImpl->getNormalizationQuantizeScale();
7232     }
7233
7242     bool setNormalizationQuantizeToType(DataType type) noexcept
7243     {
7244         return mImpl->setNormalizationQuantizeToType(type);
7245     }
7246
7254     DataType getNormalizationQuantizeToType() const noexcept
7255     {
7256         return mImpl->getNormalizationQuantizeToType();
7257     }
7258
7274     bool setMetadata(char const* metadata) noexcept
7275     {
7276         return mImpl->setMetadata(metadata);
7277     }
7278
7287     char const* getMetadata() const noexcept
7288     {
7289         return mImpl->getMetadata();
7290     }
7291
7303     bool setNbRanks(int32_t nbRanks) noexcept
7304     {
7305         return mImpl->setNbRanks(nbRanks);
7306     }
7307
7315     int32_t getNbRanks() const noexcept
7316     {
7317         return mImpl->getNbRanks();
7318     }
7319
7332     TRT_NODISCARD bool setQueryForm(AttentionIOForm form) noexcept
7333     {
7334         return mImpl->setQueryForm(form);
7335     }
7336
7345     TRT_NODISCARD AttentionIOForm getQueryForm() const noexcept
7346     {
7347         return mImpl->getQueryForm();
7348     }
7349
7362     TRT_NODISCARD bool setKeyValueForm(AttentionIOForm form) noexcept
```

```
7363     {
7364         return mImpl->setKeyValueForm(form);
7365     }
7366
7375     TRT_NODISCARD AttentionIOForm getKeyValueForm() const noexcept
7376     {
7377         return mImpl->getKeyValueForm();
7378     }
7379
7399     TRT_NODISCARD bool setQueryLengths(ITensor* lengths) noexcept
7400     {
7401         return mImpl->setQueryLengths(lengths);
7402     }
7403
7411     TRT_NODISCARD ITensor* getQueryLengths() const noexcept
7412     {
7413         return mImpl->getQueryLengths();
7414     }
7415
7438     TRT_NODISCARD bool setKeyValueLengths(ITensor* lengths) noexcept
7439     {
7440         return mImpl->setKeyValueLengths(lengths);
7441     }
7442
7450     TRT_NODISCARD ITensor* getKeyValueLengths() const noexcept
7451     {
7452         return mImpl->getKeyValueLengths();
7453     }
7454
7455 protected:
7456     apiv::VAttention* mImpl;
7457     virtual ~IAttention() noexcept = 0;
7458 };
7459
7460 inline IAttention::~IAttention() noexcept = default;
7461
7468 class IRotaryEmbeddingLayer :  public ILayer
7469 {
7470 public:
7476     void setInterleaved(bool interleaved) noexcept
7477     {
7478         mImpl->setInterleaved(interleaved);
7479     }
7480
7481
7487     TRT_NODISCARD bool getInterleaved() const noexcept
7488     {
7489         return mImpl->getInterleaved();
7490     }
7491
7492
7498     TRT_NODISCARD bool setRotaryEmbeddingDim(int32_t rotaryEmbeddingDim) noexcept
7499     {
7500         return mImpl->setRotaryEmbeddingDim(rotaryEmbeddingDim);
7501     }
7502
7503
7509     TRT_NODISCARD int32_t getRotaryEmbeddingDim() const noexcept
7510     {
7511         return mImpl->getRotaryEmbeddingDim();
7512     }
7513
7514
7528     using ILayer::setInput;
7529
7530 protected:
7531     apiv::VRotaryEmbeddingLayer* mImpl;
7532     virtual ~IRotaryEmbeddingLayer() noexcept = 0;
7533 };
7534
7535 inline IRotaryEmbeddingLayer::~IRotaryEmbeddingLayer() noexcept = default;
7536
7542 enum class KVCacheMode :  int32_t
7543 {
7544     kLINEAR = 0,
7545 };
7546
7552 template <>
7553 struct impl::EnumMaxImpl<KVCacheMode>
7554 {
7555     static constexpr int32_t kVALUE = 1;
```

```
7556 };
7557
7577 class IKVCacheUpdateLayer :  public ILayer
7578 {
7579 public:
7594     using ILayer::setInput;
7595
7603     bool setCacheMode(KVCacheMode cacheMode) noexcept
7604     {
7605         return mImpl->setCacheMode(cacheMode);
7606     }
7607
7613     KVCacheMode getCacheMode() const noexcept
7614     {
7615         return mImpl->getCacheMode();
7616     }
7617
7631     TRT_NODISCARD bool setUpdateForm(AttentionIOForm form) noexcept
7632     {
7633         return mImpl->setUpdateForm(form);
7634     }
7635
7644     TRT_NODISCARD AttentionIOForm getUpdateForm() const noexcept
7645     {
7646         return mImpl->getUpdateForm();
7647     }
7648
7666     TRT_NODISCARD bool setUpdateLengths(ITensor* lengths) noexcept
7667     {
7668         return mImpl->setUpdateLengths(lengths);
7669     }
7670
7678     TRT_NODISCARD ITensor* getUpdateLengths() const noexcept
7679     {
7680         return mImpl->getUpdateLengths();
7681     }
7682
7683 protected:
7684     apiv::VKVCacheUpdateLayer* mImpl;
7685     virtual ~IKVCacheUpdateLayer() noexcept = 0;
7686 };
7687
7688 inline IKVCacheUpdateLayer::~IKVCacheUpdateLayer() noexcept = default;
7689
7695 enum class MoEActType :  int32_t
7696 {
7697     kNONE = 0,
7698     kSILU = 1,
7699 };
7700
7706 template <>
7707 struct impl::EnumMaxImpl<MoEActType>
7708 {
7709     static constexpr int32_t kVALUE = 2;
7710 };
7711
7712
7828 class IMoELayer :  public ILayer
7829 {
7830 public:
7842     void setGatedWeights(ITensor& fcGateWeights, ITensor& fcUpWeights, ITensor& fcDownWeights, MoEActType
     activationType) noexcept
7843     {
7844         mImpl->setGatedWeights(fcGateWeights, fcUpWeights, fcDownWeights, activationType);
7845     }
7846
7854     void setGatedBiases(ITensor& fcGateBiases, ITensor& fcUpBiases, ITensor& fcDownBiases) noexcept
7855     {
7856         mImpl->setGatedBiases(fcGateBiases, fcUpBiases, fcDownBiases);
7857     }
7858
7866     void setActivationType(MoEActType activationType) noexcept
7867     {
7868         mImpl->setActivationType(activationType);
7869     }
7870
7878     MoEActType getActivationType() const noexcept
7879     {
7880         return mImpl->getActivationType();
7881     }
7882
```

```
7904     void setQuantizationStatic(ITensor& fcDownActivationScale, DataType dataType) noexcept
7905     {
7906         mImpl->setQuantizationStatic(fcDownActivationScale, dataType);
7907     }
7908
7937     void setQuantizationDynamicDblQ(ITensor& fcDownActivationDblQScale, DataType dataType, Dims const&
     blockShape, DataType dynQOutputScaleType) noexcept
7938     {
7939         mImpl->setQuantizationDynamicDblQ(fcDownActivationDblQScale, dataType, blockShape,
     dynQOutputScaleType);
7940     }
7941
7952     void setQuantizationToType(DataType type) noexcept
7953     {
7954         mImpl->setQuantizationToType(type);
7955     }
7956
7964     DataType getQuantizationToType() const noexcept
7965     {
7966         return mImpl->getQuantizationToType();
7967     }
7968
7980     void setQuantizationBlockShape(Dims const& blockShape) noexcept
7981     {
7982         mImpl->setQuantizationBlockShape(blockShape);
7983     }
7984
7992     Dims getQuantizationBlockShape() const noexcept
7993     {
7994         return mImpl->getQuantizationBlockShape();
7995     }
7996
8004     void setDynQOutputScaleType(DataType type) noexcept
8005     {
8006         mImpl->setDynQOutputScaleType(type);
8007     }
8008
8016     DataType getDynQOutputScaleType() const noexcept
8017     {
8018         return mImpl->getDynQOutputScaleType();
8019     }
8020
8037     void setSwigluParams(float limit, float alpha, float beta) noexcept
8038     {
8039         mImpl->setSwigluParams(limit, alpha, beta);
8040     }
8041
8051     void setSwigluParamLimit(float limit) noexcept
8052     {
8053         mImpl->setSwigluParamLimit(limit);
8054     }
8055
8063     float getSwigluParamLimit() const noexcept
8064     {
8065         return mImpl->getSwigluParamLimit();
8066     }
8067
8077     void setSwigluParamAlpha(float alpha) noexcept
8078     {
8079         mImpl->setSwigluParamAlpha(alpha);
8080     }
8081
8089     float getSwigluParamAlpha() const noexcept
8090     {
8091         return mImpl->getSwigluParamAlpha();
8092     }
8093
8103     void setSwigluParamBeta(float beta) noexcept
8104     {
8105         mImpl->setSwigluParamBeta(beta);
8106     }
8107
8115     float getSwigluParamBeta() const noexcept
8116     {
8117         return mImpl->getSwigluParamBeta();
8118     }
8119
8132     void setInput(int32_t index, ITensor& tensor) noexcept
8133     {
8134         mImpl->setInput(index, tensor);
8135     }
```

```
8136
8137     using ILayer::setInput;
8138
8139 protected:
8140     virtual ~IMoELayer() noexcept = 0;
8141     apiv::VMoELayer* mImpl;
8142 };
8143
8144 inline IMoELayer::~IMoELayer() noexcept = default;
8145
8153 class IDistCollectiveLayer :  public ILayer
8154 {
8155 protected:
8156     virtual ~IDistCollectiveLayer() noexcept = 0;
8157     apiv::VDistCollectiveLayer* mImpl;
8158 }; // class IDistCollectiveLayer
8159
8160 inline IDistCollectiveLayer::~IDistCollectiveLayer() noexcept = default;
8161
8179 class INetworkDefinition :  public INoCopy
8180 {
8181 public:
8182     virtual ~INetworkDefinition() noexcept = 0;
8183
8219     ITensor* addInput(char const* name, DataType type, Dims const& dimensions) noexcept
8220     {
8221         return mImpl->addInput(name, type, dimensions);
8222     }
8223
8233     void markOutput(ITensor& tensor) noexcept
8234     {
8235         mImpl->markOutput(tensor);
8236     }
8237
8251     bool markDebug(ITensor& tensor) noexcept
8252     {
8253         return mImpl->markDebug(tensor);
8254     }
8255
8267     bool unmarkDebug(ITensor& tensor) noexcept
8268     {
8269         return mImpl->unmarkDebug(tensor);
8270     }
8271
8277     bool isDebugTensor(ITensor const& tensor) const noexcept
8278     {
8279         return mImpl->isDebugTensor(tensor);
8280     }
8281
8299     bool markUnfusedTensorsAsDebugTensors() noexcept
8300     {
8301         return mImpl->markUnfusedTensorsAsDebugTensors();
8302     }
8303
8313     bool unmarkUnfusedTensorsAsDebugTensors() noexcept
8314     {
8315         return mImpl->unmarkUnfusedTensorsAsDebugTensors();
8316     }
8317
8333     IActivationLayer* addActivation(ITensor& input, ActivationType type) noexcept
8334     {
8335         return mImpl->addActivation(input, type);
8336     }
8337
8352     ILRNLayer* addLRN(ITensor& input, int64_t window, float alpha, float beta, float k) noexcept
8353     {
8354         return mImpl->addLRN(input, window, alpha, beta, k);
8355     }
8356
8378     IScaleLayer* addScale(ITensor& input, ScaleMode mode, Weights shift, Weights scale, Weights power)
    noexcept
8379     {
8380         return mImpl->addScale(input, mode, shift, scale, power);
8381     }
8382
8391     ISoftMaxLayer* addSoftMax(ITensor& input) noexcept
8392     {
8393         return mImpl->addSoftMax(input);
8394     }
8395
8408     IConcatenationLayer* addConcatenation(ITensor* const* inputs, int32_t nbInputs) noexcept
```

```
8409     {
8410         return mImpl->addConcatenation(inputs, nbInputs);
8411     }
8412
8435     IElementWiseLayer* addElementWise(ITensor& input1, ITensor& input2, ElementWiseOperation op) noexcept
8436     {
8437         return mImpl->addElementWise(input1, input2, op);
8438     }
8439
8457     IUnaryLayer* addUnary(ITensor& input, UnaryOperation operation) noexcept
8458     {
8459         return mImpl->addUnary(input, operation);
8460     }
8461
8471     IShuffleLayer* addShuffle(ITensor& input) noexcept
8472     {
8473         return mImpl->addShuffle(input);
8474     }
8475
8488     IOneHotLayer* addOneHot(ITensor& indices, ITensor& values, ITensor& depth, int32_t axis) noexcept
8489     {
8490         return mImpl->addOneHot(indices, values, depth, axis);
8491     }
8492
8500     int32_t getNbLayers() const noexcept
8501     {
8502         return mImpl->getNbLayers();
8503     }
8504
8514     ILayer* getLayer(int32_t index) const noexcept
8515     {
8516         return mImpl->getLayer(index);
8517     }
8518
8526     int32_t getNbInputs() const noexcept
8527     {
8528         return mImpl->getNbInputs();
8529     }
8530
8542     ITensor* getInput(int32_t index) const noexcept
8543     {
8544         return mImpl->getInput(index);
8545     }
8546
8556     int32_t getNbOutputs() const noexcept
8557     {
8558         return mImpl->getNbOutputs();
8559     }
8560
8572     ITensor* getOutput(int32_t index) const noexcept
8573     {
8574         return mImpl->getOutput(index);
8575     }
8576
8598     IReduceLayer* addReduce(
8599         ITensor& input, ReduceOperation operation, uint32_t reduceAxes, bool keepDimensions) noexcept
8600     {
8601         return mImpl->addReduce(input, operation, reduceAxes, keepDimensions);
8602     }
8603
8634     TRT_DEPRECATED ITopKLayer* addTopK(ITensor& input, TopKOperation op, int32_t k, uint32_t reduceAxes)
    noexcept
8635     {
8636         return mImpl->addTopK(input, op, k, reduceAxes);
8637     }
8638
8667     ITopKLayer* addTopK(ITensor& input, TopKOperation op, int32_t k, uint32_t reduceAxes, DataType
    indicesType) noexcept
8668     {
8669         return mImpl->addTopKV2(input, op, k, reduceAxes, indicesType);
8670     }
8671
8683     IGatherLayer* addGather(ITensor& data, ITensor& indices, int32_t axis) noexcept
8684     {
8685         return mImpl->addGather(data, indices, axis);
8686     }
8687
8699     IGatherLayer* addGatherV2(ITensor& data, ITensor& indices, GatherMode mode) noexcept
8700     {
8701         return mImpl->addGatherV2(data, indices, mode);
8702     }
```

```
8703
8718     IRaggedSoftMaxLayer* addRaggedSoftMax(ITensor& input, ITensor& bounds) noexcept
8719     {
8720         return mImpl->addRaggedSoftMax(input, bounds);
8721     }
8722
8739     IMatrixMultiplyLayer* addMatrixMultiply(
8740         ITensor& input0, MatrixOperation op0, ITensor& input1, MatrixOperation op1) noexcept
8741     {
8742         return mImpl->addMatrixMultiply(input0, op0, input1, op1);
8743     }
8744
8758     TRT_DEPRECATED INonZeroLayer* addNonZero(ITensor& input) noexcept
8759     {
8760         return mImpl->addNonZero(input);
8761     }
8762
8774     INonZeroLayer* addNonZero(ITensor& input, DataType indicesType) noexcept
8775     {
8776         return mImpl->addNonZeroV2(input, indicesType);
8777     }
8778
8798     IConstantLayer* addConstant(Dims const& dimensions, Weights weights) noexcept
8799     {
8800         return mImpl->addConstant(dimensions, weights);
8801     }
8802
8812     IIdentityLayer* addIdentity(ITensor& input) noexcept
8813     {
8814         return mImpl->addIdentity(input);
8815     }
8816
8827     ICastLayer* addCast(ITensor& input, DataType toType) noexcept
8828     {
8829         return mImpl->addCast(input, toType);
8830     }
8831
8842     void removeTensor(ITensor& tensor) noexcept
8843     {
8844         mImpl->removeTensor(tensor);
8845     }
8846
8854     void unmarkOutput(ITensor& tensor) noexcept
8855     {
8856         mImpl->unmarkOutput(tensor);
8857     }
8858
8875     TRT_DEPRECATED IPluginV2Layer* addPluginV2(ITensor* const* inputs, int32_t nbInputs, IPluginV2& plugin)
    noexcept
8876     {
8877         return mImpl->addPluginV2(inputs, nbInputs, plugin);
8878     }
8879
8893     IPluginV3Layer* addPluginV3(ITensor* const* inputs, int32_t nbInputs, ITensor* const* shapeInputs,
8894         int32_t nbShapeInputs, IPluginV3& plugin) noexcept
8895     {
8896         return mImpl->addPluginV3(inputs, nbInputs, shapeInputs, nbShapeInputs, plugin);
8897     }
8898
8913     ISliceLayer* addSlice(ITensor& input, Dims const& start, Dims const& size, Dims const& stride) noexcept
8914     {
8915         return mImpl->addSlice(input, start, size, stride);
8916     }
8917
8937     void setName(char const* name) noexcept
8938     {
8939         mImpl->setName(name);
8940     }
8941
8951     char const* getName() const noexcept
8952     {
8953         return mImpl->getName();
8954     }
8955
8967     IShapeLayer* addShape(ITensor& input) noexcept
8968     {
8969         return mImpl->addShape(input);
8970     }
8971
8981     TRT_DEPRECATED bool hasImplicitBatchDimension() const noexcept
8982     {
```

```
8983         return mImpl->hasImplicitBatchDimension();
8984     }
8985
8991     NetworkDefinitionCreationFlags getFlags() const noexcept
8992     {
8993         return mImpl->getFlags();
8994     }
8995
9003     bool getFlag(NetworkDefinitionCreationFlag networkDefinitionCreationFlag) const noexcept
9004     {
9005         return mImpl->getFlag(networkDefinitionCreationFlag);
9006     }
9007
9020     bool markOutputForShapes(ITensor& tensor) noexcept
9021     {
9022         return mImpl->markOutputForShapes(tensor);
9023     }
9024
9032     bool unmarkOutputForShapes(ITensor& tensor) noexcept
9033     {
9034         return mImpl->unmarkOutputForShapes(tensor);
9035     }
9036
9050     IParametricReLULayer* addParametricReLU(ITensor& input, ITensor& slope) noexcept
9051     {
9052         return mImpl->addParametricReLU(input, slope);
9053     }
9054
9072     IConvolutionLayer* addConvolutionNd(
9073         ITensor& input, int64_t nbOutputMaps, Dims const& kernelSize, Weights kernelWeights, Weights
    biasWeights) noexcept
9074     {
9075         return mImpl->addConvolutionNd(input, nbOutputMaps, kernelSize, kernelWeights, biasWeights);
9076     }
9077
9092     IPoolingLayer* addPoolingNd(ITensor& input, PoolingType type, Dims const& windowSize) noexcept
9093     {
9094         return mImpl->addPoolingNd(input, type, windowSize);
9095     }
9096
9111     //
9114     IDeconvolutionLayer* addDeconvolutionNd(
9115         ITensor& input, int64_t nbOutputMaps, Dims kernelSize, Weights kernelWeights, Weights biasWeights)
    noexcept
9116     {
9117         return mImpl->addDeconvolutionNd(input, nbOutputMaps, kernelSize, kernelWeights, biasWeights);
9118     }
9119
9151     IScaleLayer* addScaleNd(
9152         ITensor& input, ScaleMode mode, Weights shift, Weights scale, Weights power, int32_t channelAxis)
    noexcept
9153     {
9154         return mImpl->addScaleNd(input, mode, shift, scale, power, channelAxis);
9155     }
9156
9168     IResizeLayer* addResize(ITensor& input) noexcept
9169     {
9170         return mImpl->addResize(input);
9171     }
9172
9182     ILoop* addLoop() noexcept
9183     {
9184         return mImpl->addLoop();
9185     }
9186
9197     IIfConditional* addIfConditional() noexcept
9198     {
9199         return mImpl->addIfConditional();
9200     }
9201
9236     ISelectLayer* addSelect(ITensor& condition, ITensor& thenInput, ITensor& elseInput) noexcept
9237     {
9238         return mImpl->addSelect(condition, thenInput, elseInput);
9239     }
9240
9253     IAssertionLayer* addAssertion(ITensor& condition, char const* message) noexcept
9254     {
9255         return mImpl->addAssertion(condition, message);
9256     }
9257
9279     IFillLayer* addFill(Dims const& dimensions, FillOperation op, DataType outputType) noexcept
```

```
9280     {
9281         return mImpl->addFillV2(dimensions, op, outputType);
9282     }
9283
9295     IPaddingLayer* addPaddingNd(ITensor& input, Dims const& prePadding, Dims const& postPadding) noexcept
9296     {
9297         return mImpl->addPaddingNd(input, prePadding, postPadding);
9298     }
9299
9319     bool setWeightsName(Weights weights, char const* name) noexcept
9320     {
9321         return mImpl->setWeightsName(weights, name);
9322     }
9323
9335     //
9338     void setErrorRecorder(IErrorRecorder* recorder) noexcept
9339     {
9340         mImpl->setErrorRecorder(recorder);
9341     }
9342
9353     IErrorRecorder* getErrorRecorder() const noexcept
9354     {
9355         return mImpl->getErrorRecorder();
9356     }
9357
9376     IDequantizeLayer* addDequantize(ITensor& input, ITensor& scale, DataType outputType) noexcept
9377     {
9378         return mImpl->addDequantizeV2(input, scale, outputType);
9379     }
9380
9396     IScatterLayer* addScatter(ITensor& data, ITensor& indices, ITensor& updates, ScatterMode mode) noexcept
9397     {
9398         return mImpl->addScatter(data, indices, updates, mode);
9399     }
9400
9420     IQuantizeLayer* addQuantize(ITensor& input, ITensor& scale, DataType outputType) noexcept
9421     {
9422         return mImpl->addQuantizeV2(input, scale, outputType);
9423     }
9424
9447     TRT_DEPRECATED IDynamicQuantizeLayer* addDynamicQuantize(
9448         ITensor& input, int32_t axis, int32_t blockSize, DataType outputType, DataType scaleType) noexcept
9449     {
9450         return mImpl->addDynamicQuantize(input, axis, blockSize, outputType, scaleType);
9451     }
9452
9471     IDynamicQuantizeLayer* addDynamicQuantizeV2(
9472         ITensor& input, Dims const& blockShape, DataType outputType, DataType scaleType) noexcept
9473     {
9474         return mImpl->addDynamicQuantizeV2(input, blockShape, outputType, scaleType);
9475     }
9476
9487     IEinsumLayer* addEinsum(ITensor* const* inputs, int32_t nbInputs, char const* equation) noexcept
9488     {
9489         return mImpl->addEinsum(inputs, nbInputs, equation);
9490     }
9491
9505     IGridSampleLayer* addGridSample(ITensor& input, ITensor& grid) noexcept
9506     {
9507         return mImpl->addGridSample(input, grid);
9508     }
9509
9527     TRT_DEPRECATED INMSLayer* addNMS(ITensor& boxes, ITensor& scores, ITensor& maxOutputBoxesPerClass)
    noexcept
9528     {
9529         return mImpl->addNMS(boxes, scores, maxOutputBoxesPerClass);
9530     }
9531
9547     INMSLayer* addNMS(ITensor& boxes, ITensor& scores, ITensor& maxOutputBoxesPerClass, DataType
    indicesType) noexcept
9548     {
9549         return mImpl->addNMSV2(boxes, scores, maxOutputBoxesPerClass, indicesType);
9550     }
9551
9564     IReverseSequenceLayer* addReverseSequence(ITensor& input, ITensor& sequenceLens) noexcept
9565     {
9566         return mImpl->addReverseSequence(input, sequenceLens);
9567     }
9568
9596     TRT_DEPRECATED INormalizationLayer* addNormalization(ITensor& input, ITensor& scale, ITensor& bias,
    uint32_t axesMask) noexcept
```

```
9597     {
9598         return mImpl->addNormalization(input, scale, bias, axesMask);
9599     }
9600 
9618     ICumulativeLayer* addCumulative(ITensor& input, ITensor& axis, CumulativeOperation operation, bool
     exclusive, bool reverse) noexcept
9619     {
9620         return mImpl->addCumulative(input, axis, operation, exclusive, reverse);
9621     }
9622 
9648     TRT_DEPRECATED IAttention* addAttention(
9649         ITensor& query, ITensor& key, ITensor& value, AttentionNormalizationOp normOp, bool causal)
     noexcept
9650     {
9651         return mImpl->addAttention(query, key, value, normOp, causal);
9652     }
9653 
9678     IAttention* addAttentionV2(ITensor& query, ITensor& key, ITensor& value, AttentionNormalizationOp
     normOp,
9679         CausalMaskKind causalKind) noexcept
9680     {
9681         return mImpl->addAttentionV2(query, key, value, normOp, causalKind);
9682     }
9683 
9703     IRotaryEmbeddingLayer* addRotaryEmbedding(ITensor& input, ITensor& cosCache, ITensor& sinCache, bool
     interleaved, int32_t rotaryEmbeddingDim) noexcept
9704     {
9705         return mImpl->addRotaryEmbedding(input, cosCache, sinCache, interleaved, rotaryEmbeddingDim);
9706     }
9707 
9737     IKVCacheUpdateLayer* addKVCacheUpdate(
9738         ITensor& cache, ITensor& update, ITensor& writeIndices, KVCacheMode cacheMode) noexcept
9739     {
9740         return mImpl->addKVCacheUpdate(cache, update, writeIndices, cacheMode);
9741     }
9742 
9759     IMoELayer* addMoE(ITensor& hiddenStates, ITensor& selectedExpertsForTokens, ITensor&
     scoresForSelectedExperts) noexcept
9760     {
9761         return mImpl->addMoE(hiddenStates, selectedExpertsForTokens, scoresForSelectedExperts);
9762     }
9763 
9791     TRT_NODISCARD IDistCollectiveLayer* addDistCollective(ITensor& input, CollectiveOperation
     distCollectiveOp,
9792         ReduceOperation reduceOp, int64_t root, int64_t* groups, int64_t groupSize) noexcept
9793     {
9794         return mImpl->addDistCollective(input, distCollectiveOp, reduceOp, root, groups, groupSize);
9795     }
9796 
9803     virtual IBuilder& getBuilder() const noexcept
9804     {
9805         return mImpl->getBuilder();
9806     }
9807 
9816     bool markWeightsRefittable(char const* name) noexcept
9817     {
9818         return mImpl->markWeightsRefittable(name);
9819     }
9820 
9828     bool unmarkWeightsRefittable(char const* name) noexcept
9829     {
9830         return mImpl->unmarkWeightsRefittable(name);
9831     }
9832 
9841     bool areWeightsMarkedRefittable(char const* name) const noexcept
9842     {
9843         return mImpl->areWeightsMarkedRefittable(name);
9844     }
9845 
9860     ISqueezeLayer* addSqueeze(ITensor& input, ITensor& axes) noexcept
9861     {
9862         return mImpl->addSqueeze(input, axes);
9863     }
9864 
9881     IUnsqueezeLayer* addUnsqueeze(ITensor& input, ITensor& axes) noexcept
9882     {
9883         return mImpl->addUnsqueeze(input, axes);
9884     }
9885 
9907     TRT_NODISCARD INormalizationLayer* addNormalizationV2(ITensor& input, ITensor& scale, ITensor& bias,
     uint32_t axesMask) noexcept
```

```
9908     {
9909         return mImpl->addNormalizationV2(input, scale, bias, axesMask);
9910     }
9911
9912 protected:
9913     apiv::VNetworkDefinition* mImpl;
9914 };
9915
9916 inline INetworkDefinition::~INetworkDefinition() noexcept = default;
9917
9935 enum class RuntimePlatform :  int32_t
9936 {
9939     kSAME_AS_BUILD = 0,
9940
9943     kWINDOWS_AMD64 = 1,
9944
9945
9946 };
9947
9953 template <>
9954 struct impl::EnumMaxImpl<RuntimePlatform>
9955 {
9956     static constexpr int32_t kVALUE = 2;
9957 };
9958
9965 using BuilderFlags = uint32_t;
9966
9974 enum class BuilderFlag :  int32_t
9975 {
9976
9978     kDEBUG = 2,
9979
9981     kGPU_FALLBACK = 3,
9982
9984     kREFIT = 4,
9985
9987     kDISABLE_TIMING_CACHE = 5,
9988
9992     kTF32 = 6,
9993
9995     kSPARSE_WEIGHTS = 7,
9996
10008     kSAFETY_SCOPE TRT_DEPRECATED_ENUM = 8,
10009
10010
10015     kDIRECT_IO TRT_DEPRECATED_ENUM = 11,
10016
10022     kVERSION_COMPATIBLE = 12,
10023
10028     kEXCLUDE_LEAN_RUNTIME = 13,
10029
10030
10033     kERROR_ON_TIMING_CACHE_MISS = 15,
10034
10035
10040     kDISABLE_COMPILATION_CACHE = 17,
10041
10043     kSTRIP_PLAN = 18,
10044
10051     kREFIT_IDENTICAL = 19,
10052
10078     kWEIGHT_STREAMING = 20,
10079
10080
10085     kREFIT_INDIVIDUAL = 22,
10086
10095     kSTRICT_NANS = 23,
10096
10098     kMONITOR_MEMORY = 24,
10099
10100
10102     kEDITABLE_TIMING_CACHE = 26,
10103
10114     kDISTRIBUTIVE_INDEPENDENCE = 27,
10115
10116
10117 };
10118
10124 template <>
10125 struct impl::EnumMaxImpl<BuilderFlag>
10126 {
```

```
10127     static constexpr int32_t kVALUE = 28;
10128 };
10129
10130 namespace v_1_0
10131 {
10144 struct TimingCacheKey
10145 {
10146     uint8_t data[16];
10147 };
10148
10156 struct TimingCacheValue
10157 {
10159     uint64_t tacticHash;
10161     float timingMSec;
10163     static constexpr uint64_t kINVALID_TACTIC_HASH = UINT64_MAX;
10164 };
10165 } // namespace v_1_0
10166
10180 class ITimingCache :  public INoCopy
10181 {
10182 public:
10183     virtual ~ITimingCache() noexcept = 0;
10184
10194     nvinfer1::IHostMemory* serialize() const noexcept
10195     {
10196         return mImpl->serialize();
10197     }
10198
10218     bool combine(ITimingCache const& inputCache, bool ignoreMismatch) noexcept
10219     {
10220         return mImpl->combine(inputCache, ignoreMismatch);
10221     }
10222
10228     bool reset() noexcept
10229     {
10230         return mImpl->reset();
10231     }
10232
10247     int64_t queryKeys(TimingCacheKey* keyBuffer, int64_t capacity) const noexcept
10248     {
10249         return mImpl->queryKeys(keyBuffer, capacity);
10250     }
10251
10264     TimingCacheValue query(TimingCacheKey const& key) const noexcept
10265     {
10266         return mImpl->query(key);
10267     }
10268
10286     bool update(TimingCacheKey const& key, TimingCacheValue const& value) noexcept
10287     {
10288         return mImpl->update(key, value);
10289     }
10290
10291 protected:
10292     apiv::VTimingCache* mImpl;
10293 };
10294
10295 inline ITimingCache::~ITimingCache() noexcept = default;
10296
10304 enum class MemoryPoolType :  int32_t
10305 {
10311     kWORKSPACE = 0,
10312
10319     kDLA_MANAGED_SRAM = 1,
10320
10326     kDLA_LOCAL_DRAM = 2,
10327
10333     kDLA_GLOBAL_DRAM = 3,
10334
10342     kTACTIC_DRAM = 4,
10343
10356     kTACTIC_SHARED_MEMORY = 5,
10357 };
10358
10364 template <>
10365 struct impl::EnumMaxImpl<MemoryPoolType>
10366 {
10367     static constexpr int32_t kVALUE = 6;
10368 };
10369
10378 enum class PreviewFeature :  int32_t
```

```
10379 {
10383     kALIASED_PLUGIN_IO_10_03 = 0,
10384
10390     kRUNTIME_ACTIVATION_RESIZE_10_10 = 1,
10391 };
10392
10398 template <>
10399 struct impl::EnumMaxImpl<PreviewFeature>
10400 {
10401     static constexpr int32_t kVALUE = 2;
10402 };
10403
10412 enum class HardwareCompatibilityLevel :  int32_t
10413 {
10416     kNONE = 0,
10417
10429     kAMPERE_PLUS = 1,
10430
10439     kSAME_COMPUTE_CAPABILITY = 2,
10440 };
10441
10447 template <>
10448 struct impl::EnumMaxImpl<HardwareCompatibilityLevel>
10449 {
10450     static constexpr int32_t kVALUE = 3;
10451 };
10452
10453
10462 enum class TilingOptimizationLevel :  int32_t
10463 {
10465     kNONE = 0,
10466
10468     kFAST = 1,
10469
10472     kMODERATE = 2,
10473
10475     kFULL = 3
10476
10477 };
10478
10484 template <>
10485 struct impl::EnumMaxImpl<TilingOptimizationLevel>
10486 {
10487     static constexpr int32_t kVALUE = 4;
10488 };
10489
10490 namespace v_1_0
10491 {
10492 class IProgressMonitor :  public IVersionedInterface
10493 {
10494 public:
10495     IProgressMonitor() = default;
10496     virtual ~IProgressMonitor() noexcept = default;
10497
10501     InterfaceInfo getInterfaceInfo() const noexceptoverride
10502 {
10503         return InterfaceInfo{"IProgressMonitor", 1, 0};
10504     }
10505
10525     virtual void phaseStart(char const* phaseName, char const* parentPhase, int32_t nbSteps) noexcept = 0;
10526
10539     virtual bool stepComplete(char const* phaseName, int32_t step) noexcept = 0;
10540
10552     virtual void phaseFinish(char const* phaseName) noexcept = 0;
10553
10554 }; // class IProgressMonitor
10555 } // namespace v_1_0
10556
10576 using IProgressMonitor = v_1_0::IProgressMonitor;
10577
10585 class IBuilderConfig :  public INoCopy
10586 {
10587 public:
10588     virtual ~IBuilderConfig() noexcept = 0;
10589
10598     virtual void setAvgTimingIterations(int32_t avgTiming) noexcept
10599     {
10600         mImpl->setAvgTimingIterations(avgTiming);
10601     }
10602
10610     int32_t getAvgTimingIterations() const noexcept
```

```
10611     {
10612         return mImpl->getAvgTimingIterations();
10613     }
10614
10623     void setEngineCapability(EngineCapability capability) noexcept
10624     {
10625         mImpl->setEngineCapability(capability);
10626     }
10627
10635     EngineCapability getEngineCapability() const noexcept
10636     {
10637         return mImpl->getEngineCapability();
10638     }
10639
10652     void setFlags(BuilderFlags builderFlags) noexcept
10653     {
10654         mImpl->setFlags(builderFlags);
10655     }
10656
10664     BuilderFlags getFlags() const noexcept
10665     {
10666         return mImpl->getFlags();
10667     }
10668
10676     void clearFlag(BuilderFlag builderFlag) noexcept
10677     {
10678         mImpl->clearFlag(builderFlag);
10679     }
10680
10688     void setFlag(BuilderFlag builderFlag) noexcept
10689     {
10690         mImpl->setFlag(builderFlag);
10691     }
10692
10700     bool getFlag(BuilderFlag builderFlag) const noexcept
10701     {
10702         return mImpl->getFlag(builderFlag);
10703     }
10704
10717     void setDeviceType(ILayer const* layer, DeviceType deviceType) noexcept
10718     {
10719         mImpl->setDeviceType(layer, deviceType);
10720     }
10721
10727     DeviceType getDeviceType(ILayer const* layer) const noexcept
10728     {
10729         return mImpl->getDeviceType(layer);
10730     }
10731
10739     bool isDeviceTypeSet(ILayer const* layer) const noexcept
10740     {
10741         return mImpl->isDeviceTypeSet(layer);
10742     }
10743
10749     void resetDeviceType(ILayer const* layer) noexcept
10750     {
10751         mImpl->resetDeviceType(layer);
10752     }
10753
10759     bool canRunOnDLA(ILayer const* layer) const noexcept
10760     {
10761         return mImpl->canRunOnDLA(layer);
10762     }
10763
10775     void setDLACore(int32_t dlaCore) noexcept
10776     {
10777         mImpl->setDLACore(dlaCore);
10778     }
10779
10785     int32_t getDLACore() const noexcept
10786     {
10787         return mImpl->getDLACore();
10788     }
10789
10796     void setDefaultDeviceType(DeviceType deviceType) noexcept
10797     {
10798         mImpl->setDefaultDeviceType(deviceType);
10799     }
10800
10806     DeviceType getDefaultDeviceType() const noexcept
10807     {
```

```
10808         return mImpl->getDefaultDeviceType();
10809     }
10810
10816     void reset() noexcept
10817     {
10818         mImpl->reset();
10819     }
10820
10828     void setProfileStream(const cudaStream_t stream) noexcept
10829     {
10830         return mImpl->setProfileStream(stream);
10831     }
10832
10840     cudaStream_t getProfileStream() const noexcept
10841     {
10842         return mImpl->getProfileStream();
10843     }
10844
10857     int32_t addOptimizationProfile(IOptimizationProfile const* profile) noexcept
10858     {
10859         return mImpl->addOptimizationProfile(profile);
10860     }
10861
10870     int32_t getNbOptimizationProfiles() const noexcept
10871     {
10872         return mImpl->getNbOptimizationProfiles();
10873     }
10874
10882     void setProfilingVerbosity(ProfilingVerbosity verbosity) noexcept
10883     {
10884         mImpl->setProfilingVerbosity(verbosity);
10885     }
10886
10895     ProfilingVerbosity getProfilingVerbosity() const noexcept
10896     {
10897         return mImpl->getProfilingVerbosity();
10898     }
10899
10917     bool setTacticSources(TacticSources tacticSources) noexcept
10918     {
10919         return mImpl->setTacticSources(tacticSources);
10920     }
10921
10932     TacticSources getTacticSources() const noexcept
10933     {
10934         return mImpl->getTacticSources();
10935     }
10936
10952     nvinfer1::ITimingCache* createTimingCache(void const* blob, std::size_t size) const noexcept
10953     {
10954         return mImpl->createTimingCache(blob, size);
10955     }
10956
10975     bool setTimingCache(ITimingCache const& cache, bool ignoreMismatch) noexcept
10976     {
10977         return mImpl->setTimingCache(cache, ignoreMismatch);
10978     }
10979
10985     nvinfer1::ITimingCache const* getTimingCache() const noexcept
10986     {
10987         return mImpl->getTimingCache();
10988     }
10989
11017     void setMemoryPoolLimit(MemoryPoolType pool, std::size_t poolSize) noexcept
11018     {
11019         mImpl->setMemoryPoolLimit(pool, poolSize);
11020     }
11021
11036     std::size_t getMemoryPoolLimit(MemoryPoolType pool) const noexcept
11037     {
11038         return mImpl->getMemoryPoolLimit(pool);
11039     }
11040
11054     void setPreviewFeature(PreviewFeature feature, bool enable) noexcept
11055     {
11056         mImpl->setPreviewFeature(feature, enable);
11057     }
11058
11068     bool getPreviewFeature(PreviewFeature feature) const noexcept
11069     {
11070         return mImpl->getPreviewFeature(feature);
```

```
11071     }
11072
11101     void setBuilderOptimizationLevel(int32_t level) noexcept
11102     {
11103         mImpl->setBuilderOptimizationLevel(level);
11104     }
11105
11113     int32_t getBuilderOptimizationLevel() noexcept
11114     {
11115         return mImpl->getBuilderOptimizationLevel();
11116     }
11117
11130     void setHardwareCompatibilityLevel(HardwareCompatibilityLevel hardwareCompatibilityLevel) noexcept
11131     {
11132         mImpl->setHardwareCompatibilityLevel(hardwareCompatibilityLevel);
11133     }
11134
11143     HardwareCompatibilityLevel getHardwareCompatibilityLevel() const noexcept
11144     {
11145         return mImpl->getHardwareCompatibilityLevel();
11146     }
11147
11156     void setPluginsToSerialize(char const* const* paths, int32_t nbPaths) noexcept
11157     {
11158         mImpl->setPluginsToSerialize(paths, nbPaths);
11159     }
11160
11169     char const* getPluginToSerialize(int32_t index) const noexcept
11170     {
11171         return mImpl->getPluginToSerialize(index);
11172     }
11173
11179     int32_t getNbPluginsToSerialize() const noexcept
11180     {
11181         return mImpl->getNbPluginsToSerialize();
11182     }
11183
11210     bool setMaxAuxStreams(int32_t nbStreams) noexcept
11211     {
11212         return mImpl->setMaxAuxStreams(nbStreams);
11213     }
11214
11220     int32_t getMaxAuxStreams() const noexcept
11221     {
11222         return mImpl->getMaxAuxStreams();
11223     }
11224
11236     void setProgressMonitor(IProgressMonitor* monitor) noexcept
11237     {
11238         return mImpl->setProgressMonitor(monitor);
11239     }
11240
11246     IProgressMonitor* getProgressMonitor() const noexcept
11247     {
11248         return mImpl->getProgressMonitor();
11249     }
11250
11262     void setRuntimePlatform(RuntimePlatform runtimePlatform) noexcept
11263     {
11264         mImpl->setRuntimePlatform(runtimePlatform);
11265     }
11266
11274     RuntimePlatform getRuntimePlatform() const noexcept
11275     {
11276         return mImpl->getRuntimePlatform();
11277     }
11278
11286     void setMaxNbTactics(int32_t maxNbTactics) noexcept
11287     {
11288         mImpl->setMaxNbTactics(maxNbTactics);
11289     }
11290
11298     int32_t getMaxNbTactics() const noexcept
11299     {
11300         return mImpl->getMaxNbTactics();
11301     }
11302
11314     bool setTilingOptimizationLevel(TilingOptimizationLevel level) noexcept
11315     {
11316         return mImpl->setTilingOptimizationLevel(level);
11317     }
```

```
11318
11326     TilingOptimizationLevel getTilingOptimizationLevel() const noexcept
11327     {
11328         return mImpl->getTilingOptimizationLevel();
11329     }
11330
11342     bool setL2LimitForTiling(int64_t size) noexcept
11343     {
11344         return mImpl->setL2LimitForTiling(size);
11345     }
11346
11354     int64_t getL2LimitForTiling() const noexcept
11355     {
11356         return mImpl->getL2LimitForTiling();
11357     }
11358
11368     bool setRemoteAutoTuningConfig(char const* config) noexcept
11369     {
11370         return mImpl->setRemoteAutoTuningConfig(config);
11371     }
11372
11378     char const* getRemoteAutoTuningConfig() const noexcept
11379     {
11380         return mImpl->getRemoteAutoTuningConfig();
11381     }
11382
11383 protected:
11384     apiv::VBuilderConfig* mImpl;
11385 };
11386
11387 inline IBuilderConfig::~IBuilderConfig() noexcept = default;
11388
11396 using NetworkDefinitionCreationFlags = uint32_t;
11397
11406 enum class NetworkDefinitionCreationFlag :  int32_t
11407 {
11415     kSTRONGLY_TYPED TRT_DEPRECATED_ENUM = 0,
11419     kPREFER_JIT_PYTHON_PLUGINS = 1,
11420
11424     kPREFER_AOT_PYTHON_PLUGINS = 2,
11425 };
11426
11432 template <>
11433 struct impl::EnumMaxImpl<NetworkDefinitionCreationFlag>
11434 {
11435     static constexpr int32_t kVALUE = 3;
11436 };
11437
11445 class IBuilder :  public INoCopy
11446 {
11447 public:
11448     virtual ~IBuilder() noexcept = 0;
11449
11457     int32_t getMaxDLABatchSize() const noexcept
11458     {
11459         return mImpl->getMaxDLABatchSize();
11460     }
11461
11465     int32_t getNbDLACores() const noexcept
11466     {
11467         return mImpl->getNbDLACores();
11468     }
11469
11483     void setGpuAllocator(IGpuAllocator* allocator) noexcept
11484     {
11485         mImpl->setGpuAllocator(allocator);
11486     }
11487
11497     nvinfer1::IBuilderConfig* createBuilderConfig() noexcept
11498     {
11499         return mImpl->createBuilderConfig();
11500     }
11501
11523     nvinfer1::INetworkDefinition* createNetworkV2(NetworkDefinitionCreationFlags flags) noexcept
11524     {
11525         return mImpl->createNetworkV2(flags);
11526     }
11527
11538     nvinfer1::IOptimizationProfile* createOptimizationProfile() noexcept
11539     {
11540         return mImpl->createOptimizationProfile();
```

```
11541     }
11542
11557     void setErrorRecorder(IErrorRecorder* recorder) noexcept
11558     {
11559         mImpl->setErrorRecorder(recorder);
11560     }
11561
11572     IErrorRecorder* getErrorRecorder() const noexcept
11573     {
11574         return mImpl->getErrorRecorder();
11575     }
11576
11580     void reset() noexcept
11581     {
11582         mImpl->reset();
11583     }
11584
11599     nvinfer1::IHostMemory* buildSerializedNetwork(INetworkDefinition& network, IBuilderConfig& config)
      noexcept
11600     {
11601         return mImpl->buildSerializedNetwork(network, config);
11602     }
11603
11620     bool buildSerializedNetworkToStream(
11621         INetworkDefinition& network, IBuilderConfig& config, IStreamWriter& writer) noexcept
11622     {
11623         return mImpl->buildSerializedNetworkToStream(network, config, writer);
11624     }
11625
11644     nvinfer1::IHostMemory* buildSerializedNetwork(
11645         INetworkDefinition& network, IBuilderConfig& config, IHostMemory*& kernelText) noexcept
11646     {
11647         return mImpl->buildSerializedNetworkWithKernelText(network, config, kernelText);
11648     }
11649
11665     nvinfer1::ICudaEngine* buildEngineWithConfig(INetworkDefinition& network, IBuilderConfig& config)
      noexcept
11666     {
11667         return mImpl->buildEngineWithConfig(network, config);
11668     }
11669
11691     bool isNetworkSupported(INetworkDefinition const& network, IBuilderConfig const& config) const
      noexcept
11692     {
11693         return mImpl->isNetworkSupported(network, config);
11694     }
11695
11701     ILogger* getLogger() const noexcept
11702     {
11703         return mImpl->getLogger();
11704     }
11705
11717     bool setMaxThreads(int32_t maxThreads) noexcept
11718     {
11719         return mImpl->setMaxThreads(maxThreads);
11720     }
11721
11731     int32_t getMaxThreads() const noexcept
11732     {
11733         return mImpl->getMaxThreads();
11734     }
11735
11741     IPluginRegistry& getPluginRegistry() noexcept
11742     {
11743         return mImpl->getPluginRegistry();
11744     }
11745
11746 protected:
11747     apiv::VBuilder* mImpl;
11748 };
11749
11750 inline IBuilder::~IBuilder() noexcept = default;
11751
11752 } // namespace nvinfer1
11753
11758 extern "C" TENSORRTAPI void* createInferBuilder_INTERNAL(void* logger, int32_t version) noexcept;
11759
11760 namespace nvinfer1
11761 {
11762 namespace
11763 {
```

```
11764
11772 inline IBuilder* createInferBuilder(ILogger& logger) noexcept
11773 {
11774     return static_cast<IBuilder*>(createInferBuilder_INTERNAL(&logger, NV_TENSORRT_VERSION));
11775 }
11776
11777 } // namespace
11778
11790 extern "C" TENSORRTAPI nvinfer1::IPluginRegistry* getBuilderPluginRegistry(
11791     nvinfer1::EngineCapability capability) noexcept;
11792
11802 extern "C" TENSORRTAPI bool setInternalLibraryPath(AsciiChar const* path) noexcept;
11803
11804 namespace safe
11805 {
11807 class IPluginRegistry;
11808 } // namespace safe
11809
11817 extern "C" TRT_DEPRECATED_API nvinfer1::safe::IPluginRegistry* getBuilderSafePluginRegistry(
11818     nvinfer1::EngineCapability capability) noexcept;
11819
11820 } // namespace nvinfer1
11821
11822 #endif // NV_INFER_H
```

10.3 NvInferImpl.h File Reference

```
#include "NvInferLegacyDims.h"
#include "NvInferRuntimeCommon.h"
```

10.4 NvInferImpl.h

Go to the documentation of this file.

```
1 /*
2 * SPDX-FileCopyrightText:  Copyright (c) 1993-2026 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3 * SPDX-License-Identifier:  Apache-2.0
4 *
5 * Licensed under the Apache License, Version 2.0 (the "License");
6 * you may not use this file except in compliance with the License.
7 * You may obtain a copy of the License at
8 *
9 * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17
18 #ifndef NV_INFER_IMPL_H
19 #define NV_INFER_IMPL_H
20
21 #include "NvInferLegacyDims.h"
22 #include "NvInferRuntimeCommon.h"
23
24 // @cond SuppressDoxyWarnings
25
26 namespace nvinfer1
27 {
28
29 namespace v_1_0
30 {
31 class ILogger;
32 class IProgressMonitor;
33 } // namespace v_1_0
34 using ILogger = v_1_0::ILogger;
```

```
35 using IProgressMonitor = v_1_0::IProgressMonitor;
36
37 namespace v_1_0
38 {
39 class IProfiler;
40 } // namespace v_1_0
41 using IProfiler = v_1_0::IProfiler;
42
43 namespace v_1_0
44 {
45 class IOutputAllocator;
46 } // namespace v_1_0
47 using IOutputAllocator = v_1_0::IOutputAllocator;
48
49 namespace v_1_0
50 {
51 class IDebugListener;
52 } // namespace v_1_0
53 using IDebugListener = v_1_0::IDebugListener;
54
55 class IActivationLayer;
56 class IAssertionLayer;
57 class IAttention;
58 class IBuilder;
59 class IBuilderConfig;
60 class IConcatenationLayer;
61 class IConditionLayer;
62 class IConstantLayer;
63 class IConvolutionLayer;
64 class ICudaEngine;
65 class ICumulativeLayer;
66 class IDeconvolutionLayer;
67 class IDequantizeLayer;
68 class IDimensionExpr;
69 class IDynamicQuantizeLayer;
70 class IEinsumLayer;
71 class IElementWiseLayer;
72 class IEngineInspector;
73 class IExecutionContext;
74 class IFillLayer;
75 class IGatherLayer;
76 class IGridSampleLayer;
77 class IHostMemory;
78 class IIdentityLayer;
79 class ICastLayer;
80 class IIfConditional;
81 class IIfConditionalInputLayer;
82 class IIfConditionalOutputLayer;
83 class IIteratorLayer;
84 class IKVCacheUpdateLayer;
85 class ILayer;
86 class ILoop;
87 class ILoopOutputLayer;
88 class ILRNLayer;
89 class IMatrixMultiplyLayer;
90 class IMoELayer;
91 class IDistCollectiveLayer;
92 class INetworkDefinition;
93 class INormalizationLayer;
94 class INMSLayer;
95 class INonZeroLayer;
96 class IOneHotLayer;
97 class IOptimizationProfile;
98 class IPaddingLayer;
99 class IParametricReLULayer;
100 class IPlugin;
101 class IPluginExt;
102 class IPluginFactory;
103 class IPluginLayer;
104 class IPluginRegistry;
105 class IPluginV2Layer;
106 class IRotaryEmbeddingLayer;
107 class IRuntimeConfig;
108
109 namespace v_1_0
110 {
111 class IPluginV3;
112 } // namespace v_1_0
113 using IPluginV3 = v_1_0::IPluginV3;
114
115 namespace v_1_0
```

```
116 {
117 class IStreamReader;
118 class IStreamWriter;
119 } // namespace v_1_0
120 using IStreamReader = v_1_0::IStreamReader;
121 using IStreamWriter = v_1_0::IStreamWriter;
122 namespace v_1_0
123 {
124 class IStreamReaderV2;
125 } // namespace v_1_0
126 using IStreamReaderV2 = v_1_0::IStreamReaderV2;
127
128 class IPluginV3Layer;
129 class IPoolingLayer;
130 class IQuantizeLayer;
131 class IRaggedSoftMaxLayer;
132 class IRecurrenceLayer;
133 class IReduceLayer;
134 class IRefitter;
135 class IResizeLayer;
136 class IReverseSequenceLayer;
137 class IRuntime;
138 class IScaleLayer;
139 class IScatterLayer;
140 class ISelectLayer;
141 class ISerializationConfig;
142 class IShapeLayer;
143 class IShuffleLayer;
144 class ISliceLayer;
145 class ISoftMaxLayer;
146 class ISqueezeLayer;
147 class ITensor;
148
149 namespace v_1_0
150 {
151 struct TimingCacheKey;
152 struct TimingCacheValue;
153 } // namespace v_1_0
154 using TimingCacheKey = v_1_0::TimingCacheKey;
155 using TimingCacheValue = v_1_0::TimingCacheValue;
156
157 class ITimingCache;
158 class ITopKLayer;
159 class ITripLimitLayer;
160 class IUnaryLayer;
161 class IUnsqueezeLayer;
162 struct Permutation;
163 class Weights;
164
165 enum class ActivationType : int32_t;
166 enum class AttentionIOForm : int32_t;
167 enum class AttentionNormalizationOp : int32_t;
168 enum class BoundingBoxFormat : int32_t;
169 enum class CausalMaskKind : int32_t;
170 enum class BuilderFlag : int32_t;
171 enum class CumulativeOperation : int32_t;
172 enum class DeviceType : int32_t;
173 enum class DimensionOperation : int32_t;
174 enum class ElementWiseOperation : int32_t;
175 enum class EngineCapability : int32_t;
176 enum class FillOperation : int32_t;
177 enum class GatherMode : int32_t;
178 enum class KVCacheMode : int32_t;
179 enum class LayerInformationFormat : int32_t;
180 enum class LayerType : int32_t;
181 enum class LoopOutput : int32_t;
182 enum class MatrixOperation : int32_t;
183 enum class MemoryPoolType : int32_t;
184 enum class MoEActType : int32_t;
185 enum class NetworkDefinitionCreationFlag : int32_t;
186 enum class OptProfileSelector : int32_t;
187 enum class PaddingMode : int32_t;
188 enum class PoolingType : int32_t;
189 enum class ProfilingVerbosity : int32_t;
190 enum class ReduceOperation : int32_t;
191 enum class CollectiveOperation : int32_t;
192 enum class ResizeCoordinateTransformation : int32_t;
193 enum class InterpolationMode : int32_t;
194 enum class ResizeRoundMode : int32_t;
195 enum class ResizeSelector : int32_t;
196 enum class ScaleMode : int32_t;
```

```
197 enum class ScatterMode :  int32_t;
198 enum class SampleMode :  int32_t;
199 enum class SerializationFlag :  int32_t;
200 enum class TensorIOMode :  int32_t;
201 enum class TensorLocation :  int32_t;
202 enum class TopKOperation :  int32_t;
203 enum class TripLimit :  int32_t;
204 enum class UnaryOperation :  int32_t;
205 enum class WeightsRole :  int32_t;
206 enum class PreviewFeature :  int32_t;
207 enum class HardwareCompatibilityLevel :  int32_t;
208 enum class ExecutionContextAllocationStrategy :  int32_t;
209 enum class RuntimePlatform :  int32_t;
210 enum class TilingOptimizationLevel :  int32_t;
211 enum class EngineStat :  int32_t;
212
213
214 using TacticSources = uint32_t;
215 using TensorFormats = uint32_t;
216 using BuilderFlags = uint32_t;
217 using NetworkDefinitionCreationFlags = uint32_t;
218 using TempfileControlFlags = uint32_t;
219 using SerializationFlags = uint32_t;
220
228
229 namespace apiv
230 {
231
232 class VRoot
233 {
234 public:
235     virtual ~VRoot() noexcept = default;
236 };
237
238 class VHostMemory :  public VRoot
239 {
240 public:
241     virtual void* data() const noexcept = 0;
242     virtual std::size_t size() const noexcept = 0;
243     virtual DataType type() const noexcept = 0;
244 };
245
246 class VDimensionExpr :  public VRoot
247 {
248 public:
249     virtual bool isConstant() const = 0;
250     virtual int64_t getConstantValue() const = 0;
251     virtual bool isSizeTensor() const = 0;
252 };
253
254 class VExprBuilder :  public VRoot
255 {
256 public:
257     virtual IDimensionExpr const* constant(int64_t value) = 0;
258     virtual IDimensionExpr const* operation(
259         DimensionOperation op, IDimensionExpr const& first, IDimensionExpr const& second)
260         = 0;
261     virtual IDimensionExpr const* declareSizeTensor(
262         int32_t outputIndex, IDimensionExpr const& opt, IDimensionExpr const& upper)
263         = 0;
264 };
265
266 class VRuntime :  public VRoot
267 {
268 public:
269     virtual IRuntime* getPImpl() noexcept = 0;
270     virtual nvinfer1::ICudaEngine* deserializeCudaEngine(void const* blob, std::size_t size) noexcept = 0;
271     virtual void setDLACore(int32_t dlaCore) noexcept = 0;
272     virtual int32_t getDLACore() const noexcept = 0;
273     virtual int32_t getNbDLACores() const noexcept = 0;
274     virtual void setGpuAllocator(IGpuAllocator* allocator) noexcept = 0;
275     virtual void setErrorRecorder(IErrorRecorder* recorder) noexcept = 0;
276     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
277     virtual ILogger* getLogger() const noexcept = 0;
278     virtual bool setMaxThreads(int32_t maxThreads) noexcept = 0;
279     virtual int32_t getMaxThreads() const noexcept = 0;
280     virtual void setTemporaryDirectory(char const*) noexcept = 0;
281     virtual char const* getTemporaryDirectory() const noexcept = 0;
282     virtual void setTempfileControlFlags(TempfileControlFlags) noexcept = 0;
283     virtual TempfileControlFlags getTempfileControlFlags() const noexcept = 0;
284     virtual IPluginRegistry& getPluginRegistry() noexcept = 0;
```

```
285     virtual void setPluginRegistryParent(IPluginRegistry* parent) noexcept = 0;
286     virtual IRuntime* loadRuntime(char const* path) noexcept = 0;
287     virtual void setEngineHostCodeAllowed(bool allowed) noexcept = 0;
288     virtual bool getEngineHostCodeAllowed() const noexcept = 0;
289     // Added in TensorRT version 10.7
290     virtual nvinfer1::ICudaEngine* deserializeCudaEngineV2(IStreamReaderV2& streamReader) noexcept = 0;
291 };
292
293 class VRefitter :  public VRoot
294 {
295 public:
296     virtual IRefitter* getPImpl() noexcept = 0;
297     virtual bool setWeights(char const* layerName, WeightsRole role, const Weights weights) noexcept = 0;
298     virtual bool refitCudaEngine() noexcept = 0;
299     virtual int32_t getMissing(int32_t size, char const** layerNames, WeightsRole* roles) noexcept = 0;
300     virtual int32_t getAll(int32_t size, char const** layerNames, WeightsRole* roles) noexcept = 0;
301     virtual void setErrorRecorder(IErrorRecorder* recorder) noexcept = 0;
302     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
303     virtual bool setNamedWeights(char const* name, Weights weights) noexcept = 0;
304     virtual int32_t getMissingWeights(int32_t size, char const** weightsNames) noexcept = 0;
305     virtual int32_t getAllWeights(int32_t size, char const** weightsNames) noexcept = 0;
306     virtual ILogger* getLogger() const noexcept = 0;
307     virtual bool setMaxThreads(int32_t maxThreads) noexcept = 0;
308     virtual int32_t getMaxThreads() const noexcept = 0;
309     virtual bool setNamedWeightsWithLocation(char const* name, Weights weights, TensorLocation location)
    noexcept = 0;
310     virtual Weights getNamedWeights(char const* weightsName) const noexcept = 0;
311     virtual TensorLocation getWeightsLocation(char const* weightsName) const noexcept = 0;
312     virtual bool unsetNamedWeights(char const* weightsName) noexcept = 0;
313     virtual void setWeightsValidation(bool weightsValidation) noexcept = 0;
314     virtual bool getWeightsValidation() const noexcept = 0;
315     virtual bool refitCudaEngineAsync(cudaStream_t stream) noexcept = 0;
316     virtual Weights getWeightsPrototype(char const* weightsName) const noexcept = 0;
317 };
318
319 class VOptimizationProfile :  public VRoot
320 {
321 public:
322     virtual bool setDimensions(char const* inputName, OptProfileSelector select, Dims const& dims) noexcept
    = 0;
323     virtual Dims getDimensions(char const* inputName, OptProfileSelector select) const noexcept = 0;
324     virtual int32_t getNbShapeValues(char const* inputName) const noexcept = 0;
325     virtual bool setExtraMemoryTarget(float target) noexcept = 0;
326     virtual float getExtraMemoryTarget() const noexcept = 0;
327     virtual bool isValid() const noexcept = 0;
328     // Added in TensorRT 10.11
329     TRT_NODISCARD virtual bool setShapeValuesV2(
330         char const* inputName, OptProfileSelector select, int64_t const* values, int32_t nbValues) noexcept =
    0;
331     TRT_NODISCARD virtual int64_t const* getShapeValuesV2(
332         char const* inputName, OptProfileSelector select) const noexcept = 0;
333 };
334
335 class VCudaEngine :  public VRoot
336 {
337 public:
338     virtual ICudaEngine* getPImpl() noexcept = 0;
339     virtual int32_t getNbLayers() const noexcept = 0;
340     virtual IHostMemory* serialize() const noexcept = 0;
341     virtual IExecutionContext* createExecutionContext(ExecutionContextAllocationStrategy strategy) noexcept
    = 0;
342     virtual bool isRefittable() const noexcept = 0;
343     virtual char const* getName() const noexcept = 0;
344     virtual int32_t getNbOptimizationProfiles() const noexcept = 0;
345     virtual EngineCapability getEngineCapability() const noexcept = 0;
346     virtual void setErrorRecorder(IErrorRecorder* recorder) noexcept = 0;
347     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
348     virtual TacticSources getTacticSources() const noexcept = 0;
349     virtual ProfilingVerbosity getProfilingVerbosity() const noexcept = 0;
350     virtual IEngineInspector* createEngineInspector() const noexcept = 0;
351     virtual Dims getTensorShape(char const* tensorName) const noexcept = 0;
352     virtual DataType getTensorDataType(char const* tensorName) const noexcept = 0;
353     virtual TensorLocation getTensorLocation(char const* tensorName) const noexcept = 0;
354     virtual bool isShapeInferenceIO(char const* tensorName) const noexcept = 0;
355     virtual TensorIOMode getTensorIOMode(char const* tensorName) const noexcept = 0;
356     virtual int32_t getTensorBytesPerComponent(char const* tensorName) const noexcept = 0;
357     virtual int32_t getTensorComponentsPerElement(char const* tensorName) const noexcept = 0;
358     virtual TensorFormat getTensorFormat(char const* tensorName) const noexcept = 0;
359     virtual char const* getTensorFormatDesc(char const* tensorName) const noexcept = 0;
360     virtual int32_t getTensorVectorizedDim(char const* tensorName) const noexcept = 0;
361     virtual Dims getProfileShape(
```

```
362         char const* tensorName, int32_t profileIndex, OptProfileSelector select) const noexcept = 0;
363     virtual int32_t getNbIOTensors() const noexcept = 0;
364     virtual char const* getIOTensorName(int32_t index) const noexcept = 0;
365     virtual HardwareCompatibilityLevel getHardwareCompatibilityLevel() const noexcept = 0;
366     virtual int32_t getNbAuxStreams() const noexcept = 0;
367
368     virtual int32_t getTensorBytesPerComponentV2(char const* tensorName, int32_t profileIndex) const noexcept
    = 0;
369     virtual int32_t getTensorComponentsPerElementV2(char const* tensorName, int32_t profileIndex) const
    noexcept = 0;
370     virtual TensorFormat getTensorFormatV2(char const* tensorName, int32_t profileIndex) const noexcept = 0;
371     virtual char const* getTensorFormatDescV2(char const* tensorName, int32_t profileIndex) const noexcept =
    0;
372     virtual int32_t getTensorVectorizedDimV2(char const* tensorName, int32_t profileIndex) const noexcept =
    0;
373
374     virtual ISerializationConfig* createSerializationConfig() noexcept = 0;
375     virtual IHostMemory* serializeWithConfig(ISerializationConfig& config) const noexcept = 0;
376
377     virtual IRefitter* createRefitter(ILogger& logger) noexcept = 0;
378
379     virtual int64_t getStreamableWeightsSize() const noexcept = 0;
380
381     virtual bool isDebugTensor(char const* name) const noexcept = 0;
382
383     // Added in TensorRT 10.1
384     virtual bool setWeightStreamingBudgetV2(int64_t gpuMemoryBudget) noexcept = 0;
385     virtual int64_t getWeightStreamingBudgetV2() const noexcept = 0;
386     virtual int64_t getWeightStreamingAutomaticBudget() const noexcept = 0;
387     virtual int64_t getWeightStreamingScratchMemorySize() const noexcept = 0;
388     virtual int64_t getDeviceMemorySizeV2() const noexcept = 0;
389     virtual int64_t getDeviceMemorySizeForProfileV2(int32_t profileIndex) const noexcept = 0;
390     // Added in TensorRT 10.11
391     TRT_NODISCARD virtual int64_t const* getProfileTensorValuesV2(
392         char const* tensorName, int32_t profileIndex, OptProfileSelector select) const noexcept = 0;
393     TRT_NODISCARD virtual IExecutionContext* createExecutionContextWithRuntimeConfig(
394         IRuntimeConfig* runtimeConfig) noexcept = 0;
395     TRT_NODISCARD virtual IRuntimeConfig* createRuntimeConfig() noexcept = 0;
396     TRT_NODISCARD virtual int64_t getEngineStat(EngineStat stat) const noexcept = 0;
397     // Added in TensorRT 10.15
398     TRT_NODISCARD virtual char const* getAliasedInputTensor(char const* tensorName) const noexcept = 0;
399 };
400
401 class VExecutionContext :  public VRoot
402 {
403 public:
404     virtual IExecutionContext* getPImpl() noexcept = 0;
405     virtual void setDebugSync(bool sync) noexcept = 0;
406     virtual bool getDebugSync() const noexcept = 0;
407     virtual void setProfiler(IProfiler*) noexcept = 0;
408     virtual IProfiler* getProfiler() const noexcept = 0;
409     virtual ICudaEngine const& getEngine() const noexcept = 0;
410     virtual void setName(char const* name) noexcept = 0;
411     virtual char const* getName() const noexcept = 0;
412     virtual void setDeviceMemory(void* memory) noexcept = 0;
413     virtual int32_t getOptimizationProfile() const noexcept = 0;
414     virtual bool allInputDimensionsSpecified() const noexcept = 0;
415     virtual void setErrorRecorder(IErrorRecorder* recorder) noexcept = 0;
416     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
417     virtual bool executeV2(void* const* bindings) noexcept = 0;
418     virtual bool setOptimizationProfileAsync(int32_t profileIndex, cudaStream_t stream) noexcept = 0;
419     virtual void setEnqueueEmitsProfile(bool enqueueEmitsProfile) noexcept = 0;
420     virtual bool getEnqueueEmitsProfile() const noexcept = 0;
421     virtual bool reportToProfiler() const noexcept = 0;
422     virtual bool setInputShape(char const* tensorName, Dims const& dims) noexcept = 0;
423     virtual Dims getTensorShape(char const* tensorName) const noexcept = 0;
424     virtual Dims getTensorStrides(char const* tensorName) const noexcept = 0;
425     virtual bool setTensorAddress(char const* tensorName, void* data) noexcept = 0;
426     virtual void const* getTensorAddress(char const* tensorName) const noexcept = 0;
427     virtual bool setInputTensorAddress(char const* tensorName, void const* data) noexcept = 0;
428     virtual bool setOutputTensorAddress(char const* tensorName, void* data) noexcept = 0;
429     virtual int32_t inferShapes(int32_t nbMaxNames, char const** tensorNames) noexcept = 0;
430     virtual bool setInputConsumedEvent(cudaEvent_t event) noexcept = 0;
431     virtual cudaEvent_t getInputConsumedEvent() const noexcept = 0;
432     virtual void* getOutputTensorAddress(char const* tensorName) const noexcept = 0;
433     virtual bool setOutputAllocator(char const* tensorName, IOutputAllocator* outputAllocator) noexcept = 0;
434     virtual IOutputAllocator* getOutputAllocator(char const* name) noexcept = 0;
435     virtual int64_t getMaxOutputSize(char const* tensorName) const noexcept = 0;
436     virtual bool setTemporaryStorageAllocator(IGpuAllocator* allocator) noexcept = 0;
437     virtual IGpuAllocator* getTemporaryStorageAllocator() const noexcept = 0;
438     virtual bool enqueueV3(cudaStream_t stream) noexcept = 0;
```

```
439     virtual void setPersistentCacheLimit(size_t size) noexcept = 0;
440     virtual size_t getPersistentCacheLimit() const noexcept = 0;
441     virtual bool setNvtxVerbosity(ProfilingVerbosity verbosity) noexcept = 0;
442     virtual ProfilingVerbosity getNvtxVerbosity() const noexcept = 0;
443     virtual void setAuxStreams(cudaStream_t* auxStreams, int32_t nbStreams) noexcept = 0;
444     virtual bool setDebugListener(IDebugListener* listener) noexcept = 0;
445     virtual IDebugListener* getDebugListener() noexcept = 0;
446     virtual bool setTensorDebugState(char const* name, bool flag) noexcept = 0;
447     virtual bool getDebugState(char const* name) const noexcept = 0;
448     virtual bool setAllTensorsDebugState(bool flag) noexcept = 0;
449     virtual size_t updateDeviceMemorySizeForShapes() noexcept = 0;
450     virtual void setDeviceMemoryV2(void* memory, int64_t size) noexcept = 0;
451     TRT_NODISCARD virtual IRuntimeConfig* getRuntimeConfig() const noexcept = 0;
452     virtual bool setUnfusedTensorsDebugState(bool flag) noexcept = 0;
453     virtual bool getUnfusedTensorsDebugState() const noexcept = 0;
454     virtual bool setCommunicator(void* communicator) noexcept = 0;
455 };
456
457 class VEngineInspector :  public VRoot
458 {
459 public:
460     virtual IEngineInspector* getPImpl() noexcept = 0;
461     virtual bool setExecutionContext(IExecutionContext const* context) noexcept = 0;
462     virtual IExecutionContext const* getExecutionContext() const noexcept = 0;
463     virtual char const* getLayerInformation(int32_t layerIndex, LayerInformationFormat format) const
    noexcept = 0;
464     virtual char const* getEngineInformation(LayerInformationFormat format) const noexcept = 0;
465     virtual void setErrorRecorder(IErrorRecorder* recorder) noexcept = 0;
466     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
467 };
468
469 class VTensor :  public VRoot
470 {
471 public:
472     virtual void setName(char const* name) noexcept = 0;
473     virtual char const* getName() const noexcept = 0;
474     virtual void setDimensions(Dims const& dimensions) noexcept = 0;
475     virtual Dims getDimensions() const noexcept = 0;
476     virtual DataType getType() const noexcept = 0;
477     virtual bool isNetworkInput() const noexcept = 0;
478     virtual bool isNetworkOutput() const noexcept = 0;
479     virtual void setBroadcastAcrossBatch(bool broadcastAcrossBatch) noexcept = 0;
480     virtual bool getBroadcastAcrossBatch() const noexcept = 0;
481     virtual TensorLocation getLocation() const noexcept = 0;
482     virtual void setLocation(TensorLocation location) noexcept = 0;
483     virtual void setAllowedFormats(TensorFormats formats) noexcept = 0;
484     virtual TensorFormats getAllowedFormats() const noexcept = 0;
485     virtual bool isShapeTensor() const noexcept = 0;
486     virtual bool isExecutionTensor() const noexcept = 0;
487     virtual void setDimensionName(int32_t index, char const* name) noexcept = 0;
488     virtual char const* getDimensionName(int32_t index) const noexcept = 0;
489 };
490
491 class VLayer :  public VRoot
492 {
493 public:
494     virtual LayerType getType() const noexcept = 0;
495     virtual void setName(char const* name) noexcept = 0;
496     virtual char const* getName() const noexcept = 0;
497     virtual int32_t getNbInputs() const noexcept = 0;
498     virtual ITensor* getInput(int32_t index) const noexcept = 0;
499     virtual int32_t getNbOutputs() const noexcept = 0;
500     virtual ITensor* getOutput(int32_t index) const noexcept = 0;
501     virtual void setInput(int32_t index, ITensor& tensor) noexcept = 0;
502     virtual DataType getOutputType(int32_t index) const noexcept = 0;
503     virtual void setMetadata(char const* docString) noexcept = 0;
504     virtual char const* getMetadata() const noexcept = 0;
505     virtual bool setNbRanks(int32_t nbRanks) noexcept = 0;
506     virtual int32_t getNbRanks() const noexcept = 0;
507 };
508
509 class VConvolutionLayer :  public VRoot
510 {
511 public:
512     virtual void setNbOutputMaps(int64_t nbOutputMaps) noexcept = 0;
513     virtual int64_t getNbOutputMaps() const noexcept = 0;
514     virtual void setNbGroups(int64_t nbGroups) noexcept = 0;
515     virtual int64_t getNbGroups() const noexcept = 0;
516     virtual void setKernelWeights(Weights weights) noexcept = 0;
517     virtual Weights getKernelWeights() const noexcept = 0;
518     virtual void setBiasWeights(Weights weights) noexcept = 0;
```

```
519     virtual Weights getBiasWeights() const noexcept = 0;
520     virtual void setPrePadding(Dims const&  padding) noexcept = 0;
521     virtual Dims getPrePadding() const noexcept = 0;
522     virtual void setPostPadding(Dims const& padding) noexcept = 0;
523     virtual Dims getPostPadding() const noexcept = 0;
524     virtual void setPaddingMode(PaddingMode paddingMode) noexcept = 0;
525     virtual PaddingMode getPaddingMode() const noexcept = 0;
526     virtual void setKernelSizeNd(Dims const& kernelSize) noexcept = 0;
527     virtual Dims getKernelSizeNd() const noexcept = 0;
528     virtual void setStrideNd(Dims const& stride) noexcept = 0;
529     virtual Dims getStrideNd() const noexcept = 0;
530     virtual void setPaddingNd(Dims const& padding) noexcept = 0;
531     virtual Dims getPaddingNd() const noexcept = 0;
532     virtual void setDilationNd(Dims const& dilation) noexcept = 0;
533     virtual Dims getDilationNd() const noexcept = 0;
534 };
535
536 class VActivationLayer :  public VRoot
537 {
538 public:
539     virtual void setActivationType(ActivationType type) noexcept = 0;
540     virtual ActivationType getActivationType() const noexcept = 0;
541     virtual void setAlpha(float alpha) noexcept = 0;
542     virtual void setBeta(float beta) noexcept = 0;
543     virtual float getAlpha() const noexcept = 0;
544     virtual float getBeta() const noexcept = 0;
545 };
546
547 class VPoolingLayer :  public VRoot
548 {
549 public:
550     virtual void setPoolingType(PoolingType type) noexcept = 0;
551     virtual PoolingType getPoolingType() const noexcept = 0;
552     virtual void setBlendFactor(float blendFactor) noexcept = 0;
553     virtual float getBlendFactor() const noexcept = 0;
554     virtual void setAverageCountExcludesPadding(bool exclusive) noexcept = 0;
555     virtual bool getAverageCountExcludesPadding() const noexcept = 0;
556     virtual void setPrePadding(Dims const& padding) noexcept = 0;
557     virtual Dims getPrePadding() const noexcept = 0;
558     virtual void setPostPadding(Dims const& padding) noexcept = 0;
559     virtual Dims getPostPadding() const noexcept = 0;
560     virtual void setPaddingMode(PaddingMode paddingMode) noexcept = 0;
561     virtual PaddingMode getPaddingMode() const noexcept = 0;
562     virtual void setWindowSizeNd(Dims const& windowSize) noexcept = 0;
563     virtual Dims getWindowSizeNd() const noexcept = 0;
564     virtual void setStrideNd(Dims const& stride) noexcept = 0;
565     virtual Dims getStrideNd() const noexcept = 0;
566     virtual void setPaddingNd(Dims const& padding) noexcept = 0;
567     virtual Dims getPaddingNd() const noexcept = 0;
568 };
569
570 class VLRNLayer :  public VRoot
571 {
572 public:
573     virtual void setWindowSize(int64_t windowSize) noexcept = 0;
574     virtual int64_t getWindowSize() const noexcept = 0;
575     virtual void setAlpha(float alpha) noexcept = 0;
576     virtual float getAlpha() const noexcept = 0;
577     virtual void setBeta(float beta) noexcept = 0;
578     virtual float getBeta() const noexcept = 0;
579     virtual void setK(float k) noexcept = 0;
580     virtual float getK() const noexcept = 0;
581 };
582
583 class VScaleLayer :  public VRoot
584 {
585 public:
586     virtual void setMode(ScaleMode mode) noexcept = 0;
587     virtual ScaleMode getMode() const noexcept = 0;
588     virtual void setShift(Weights shift) noexcept = 0;
589     virtual Weights getShift() const noexcept = 0;
590     virtual void setScale(Weights scale) noexcept = 0;
591     virtual Weights getScale() const noexcept = 0;
592     virtual void setPower(Weights power) noexcept = 0;
593     virtual Weights getPower() const noexcept = 0;
594     virtual int32_t getChannelAxis() const noexcept = 0;
595     virtual void setChannelAxis(int32_t channelAxis) noexcept = 0;
596 };
597
598 class VSoftMaxLayer :  public VRoot
599 {
```

```
600 public:
601     virtual void setAxes(uint32_t axes) noexcept = 0;
602     virtual uint32_t getAxes() const noexcept = 0;
603 };
604
605 class VConcatenationLayer :  public VRoot
606 {
607 public:
608     virtual void setAxis(int32_t axis) noexcept = 0;
609     virtual int32_t getAxis() const noexcept = 0;
610 };
611
612 class VDeconvolutionLayer :  public VRoot
613 {
614 public:
615     virtual void setNbOutputMaps(int64_t nbOutputMaps) noexcept = 0;
616     virtual int64_t getNbOutputMaps() const noexcept = 0;
617     virtual void setNbGroups(int64_t nbGroups) noexcept = 0;
618     virtual int64_t getNbGroups() const noexcept = 0;
619     virtual void setKernelWeights(Weights weights) noexcept = 0;
620     virtual Weights getKernelWeights() const noexcept = 0;
621     virtual void setBiasWeights(Weights weights) noexcept = 0;
622     virtual Weights getBiasWeights() const noexcept = 0;
623     virtual void setPrePadding(Dims const& padding) noexcept = 0;
624     virtual Dims getPrePadding() const noexcept = 0;
625     virtual void setPostPadding(Dims const& padding) noexcept = 0;
626     virtual Dims getPostPadding() const noexcept = 0;
627     virtual void setPaddingMode(PaddingMode paddingMode) noexcept = 0;
628     virtual PaddingMode getPaddingMode() const noexcept = 0;
629     virtual void setKernelSizeNd(Dims const& kernelSize) noexcept = 0;
630     virtual Dims getKernelSizeNd() const noexcept = 0;
631     virtual void setStrideNd(Dims const& stride) noexcept = 0;
632     virtual Dims getStrideNd() const noexcept = 0;
633     virtual void setPaddingNd(Dims const& padding) noexcept = 0;
634     virtual Dims getPaddingNd() const noexcept = 0;
635     virtual void setDilationNd(Dims const& dilation) noexcept = 0;
636     virtual Dims getDilationNd() const noexcept = 0;
637 };
638
639 class VElementWiseLayer :  public VRoot
640 {
641 public:
642     virtual void setOperation(ElementWiseOperation op) noexcept = 0;
643     virtual ElementWiseOperation getOperation() const noexcept = 0;
644 };
645
646 class VGatherLayer :  public VRoot
647 {
648 public:
649     virtual void setGatherAxis(int32_t axis) noexcept = 0;
650     virtual int32_t getGatherAxis() const noexcept = 0;
651     virtual void setNbElementWiseDims(int32_t k) noexcept = 0;
652     virtual int32_t getNbElementWiseDims() const noexcept = 0;
653     virtual void setMode(GatherMode mode) noexcept = 0;
654     virtual GatherMode getMode() const noexcept = 0;
655 };
656
657 class VPluginLayer :  public VRoot
658 {
659 public:
660     virtual IPlugin& getPlugin() noexcept = 0;
661 };
662
663 class VPluginV2Layer :  public VRoot
664 {
665 public:
666     virtual IPluginV2& getPlugin() noexcept = 0;
667 };
668
669 class VPluginV3Layer :  public VRoot
670 {
671 public:
672     virtual IPluginV3& getPlugin() noexcept = 0;
673 };
674
675 class VUnaryLayer :  public VRoot
676 {
677 public:
678     virtual void setOperation(UnaryOperation op) noexcept = 0;
679     virtual UnaryOperation getOperation() const noexcept = 0;
680 };
```

```
681
682 class VReduceLayer :  public VRoot
683 {
684 public:
685     virtual void setOperation(ReduceOperation op) noexcept = 0;
686     virtual ReduceOperation getOperation() const noexcept = 0;
687     virtual void setReduceAxes(uint32_t reduceAxes) noexcept = 0;
688     virtual uint32_t getReduceAxes() const noexcept = 0;
689     virtual void setKeepDimensions(bool keepDimensions) noexcept = 0;
690     virtual bool getKeepDimensions() const noexcept = 0;
691 };
692
693 class VPaddingLayer :  public VRoot
694 {
695 public:
696     virtual void setPrePaddingNd(Dims const& padding) noexcept = 0;
697     virtual Dims getPrePaddingNd() const noexcept = 0;
698     virtual void setPostPaddingNd(Dims const& padding) noexcept = 0;
699     virtual Dims getPostPaddingNd() const noexcept = 0;
700 };
701
702 class VShuffleLayer :  public VRoot
703 {
704 public:
705     virtual void setFirstTranspose(Permutation const& permutation) noexcept = 0;
706     virtual Permutation const& getFirstTranspose() const noexcept = 0;
707     virtual void setReshapeDimensions(Dims const& dimensions) noexcept = 0;
708     virtual Dims getReshapeDimensions() const noexcept = 0;
709     virtual void setSecondTranspose(Permutation const& permutation) noexcept = 0;
710     virtual Permutation const& getSecondTranspose() const noexcept = 0;
711     virtual void setZeroIsPlaceholder(bool zeroIsPlaceholder) noexcept = 0;
712     virtual bool getZeroIsPlaceholder() const noexcept = 0;
713 };
714
715 class VSliceLayer :  public VRoot
716 {
717 public:
718     virtual void setStart(Dims const& start) noexcept = 0;
719     virtual Dims getStart() const noexcept = 0;
720     virtual void setSize(Dims const& size) noexcept = 0;
721     virtual Dims getSize() const noexcept = 0;
722     virtual void setStride(Dims const& stride) noexcept = 0;
723     virtual Dims getStride() const noexcept = 0;
724     virtual void setMode(SampleMode mode) noexcept = 0;
725     virtual SampleMode getMode() const noexcept = 0;
726     virtual void setAxes(Dims const& axes) noexcept = 0;
727     virtual Dims getAxes() const noexcept = 0;
728 };
729
730 class VShapeLayer :  public VRoot
731 {
732 public:
733 };
734
735 class VTopKLayer :  public VRoot
736 {
737 public:
738     virtual void setOperation(TopKOperation op) noexcept = 0;
739     virtual TopKOperation getOperation() const noexcept = 0;
740     virtual void setK(int32_t k) noexcept = 0;
741     virtual int32_t getK() const noexcept = 0;
742     virtual void setReduceAxes(uint32_t reduceAxes) noexcept = 0;
743     virtual uint32_t getReduceAxes() const noexcept = 0;
744     virtual bool setIndicesType(DataType type) noexcept = 0;
745     virtual DataType getIndicesType() const noexcept = 0;
746 };
747
748 class VMatrixMultiplyLayer :  public VRoot
749 {
750 public:
751     virtual void setOperation(int32_t index, MatrixOperation op) noexcept = 0;
752     virtual MatrixOperation getOperation(int32_t index) const noexcept = 0;
753 };
754
755 class VNonZeroLayer :  public VRoot
756 {
757 public:
758     virtual bool setIndicesType(DataType type) noexcept = 0;
759     virtual DataType getIndicesType() const noexcept = 0;
760 };
761
```

```
762 class VRaggedSoftMaxLayer :  public VRoot
763 {
764 public:
765 };
766
767 class VIdentityLayer :  public VRoot
768 {
769 public:
770 };
771
772 class VCastLayer :  public VRoot
773 {
774 public:
775     virtual void setToType(DataType toType) noexcept = 0;
776     virtual DataType getToType() const noexcept = 0;
777 };
778
779 class VConstantLayer :  public VRoot
780 {
781 public:
782     virtual void setWeights(Weights weights) noexcept = 0;
783     virtual Weights getWeights() const noexcept = 0;
784     virtual void setDimensions(Dims const& dimensions) noexcept = 0;
785     virtual Dims getDimensions() const noexcept = 0;
786 };
787
788 class VParametricReLULayer :  public VRoot
789 {
790 public:
791 };
792
793 class VResizeLayer :  public VRoot
794 {
795 public:
796     virtual void setOutputDimensions(Dims const& dimensions) noexcept = 0;
797     virtual Dims getOutputDimensions() const noexcept = 0;
798     virtual void setScales(float const* scales, int32_t nbScales) noexcept = 0;
799     virtual int32_t getScales(int32_t size, float* scales) const noexcept = 0;
800     virtual void setResizeMode(InterpolationMode interpolationMode) noexcept = 0;
801     virtual InterpolationMode getResizeMode() const noexcept = 0;
802     virtual void setCoordinateTransformation(ResizeCoordinateTransformation coordTransform) noexcept = 0;
803     virtual ResizeCoordinateTransformation getCoordinateTransformation() const noexcept = 0;
804     virtual void setSelectorForSinglePixel(ResizeSelector selector) noexcept = 0;
805     virtual ResizeSelector getSelectorForSinglePixel() const noexcept = 0;
806     virtual void setNearestRounding(ResizeRoundMode value) noexcept = 0;
807     virtual ResizeRoundMode getNearestRounding() const noexcept = 0;
808     virtual void setCubicCoeff(float value) noexcept = 0;
809     virtual float getCubicCoeff() const noexcept = 0;
810     virtual void setExcludeOutside(bool value) noexcept = 0;
811     virtual bool getExcludeOutside() const noexcept = 0;
812 };
813
814 class VLoopBoundaryLayer :  public VRoot
815 {
816 public:
817     virtual ILoop* getLoop() const noexcept = 0;
818 };
819
820 class VRecurrenceLayer :  public VRoot
821 {
822 public:
823 };
824
825 class VLoopOutputLayer :  public VRoot
826 {
827 public:
828     virtual LoopOutput getLoopOutput() const noexcept = 0;
829     virtual void setAxis(int32_t axis) noexcept = 0;
830     virtual int32_t getAxis() const noexcept = 0;
831 };
832
833 class VTripLimitLayer :  public VRoot
834 {
835 public:
836     virtual TripLimit getTripLimit() const noexcept = 0;
837 };
838
839 class VIteratorLayer :  public VRoot
840 {
841 public:
842     virtual void setAxis(int32_t axis) noexcept = 0;
```

```
843     virtual int32_t getAxis() const noexcept = 0;
844     virtual void setReverse(bool reverse) noexcept = 0;
845     virtual bool getReverse() const noexcept = 0;
846 };
847 class VLoop :  public VRoot
848 {
849 public:
850     virtual IRecurrenceLayer* addRecurrence(ITensor& initialValue) noexcept = 0;
851     virtual ITripLimitLayer* addTripLimit(ITensor& tensor, TripLimit limit) noexcept = 0;
852     virtual IIteratorLayer* addIterator(ITensor& tensor, int32_t axis = 0, bool reverse = false) noexcept =
    0;
853     virtual ILoopOutputLayer* addLoopOutput(ITensor& tensor, LoopOutput outputKind, int32_t axis = 0)
    noexcept = 0;
854     virtual void setName(char const* name) noexcept = 0;
855     virtual char const* getName() const noexcept = 0;
856 };
857
858 class VConditionalBoundaryLayer :  public VRoot
859 {
860 public:
861     virtual IIfConditional* getConditional() const noexcept = 0;
862 };
863
864 class VConditionLayer :  public VRoot
865 {
866 public:
867 };
868
869 class VConditionalInputLayer :  public VRoot
870 {
871 public:
872 };
873
874 class VConditionalOutputLayer :  public VRoot
875 {
876 public:
877 };
878
879 class VIfConditional :  public VRoot
880 {
881 public:
882     virtual IConditionLayer* setCondition(ITensor& tensor) noexcept = 0;
883     virtual IIfConditionalInputLayer* addInput(ITensor& tensor) noexcept = 0;
884     virtual IIfConditionalOutputLayer* addOutput(ITensor& trueTensor, ITensor& falseTensor) noexcept = 0;
885     virtual void setName(char const* name) noexcept = 0;
886     virtual char const* getName() const noexcept = 0;
887 };
888
889 class VAttentionBoundaryLayer :  public VRoot
890 {
891 public:
892     virtual IAttention* getAttention() const noexcept = 0;
893 };
894
895 class VAttentionInputLayer :  public VRoot
896 {
897 public:
898 };
899
900 class VAttentionOutputLayer :  public VRoot
901 {
902 public:
903 };
904
905 class VAttention :  public VRoot
906 {
907 public:
908     TRT_NODISCARD virtual bool setInput(int32_t index, ITensor& input) noexcept = 0;
909     TRT_NODISCARD virtual int32_t getNbInputs() const noexcept = 0;
910     TRT_NODISCARD virtual ITensor* getInput(int32_t index) const noexcept = 0;
911     TRT_NODISCARD virtual int32_t getNbOutputs() const noexcept = 0;
912     TRT_NODISCARD virtual ITensor* getOutput(int32_t index) const noexcept = 0;
913     TRT_NODISCARD virtual bool setName(char const* name) noexcept = 0;
914     TRT_NODISCARD virtual char const* getName() const noexcept = 0;
915     TRT_NODISCARD virtual bool setNormalizationOperation(AttentionNormalizationOp op) noexcept = 0;
916     TRT_NODISCARD virtual AttentionNormalizationOp getNormalizationOperation() const noexcept = 0;
917     TRT_DEPRECATED virtual bool setCausal(bool isCausal) noexcept = 0;
918     TRT_DEPRECATED virtual bool getCausal() const noexcept = 0;
919     TRT_NODISCARD virtual bool setMask(ITensor& mask) noexcept = 0;
920     TRT_NODISCARD virtual ITensor* getMask() const noexcept = 0;
921     TRT_NODISCARD virtual bool setDecomposable(bool decomposable) noexcept = 0;
```

```
922     TRT_NODISCARD virtual bool getDecomposable() const noexcept = 0;
923     TRT_NODISCARD virtual bool setNormalizationQuantizeScale(ITensor& tensor) noexcept = 0;
924     TRT_NODISCARD virtual ITensor* getNormalizationQuantizeScale() const noexcept = 0;
925     TRT_NODISCARD virtual bool setNormalizationQuantizeToType(DataType type) noexcept = 0;
926     TRT_NODISCARD virtual DataType getNormalizationQuantizeToType() const noexcept = 0;
927     TRT_NODISCARD virtual bool setMetadata(char const* docString) noexcept = 0;
928     TRT_NODISCARD virtual char const* getMetadata() const noexcept = 0;
929     TRT_NODISCARD virtual bool setNbRanks(int32_t nbRanks) noexcept = 0;
930     TRT_NODISCARD virtual int32_t getNbRanks() const noexcept = 0;
931     TRT_NODISCARD virtual bool setCausalKind(CausalMaskKind kind) noexcept = 0;
932     TRT_NODISCARD virtual CausalMaskKind getCausalKind() const noexcept = 0;
933     TRT_NODISCARD virtual bool setQueryForm(AttentionIOForm form) noexcept = 0;
934     TRT_NODISCARD virtual AttentionIOForm getQueryForm() const noexcept = 0;
935     TRT_NODISCARD virtual bool setKeyValueForm(AttentionIOForm form) noexcept = 0;
936     TRT_NODISCARD virtual AttentionIOForm getKeyValueForm() const noexcept = 0;
937     TRT_NODISCARD virtual bool setQueryLengths(ITensor* lengths) noexcept = 0;
938     TRT_NODISCARD virtual ITensor* getQueryLengths() const noexcept = 0;
939     TRT_NODISCARD virtual bool setKeyValueLengths(ITensor* lengths) noexcept = 0;
940     TRT_NODISCARD virtual ITensor* getKeyValueLengths() const noexcept = 0;
941 }; // class VAttention
942
943 class VSelectLayer :  public VRoot
944 {
945 };
946
947 class VAssertionLayer :  public VRoot
948 {
949 public:
950     virtual void setMessage(char const* message) noexcept = 0;
951     virtual char const* getMessage() const noexcept = 0;
952 };
953
954 class VFillLayer :  public VRoot
955 {
956 public:
957     virtual void setDimensions(Dims const& dimensions) noexcept = 0;
958     virtual Dims getDimensions() const noexcept = 0;
959     virtual void setOperation(FillOperation op) noexcept = 0;
960     virtual FillOperation getOperation() const noexcept = 0;
961     virtual void setAlpha(double alpha) noexcept = 0;
962     virtual double getAlpha() const noexcept = 0;
963     virtual void setBeta(double beta) noexcept = 0;
964     virtual double getBeta() const noexcept = 0;
965     virtual void setAlphaInt64(int64_t alpha) noexcept = 0;
966     virtual int64_t getAlphaInt64() const noexcept = 0;
967     virtual void setBetaInt64(int64_t beta) noexcept = 0;
968     virtual int64_t getBetaInt64() const noexcept = 0;
969     virtual bool isAlphaBetaInt64() const noexcept = 0;
970     virtual DataType getToType() const noexcept = 0;
971     virtual void setToType(DataType toType) noexcept = 0;
972 };
973
974 class VQuantizeLayer :  public VRoot
975 {
976 public:
977     virtual int32_t getAxis() const noexcept = 0;
978     virtual void setAxis(int32_t axis) noexcept = 0;
979     virtual DataType getToType() const noexcept = 0;
980     virtual void setToType(DataType toType) noexcept = 0;
981     virtual Dims getBlockShape() const noexcept = 0;
982     virtual bool setBlockShape(Dims const& blockShape) noexcept = 0;
983 };
984
985 class VDequantizeLayer :  public VRoot
986 {
987 public:
988     virtual int32_t getAxis() const noexcept = 0;
989     virtual void setAxis(int32_t axis) noexcept = 0;
990     virtual DataType getToType() const noexcept = 0;
991     virtual void setToType(DataType toType) noexcept = 0;
992     virtual Dims getBlockShape() const noexcept = 0;
993     virtual bool setBlockShape(Dims const& blockShape) noexcept = 0;
994 };
995
996 class VDynamicQuantizeLayer :  public VRoot
997 {
998 public:
999     TRT_DEPRECATED virtual int32_t getAxis() const noexcept = 0;
1000     TRT_DEPRECATED virtual void setAxis(int32_t axis) noexcept = 0;
1001     TRT_DEPRECATED virtual int32_t getBlockSize() const noexcept = 0;
1002     TRT_DEPRECATED virtual void setBlockSize(int32_t axis) noexcept = 0;
```

```
1003     virtual DataType getScaleType() const noexcept = 0;
1004     virtual void setScaleType(DataType axis) noexcept = 0;
1005     virtual DataType getToType() const noexcept = 0;
1006     virtual void setToType(DataType toType) noexcept = 0;
1007     virtual Dims getBlockShape() const noexcept = 0;
1008     virtual void setBlockShape(Dims const& blockShape) noexcept = 0;
1009 };
1010
1011 class VScatterLayer :  public VRoot
1012 {
1013 public:
1014    virtual void setMode(ScatterMode mode) noexcept = 0;
1015    virtual ScatterMode getMode() const noexcept = 0;
1016    virtual void setAxis(int32_t axis) noexcept = 0;
1017    virtual int32_t getAxis() const noexcept = 0;
1018 }; // class VScatterLayer
1019
1020 class VEinsumLayer :  public VRoot
1021 {
1022 public:
1023     virtual bool setEquation(char const* equation) noexcept = 0;
1024     virtual char const* getEquation() const noexcept = 0;
1025 };
1026
1027 class VOneHotLayer :  public VRoot
1028 {
1029 public:
1030     virtual int32_t getAxis() const noexcept = 0;
1031     virtual void setAxis(int32_t axis) noexcept = 0;
1032 }; // class VOneHotLayer
1033
1034 class VGridSampleLayer :  public VRoot
1035 {
1036 public:
1037     virtual void setInterpolationMode(InterpolationMode mode) noexcept = 0;
1038     virtual InterpolationMode getInterpolationMode() const noexcept = 0;
1039     virtual void setAlignCorners(bool alignCorners) noexcept = 0;
1040     virtual bool getAlignCorners() const noexcept = 0;
1041     virtual bool setSampleMode(SampleMode mode) noexcept = 0;
1042     virtual SampleMode getSampleMode() const noexcept = 0;
1043 }; // class VGridSampleLayer
1044
1045 class VNMSLayer :  public VRoot
1046 {
1047 public:
1048     virtual void setBoundingBoxFormat(BoundingBoxFormat fmt) noexcept = 0;
1049     virtual BoundingBoxFormat getBoundingBoxFormat() const noexcept = 0;
1050     virtual void setTopKBoxLimit(int32_t limit) noexcept = 0;
1051     virtual int32_t getTopKBoxLimit() const noexcept = 0;
1052     virtual bool setIndicesType(DataType type) noexcept = 0;
1053     virtual DataType getIndicesType() const noexcept = 0;
1054 }; // class VNMSLayer
1055
1056 class VReverseSequenceLayer :  public VRoot
1057 {
1058 public:
1059     virtual void setBatchAxis(int32_t batchAxis) noexcept = 0;
1060     virtual int32_t getBatchAxis() const noexcept = 0;
1061
1062     virtual void setSequenceAxis(int32_t sequenceAxis) noexcept = 0;
1063     virtual int32_t getSequenceAxis() const noexcept = 0;
1064 }; // class VReverseSequenceLayer
1065
1066 class VNormalizationLayer :  public VRoot
1067 {
1068 public:
1069     virtual void setEpsilon(float eps) noexcept = 0;
1070     virtual float getEpsilon() const noexcept = 0;
1071     virtual void setAxes(uint32_t axesMask) noexcept = 0;
1072     virtual uint32_t getAxes() const noexcept = 0;
1073     virtual void setNbGroups(int64_t nbGroups) noexcept = 0;
1074     virtual int64_t getNbGroups() const noexcept = 0;
1075     virtual bool isV2() const noexcept = 0;
1076 }; // class VNormalizationLayer
1077
1078 class VSqueezeLayer :  public VRoot
1079 {
1080 };
1081
1082 class VUnsqueezeLayer :  public VRoot
1083 {
```

```
1084 };
1085
1086 class VCumulativeLayer :  public VRoot
1087 {
1088 public:
1089     virtual bool setOperation(CumulativeOperation op) noexcept = 0;
1090     virtual CumulativeOperation getOperation() const noexcept = 0;
1091     virtual void setExclusive(bool exclusive) noexcept = 0;
1092     virtual bool getExclusive() const noexcept = 0;
1093     virtual void setReverse(bool reverse) noexcept = 0;
1094     virtual bool getReverse() const noexcept = 0;
1095 }; // class VCumulativeLayer
1096
1097 class VRotaryEmbeddingLayer :  public VRoot
1098 {
1099 public:
1100     virtual void setInterleaved(bool interleaved) noexcept = 0;
1101     virtual bool getInterleaved() const noexcept = 0;
1102     virtual bool setRotaryEmbeddingDim(int32_t rotaryEmbeddingDim) noexcept = 0;
1103     virtual int32_t getRotaryEmbeddingDim() const noexcept = 0;
1104     virtual void setInput(int32_t index, ITensor& input) noexcept = 0;
1105 }; // class VRotaryEmbeddingLayer
1106
1107 class VKVCacheUpdateLayer :  public VRoot
1108 {
1109 public:
1110     TRT_NODISCARD virtual bool setCacheMode(KVCacheMode cacheMode) noexcept = 0;
1111     TRT_NODISCARD virtual KVCacheMode getCacheMode() const noexcept = 0;
1112     TRT_NODISCARD virtual bool setUpdateForm(AttentionIOForm form) noexcept = 0;
1113     TRT_NODISCARD virtual AttentionIOForm getUpdateForm() const noexcept = 0;
1114     TRT_NODISCARD virtual bool setUpdateLengths(ITensor* lengths) noexcept = 0;
1115     TRT_NODISCARD virtual ITensor* getUpdateLengths() const noexcept = 0;
1116 }; // class VKVCacheUpdateLayer
1117
1118 class VMoELayer :  public VRoot
1119 {
1120 public:
1121     virtual void setGatedWeights(ITensor& fcGateWeights, ITensor& fcUpWeights, ITensor& fcDownWeights,
      MoEActType activationType) noexcept = 0;
1122     virtual void setGatedBiases(ITensor& fcGateBiases, ITensor& fcUpBiases, ITensor& fcDownBiases) noexcept
      = 0;
1123     virtual void setActivationType(MoEActType activationType) noexcept = 0;
1124     virtual MoEActType getActivationType() const noexcept = 0;
1125     virtual void setQuantizationStatic(ITensor& fcDownActivationScale, DataType dataType) noexcept = 0;
1126     virtual void setQuantizationDynamicDblQ(ITensor& fcDownActivationDblQScale, DataType dataType, Dims
      const& blockShape, DataType dynQOutputScaleType) noexcept = 0;
1127     virtual void setQuantizationToType(DataType type) noexcept = 0;
1128     virtual DataType getQuantizationToType() const noexcept = 0;
1129     virtual void setQuantizationBlockShape(Dims const& blockShape) noexcept = 0;
1130     virtual Dims getQuantizationBlockShape() const noexcept = 0;
1131     virtual void setDynQOutputScaleType(DataType type) noexcept = 0;
1132     virtual DataType getDynQOutputScaleType() const noexcept = 0;
1133     virtual void setSwigluParams(float limit, float alpha, float beta) noexcept = 0;
1134     virtual void setSwigluParamLimit(float limit) noexcept = 0;
1135     virtual float getSwigluParamLimit() const noexcept = 0;
1136     virtual void setSwigluParamAlpha(float alpha) noexcept = 0;
1137     virtual float getSwigluParamAlpha() const noexcept = 0;
1138     virtual void setSwigluParamBeta(float beta) noexcept = 0;
1139     virtual float getSwigluParamBeta() const noexcept = 0;
1140     virtual void setInput(int32_t index, ITensor& tensor) noexcept = 0;
1141 }; // class VMoELayer
1142
1143
1144 class VNetworkDefinition :  public VRoot
1145 {
1146 public:
1147     virtual ITensor* addInput(char const* name, DataType type, Dims const& dimensions) noexcept = 0;
1148     virtual void markOutput(ITensor& tensor) noexcept = 0;
1149     virtual IActivationLayer* addActivation(ITensor& input, ActivationType type) noexcept = 0;
1150     virtual ILRNLayer* addLRN(ITensor& input, int64_t window, float alpha, float beta, float k) noexcept =
      0;
1151     virtual IScaleLayer* addScale(
1152         ITensor& input, ScaleMode mode, Weights shift, Weights scale, Weights power) noexcept = 0;
1153     virtual ISoftMaxLayer* addSoftMax(ITensor& input) noexcept = 0;
1154     virtual IConcatenationLayer* addConcatenation(ITensor* const* inputs, int32_t nbInputs) noexcept = 0;
1155     virtual IElementWiseLayer* addElementWise(ITensor& input1, ITensor& input2, ElementWiseOperation op)
      noexcept = 0;
1156     virtual IUnaryLayer* addUnary(ITensor& input, UnaryOperation operation) noexcept = 0;
1157     virtual IShuffleLayer* addShuffle(ITensor& input) noexcept = 0;
1158     virtual int32_t getNbLayers() const noexcept = 0;
1159     virtual ILayer* getLayer(int32_t index) const noexcept = 0;
```

```
1160     virtual int32_t getNbInputs() const noexcept = 0;
1161     virtual ITensor* getInput(int32_t index) const noexcept = 0;
1162     virtual int32_t getNbOutputs() const noexcept = 0;
1163     virtual ITensor* getOutput(int32_t index) const noexcept = 0;
1164     virtual IReduceLayer* addReduce(
1165         ITensor& input, ReduceOperation operation, uint32_t reduceAxes, bool keepDimensions) noexcept
1166         = 0;
1167     virtual ITopKLayer* addTopK(ITensor& input, TopKOperation op, int32_t k, uint32_t reduceAxes) noexcept =
     0;
1168     virtual IGatherLayer* addGather(ITensor& data, ITensor& indices, int32_t axis) noexcept = 0;
1169     virtual IRaggedSoftMaxLayer* addRaggedSoftMax(ITensor& input, ITensor& bounds) noexcept = 0;
1170     virtual IMatrixMultiplyLayer* addMatrixMultiply(
1171         ITensor& input0, MatrixOperation op0, ITensor& input1, MatrixOperation op1) noexcept = 0;
1172     virtual IConstantLayer* addConstant(Dims const& dimensions, Weights weights) noexcept = 0;
1173     virtual IIdentityLayer* addIdentity(ITensor& input) noexcept = 0;
1174     virtual void removeTensor(ITensor& tensor) noexcept = 0;
1175     virtual void unmarkOutput(ITensor& tensor) noexcept = 0;
1176     virtual IPluginV2Layer* addPluginV2(ITensor* const* inputs, int32_t nbInputs, IPluginV2& plugin)
     noexcept = 0;
1177     virtual IPluginV3Layer* addPluginV3(ITensor* const* inputs, int32_t nbInputs, ITensor* const*
     shapeInputs,
1178         int32_t nbShapeInputs, IPluginV3& plugin) noexcept = 0;
1179     virtual ISliceLayer* addSlice(ITensor& input, Dims const& start, Dims const& size, Dims const& stride)
     noexcept = 0;
1180     virtual void setName(char const* name) noexcept = 0;
1181     virtual char const* getName() const noexcept = 0;
1182     virtual IShapeLayer* addShape(ITensor& input) noexcept = 0;
1183     virtual bool hasImplicitBatchDimension() const noexcept = 0;
1184     virtual bool markOutputForShapes(ITensor& tensor) noexcept = 0;
1185     virtual bool unmarkOutputForShapes(ITensor& tensor) noexcept = 0;
1186     virtual IParametricReLULayer* addParametricReLU(ITensor& input, ITensor& slope) noexcept = 0;
1187     virtual IConvolutionLayer* addConvolutionNd(
1188         ITensor& input, int64_t nbOutputMaps, Dims const& kernelSize, Weights kernelWeights, Weights
     biasWeights) noexcept
1189         = 0;
1190     virtual IPoolingLayer* addPoolingNd(ITensor& input, PoolingType type, Dims const& windowSize) noexcept
     = 0;
1191     virtual IDeconvolutionLayer* addDeconvolutionNd(
1192         ITensor& input, int64_t nbOutputMaps, Dims const& kernelSize, Weights kernelWeights, Weights
     biasWeights) noexcept
1193         = 0;
1194     virtual IScaleLayer* addScaleNd(
1195         ITensor& input, ScaleMode mode, Weights shift, Weights scale, Weights power, int32_t channelAxis)
     noexcept = 0;
1196     virtual IResizeLayer* addResize(ITensor& input) noexcept = 0;
1197     virtual ILoop* addLoop() noexcept = 0;
1198     virtual ISelectLayer* addSelect(ITensor& condition, ITensor& thenInput, ITensor& elseInput) noexcept =
     0;
1199     virtual IPaddingLayer* addPaddingNd(ITensor& input, Dims const& prePadding, Dims const& postPadding)
     noexcept = 0;
1200     virtual bool setWeightsName(Weights weights, char const* name) noexcept = 0;
1201     virtual void setErrorRecorder(IErrorRecorder* recorder) noexcept = 0;
1202     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
1203     virtual IGatherLayer* addGatherV2(ITensor& data, ITensor& indices, GatherMode mode) noexcept = 0;
1204     virtual IIfConditional* addIfConditional() noexcept = 0;
1205     virtual IScatterLayer* addScatter(ITensor& data, ITensor& indices, ITensor& updates, ScatterMode mode)
     noexcept = 0;
1206     virtual IEinsumLayer* addEinsum(ITensor* const* inputs, int32_t nbInputs, char const* equation)
     noexcept = 0;
1207     virtual IAssertionLayer* addAssertion(ITensor& condition, char const* message) noexcept = 0;
1208     virtual IOneHotLayer* addOneHot(ITensor& indices, ITensor& values, ITensor& depth, int32_t axis)
     noexcept = 0;
1209     virtual INonZeroLayer* addNonZero(ITensor& input) noexcept = 0;
1210     virtual IGridSampleLayer* addGridSample(ITensor& input, ITensor& grid) noexcept = 0;
1211     virtual INMSLayer* addNMS(ITensor& boxes, ITensor& scores, ITensor& maxOutputBoxesPerClass) noexcept =
     0;
1212     virtual IReverseSequenceLayer* addReverseSequence(ITensor& input, ITensor& sequenceLens) noexcept = 0;
1213     virtual INormalizationLayer* addNormalization(
1214         ITensor& input, ITensor& scale, ITensor& bias, uint32_t axesMask) noexcept = 0;
1215     virtual ICastLayer* addCast(ITensor& input, DataType toType) noexcept = 0;
1216     virtual IBuilder& getBuilder() const noexcept = 0;
1217     virtual NetworkDefinitionCreationFlags getFlags() const noexcept = 0;
1218     virtual bool getFlag(NetworkDefinitionCreationFlag networkDefinitionCreationFlag) const noexcept = 0;
1219     virtual IQuantizeLayer* addQuantizeV2(ITensor& input, ITensor& scale, DataType outputType) noexcept =
     0;
1220     virtual IDequantizeLayer* addDequantizeV2(ITensor& input, ITensor& scale, DataType outputType) noexcept
     = 0;
1221     virtual IFillLayer* addFillV2(Dims const& dimensions, FillOperation op, DataType outputType) noexcept =
     0;
1222     virtual bool markDebug(ITensor& tensor) noexcept = 0;
1223     virtual bool unmarkDebug(ITensor& tensor) noexcept = 0;
```

```
1224     virtual bool isDebugTensor(ITensor const& tensor) const noexcept = 0;
1225     virtual bool markWeightsRefittable(char const* name) noexcept = 0;
1226     virtual bool unmarkWeightsRefittable(char const* name) noexcept = 0;
1227     virtual bool areWeightsMarkedRefittable(char const* name) const noexcept = 0;
1228     virtual ISqueezeLayer* addSqueeze(ITensor& input, ITensor& axes) noexcept = 0;
1229     virtual IUnsqueezeLayer* addUnsqueeze(ITensor& input, ITensor& axes) noexcept = 0;
1230     virtual IDynamicQuantizeLayer* addDynamicQuantize(
1231         ITensor& input, int32_t axis, int32_t blockSize, DataType toType, DataType scaleType) noexcept = 0;
1232     virtual ICumulativeLayer* addCumulative(
1233         ITensor& input, ITensor& axis, CumulativeOperation operation, bool exclusive, bool reverse)
     noexcept = 0;
1234     virtual bool markUnfusedTensorsAsDebugTensors() noexcept = 0;
1235     virtual bool unmarkUnfusedTensorsAsDebugTensors() noexcept = 0;
1236     virtual ITopKLayer* addTopKV2(
1237         ITensor& input, TopKOperation op, int32_t k, uint32_t reduceAxes, DataType indicesType) noexcept =
     0;
1238     virtual INonZeroLayer* addNonZeroV2(ITensor& input, DataType indicesType) noexcept = 0;
1239     virtual INMSLayer* addNMSV2(
1240         ITensor& boxes, ITensor& scores, ITensor& maxOutputBoxesPerClass, DataType indicesType) noexcept =
     0;
1241     TRT_DEPRECATED virtual IAttention* addAttention(
1242         ITensor& query, ITensor& key, ITensor& value, AttentionNormalizationOp normOp, bool isCausal)
     noexcept = 0;
1243     virtual IRotaryEmbeddingLayer* addRotaryEmbedding(ITensor& input, ITensor& cosCache, ITensor& sinCache,
1244         bool interleaved, int32_t rotaryEmbeddingDim) noexcept = 0;
1245     virtual IDynamicQuantizeLayer* addDynamicQuantizeV2(
1246         ITensor& input, Dims const& blockShape, DataType toType, DataType scaleType) noexcept = 0;
1247     virtual IKVCacheUpdateLayer* addKVCacheUpdate(
1248         ITensor& cache, ITensor& update, ITensor& writeIndices, KVCacheMode cacheMode) noexcept = 0;
1249     virtual INormalizationLayer* addNormalizationV2(
1250         ITensor& input, ITensor& scale, ITensor& bias, uint32_t axesMask) noexcept = 0;
1251     virtual IMoELayer* addMoE(
1252         ITensor& hiddenStates, ITensor& selectedExpertsForTokens, ITensor& scoresForSelectedExperts)
     noexcept = 0;
1253     virtual IDistCollectiveLayer* addDistCollective(ITensor& input, CollectiveOperation distCollectiveOp,
1254         ReduceOperation reduceOp, int64_t root, int64_t* groups, int64_t groupSize) noexcept = 0;
1255     virtual IAttention* addAttentionV2(ITensor& query, ITensor& key, ITensor& value,
     AttentionNormalizationOp normOp,
1256         CausalMaskKind causalKind) noexcept = 0;
1257 };
1258
1259 class VTimingCache :  public VRoot
1260 {
1261 public:
1262     virtual nvinfer1::IHostMemory* serialize() const noexcept = 0;
1263     virtual bool combine(ITimingCache const& inputCache, bool ignoreMismatch) noexcept = 0;
1264     virtual bool reset() noexcept = 0;
1265     virtual int64_t queryKeys(TimingCacheKey* keyBuffer, int64_t capacity) const noexcept = 0;
1266     virtual TimingCacheValue query(TimingCacheKey const& key) const noexcept = 0;
1267     virtual bool update(TimingCacheKey const& key, TimingCacheValue const& value) noexcept = 0;
1268 };
1269
1270 class VBuilderConfig :  public VRoot
1271 {
1272 public:
1273     virtual void setAvgTimingIterations(int32_t avgTiming) noexcept = 0;
1274     virtual int32_t getAvgTimingIterations() const noexcept = 0;
1275     virtual void setEngineCapability(EngineCapability capability) noexcept = 0;
1276     virtual EngineCapability getEngineCapability() const noexcept = 0;
1277     virtual void setFlags(BuilderFlags builderFlags) noexcept = 0;
1278     virtual BuilderFlags getFlags() const noexcept = 0;
1279     virtual void clearFlag(BuilderFlag builderFlag) noexcept = 0;
1280     virtual void setFlag(BuilderFlag builderFlag) noexcept = 0;
1281     virtual bool getFlag(BuilderFlag builderFlag) const noexcept = 0;
1282     virtual void setDeviceType(ILayer const* layer, DeviceType deviceType) noexcept = 0;
1283     virtual DeviceType getDeviceType(ILayer const* layer) const noexcept = 0;
1284     virtual bool isDeviceTypeSet(ILayer const* layer) const noexcept = 0;
1285     virtual void resetDeviceType(ILayer const* layer) noexcept = 0;
1286     virtual bool canRunOnDLA(ILayer const* layer) const noexcept = 0;
1287     virtual void setDLACore(int32_t dlaCore) noexcept = 0;
1288     virtual int32_t getDLACore() const noexcept = 0;
1289     virtual void setDefaultDeviceType(DeviceType deviceType) noexcept = 0;
1290     virtual DeviceType getDefaultDeviceType() const noexcept = 0;
1291     virtual void reset() noexcept = 0;
1292     virtual void setProfileStream(const cudaStream_t stream) noexcept = 0;
1293     virtual cudaStream_t getProfileStream() const noexcept = 0;
1294     virtual int32_t addOptimizationProfile(IOptimizationProfile const* profile) noexcept = 0;
1295     virtual int32_t getNbOptimizationProfiles() const noexcept = 0;
1296     virtual void setProfilingVerbosity(ProfilingVerbosity verbosity) noexcept = 0;
1297     virtual ProfilingVerbosity getProfilingVerbosity() const noexcept = 0;
1298     virtual bool setTacticSources(TacticSources tacticSources) noexcept = 0;
```

```
1299     virtual TacticSources getTacticSources() const noexcept = 0;
1300     virtual nvinfer1::ITimingCache* createTimingCache(void const* blob, std::size_t size) const noexcept =
     0;
1301     virtual bool setTimingCache(ITimingCache const& cache, bool ignoreMismatch) noexcept = 0;
1302     virtual nvinfer1::ITimingCache const* getTimingCache() const noexcept = 0;
1303     virtual void setMemoryPoolLimit(MemoryPoolType pool, std::size_t poolSize) noexcept = 0;
1304     virtual std::size_t getMemoryPoolLimit(MemoryPoolType pool) const noexcept = 0;
1305     virtual void setPreviewFeature(PreviewFeature feature, bool enable) noexcept = 0;
1306     virtual bool getPreviewFeature(PreviewFeature feature) const noexcept = 0;
1307     virtual void setBuilderOptimizationLevel(int32_t level) noexcept = 0;
1308     virtual int32_t getBuilderOptimizationLevel() const noexcept = 0;
1309     virtual void setHardwareCompatibilityLevel(HardwareCompatibilityLevel hardwareCompatibilityLevel)
     noexcept = 0;
1310     virtual HardwareCompatibilityLevel getHardwareCompatibilityLevel() const noexcept = 0;
1311     virtual void setPluginsToSerialize(char const* const* paths, int32_t nbPaths) noexcept = 0;
1312     virtual char const* getPluginToSerialize(int32_t index) const noexcept = 0;
1313     virtual int32_t getNbPluginsToSerialize() const noexcept = 0;
1314     virtual bool setMaxAuxStreams(int32_t nbStreams) noexcept = 0;
1315     virtual int32_t getMaxAuxStreams() const noexcept = 0;
1316     virtual void setProgressMonitor(IProgressMonitor* monitor) noexcept = 0;
1317     virtual IProgressMonitor* getProgressMonitor() const noexcept = 0;
1318     virtual void setRuntimePlatform(RuntimePlatform runtimePlatform) noexcept = 0;
1319     virtual RuntimePlatform getRuntimePlatform() const noexcept = 0;
1320     virtual void setMaxNbTactics(int32_t maxTactics) noexcept = 0;
1321     virtual int32_t getMaxNbTactics() const noexcept = 0;
1322     virtual bool setTilingOptimizationLevel(TilingOptimizationLevel level) noexcept = 0;
1323     virtual TilingOptimizationLevel getTilingOptimizationLevel() const noexcept = 0;
1324     virtual bool setL2LimitForTiling(int64_t size) noexcept = 0;
1325     virtual int64_t getL2LimitForTiling() const noexcept = 0;
1326     virtual bool setRemoteAutoTuningConfig(char const* config) noexcept = 0;
1327     virtual char const* getRemoteAutoTuningConfig() const noexcept = 0;
1328 };
1329
1330 class VSerializationConfig :  public VRoot
1331 {
1332 public:
1333     virtual bool setFlags(SerializationFlags serializationFlags) noexcept = 0;
1334     virtual SerializationFlags getFlags() const noexcept = 0;
1335     virtual bool clearFlag(SerializationFlag serializationFlag) noexcept = 0;
1336     virtual bool setFlag(SerializationFlag serializationFlag) noexcept = 0;
1337     virtual bool getFlag(SerializationFlag serializationFlag) const noexcept = 0;
1338 };
1339
1340 class VBuilder :  public VRoot
1341 {
1342 public:
1343     virtual int32_t getMaxDLABatchSize() const noexcept = 0;
1344     virtual int32_t getNbDLACores() const noexcept = 0;
1345     virtual void setGpuAllocator(IGpuAllocator* allocator) noexcept = 0;
1346     virtual nvinfer1::IBuilderConfig* createBuilderConfig() noexcept = 0;
1347     virtual nvinfer1::INetworkDefinition* createNetworkV2(NetworkDefinitionCreationFlags flags) noexcept =
     0;
1348     virtual nvinfer1::IOptimizationProfile* createOptimizationProfile() noexcept = 0;
1349     virtual void setErrorRecorder(IErrorRecorder* recorder) noexcept = 0;
1350     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
1351     virtual void reset() noexcept = 0;
1352     virtual nvinfer1::IHostMemory* buildSerializedNetwork(
1353         INetworkDefinition& network, IBuilderConfig& config) noexcept = 0;
1354     virtual bool isNetworkSupported(INetworkDefinition const& network, IBuilderConfig const& config) const
     noexcept = 0;
1355     virtual ILogger* getLogger() const noexcept = 0;
1356     virtual bool setMaxThreads(int32_t maxThreads) noexcept = 0;
1357     virtual int32_t getMaxThreads() const noexcept = 0;
1358     virtual IPluginRegistry& getPluginRegistry() noexcept = 0;
1359     virtual ICudaEngine* buildEngineWithConfig(INetworkDefinition& network, IBuilderConfig& config)
     noexcept = 0;
1360     virtual bool buildSerializedNetworkToStream(
1361         INetworkDefinition& network, IBuilderConfig& config, IStreamWriter& writer) noexcept = 0;
1362     virtual nvinfer1::IHostMemory* buildSerializedNetworkWithKernelText(
1363         INetworkDefinition& network, IBuilderConfig& config, IHostMemory*& kernelText) noexcept
1364         = 0;
1365 };
1366
1367 class VRuntimeConfig :  public VRoot
1368 {
1369 public:
1370     virtual IRuntimeConfig* getPImpl() noexcept = 0;
1371     virtual void setExecutionContextAllocationStrategy(ExecutionContextAllocationStrategy strategy)
     noexcept = 0;
1372     virtual ExecutionContextAllocationStrategy getExecutionContextAllocationStrategy() const noexcept = 0;
1373 };
```

```
1374
1375
1376 class VDistCollectiveLayer :  public VRoot
1377 {
1378 }; // VDistCollectiveLayer
1379
1380 } // namespace apiv
1381 } // namespace nvinfer1
1382
1383 // @endcond
1384
1385 #endif // NV_INFER_RUNTIME_IMPL_H
```

10.5 NvInferLegacyDims.h File Reference

```
#include "NvInferRuntimeBase.h"
```

Classes

- class nvinfer1::Dims2

  Descriptor for two-dimensional data.
- class nvinfer1::DimsHW

  Descriptor for two-dimensional spatial data.
- class nvinfer1::Dims3

  Descriptor for three-dimensional data.
- class nvinfer1::Dims4

  Descriptor for four-dimensional data.

Namespaces

- namespace nvinfer1

  The TensorRT API version 1 namespace.

Macros

- #define NV_INFER_INTERNAL_INCLUDE 1

10.5.1 Detailed Description

This file contains declarations of legacy dimensions types which use channel semantics in their names, and declarations on which those types rely.

10.5.2 Macro Definition Documentation

10.5.2.1 NV_INFER_INTERNAL_INCLUDE

```
#define NV_INFER_INTERNAL_INCLUDE 1
```

10.6 NvInferLegacyDims.h

Go to the documentation of this file.

```
1 /*
2 * SPDX-FileCopyrightText: Copyright (c) 1993-2025 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3 * SPDX-License-Identifier: Apache-2.0
4 *
5 * Licensed under the Apache License, Version 2.0 (the "License");
6 * you may not use this file except in compliance with the License.
7 * You may obtain a copy of the License at
8 *
9 * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17
18 #ifndef NV_INFER_LEGACY_DIMS_H
19 #define NV_INFER_LEGACY_DIMS_H
20
21 #define NV_INFER_INTERNAL_INCLUDE 1
22 #include "NvInferRuntimeBase.h" // IWYU pragma: exports
23 #undef NV_INFER_INTERNAL_INCLUDE
24
31
37 namespace nvinfer1
38 {
44 class Dims2 : public Dims
45 {
46 public:
50     Dims2()
51         : Dims2(0, 0)
52     {
53     }
54
61     Dims2(int64_t d0, int64_t d1)
62     {
63         nbDims = 2;
64         d[0] = d0;
65         d[1] = d1;
66         for (int64_t i{nbDims}; i < Dims::MAX_DIMS; ++i)
67         {
68             d[i] = 0;
69         }
70     }
71 };
72
78 class DimsHW : public Dims2
79 {
80 public:
84     DimsHW()
85         : Dims2()
86     {
87     }
88
95     DimsHW(int64_t height, int64_t width)
96         : Dims2(height, width)
97     {
98     }
99
105     int64_t& h()
106     {
107         return d[0];
108     }
109
115     int64_t h()const
```

```
116 {
117         return d[0];
118     }
119 
125     int64_t& w()
126     {
127         return d[1];
128     }
129 
135     int64_t w()const
136 {
137         return d[1];
138     }
139 };
140 
146 class Dims3 :  public Dims2
147 {
148 public:
152     Dims3()
153         :  Dims3(0, 0, 0)
154     {
155     }
156 
164     Dims3(int64_t d0, int64_t d1, int64_t d2)
165         :  Dims2(d0, d1)
166     {
167         nbDims = 3;
168         d[2] = d2;
169     }
170 };
171 
177 class Dims4 :  public Dims3
178 {
179 public:
183     Dims4()
184         :  Dims4(0, 0, 0, 0)
185     {
186     }
187 
196     Dims4(int64_t d0, int64_t d1, int64_t d2, int64_t d3)
197         :  Dims3(d0, d1, d2)
198     {
199         nbDims = 4;
200         d[3] = d3;
201     }
202 };
203 
204 } // namespace nvinfer1
205 
206 #endif // NV_INFER_LEGCY_DIMS_H
```

10.7 NvInferPlugin.h File Reference

```
#include "NvInfer.h"
#include "NvInferPluginUtils.h"
```

Functions

- TENSORRTAPI bool initLibNvInferPlugins (void ∗logger, char const ∗libNamespace)

  Initialize and register all the existing TensorRT plugins to the Plugin Registry with an optional namespace. The plugin library author should ensure that this function name is unique to the library. This function should be called once before accessing the Plugin Registry.

10.7.1 Detailed Description

This is the API for the Nvidia provided TensorRT plugins.

10.7.2 Function Documentation

10.7.2.1 initLibNvInferPlugins()

```
TENSORRTAPI bool initLibNvInferPlugins (
             void * logger,
             char const * libNamespace )
```

Initialize and register all the existing TensorRT plugins to the Plugin Registry with an optional namespace. The plugin library author should ensure that this function name is unique to the library. This function should be called once before accessing the Plugin Registry.

Parameters

logger	Logger object to print plugin registration information
libNamespace	Namespace used to register all the plugins in this library

10.8 NvInferPlugin.h

Go to the documentation of this file.

```
1 /*
2 * SPDX-FileCopyrightText:  Copyright (c) 1993-2024 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3 * SPDX-License-Identifier:  Apache-2.0
4 *
5 * Licensed under the Apache License, Version 2.0 (the "License");
6 * you may not use this file except in compliance with the License.
7 * You may obtain a copy of the License at
8 *
9 * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17
18 #ifndef NV_INFER_PLUGIN_H
19 #define NV_INFER_PLUGIN_H
20
21 #include "NvInfer.h"
22 #include "NvInferPluginUtils.h"
28
29 extern "C"
30 {
38     TENSORRTAPI bool initLibNvInferPlugins(void* logger, char const* libNamespace);
39
40 } // extern "C"
41
42 #endif // NV_INFER_PLUGIN_H
```

10.9 NvInferPluginBase.h File Reference

```
#include "NvInferRuntimeBase.h"
```

Classes

- class nvinfer1::PluginField

  Structure containing plugin attribute field names and associated data This information can be parsed to decode necessary plugin metadata.
- struct nvinfer1::PluginFieldCollection

  Plugin field collection struct.
- class nvinfer1::v_1_0::IPluginCapability
- class nvinfer1::v_1_0::IPluginResource
- class nvinfer1::v_1_0::IPluginCreatorInterface
- class nvinfer1::v_1_0::IPluginV3

Namespaces

- namespace nvinfer1

  The TensorRT API version 1 namespace.
- namespace nvinfer1::v_1_0

  Forward declare IErrorRecorder and ILogger for use in other interfaces.

Macros

- #define NV_INFER_INTERNAL_INCLUDE 1

Typedefs

- using nvinfer1::IPluginResource = v_1_0::IPluginResource
- using nvinfer1::IPluginCreatorInterface = v_1_0::IPluginCreatorInterface
- using nvinfer1::IPluginV3 = v_1_0::IPluginV3
- using nvinfer1::IPluginCapability = v_1_0::IPluginCapability

Enumerations

- enum class nvinfer1::PluginFieldType : int32_t { nvinfer1::kFLOAT16 = 0 , nvinfer1::kFLOAT32 = 1 , nvinfer1::kFLOAT64 = 2 , nvinfer1::kINT8 = 3 , nvinfer1::kINT16 = 4 , nvinfer1::kINT32 = 5 , nvinfer1::kCHAR = 6 , nvinfer1::kDIMS = 7 , nvinfer1::kUNKNOWN = 8 , nvinfer1::kBF16 = 9 , nvinfer1::kINT64 = 10 , nvinfer1::kFP8 = 11 , nvinfer1::kINT4 = 12 , nvinfer1::kFP4 = 13 }

  The possible field types for custom layer.
- enum class nvinfer1::TensorRTPhase : int32_t { nvinfer1::kBUILD = 0 , nvinfer1::kRUNTIME = 1 }

  Indicates a phase of operation of TensorRT.
- enum class nvinfer1::PluginCapabilityType : int32_t { nvinfer1::kCORE = 0 , nvinfer1::kBUILD = 1 , nvinfer1::kRUNTIME = 2 }

  Enumerates the different capability types a IPluginV3 object may have.

10.9.1 Macro Definition Documentation

10.9.1.1 NV_INFER_INTERNAL_INCLUDE

```
#define NV_INFER_INTERNAL_INCLUDE 1
```

10.10 NvInferPluginBase.h

Go to the documentation of this file.

```
1 /*
2 * SPDX-FileCopyrightText: Copyright (c) 2024-2026 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3 * SPDX-License-Identifier: Apache-2.0
4 *
5 * Licensed under the Apache License, Version 2.0 (the "License");
6 * you may not use this file except in compliance with the License.
7 * You may obtain a copy of the License at
8 *
9 * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17
18 #ifndef NV_INFER_PLUGIN_BASE_H
19 #define NV_INFER_PLUGIN_BASE_H
20
21 #if !defined(NV_INFER_INTERNAL_INCLUDE)
22 static_assert(false, "Do not directly include this file. Include NvInferRuntime.h or NvInferPluginUtils.h");
23 #endif
24
25 #define NV_INFER_INTERNAL_INCLUDE 1
26 #include "NvInferRuntimeBase.h" // IWYU pragma: exports
27 #undef NV_INFER_INTERNAL_INCLUDE
28 namespace nvinfer1
29 {
30
36 enum class PluginFieldType : int32_t
37 {
39     kFLOAT16 = 0,
41     kFLOAT32 = 1,
43     kFLOAT64 = 2,
45     kINT8 = 3,
47     kINT16 = 4,
49     kINT32 = 5,
51     kCHAR = 6,
53     kDIMS = 7,
55     kUNKNOWN = 8,
57     kBF16 = 9,
59     kINT64 = 10,
61     kFP8 = 11,
63     kINT4 = 12,
65     kFP4 = 13,
66 };
67
75 class PluginField
76 {
77 public:
79     AsciiChar const* name;
81     void const* data;
83     PluginFieldType type;
85     int32_t length;
86
87     PluginField(AsciiChar const* const name_ = nullptr, void const* const data_ = nullptr,
88         PluginFieldType const type_ = PluginFieldType::kUNKNOWN, int32_t const length_ = 0) noexcept
```

```
89          :  name(name_)
90          , data(data_)
91          , type(type_)
92          , length(length_)
93      {
94      }
95 };
96
102 struct PluginFieldCollection
103 {
105     int32_t nbFields{};
107     PluginField const* fields{};
108 };
109
115 enum class TensorRTPhase :  int32_t
116 {
118     kBUILD = 0,
120     kRUNTIME = 1
121 };
122
128 enum class PluginCapabilityType :  int32_t
129 {
131     kCORE = 0,
133     kBUILD = 1,
135     kRUNTIME = 2
136 };
137
138 namespace v_1_0
139 {
140 class IPluginCapability :  public IVersionedInterface
141 {
142 };
143
144 class IPluginResource :  public IVersionedInterface
145 {
146 public:
150     InterfaceInfo getInterfaceInfo() const noexceptoverride
151 {
152         return InterfaceInfo{"IPluginResource", 1, 0};
153     }
167     virtual int32_t release() noexcept = 0;
168
181     virtual IPluginResource* clone() noexcept = 0;
182
183     ~IPluginResource() noexcept override = default;
184
185     IPluginResource() = default;
186     IPluginResource(IPluginResource const&) = default;
187     IPluginResource(IPluginResource&&) = default;
188     IPluginResource& operator=(IPluginResource const&) & = default;
189     IPluginResource& operator=(IPluginResource&&) & = default;
190 }; // class IPluginResource
191
192 class IPluginCreatorInterface :  public IVersionedInterface
193 {
194 public:
195     ~IPluginCreatorInterface() noexcept override = default;
196
197 protected:
198     IPluginCreatorInterface() = default;
199     IPluginCreatorInterface(IPluginCreatorInterface const&) = default;
200     IPluginCreatorInterface(IPluginCreatorInterface&&) = default;
201     IPluginCreatorInterface& operator=(IPluginCreatorInterface const&) & = default;
202     IPluginCreatorInterface& operator=(IPluginCreatorInterface&&) & = default;
203 };
204
205 class IPluginV3 :  public IVersionedInterface
206 {
207 public:
211     InterfaceInfo getInterfaceInfo() const noexceptoverride
212 {
213         return InterfaceInfo{"PLUGIN", 1, 0};
214     }
215
227     virtual IPluginCapability* getCapabilityInterface(PluginCapabilityType type) noexcept = 0;
228
239     virtual IPluginV3* clone() noexcept = 0;
240 };
241 } // namespace v_1_0
242
251 using IPluginResource = v_1_0::IPluginResource;
```

```
252
260 using IPluginCreatorInterface = v_1_0::IPluginCreatorInterface;
261
273 using IPluginV3 = v_1_0::IPluginV3;
274
288 using IPluginCapability = v_1_0::IPluginCapability;
289 } // namespace nvinfer1
290
291 #endif /* NV_INFER_PLUGIN_BASE_H */
```

10.11 NvInferPluginUtils.h File Reference

```
#include "NvInferRuntimeCommon.h"
```

Classes

- struct nvinfer1::plugin::PriorBoxParameters

  The PriorBox plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions (H x W).
- struct nvinfer1::plugin::RPROIParams

  RPROIParams is used to create the RPROIPlugin instance.
- struct nvinfer1::plugin::GridAnchorParameters

  The Anchor Generator plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions (H x W). GridAnchorParameters defines a set of parameters for creating the plugin layer for all feature maps.
- struct nvinfer1::plugin::softmaxTree

  When performing yolo9000, softmaxTree is helping to do softmax on confidence scores, for element to get the precise classification through word-tree structured classification definition.
- struct nvinfer1::plugin::RegionParameters

  The Region plugin layer performs region proposal calculation.

Namespaces

- namespace nvinfer1

  The TensorRT API version 1 namespace.
- namespace nvinfer1::plugin

10.11.1 Detailed Description

This is the API for the Nvidia provided TensorRT plugin utilities. It lists all the parameters utilized by the TensorRT plugins.

10.12 NvInferPluginUtils.h

Go to the documentation of this file.

```
1 /*
2 * SPDX-FileCopyrightText:  Copyright (c) 1993-2026 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3 * SPDX-License-Identifier:  Apache-2.0
4 *
5 * Licensed under the Apache License, Version 2.0 (the "License");
6 * you may not use this file except in compliance with the License.
7 * You may obtain a copy of the License at
8 *
9 * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17
18 #ifndef NV_INFER_PLUGIN_UTILS_H
19 #define NV_INFER_PLUGIN_UTILS_H
20
21 #include "NvInferRuntimeCommon.h"
22
23
30 namespace nvinfer1
31 {
32 namespace plugin
33 {
34
43 struct PriorBoxParameters
44 {
45     float *minSize;
46     float *maxSize;
47     float *aspectRatios;
48     int32_t numMinSize;
49     int32_t numMaxSize;
50     int32_t numAspectRatios;
51     bool flip;
53     bool clip;
54     float variance[4];
55     int32_t imgH;
56     int32_t imgW;
57     float stepH;
58     float stepW;
59     float offset;
60 };
61
67 struct RPROIParams
68 {
69     int32_t poolingH;
70     int32_t poolingW;
71     int32_t featureStride;
73     int32_t preNmsTop;
74     int32_t nmsMaxOut;
75     int32_t anchorsRatioCount;
76     int32_t anchorsScaleCount;
77     float iouThreshold;
78     float minBoxSize;
79     float spatialScale;
80 };
81
88 struct GridAnchorParameters
89 {
90     float minSize;
91     float maxSize;
92     float* aspectRatios;
93     int32_t numAspectRatios;
94     int32_t H;
95     int32_t W;
96     float variance[4];
97 };
98
103 struct softmaxTree
104 {
105     int32_t* leaf;
106     int32_t n;
107     int32_t* parent;
```

```
108     int32_t* child;
109     int32_t* group;
110     char** name;
111     int32_t groups;
112     int32_t* groupSize;
113     int32_t* groupOffset;
114 };
115
124 struct RegionParameters
125 {
126     int32_t num;
127     int32_t coords;
128     int32_t classes;
129     softmaxTree* smTree;
130 };
131
132 } // namespace plugin
133 } // namespace nvinfer1
134
135 #endif // NV_INFER_PLUGIN_UTILS_H
```

10.13 NvInferRuntime.h File Reference

```
#include "NvInferImpl.h"
#include "NvInferPluginBase.h"
#include "NvInferRuntimeCommon.h"
```

Classes

- class nvinfer1::INoCopy

  Forward declaration of IEngineInspector for use by other interfaces.
- struct nvinfer1::impl::EnumMaxImpl< EngineCapability >

  Maximum number of elements in EngineCapability enum.
- class nvinfer1::Weights

  An array of weights used as a layer parameter.
- class nvinfer1::IHostMemory

  Class to handle library allocated memory that is accessible to the user.
- struct nvinfer1::impl::EnumMaxImpl< DimensionOperation >

  Maximum number of elements in DimensionOperation enum.
- struct nvinfer1::impl::EnumMaxImpl< TensorLocation >

  Maximum number of elements in TensorLocation enum.
- class nvinfer1::IDimensionExpr

  An IDimensionExpr represents an integer expression constructed from constants, input dimensions, and binary operations. These expressions are can be used in overrides of IPluginV2DynamicExt::getOutputDimensions or IPluginV3OneBuild::getOutputShapes() to define output dimensions in terms of input dimensions.
- class nvinfer1::IExprBuilder

  Object for constructing IDimensionExpr.
- class nvinfer1::DimsExprs

  Analog of class Dims with expressions instead of constants for the dimensions.
- struct nvinfer1::DynamicPluginTensorDesc

  Summarizes tensors that a plugin might see for an input or output.
- class nvinfer1::IPluginV2DynamicExt

Similar to IPluginV2Ext, but with support for dynamic shapes.

- class nvinfer1::v_1_0::IStreamReader
- class nvinfer1::v_1_0::IStreamWriter
- class nvinfer1::v_1_0::IStreamReaderV2
- class nvinfer1::IPluginResourceContext

  Interface for plugins to access per context resources provided by TensorRT.
- class nvinfer1::v_1_0::IPluginV3OneCore
- class nvinfer1::v_1_0::IPluginV3OneBuild
- class nvinfer1::v_1_0::IPluginV3OneRuntime
- class nvinfer1::v_2_0::IPluginV3OneBuild
- class nvinfer1::v_1_0::IProfiler
- struct nvinfer1::impl::EnumMaxImpl< WeightsRole >

  Maximum number of elements in WeightsRole enum.
- struct nvinfer1::impl::EnumMaxImpl< DeviceType >

  Maximum number of elements in DeviceType enum.
- struct nvinfer1::impl::EnumMaxImpl< TempfileControlFlag >

  Maximum number of elements in TempfileControlFlag enum.
- struct nvinfer1::impl::EnumMaxImpl< TensorFormat >

  Maximum number of elements in TensorFormat enum.
- struct nvinfer1::impl::EnumMaxImpl< AllocatorFlag >

  Maximum number of elements in AllocatorFlag enum.
- class nvinfer1::v_1_0::ILogger

  Application-implemented logging interface for the builder, refitter and runtime.
- struct nvinfer1::impl::EnumMaxImpl< ILogger::Severity >

  Maximum number of elements in ILogger::Severity enum.
- class nvinfer1::v_1_0::IGpuAllocator
- class nvinfer1::IRuntime

  Allows a serialized functionally unsafe engine to be deserialized.
- class nvinfer1::IRefitter

  Updates weights in an engine.
- struct nvinfer1::impl::EnumMaxImpl< OptProfileSelector >

  Number of different values of OptProfileSelector enum.
- class nvinfer1::IOptimizationProfile

  Optimization profile for dynamic input dimensions and shape tensors.
- struct nvinfer1::impl::EnumMaxImpl< TacticSource >

  Maximum number of tactic sources in TacticSource enum.
- struct nvinfer1::impl::EnumMaxImpl< ProfilingVerbosity >

  Maximum number of profile verbosity levels in ProfilingVerbosity enum.
- struct nvinfer1::impl::EnumMaxImpl< SerializationFlag >

  Maximum number of serialization flags in SerializationFlag enum.
- class nvinfer1::ISerializationConfig

  Holds properties for configuring an engine to serialize the binary.
- struct nvinfer1::impl::EnumMaxImpl< ExecutionContextAllocationStrategy >
- class nvinfer1::IRuntimeConfig

  A class for runtime configuration. This class is used during execution context creation.
- struct nvinfer1::impl::EnumMaxImpl< EngineStat >

  Maximum number of engine statistic kinds in EngineStat enum.
- class nvinfer1::ICudaEngine

An engine for executing inference on a built network, with functionally unsafe features.

- class nvinfer1::v_1_0::IOutputAllocator
- class nvinfer1::v_1_0::IDebugListener
- class nvinfer1::IExecutionContext

  Context for executing inference using an engine, with functionally unsafe features.
- struct nvinfer1::impl::EnumMaxImpl< LayerInformationFormat >

  Maximum number of layer information formats in LayerInformationFormat enum.
- class nvinfer1::IEngineInspector

  An engine inspector which prints out the layer information of an engine or an execution context.
- class nvinfer1::PluginRegistrar< T >

  Register the plugin creator to the registry The static registry object will be instantiated when the plugin library is loaded. This static object will register all creators available in the library to the registry.
- class nvinfer1::v_1_0::ILoggerFinder

  A virtual base class to find a logger. Allows a plugin to find an instance of a logger if it needs to emit a log message. A pointer to an instance of this class is passed to a plugin shared library on initialization when that plugin is serialized as part of a version-compatible plan. See the plugin chapter in the developer guide for details.
- class nvinfer1::v_1_0::IGpuAsyncAllocator
- class nvinfer1::v_1_0::IPluginCreatorV3One

Namespaces

- namespace nvinfer1

  The TensorRT API version 1 namespace.
- namespace nvinfer1::v_1_0

  Forward declare IErrorRecorder and ILogger for use in other interfaces.
- namespace nvinfer1::v_2_0
- namespace nvinfer1::anonymous_namespace{NvInferRuntime.h}

Macros

- #define NV_INFER_INTERNAL_INCLUDE 1
- #define REGISTER_TENSORRT_PLUGIN(name) static nvinfer1::PluginRegistrar<name> pluginRegistrar##name {}

Typedefs

- using nvinfer1::IStreamReader = v_1_0::IStreamReader
- using nvinfer1::IStreamWriter = v_1_0::IStreamWriter
- using nvinfer1::IStreamReaderV2 = v_1_0::IStreamReaderV2
- using nvinfer1::IPluginV3OneCore = v_1_0::IPluginV3OneCore
- using nvinfer1::IPluginV3OneBuild = v_1_0::IPluginV3OneBuild
- using nvinfer1::IPluginV3OneRuntime = v_1_0::IPluginV3OneRuntime
- using nvinfer1::IPluginV3OneBuildV2 = v_2_0::IPluginV3OneBuild
- using nvinfer1::IProfiler = v_1_0::IProfiler
- using nvinfer1::TempfileControlFlags = uint32_t

  Represents a collection of one or more TempfileControlFlag values combined using bitwise-OR operations.
- using nvinfer1::AllocatorFlags = uint32_t

- using nvinfer1::IGpuAllocator = v_1_0::IGpuAllocator
- using nvinfer1::TacticSources = uint32_t

  Represents a collection of one or more TacticSource values combine using bitwise-OR operations.
- using nvinfer1::SerializationFlags = uint32_t

  Represents one or more SerializationFlag values using binary OR operations, e.g., 1U << SerializationFlag::kEXCLUDE_LEAN_RUNTIME
- using nvinfer1::IOutputAllocator = v_1_0::IOutputAllocator
- using nvinfer1::IDebugListener = v_1_0::IDebugListener
- using nvinfer1::ILoggerFinder = v_1_0::ILoggerFinder
- using nvinfer1::IGpuAsyncAllocator = v_1_0::IGpuAsyncAllocator
- using nvinfer1::IPluginCreatorV3One = v_1_0::IPluginCreatorV3One

Enumerations

- enum class nvinfer1::EngineCapability : int32_t { nvinfer1::kSTANDARD = 0 , nvinfer1::kSAFETY = 1 , nvinfer1::kDLA_STANDALONE = 2 }

  List of supported engine capability flows.
- enum class nvinfer1::DimensionOperation : int32_t { nvinfer1::kSUM = 0 , nvinfer1::kPROD = 1 , nvinfer1::kMAX = 2 , nvinfer1::kMIN = 3 , nvinfer1::kSUB = 4 , nvinfer1::kEQUAL = 5 , nvinfer1::kLESS = 6 , nvinfer1::kFLOOR_DIV = 7 , nvinfer1::kCEIL_DIV = 8 }

  An operation on two IDimensionExpr, which represent integer expressions used in dimension computations.
- enum class nvinfer1::TensorLocation : int32_t { nvinfer1::kDEVICE = 0 , nvinfer1::kHOST = 1 }

  The location for tensor data storage, device or host.
- enum class nvinfer1::SeekPosition : int32_t { nvinfer1::kSET = 0 , nvinfer1::kCUR = 1 , nvinfer1::kEND = 2 }

  Controls the seek mode of IStreamReaderV2.
- enum class nvinfer1::WeightsRole : int32_t { nvinfer1::kKERNEL = 0 , nvinfer1::kBIAS = 1 , nvinfer1::kSHIFT = 2 , nvinfer1::kSCALE = 3 , nvinfer1::kCONSTANT = 4 , nvinfer1::kANY = 5 }

  How a layer uses particular Weights.
- enum class nvinfer1::DeviceType : int32_t { nvinfer1::kGPU = 0 , nvinfer1::kDLA = 1 }

  The device that this layer/network will execute on.
- enum class nvinfer1::TempfileControlFlag : int32_t { nvinfer1::kALLOW_IN_MEMORY_FILES = 0 , nvinfer1::kALLOW_TEMPORARY_FILES = 1 }

  Flags used to control TensorRT's behavior when creating executable temporary files.
- enum class nvinfer1::TensorFormat : int32_t { nvinfer1::kLINEAR = 0 , nvinfer1::kCHW2 = 1 , nvinfer1::kHWC8 = 2 , nvinfer1::kCHW4 = 3 , nvinfer1::kCHW16 = 4 , nvinfer1::kCHW32 = 5 , nvinfer1::kDHWC8 = 6 , nvinfer1::kCDHW32 = 7 , nvinfer1::kHWC = 8 , nvinfer1::kDLA_LINEAR = 9 , nvinfer1::kDLA_HWC4 = 10 , nvinfer1::kHWC16 = 11 , nvinfer1::kDHWC = 12 }

  Format of the input/output tensors.
- enum class nvinfer1::AllocatorFlag : int32_t { nvinfer1::kRESIZABLE = 0 }

  Allowed type of memory allocation.
- enum class nvinfer1::OptProfileSelector : int32_t { nvinfer1::kMIN = 0 , nvinfer1::kOPT = 1 , nvinfer1::kMAX = 2 }

  When setting or querying optimization profile parameters (such as shape tensor inputs or dynamic dimensions), select whether we are interested in the minimum, optimum, or maximum values for these parameters. The minimum and maximum specify the permitted range that is supported at runtime, while the optimum value is used for the kernel selection. This should be the "typical" value that is expected to occur at runtime.

- enum class nvinfer1::TacticSource : int32_t { nvinfer1::kEDGE_MASK_CONVOLUTIONS = 0 , nvinfer1::kJIT_CONVOLUTIONS = 1 }

  List of tactic sources for TensorRT.
- enum class nvinfer1::ProfilingVerbosity : int32_t { nvinfer1::kLAYER_NAMES_ONLY = 0 , nvinfer1::kNONE = 1 , nvinfer1::kDETAILED = 2 }

  List of verbosity levels of layer information exposed in NVTX annotations and in IEngineInspector.
- enum class nvinfer1::SerializationFlag : int32_t { nvinfer1::kEXCLUDE_WEIGHTS = 0 , nvinfer1::kEXCLUDE_LEAN_RUNTIM = 1 , nvinfer1::kINCLUDE_REFIT = 2 }

  List of valid flags that the engine can enable when serializing the bytes.
- enum class nvinfer1::ExecutionContextAllocationStrategy : int32_t { nvinfer1::kSTATIC = 0 , nvinfer1::kON_PROFILE_CHANGE = 1 , nvinfer1::kUSER_MANAGED = 2 }

  Different memory allocation behaviors for IExecutionContext.
- enum class nvinfer1::EngineStat : int32_t { nvinfer1::kTOTAL_WEIGHTS_SIZE = 0 , nvinfer1::kSTRIPPED_WEIGHTS_SIZE = 1 }

  The kind of engine statistics that queried from the ICudaEngine.
- enum class nvinfer1::LayerInformationFormat : int32_t { nvinfer1::kONELINE = 0 , nvinfer1::kJSON = 1 }

  The format in which the IEngineInspector prints the layer information.

Functions

- TENSORRTAPI nvinfer1::IPluginRegistry * getPluginRegistry () noexcept

  Return the plugin registry.
- TENSORRTAPI nvinfer1::ILogger * getLogger () noexcept

  Return the logger object.
- IRuntime * nvinfer1::anonymous_namespace{NvInferRuntime.h}::createInferRuntime (ILogger &logger) noexcept

  Create an instance of an IRuntime class.
- IRefitter * nvinfer1::anonymous_namespace{NvInferRuntime.h}::createInferRefitter (ICudaEngine &engine, ILogger &logger) noexcept

  Create an instance of an IRefitter class.
- TENSORRTAPI int32_t getInferLibMajorVersion () noexcept

  Return the library major version number.
- TENSORRTAPI int32_t getInferLibMinorVersion () noexcept

  Return the library minor version number.
- TENSORRTAPI int32_t getInferLibPatchVersion () noexcept

  Return the library patch version number.
- TENSORRTAPI int32_t getInferLibBuildVersion () noexcept

  Return the library build version number.

10.13.1 Detailed Description

This is the top-level API file for TensorRT extended runtime library.

10.13.2 Macro Definition Documentation

10.13.2.1 NV_INFER_INTERNAL_INCLUDE

```
#define NV_INFER_INTERNAL_INCLUDE 1
```

10.13.2.2 REGISTER_TENSORRT_PLUGIN

```
#define REGISTER_TENSORRT_PLUGIN(
              name )  static nvinfer1::PluginRegistrar<name> pluginRegistrar##name {}
```

10.13.3 Function Documentation

10.13.3.1 getInferLibBuildVersion()

```
TENSORRTAPI int32_t getInferLibBuildVersion ( )  [noexcept]
```

Return the library build version number.

10.13.3.2 getInferLibMajorVersion()

```
TENSORRTAPI int32_t getInferLibMajorVersion ( )  [noexcept]
```

Return the library major version number.

10.13.3.3 getInferLibMinorVersion()

```
TENSORRTAPI int32_t getInferLibMinorVersion ( )  [noexcept]
```

Return the library minor version number.

10.13.3.4 getInferLibPatchVersion()

```
TENSORRTAPI int32_t getInferLibPatchVersion ( )  [noexcept]
```

Return the library patch version number.

10.13.3.5 getLogger()

```
TENSORRTAPI nvinfer1::ILogger * getLogger ( )  [noexcept]
```

Return the logger object.

Note

the global logger is used only by standalone functions which have no associated builder, runtime or refitter.

10.13.3.6 getPluginRegistry()

```
TENSORRTAPI nvinfer1::IPluginRegistry * getPluginRegistry ( )  [noexcept]
```

Return the plugin registry.

10.14 NvInferRuntime.h

Go to the documentation of this file.

```
1 /*
2 * SPDX-FileCopyrightText:  Copyright (c) 1993-2026 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3 * SPDX-License-Identifier:  Apache-2.0
4 *
5 * Licensed under the Apache License, Version 2.0 (the "License");
6 * you may not use this file except in compliance with the License.
7 * You may obtain a copy of the License at
8 *
9 * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17
18 #ifndef NV_INFER_RUNTIME_H
19 #define NV_INFER_RUNTIME_H
20
26
27 #include "NvInferImpl.h" // IWYU pragma:  export
28 #define NV_INFER_INTERNAL_INCLUDE 1
29 #include "NvInferPluginBase.h" // IWYU pragma:  export
30 #undef NV_INFER_INTERNAL_INCLUDE
31 #include "NvInferRuntimeCommon.h" // IWYU pragma:  export
32
33 namespace nvinfer1
34 {
35
36 class IExecutionContext;
37 class ICudaEngine;
38 class IPluginFactory;
39 class IEngineInspector;
40
49
50 class INoCopy
51 {
52 protected:
53     INoCopy() = default;
54     virtual ~INoCopy() = default;
55     INoCopy(INoCopy const& other) = delete;
```

```
56      INoCopy& operator=(INoCopy const& other) = delete;
57      INoCopy(INoCopy&& other) = delete;
58      INoCopy& operator=(INoCopy&& other) = delete;
59  };
60
75  enum class EngineCapability :  int32_t
76  {
81      kSTANDARD = 0,
82
89      kSAFETY = 1,
90
96      kDLA_STANDALONE = 2,
97  };
98
100 template <>
101 struct impl::EnumMaxImpl<EngineCapability>
102 {
103     static constexpr int32_t kVALUE = 3;
104 };
105
120 class Weights
121 {
122 public:
123     DataType type;
124     void const* values;
125     int64_t count;
126 };
127
138 class IHostMemory :  public INoCopy
139 {
140 public:
141     virtual ~IHostMemory() noexcept = 0;
142
144     void* data() const noexcept
145     {
146         return mImpl->data();
147     }
148
150     std::size_t size() const noexcept
151     {
152         return mImpl->size();
153     }
154
156     DataType type() const noexcept
157     {
158         return mImpl->type();
159     }
160
161 protected:
162     apiv::VHostMemory* mImpl;
163 };
164
165 inline IHostMemory::~IHostMemory() noexcept = default;
166
177 enum class DimensionOperation :  int32_t
178 {
179     kSUM = 0,
180     kPROD = 1,
181     kMAX = 2,
182     kMIN = 3,
183     kSUB = 4,
184     kEQUAL = 5,
185     kLESS = 6,
186     kFLOOR_DIV = 7,
187     kCEIL_DIV = 8
188 };
189
191 template <>
192 struct impl::EnumMaxImpl<DimensionOperation>
193 {
194     static constexpr int32_t kVALUE = 9;
195 };
196
202 enum class TensorLocation :  int32_t
203 {
204     kDEVICE = 0,
205     kHOST = 1,
206 };
207
209 template <>
210 struct impl::EnumMaxImpl<TensorLocation>
```

```
211 {
212     static constexpr int32_t kVALUE = 2;
213 };
214
227 class IDimensionExpr :  public INoCopy
228 {
229 public:
233     bool isConstant() const noexcept
234     {
235         return mImpl->isConstant();
236     }
237
244     int64_t getConstantValue() const noexcept
245     {
246         return mImpl->getConstantValue();
247     }
248
249 protected:
250     apiv::VDimensionExpr* mImpl;
251     virtual ~IDimensionExpr() noexcept = 0;
252
253 public:
259     bool isSizeTensor() const noexcept
260     {
261         return mImpl->isSizeTensor();
262     }
263 };
264
265 inline IDimensionExpr::~IDimensionExpr() noexcept = default;
266
284 class IExprBuilder :  public INoCopy
285 {
286 public:
290     IDimensionExpr const* constant(int64_t value) noexcept
291     {
292         return mImpl->constant(value);
293     }
294
301     IDimensionExpr const* operation(
302         DimensionOperation op, IDimensionExpr const& first, IDimensionExpr const& second) noexcept
303     {
304         return mImpl->operation(op, first, second);
305     }
306
307 protected:
308     apiv::VExprBuilder* mImpl;
309     virtual ~IExprBuilder() noexcept = 0;
310
311 public:
336     IDimensionExpr const* declareSizeTensor(int32_t outputIndex, IDimensionExpr const& opt, IDimensionExpr
    const& upper)
337     {
338         return mImpl->declareSizeTensor(outputIndex, opt, upper);
339     }
340 };
341
342 inline IExprBuilder::~IExprBuilder() noexcept = default;
343
349 class DimsExprs
350 {
351 public:
352     int32_t nbDims;
353     IDimensionExpr const* d[Dims::MAX_DIMS];
354 };
355
361 struct DynamicPluginTensorDesc
362 {
364     PluginTensorDesc desc;
365
367     Dims min;
368
370     Dims max;
371
373     Dims opt;
374 };
375
406 class TRT_DEPRECATED IPluginV2DynamicExt :  public nvinfer1::IPluginV2Ext
407 {
408 public:
409     IPluginV2DynamicExt* clone() const noexcept override = 0;
410
```

```
435     virtual DimsExprs getOutputDimensions(
436         int32_t outputIndex, DimsExprs const* inputs, int32_t nbInputs, IExprBuilder& exprBuilder) noexcept =
    0;
437
441     static constexpr int32_t kFORMAT_COMBINATION_LIMIT = 100;
442
475     virtual bool supportsFormatCombination(
476         int32_t pos, PluginTensorDesc const* inOut, int32_t nbInputs, int32_t nbOutputs) noexcept = 0;
477
513     virtual void configurePlugin(DynamicPluginTensorDesc const* in, int32_t nbInputs,
514         DynamicPluginTensorDesc const* out, int32_t nbOutputs) noexcept = 0;
515
525     virtual size_t getWorkspaceSize(PluginTensorDesc const* inputs, int32_t nbInputs, PluginTensorDesc const*
    outputs,
526         int32_t nbOutputs) const noexcept = 0;
527
540     virtual int32_t enqueue(PluginTensorDesc const* inputDesc, PluginTensorDesc const* outputDesc,
541         void const* const* inputs, void* const* outputs, void* workspace, cudaStream_t stream) noexcept = 0;
542
543 protected:
551     int32_t getTensorRTVersion() const noexceptoverride
552 {
553         return (static_cast<int32_t>(PluginVersion::kV2_DYNAMICEXT) << 24 | (NV_TENSORRT_VERSION & 0xFFFFFF));
554     }
555
556     virtual ~IPluginV2DynamicExt() noexcept {}
557
558 private:
559     // Following are obsolete base class methods, and must not be implemented or used.
560
564     void configurePlugin(Dims const*, int32_t, Dims const*, int32_t, DataType const*, DataType const*, bool
    const*,
565         bool const*, PluginFormat, int32_t) noexcept final
566     {
567     }
568
572     bool supportsFormat(DataType, PluginFormat) const noexcept final
573     {
574         return false;
575     }
576
580     Dims getOutputDimensions(int32_t, Dims const*, int32_t) noexcept final
581     {
582         return Dims{-1, {}};
583     }
584
588     size_t getWorkspaceSize(int32_t) const noexcept final
589     {
590         return 0;
591     }
592
596     int32_t enqueue(int32_t, void const* const*, void* const*, void*, cudaStream_t) noexcept final
597     {
598         return 1;
599     }
600 };
601
602 namespace v_1_0
603 {
607 class TRT_DEPRECATED IStreamReader :  public IVersionedInterface
608 {
609 public:
614     ~IStreamReader() override = default;
615     IStreamReader() = default;
616
620     InterfaceInfo getInterfaceInfo() const noexceptoverride
621 {
622         return InterfaceInfo{"IStreamReader", 1, 0};
623     }
624
633     virtual int64_t read(void* destination, int64_t nbBytes) = 0;
634
635 protected:
636     IStreamReader(IStreamReader const&) = default;
637     IStreamReader(IStreamReader&&) = default;
638     IStreamReader& operator=(IStreamReader const&) & = default;
639     IStreamReader& operator=(IStreamReader&&) & = default;
640 };
641
642 class IStreamWriter :  public IVersionedInterface
643 {
```

```
644 public:
649     ~IStreamWriter() override = default;
650     IStreamWriter() = default;
651
655     InterfaceInfo getInterfaceInfo() const noexceptoverride
656 {
657         return InterfaceInfo{"IStreamWriter", 1, 0};
658     }
659
669     virtual int64_t write(void const* data, int64_t nbBytes) = 0;
670
671 protected:
672     IStreamWriter(IStreamWriter const&) = default;
673     IStreamWriter(IStreamWriter&&) = default;
674     IStreamWriter& operator=(IStreamWriter const&) & = default;
675     IStreamWriter& operator=(IStreamWriter&&) & = default;
676 };
677 } // namespace v_1_0
678
689 using IStreamReader = v_1_0::IStreamReader;
690
699 using IStreamWriter = v_1_0::IStreamWriter;
700
705 enum class SeekPosition :  int32_t
706 {
708     kSET = 0,
709
711     kCUR = 1,
712
714     kEND = 2,
715 };
716
717 namespace v_1_0
718 {
719 class IStreamReaderV2 :  public IVersionedInterface
720 {
721 public:
726     ~IStreamReaderV2() override = default;
727     IStreamReaderV2() = default;
728
732     InterfaceInfo getInterfaceInfo() const noexceptoverride
733 {
734         return InterfaceInfo{"IStreamReaderV2", 1, 0};
735     }
736
747     virtual int64_t read(void* destination, int64_t nbBytes, cudaStream_t stream) noexcept = 0;
748
757     virtual bool seek(int64_t offset, SeekPosition where) noexcept = 0;
758
759 protected:
760     IStreamReaderV2(IStreamReaderV2 const&) = default;
761     IStreamReaderV2(IStreamReaderV2&&) = default;
762     IStreamReaderV2& operator=(IStreamReaderV2 const&) & = default;
763     IStreamReaderV2& operator=(IStreamReaderV2&&) & = default;
764 };
765 } // namespace v_1_0
766
776 using IStreamReaderV2 = v_1_0::IStreamReaderV2;
777
791 class IPluginResourceContext
792 {
793 public:
798     virtual IGpuAllocator* getGpuAllocator() const noexcept = 0;
799
804     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
805     virtual ~IPluginResourceContext() noexcept = 0;
806
807 protected:
808     IPluginResourceContext() = default;
809     IPluginResourceContext(IPluginResourceContext const&) = default;
810     IPluginResourceContext(IPluginResourceContext&&) = default;
811     IPluginResourceContext& operator=(IPluginResourceContext const&) & = default;
812     IPluginResourceContext& operator=(IPluginResourceContext&&) & = default;
813 };
814
815 inline IPluginResourceContext::~IPluginResourceContext() noexcept = default;
816
817 namespace v_1_0
818 {
819 class IPluginV3OneCore :  public IPluginCapability
820 {
```

```
821 public:
825     InterfaceInfo getInterfaceInfo() const noexceptoverride
826 {
827         return InterfaceInfo{"PLUGIN_V3ONE_CORE", 1, 0};
828     }
829
838     virtual AsciiChar const* getPluginName() const noexcept = 0;
839
848     virtual AsciiChar const* getPluginVersion() const noexcept = 0;
849
859     virtual AsciiChar const* getPluginNamespace() const noexcept = 0;
860 };
861
862 class IPluginV3OneBuild :  public IPluginCapability
863 {
864 public:
870     static constexpr int32_t kDEFAULT_FORMAT_COMBINATION_LIMIT = 100;
871
875     InterfaceInfo getInterfaceInfo() const noexceptoverride
876 {
877         return InterfaceInfo{"PLUGIN_V3ONE_BUILD", 1, 0};
878     }
879
899     virtual int32_t configurePlugin(DynamicPluginTensorDesc const* in, int32_t nbInputs,
900         DynamicPluginTensorDesc const* out, int32_t nbOutputs) noexcept = 0;
901
918     virtual int32_t getOutputDataTypes(
919         DataType* outputTypes, int32_t nbOutputs, const DataType* inputTypes, int32_t nbInputs) const
      noexcept = 0;
920
942     virtual int32_t getOutputShapes(DimsExprs const* inputs, int32_t nbInputs, DimsExprs const* shapeInputs,
943         int32_t nbShapeInputs, DimsExprs* outputs, int32_t nbOutputs, IExprBuilder& exprBuilder) noexcept =
      0;
944
979     virtual bool supportsFormatCombination(
980         int32_t pos, DynamicPluginTensorDesc const* inOut, int32_t nbInputs, int32_t nbOutputs) noexcept = 0;
981
987     virtual int32_t getNbOutputs() const noexcept = 0;
988
998     virtual size_t getWorkspaceSize(DynamicPluginTensorDesc const* /* inputs */, int32_t /* nbInputs */,
999         DynamicPluginTensorDesc const* /* outputs */, int32_t /* nbOutputs */) const noexcept
1000     {
1001         return 0;
1002     }
1003
1035     virtual int32_t getValidTactics(int32_t* /* tactics */, int32_t /* nbTactics */) noexcept
1036     {
1037         return 0;
1038     }
1039
1043     virtual int32_t getNbTactics() noexcept
1044     {
1045         return 0;
1046     }
1047
1059     virtual char const* getTimingCacheID() noexcept
1060     {
1061         return nullptr;
1062     }
1063
1067     virtual int32_t getFormatCombinationLimit() noexcept
1068     {
1069         return kDEFAULT_FORMAT_COMBINATION_LIMIT;
1070     }
1071
1078     virtual char const* getMetadataString() noexcept
1079     {
1080         return nullptr;
1081     }
1082 };
1083
1084 class IPluginV3OneRuntime :  public IPluginCapability
1085 {
1086 public:
1090     InterfaceInfo getInterfaceInfo() const noexceptoverride
1091 {
1092         return InterfaceInfo{"PLUGIN_V3ONE_RUNTIME", 1, 0};
1093     }
1094
1102     virtual int32_t setTactic(int32_t /* tactic */) noexcept
1103     {
```

```
1104         return 0;
1105     }
1106
1125     virtual int32_t onShapeChange(
1126         PluginTensorDesc const* in, int32_t nbInputs, PluginTensorDesc const* out, int32_t nbOutputs)
      noexcept = 0;
1127
1141     virtual int32_t enqueue(PluginTensorDesc const* inputDesc, PluginTensorDesc const* outputDesc,
1142         void const* const* inputs, void* const* outputs, void* workspace, cudaStream_t stream) noexcept =
      0;
1143
1163     virtual IPluginV3* attachToContext(IPluginResourceContext* context) noexcept = 0;
1164
1170
1174     virtual PluginFieldCollection const* getFieldsToSerialize() noexcept = 0;
1175 };
1176 } // namespace v_1_0
1177
1178 namespace v_2_0
1179 {
1180
1181 class IPluginV3OneBuild :  public v_1_0::IPluginV3OneBuild
1182 {
1183 public:
1184     InterfaceInfo getInterfaceInfo() const noexceptoverride
1185 {
1186         return InterfaceInfo{"PLUGIN_V3ONE_BUILD", 2, 0};
1187     }
1188
1218     virtual int32_t getAliasedInput(int32_t /* outputIndex */) noexcept
1219     {
1220         return -1;
1221     }
1222 };
1223
1224 } // namespace v_2_0
1225
1235 using IPluginV3OneCore = v_1_0::IPluginV3OneCore;
1236
1247 using IPluginV3OneBuild = v_1_0::IPluginV3OneBuild;
1248
1259 using IPluginV3OneRuntime = v_1_0::IPluginV3OneRuntime;
1260
1268 using IPluginV3OneBuildV2 = v_2_0::IPluginV3OneBuild;
1269
1272 namespace v_1_0
1273 {
1274 class IProfiler :  public IVersionedInterface
1275 {
1276 public:
1280     InterfaceInfo getInterfaceInfo() const noexceptoverride
1281 {
1282         return {"IProfiler", 1, 0};
1283     }
1284
1292     virtual void reportLayerTime(char const* layerName, float ms) noexcept = 0;
1293
1294     ~IProfiler() override = default;
1295 };
1296 } // namespace v_1_0
1297
1309 using IProfiler = v_1_0::IProfiler;
1310
1318 enum class WeightsRole :  int32_t
1319 {
1320     kKERNEL = 0,
1321     kBIAS = 1,
1322     kSHIFT = 2,
1323     kSCALE = 3,
1324     kCONSTANT = 4,
1325     kANY = 5,
1326 };
1327
1329 template <>
1330 struct impl::EnumMaxImpl<WeightsRole>
1331 {
1332     static constexpr int32_t kVALUE = 6;
1333 };
1334
1340 enum class DeviceType :  int32_t
1341 {
```

```
1342     kGPU = 0,
1343     kDLA = 1,
1344 };
1345
1347 template <>
1348 struct impl::EnumMaxImpl<DeviceType>
1349 {
1350     static constexpr int32_t kVALUE = 2;
1351 };
1352
1363 enum class TempfileControlFlag :  int32_t
1364 {
1366     kALLOW_IN_MEMORY_FILES = 0,
1367
1371     kALLOW_TEMPORARY_FILES = 1,
1372 };
1373
1375 template <>
1376 struct impl::EnumMaxImpl<TempfileControlFlag>
1377 {
1378     static constexpr int32_t kVALUE = 2;
1379 };
1380
1387 using TempfileControlFlags = uint32_t;
1388
1420 enum class TensorFormat :  int32_t
1421 {
1428     kLINEAR = 0,
1429
1434     kCHW2 = 1,
1435
1439     kHWC8 = 2,
1440
1454     kCHW4 = 3,
1455
1462     kCHW16 = 4,
1463
1471     kCHW32 = 5,
1472
1477     kDHWC8 = 6,
1478
1483     kCDHW32 = 7,
1484
1488     kHWC = 8,
1489
1498     kDLA_LINEAR = 9,
1499
1513     kDLA_HWC4 = 10,
1514
1519     kHWC16 = 11,
1520
1525     kDHWC = 12
1526 };
1527
1529 template <>
1530 struct impl::EnumMaxImpl<TensorFormat>
1531 {
1533     static constexpr int32_t kVALUE = 13;
1534 };
1535
1541 enum class AllocatorFlag :  int32_t
1542 {
1544     kRESIZABLE = 0,
1545 };
1546
1548 template <>
1549 struct impl::EnumMaxImpl<AllocatorFlag>
1550 {
1552     static constexpr int32_t kVALUE = 1;
1553 };
1554
1555 using AllocatorFlags = uint32_t;
1556
1559 namespace v_1_0
1560 {
1561
1574 class ILogger :  public IVersionedInterface
1575 {
1576 public:
1580     InterfaceInfo getInterfaceInfo() const noexceptoverride
1581 {
```

```
1582         return {"ILogger", 1, 0};
1583     }
1584
1590     enum class Severity :  int32_t
1591     {
1593         kINTERNAL_ERROR = 0,
1595         kERROR = 1,
1597         kWARNING = 2,
1599         kINFO = 3,
1601         kVERBOSE = 4,
1602     };
1603
1622     virtual void log(Severity severity, AsciiChar const* msg) noexcept = 0;
1623
1624     ILogger() = default;
1625     ~ILogger() override = default;
1626
1627 protected:
1628     // @cond SuppressDoxyWarnings
1629     ILogger(ILogger const&) = default;
1630     ILogger(ILogger&&) = default;
1631     ILogger& operator=(ILogger const&) & = default;
1632     ILogger& operator=(ILogger&&) & = default;
1633     // @endcond
1634 };
1635
1636 } // namespace v_1_0
1637
1638 using ILogger = v_1_0::ILogger;
1639
1641 template <>
1642 struct impl::EnumMaxImpl<ILogger::Severity>
1643 {
1645     static constexpr int32_t kVALUE = 5;
1646 };
1647
1648 namespace v_1_0
1649 {
1650
1651 class IGpuAllocator :  public IVersionedInterface
1652 {
1653 public:
1678     TRT_DEPRECATED virtual void* allocate(
1679         uint64_t const size, uint64_t const alignment, AllocatorFlags const flags) noexcept = 0;
1680
1681     ~IGpuAllocator() override = default;
1682     IGpuAllocator() = default;
1683
1721     virtual void* reallocate(void* const /*baseAddr*/, uint64_t /*alignment*/, uint64_t /*newSize*/)
      noexcept
1722     {
1723         return nullptr;
1724     }
1725
1744     TRT_DEPRECATED virtual bool deallocate(void* const memory) noexcept = 0;
1745
1774     virtual void* allocateAsync(
1775         uint64_t const size, uint64_t const alignment, AllocatorFlags const flags, cudaStream_t /*stream*/)
      noexcept
1776     {
1777         return allocate(size, alignment, flags);
1778     }
1807     virtual bool deallocateAsync(void* const memory, cudaStream_t /*stream*/) noexcept
1808     {
1809         return deallocate(memory);
1810     }
1811
1815     InterfaceInfo getInterfaceInfo() const noexceptoverride
1816 {
1817         return {"IGpuAllocator", 1, 0};
1818     }
1819
1820 protected:
1821     // @cond SuppressDoxyWarnings
1822     IGpuAllocator(IGpuAllocator const&) = default;
1823     IGpuAllocator(IGpuAllocator&&) = default;
1824     IGpuAllocator& operator=(IGpuAllocator const&) & = default;
1825     IGpuAllocator& operator=(IGpuAllocator&&) & = default;
1826     // @endcond
1827 };
1828
```

```
1829 } // namespace v_1_0
1830
1851 using IGpuAllocator = v_1_0::IGpuAllocator;
1852
1853
1861 class IRuntime :  public INoCopy
1862 {
1863 public:
1864     virtual ~IRuntime() noexcept = 0;
1865
1877     void setDLACore(int32_t dlaCore) noexcept
1878     {
1879         mImpl->setDLACore(dlaCore);
1880     }
1881
1887     int32_t getDLACore() const noexcept
1888     {
1889         return mImpl->getDLACore();
1890     }
1891
1895     int32_t getNbDLACores() const noexcept
1896     {
1897         return mImpl->getNbDLACores();
1898     }
1899
1911     void setGpuAllocator(IGpuAllocator* allocator) noexcept
1912     {
1913         mImpl->setGpuAllocator(allocator);
1914     }
1915
1927     //
1930     void setErrorRecorder(IErrorRecorder* recorder) noexcept
1931     {
1932         mImpl->setErrorRecorder(recorder);
1933     }
1934
1945     IErrorRecorder* getErrorRecorder() const noexcept
1946     {
1947         return mImpl->getErrorRecorder();
1948     }
1949
1963     ICudaEngine* deserializeCudaEngine(void const* blob, std::size_t size) noexcept
1964     {
1965         return mImpl->deserializeCudaEngine(blob, size);
1966     }
1967
1986     ICudaEngine* deserializeCudaEngine(IStreamReaderV2& streamReader)
1987     {
1988         return mImpl->deserializeCudaEngineV2(streamReader);
1989     }
1990
1996     ILogger* getLogger() const noexcept
1997     {
1998         return mImpl->getLogger();
1999     }
2000
2011     bool setMaxThreads(int32_t maxThreads) noexcept
2012     {
2013         return mImpl->setMaxThreads(maxThreads);
2014     }
2015
2025     int32_t getMaxThreads() const noexcept
2026     {
2027         return mImpl->getMaxThreads();
2028     }
2029
2060     void setTemporaryDirectory(char const* path) noexcept
2061     {
2062         return mImpl->setTemporaryDirectory(path);
2063     }
2064
2071     char const* getTemporaryDirectory() const noexcept
2072     {
2073         return mImpl->getTemporaryDirectory();
2074     }
2075
2087     void setTempfileControlFlags(TempfileControlFlags flags) noexcept
2088     {
2089         return mImpl->setTempfileControlFlags(flags);
2090     }
2091
```

```
2099     TempfileControlFlags getTempfileControlFlags() const noexcept
2100     {
2101         return mImpl->getTempfileControlFlags();
2102     }
2103
2109     IPluginRegistry& getPluginRegistry() noexcept
2110     {
2111         return mImpl->getPluginRegistry();
2112     }
2113
2127     IRuntime* loadRuntime(char const* path) noexcept
2128     {
2129         return mImpl->loadRuntime(path);
2130     }
2131
2139     void setEngineHostCodeAllowed(bool allowed) noexcept
2140     {
2141         return mImpl->setEngineHostCodeAllowed(allowed);
2142     }
2143
2149     bool getEngineHostCodeAllowed() const noexcept
2150     {
2151         return mImpl->getEngineHostCodeAllowed();
2152     }
2153
2154
2155 protected:
2156     apiv::VRuntime* mImpl{};
2157 };
2158
2159 inline IRuntime::~IRuntime() noexcept = default;
2160
2168 class IRefitter :  public INoCopy
2169 {
2170 public:
2171     virtual ~IRefitter() noexcept = 0;
2172
2188     bool setWeights(char const* layerName, WeightsRole role, Weights weights) noexcept
2189     {
2190         return mImpl->setWeights(layerName, role, weights);
2191     }
2192
2205     bool refitCudaEngine() noexcept
2206     {
2207         return mImpl->refitCudaEngine();
2208     }
2209
2226     int32_t getMissing(int32_t size, char const** layerNames, WeightsRole* roles) noexcept
2227     {
2228         return mImpl->getMissing(size, layerNames, roles);
2229     }
2230
2243     int32_t getAll(int32_t size, char const** layerNames, WeightsRole* roles) noexcept
2244     {
2245         return mImpl->getAll(size, layerNames, roles);
2246     }
2247
2259     //
2262     void setErrorRecorder(IErrorRecorder* recorder) noexcept
2263     {
2264         mImpl->setErrorRecorder(recorder);
2265     }
2266
2277     IErrorRecorder* getErrorRecorder() const noexcept
2278     {
2279         return mImpl->getErrorRecorder();
2280     }
2281
2302     bool setNamedWeights(char const* name, Weights weights) noexcept
2303     {
2304         return mImpl->setNamedWeights(name, weights);
2305     }
2306
2322     int32_t getMissingWeights(int32_t size, char const** weightsNames) noexcept
2323     {
2324         return mImpl->getMissingWeights(size, weightsNames);
2325     }
2326
2338     int32_t getAllWeights(int32_t size, char const** weightsNames) noexcept
2339     {
2340         return mImpl->getAllWeights(size, weightsNames);
```

```
2341     }
2342
2348     ILogger* getLogger() const noexcept
2349     {
2350         return mImpl->getLogger();
2351     }
2352
2364     bool setMaxThreads(int32_t maxThreads) noexcept
2365     {
2366         return mImpl->setMaxThreads(maxThreads);
2367     }
2368
2378     int32_t getMaxThreads() const noexcept
2379     {
2380         return mImpl->getMaxThreads();
2381     }
2382
2405     bool setNamedWeights(char const* name, Weights weights, TensorLocation location) noexcept
2406     {
2407         return mImpl->setNamedWeightsWithLocation(name, weights, location);
2408     }
2409
2421     Weights getNamedWeights(char const* weightsName) const noexcept
2422     {
2423         return mImpl->getNamedWeights(weightsName);
2424     }
2425
2437     TensorLocation getWeightsLocation(char const* weightsName) const noexcept
2438     {
2439         return mImpl->getWeightsLocation(weightsName);
2440     }
2441
2453     bool unsetNamedWeights(char const* weightsName) noexcept
2454     {
2455         return mImpl->unsetNamedWeights(weightsName);
2456     }
2457
2469     void setWeightsValidation(bool weightsValidation) noexcept
2470     {
2471         return mImpl->setWeightsValidation(weightsValidation);
2472     }
2473
2477     bool getWeightsValidation() const noexcept
2478     {
2479         return mImpl->getWeightsValidation();
2480     }
2481
2499     bool refitCudaEngineAsync(cudaStream_t stream) noexcept
2500     {
2501         return mImpl->refitCudaEngineAsync(stream);
2502     }
2503
2517     Weights getWeightsPrototype(char const* weightsName) const noexcept
2518     {
2519         return mImpl->getWeightsPrototype(weightsName);
2520     }
2521
2522 protected:
2523     apiv::VRefitter* mImpl;
2524 };
2525
2526 inline IRefitter::~IRefitter() noexcept = default;
2527
2538 enum class OptProfileSelector :  int32_t
2539 {
2540     kMIN = 0,
2541     kOPT = 1,
2542     kMAX = 2
2543 };
2544
2546 template <>
2547 struct impl::EnumMaxImpl<OptProfileSelector>
2548 {
2549     static constexpr int32_t kVALUE = 3;
2550 };
2551
2574 class IOptimizationProfile :  public INoCopy
2575 {
2576 public:
2604     bool setDimensions(char const* inputName, OptProfileSelector select, Dims const& dims) noexcept
2605     {
```

```
2606         return mImpl->setDimensions(inputName, select, dims);
2607     }
2608
2616     Dims getDimensions(char const* inputName, OptProfileSelector select) const noexcept
2617     {
2618         return mImpl->getDimensions(inputName, select);
2619     }
2620
2629     int32_t getNbShapeValues(char const* inputName) const noexcept
2630     {
2631         return mImpl->getNbShapeValues(inputName);
2632     }
2633
2647     bool setExtraMemoryTarget(float target) noexcept
2648     {
2649         return mImpl->setExtraMemoryTarget(target);
2650     }
2651
2659     float getExtraMemoryTarget() const noexcept
2660     {
2661         return mImpl->getExtraMemoryTarget();
2662     }
2663
2676     bool isValid() const noexcept
2677     {
2678         return mImpl->isValid();
2679     }
2680
2723     bool setShapeValuesV2(
2724         char const* inputName, OptProfileSelector select, int64_t const* values, int32_t nbValues) noexcept
2725     {
2726         return mImpl->setShapeValuesV2(inputName, select, values, nbValues);
2727     }
2728
2736     int64_t const* getShapeValuesV2(char const* inputName, OptProfileSelector select) const noexcept
2737     {
2738         return mImpl->getShapeValuesV2(inputName, select);
2739     }
2740
2741 protected:
2742     apiv::VOptimizationProfile* mImpl;
2743     virtual ~IOptimizationProfile() noexcept = 0;
2744 };
2745
2746 inline IOptimizationProfile::~IOptimizationProfile() noexcept = default;
2747
2755 enum class TacticSource :  int32_t
2756 {
2760     kEDGE_MASK_CONVOLUTIONS = 0,
2761
2764     kJIT_CONVOLUTIONS = 1,
2765 };
2766
2768 template <>
2769 struct impl::EnumMaxImpl<TacticSource>
2770 {
2771     static constexpr int32_t kVALUE = 2;
2772 };
2773
2780 using TacticSources = uint32_t;
2781
2791 enum class ProfilingVerbosity :  int32_t
2792 {
2793     kLAYER_NAMES_ONLY = 0,
2794     kNONE = 1,
2795     kDETAILED = 2,
2796 };
2797
2799 template <>
2800 struct impl::EnumMaxImpl<ProfilingVerbosity>
2801 {
2802     static constexpr int32_t kVALUE = 3;
2803 };
2804
2811 using SerializationFlags = uint32_t;
2812
2820 enum class SerializationFlag :  int32_t
2821 {
2822     kEXCLUDE_WEIGHTS = 0,
2823     kEXCLUDE_LEAN_RUNTIME = 1,
2824     kINCLUDE_REFIT = 2,
```

```
2825 };
2826
2828 template <>
2829 struct impl::EnumMaxImpl<SerializationFlag>
2830 {
2831     static constexpr int32_t kVALUE = 3;
2832 };
2833
2841 class ISerializationConfig :  public INoCopy
2842 {
2843 public:
2844     virtual ~ISerializationConfig() noexcept = 0;
2845
2857     bool setFlags(SerializationFlags serializationFlags) noexcept
2858     {
2859         return mImpl->setFlags(serializationFlags);
2860     }
2861
2869     SerializationFlags getFlags() const noexcept
2870     {
2871         return mImpl->getFlags();
2872     }
2873
2881     bool clearFlag(SerializationFlag serializationFlag) noexcept
2882     {
2883         return mImpl->clearFlag(serializationFlag);
2884     }
2885
2893     bool setFlag(SerializationFlag serializationFlag) noexcept
2894     {
2895         return mImpl->setFlag(serializationFlag);
2896     }
2897
2905     bool getFlag(SerializationFlag serializationFlag) const noexcept
2906     {
2907         return mImpl->getFlag(serializationFlag);
2908     }
2909
2910 protected:
2911     apiv::VSerializationConfig* mImpl;
2912 };
2913
2914 inline ISerializationConfig::~ISerializationConfig() noexcept = default;
2915
2927 enum class ExecutionContextAllocationStrategy :  int32_t
2928 {
2929     kSTATIC = 0,
2930     kON_PROFILE_CHANGE = 1,
2931     kUSER_MANAGED = 2,
2932 };
2933
2936 template <>
2937 struct impl::EnumMaxImpl<ExecutionContextAllocationStrategy>
2938 {
2939     static constexpr int32_t kVALUE = 3;
2940 };
2941
2942
2949 class IRuntimeConfig :  public INoCopy
2950 {
2951 public:
2952     virtual ~IRuntimeConfig() noexcept = 0;
2953
2959     void setExecutionContextAllocationStrategy(ExecutionContextAllocationStrategy strategy) noexcept
2960     {
2961         return mImpl->setExecutionContextAllocationStrategy(strategy);
2962     }
2963
2969     ExecutionContextAllocationStrategy getExecutionContextAllocationStrategy() const noexcept
2970     {
2971         return mImpl->getExecutionContextAllocationStrategy();
2972     }
2973
2974
2975 protected:
2976     apiv::VRuntimeConfig* mImpl;
2977 }; // class IRuntimeConfig
2978
2979 inline IRuntimeConfig::~IRuntimeConfig() noexcept = default;
2980
2989 enum class EngineStat :  int32_t
```

```
2990 {
2992     kTOTAL_WEIGHTS_SIZE = 0,
2993
2995     kSTRIPPED_WEIGHTS_SIZE = 1,
2996 };
2997
2999 template <>
3000 struct impl::EnumMaxImpl<EngineStat>
3001 {
3002     static constexpr int32_t kVALUE = 2;
3003 };
3004
3012 class ICudaEngine :  public INoCopy
3013 {
3014 public:
3015     virtual ~ICudaEngine() noexcept = 0;
3016
3027     Dims getTensorShape(char const* tensorName) const noexcept
3028     {
3029         return mImpl->getTensorShape(tensorName);
3030     }
3031
3042     DataType getTensorDataType(char const* tensorName) const noexcept
3043     {
3044         return mImpl->getTensorDataType(tensorName);
3045     }
3046
3056     int32_t getNbLayers() const noexcept
3057     {
3058         return mImpl->getNbLayers();
3059     }
3060
3070     IHostMemory* serialize() const noexcept
3071     {
3072         return mImpl->serialize();
3073     }
3074
3087     IExecutionContext* createExecutionContext(
3088         ExecutionContextAllocationStrategy strategy = ExecutionContextAllocationStrategy::kSTATIC) noexcept
3089     {
3090         return mImpl->createExecutionContext(strategy);
3091     }
3092
3105     TensorLocation getTensorLocation(char const* tensorName) const noexcept
3106     {
3107         return mImpl->getTensorLocation(tensorName);
3108     }
3109
3125     bool isShapeInferenceIO(char const* tensorName) const noexcept
3126     {
3127         return mImpl->isShapeInferenceIO(tensorName);
3128     }
3129
3139     TensorIOMode getTensorIOMode(char const* tensorName) const noexcept
3140     {
3141         return mImpl->getTensorIOMode(tensorName);
3142     }
3143
3158     TRT_NODISCARD char const* getAliasedInputTensor(char const* tensorName) const noexcept
3159     {
3160         return mImpl->getAliasedInputTensor(tensorName);
3161     }
3162
3170     IExecutionContext* createExecutionContext(IRuntimeConfig* runtimeConfig) noexcept
3171     {
3172         return mImpl->createExecutionContextWithRuntimeConfig(runtimeConfig);
3173     }
3174
3183     IRuntimeConfig* createRuntimeConfig() noexcept
3184     {
3185         return mImpl->createRuntimeConfig();
3186     }
3187
3197     int64_t getDeviceMemorySizeV2() const noexcept
3198     {
3199         return mImpl->getDeviceMemorySizeV2();
3200     }
3201
3211     int64_t getDeviceMemorySizeForProfileV2(int32_t profileIndex) const noexcept
3212     {
3213         return mImpl->getDeviceMemorySizeForProfileV2(profileIndex);
```

```
3214     }
3215
3221     bool isRefittable() const noexcept
3222     {
3223         return mImpl->isRefittable();
3224     }
3225
3242     int32_t getTensorBytesPerComponent(char const* tensorName) const noexcept
3243     {
3244         return mImpl->getTensorBytesPerComponent(tensorName);
3245     }
3246
3260     int32_t getTensorBytesPerComponent(char const* tensorName, int32_t profileIndex) const noexcept
3261     {
3262         return mImpl->getTensorBytesPerComponentV2(tensorName, profileIndex);
3263     }
3264
3281     int32_t getTensorComponentsPerElement(char const* tensorName) const noexcept
3282     {
3283         return mImpl->getTensorComponentsPerElement(tensorName);
3284     }
3285
3299     int32_t getTensorComponentsPerElement(char const* tensorName, int32_t profileIndex) const noexcept
3300     {
3301         return mImpl->getTensorComponentsPerElementV2(tensorName, profileIndex);
3302     }
3303
3314     TensorFormat getTensorFormat(char const* tensorName) const noexcept
3315     {
3316         return mImpl->getTensorFormat(tensorName);
3317     }
3318
3328     TensorFormat getTensorFormat(char const* tensorName, int32_t profileIndex) const noexcept
3329     {
3330         return mImpl->getTensorFormatV2(tensorName, profileIndex);
3331     }
3332
3352     char const* getTensorFormatDesc(char const* tensorName) const noexcept
3353     {
3354         return mImpl->getTensorFormatDesc(tensorName);
3355     }
3356
3375     char const* getTensorFormatDesc(char const* tensorName, int32_t profileIndex) const noexcept
3376     {
3377         return mImpl->getTensorFormatDescV2(tensorName, profileIndex);
3378     }
3379
3392     int32_t getTensorVectorizedDim(char const* tensorName) const noexcept
3393     {
3394         return mImpl->getTensorVectorizedDim(tensorName);
3395     }
3396
3408     int32_t getTensorVectorizedDim(char const* tensorName, int32_t profileIndex) const noexcept
3409     {
3410         return mImpl->getTensorVectorizedDimV2(tensorName, profileIndex);
3411     }
3412
3423     char const* getName() const noexcept
3424     {
3425         return mImpl->getName();
3426     }
3427
3434     int32_t getNbOptimizationProfiles() const noexcept
3435     {
3436         return mImpl->getNbOptimizationProfiles();
3437     }
3438
3454     Dims getProfileShape(char const* tensorName, int32_t profileIndex, OptProfileSelector select) const
    noexcept
3455     {
3456         return mImpl->getProfileShape(tensorName, profileIndex, select);
3457     }
3458
3469     EngineCapability getEngineCapability() const noexcept
3470     {
3471         return mImpl->getEngineCapability();
3472     }
3473
3488     void setErrorRecorder(IErrorRecorder* recorder) noexcept
3489     {
3490         return mImpl->setErrorRecorder(recorder);
```

```
3491     }
3492
3503     IErrorRecorder* getErrorRecorder() const noexcept
3504     {
3505         return mImpl->getErrorRecorder();
3506     }
3507
3519     TacticSources getTacticSources() const noexcept
3520     {
3521         return mImpl->getTacticSources();
3522     }
3523
3531     ProfilingVerbosity getProfilingVerbosity() const noexcept
3532     {
3533         return mImpl->getProfilingVerbosity();
3534     }
3535
3541     IEngineInspector* createEngineInspector() const noexcept
3542     {
3543         return mImpl->createEngineInspector();
3544     }
3545
3554     int32_t getNbIOTensors() const noexcept
3555     {
3556         return mImpl->getNbIOTensors();
3557     }
3558
3566     char const* getIOTensorName(int32_t index) const noexcept
3567     {
3568         return mImpl->getIOTensorName(index);
3569     }
3570
3577     HardwareCompatibilityLevel getHardwareCompatibilityLevel() const noexcept
3578     {
3579         return mImpl->getHardwareCompatibilityLevel();
3580     }
3581
3592     int32_t getNbAuxStreams() const noexcept
3593     {
3594         return mImpl->getNbAuxStreams();
3595     }
3596
3602     ISerializationConfig* createSerializationConfig() noexcept
3603     {
3604         return mImpl->createSerializationConfig();
3605     }
3606
3622     IHostMemory* serializeWithConfig(ISerializationConfig& config) const noexcept
3623     {
3624         return mImpl->serializeWithConfig(config);
3625     }
3626
3638     int64_t getStreamableWeightsSize() const noexcept
3639     {
3640         return mImpl->getStreamableWeightsSize();
3641     }
3642
3677     bool setWeightStreamingBudgetV2(int64_t gpuMemoryBudget) noexcept
3678     {
3679         return mImpl->setWeightStreamingBudgetV2(gpuMemoryBudget);
3680     }
3681
3693     int64_t getWeightStreamingBudgetV2() const noexcept
3694     {
3695         return mImpl->getWeightStreamingBudgetV2();
3696     }
3697
3716     int64_t getWeightStreamingAutomaticBudget() const noexcept
3717     {
3718         return mImpl->getWeightStreamingAutomaticBudget();
3719     }
3720
3742     int64_t getWeightStreamingScratchMemorySize() const noexcept
3743     {
3744         return mImpl->getWeightStreamingScratchMemorySize();
3745     }
3746
3756     bool isDebugTensor(char const* name) const noexcept
3757     {
3758         return mImpl->isDebugTensor(name);
3759     }
```

```
3760
3778     int64_t const* getProfileTensorValuesV2(
3779         char const* tensorName, int32_t profileIndex, OptProfileSelector select) const noexcept
3780     {
3781         return mImpl->getProfileTensorValuesV2(tensorName, profileIndex, select);
3782     }
3783
3806     int64_t getEngineStat(EngineStat stat) const noexcept
3807     {
3808         return mImpl->getEngineStat(stat);
3809     }
3810
3811 protected:
3812     apiv::VCudaEngine* mImpl;
3813 };
3814
3815 inline ICudaEngine::~ICudaEngine() noexcept = default;
3816
3817 namespace v_1_0
3818 {
3819 class IOutputAllocator :  public IVersionedInterface
3820 {
3821 public:
3825     InterfaceInfo getInterfaceInfo() const noexceptoverride
3826 {
3827         return {"IOutputAllocator", 1, 0};
3828     }
3829
3849     TRT_DEPRECATED virtual void* reallocateOutput(
3850         char const* /* tensorName */, void* /* currentMemory */, uint64_t /* size */, uint64_t /* alignment
     */) noexcept
3851     {
3852         return nullptr;
3853     }
3854
3877     virtual void* reallocateOutputAsync(
3878         [[maybe_unused]] char const* tensorName, [[maybe_unused]] void* currentMemory, [[maybe_unused]]
     uint64_t size,
3879         [[maybe_unused]] uint64_t alignment, cudaStream_t /* stream */)
3880     {
3881         return reallocateOutput(tensorName, currentMemory, size, alignment);
3882     }
3883
3892     virtual void notifyShape(char const* tensorName, Dims const& dims) noexcept = 0;
3893 };
3894 } // namespace v_1_0
3895
3903 using IOutputAllocator = v_1_0::IOutputAllocator;
3904
3905 namespace v_1_0
3906 {
3907 class IDebugListener :  public IVersionedInterface
3908 {
3909 public:
3913     InterfaceInfo getInterfaceInfo() const noexceptoverride
3914 {
3915         return {"IDebugListener", 1, 0};
3916     }
3917
3931     virtual bool processDebugTensor(void const* addr, TensorLocation location, DataType type, Dims const&
     shape,
3932         char const* name, cudaStream_t stream)
3933         = 0;
3934
3935     ~IDebugListener() override = default;
3936 };
3937 } // namespace v_1_0
3938
3944 using IDebugListener = v_1_0::IDebugListener;
3945
3956 class IExecutionContext :  public INoCopy
3957 {
3958 public:
3959     virtual ~IExecutionContext() noexcept = 0;
3960
3969     void setDebugSync(bool sync) noexcept
3970     {
3971         mImpl->setDebugSync(sync);
3972     }
3973
3979     bool getDebugSync() const noexcept
```

```
3980     {
3981         return mImpl->getDebugSync();
3982     }
3983
3989     void setProfiler(IProfiler* profiler) noexcept
3990     {
3991         mImpl->setProfiler(profiler);
3992     }
3993
3999     IProfiler* getProfiler() const noexcept
4000     {
4001         return mImpl->getProfiler();
4002     }
4003
4009     ICudaEngine const& getEngine() const noexcept
4010     {
4011         return mImpl->getEngine();
4012     }
4013
4023     void setName(char const* name) noexcept
4024     {
4025         mImpl->setName(name);
4026     }
4027
4033     char const* getName() const noexcept
4034     {
4035         return mImpl->getName();
4036     }
4037
4059     void setDeviceMemory(void* memory) noexcept
4060     {
4061         mImpl->setDeviceMemory(memory);
4062     }
4063
4080     void setDeviceMemoryV2(void* memory, int64_t size) noexcept
4081     {
4082         return mImpl->setDeviceMemoryV2(memory, size);
4083     }
4084
4101     Dims getTensorStrides(char const* tensorName) const noexcept
4102     {
4103         return mImpl->getTensorStrides(tensorName);
4104     }
4105
4106 public:
4116     int32_t getOptimizationProfile() const noexcept
4117     {
4118         return mImpl->getOptimizationProfile();
4119     }
4120
4134     bool setInputShape(char const* tensorName, Dims const& dims) noexcept
4135     {
4136         return mImpl->setInputShape(tensorName, dims);
4137     }
4138
4171     Dims getTensorShape(char const* tensorName) const noexcept
4172     {
4173         return mImpl->getTensorShape(tensorName);
4174     }
4175
4187     bool allInputDimensionsSpecified() const noexcept
4188     {
4189         return mImpl->allInputDimensionsSpecified();
4190     }
4191
4206     void setErrorRecorder(IErrorRecorder* recorder) noexcept
4207     {
4208         mImpl->setErrorRecorder(recorder);
4209     }
4210
4221     IErrorRecorder* getErrorRecorder() const noexcept
4222     {
4223         return mImpl->getErrorRecorder();
4224     }
4225
4238     bool executeV2(void* const* bindings) noexcept
4239     {
4240         return mImpl->executeV2(bindings);
4241     }
4242
4282     bool setOptimizationProfileAsync(int32_t profileIndex, cudaStream_t stream) noexcept
```

```
4283     {
4284         return mImpl->setOptimizationProfileAsync(profileIndex, stream);
4285     }
4286
4298     void setEnqueueEmitsProfile(bool enqueueEmitsProfile) noexcept
4299     {
4300         mImpl->setEnqueueEmitsProfile(enqueueEmitsProfile);
4301     }
4302
4310     bool getEnqueueEmitsProfile() const noexcept
4311     {
4312         return mImpl->getEnqueueEmitsProfile();
4313     }
4314
4340     bool reportToProfiler() const noexcept
4341     {
4342         return mImpl->reportToProfiler();
4343     }
4344
4384     bool setTensorAddress(char const* tensorName, void* data) noexcept
4385     {
4386         return mImpl->setTensorAddress(tensorName, data);
4387     }
4388
4401     void const* getTensorAddress(char const* tensorName) const noexcept
4402     {
4403         return mImpl->getTensorAddress(tensorName);
4404     }
4405
4424     bool setOutputTensorAddress(char const* tensorName, void* data) noexcept
4425     {
4426         return mImpl->setOutputTensorAddress(tensorName, data);
4427     }
4428
4446     bool setInputTensorAddress(char const* tensorName, void const* data) noexcept
4447     {
4448         return mImpl->setInputTensorAddress(tensorName, data);
4449     }
4450
4465     void* getOutputTensorAddress(char const* tensorName) const noexcept
4466     {
4467         return mImpl->getOutputTensorAddress(tensorName);
4468     }
4469
4498     int32_t inferShapes(int32_t nbMaxNames, char const** tensorNames) noexcept
4499     {
4500         return mImpl->inferShapes(nbMaxNames, tensorNames);
4501     }
4502
4515     size_t updateDeviceMemorySizeForShapes() noexcept
4516     {
4517         return mImpl->updateDeviceMemorySizeForShapes();
4518     }
4519
4531     bool setInputConsumedEvent(cudaEvent_t event) noexcept
4532     {
4533         return mImpl->setInputConsumedEvent(event);
4534     }
4535
4541     cudaEvent_t getInputConsumedEvent() const noexcept
4542     {
4543         return mImpl->getInputConsumedEvent();
4544     }
4545
4560     bool setOutputAllocator(char const* tensorName, IOutputAllocator* outputAllocator) noexcept
4561     {
4562         return mImpl->setOutputAllocator(tensorName, outputAllocator);
4563     }
4564
4573     IOutputAllocator* getOutputAllocator(char const* tensorName) const noexcept
4574     {
4575         return mImpl->getOutputAllocator(tensorName);
4576     }
4577
4591     int64_t getMaxOutputSize(char const* tensorName) const noexcept
4592     {
4593         return mImpl->getMaxOutputSize(tensorName);
4594     }
4595
4611     bool setTemporaryStorageAllocator(IGpuAllocator* allocator) noexcept
4612     {
```

```
4613         return mImpl->setTemporaryStorageAllocator(allocator);
4614     }
4615
4621     IGpuAllocator* getTemporaryStorageAllocator() const noexcept
4622     {
4623         return mImpl->getTemporaryStorageAllocator();
4624     }
4625
4645     bool enqueueV3(cudaStream_t stream) noexcept
4646     {
4647         return mImpl->enqueueV3(stream);
4648     }
4649
4661     void setPersistentCacheLimit(size_t size) noexcept
4662     {
4663         mImpl->setPersistentCacheLimit(size);
4664     }
4665
4672     size_t getPersistentCacheLimit() const noexcept
4673     {
4674         return mImpl->getPersistentCacheLimit();
4675     }
4676
4696     bool setNvtxVerbosity(ProfilingVerbosity verbosity) noexcept
4697     {
4698         return mImpl->setNvtxVerbosity(verbosity);
4699     }
4700
4708     ProfilingVerbosity getNvtxVerbosity() const noexcept
4709     {
4710         return mImpl->getNvtxVerbosity();
4711     }
4712
4739     void setAuxStreams(cudaStream_t* auxStreams, int32_t nbStreams) noexcept
4740     {
4741         mImpl->setAuxStreams(auxStreams, nbStreams);
4742     }
4743
4751     bool setDebugListener(IDebugListener* listener) noexcept
4752     {
4753         return mImpl->setDebugListener(listener);
4754     }
4755
4761     IDebugListener* getDebugListener() noexcept
4762     {
4763         return mImpl->getDebugListener();
4764     }
4765
4780     bool setTensorDebugState(char const* name, bool flag) noexcept
4781     {
4782         return mImpl->setTensorDebugState(name, flag);
4783     }
4784
4792     bool getDebugState(char const* name) const noexcept
4793     {
4794         return mImpl->getDebugState(name);
4795     }
4796
4802     IRuntimeConfig* getRuntimeConfig() const noexcept
4803     {
4804         return mImpl->getRuntimeConfig();
4805     }
4806
4815     bool setAllTensorsDebugState(bool flag) noexcept
4816     {
4817         return mImpl->setAllTensorsDebugState(flag);
4818     }
4819
4831     bool setUnfusedTensorsDebugState(bool flag) noexcept
4832     {
4833         return mImpl->setUnfusedTensorsDebugState(flag);
4834     }
4835
4841     bool getUnfusedTensorsDebugState() const noexcept
4842     {
4843         return mImpl->getUnfusedTensorsDebugState();
4844     }
4845
4859     bool setCommunicator(void* communicator) noexcept
4860     {
4861         return mImpl->setCommunicator(communicator);
```

```
4862     }
4863
4864 protected:
4865     apiv::VExecutionContext* mImpl;
4866 }; // class IExecutionContext
4867
4868 inline IExecutionContext::~IExecutionContext() noexcept = default;
4869
4877 enum class LayerInformationFormat :  int32_t
4878 {
4879     kONELINE = 0,
4880     kJSON = 1,
4881 };
4882
4884 template <>
4885 struct impl::EnumMaxImpl<LayerInformationFormat>
4886 {
4887     static constexpr int32_t kVALUE = 2;
4888 };
4889
4905 class IEngineInspector :  public INoCopy
4906 {
4907 public:
4908     virtual ~IEngineInspector() noexcept = 0;
4909
4922     bool setExecutionContext(IExecutionContext const* context) noexcept
4923     {
4924         return mImpl->setExecutionContext(context);
4925     }
4926
4934     IExecutionContext const* getExecutionContext() const noexcept
4935     {
4936         return mImpl->getExecutionContext();
4937     }
4938
4959     char const* getLayerInformation(int32_t layerIndex, LayerInformationFormat format) const noexcept
4960     {
4961         return mImpl->getLayerInformation(layerIndex, format);
4962     }
4963
4982     char const* getEngineInformation(LayerInformationFormat format) const noexcept
4983     {
4984         return mImpl->getEngineInformation(format);
4985     }
4986
5001     void setErrorRecorder(IErrorRecorder* recorder) noexcept
5002     {
5003         mImpl->setErrorRecorder(recorder);
5004     }
5005
5016     IErrorRecorder* getErrorRecorder() const noexcept
5017     {
5018         return mImpl->getErrorRecorder();
5019     }
5020
5021 protected:
5022     apiv::VEngineInspector* mImpl;
5023 }; // class IEngineInspector
5024
5025 inline IEngineInspector::~IEngineInspector() noexcept = default;
5026
5027 } // namespace nvinfer1
5028
5033 extern "C" TENSORRTAPI void* createInferRuntime_INTERNAL(void* logger, int32_t version) noexcept;
5034
5039 extern "C" TENSORRTAPI void* createInferRefitter_INTERNAL(void* engine, void* logger, int32_t version)
     noexcept;
5040
5044 extern "C" TENSORRTAPI nvinfer1::IPluginRegistry* getPluginRegistry() noexcept;
5045
5051 extern "C" TENSORRTAPI nvinfer1::ILogger* getLogger() noexcept;
5052
5053 namespace nvinfer1
5054 {
5055 namespace // unnamed namespace avoids linkage surprises when linking objects built with different versions
     of this
5056           // header.
5057 {
5063 inline IRuntime* createInferRuntime(ILogger& logger) noexcept
5064 {
5065     return static_cast<IRuntime*>(createInferRuntime_INTERNAL(&logger, NV_TENSORRT_VERSION));
```

```
5066 }
5067
5074 inline IRefitter* createInferRefitter(ICudaEngine& engine, ILogger& logger) noexcept
5075 {
5076     return static_cast<IRefitter*>(createInferRefitter_INTERNAL(&engine, &logger, NV_TENSORRT_VERSION));
5077 }
5078
5079 } // namespace
5080
5092 template <typename T>
5093 class PluginRegistrar
5094 {
5095 public:
5096     PluginRegistrar()
5097     {
5098         getPluginRegistry()->registerCreator(instance, "");
5099     }
5100
5101 private:
5103     T instance{};
5104 };
5105
5106 } // namespace nvinfer1
5107
5108 #define REGISTER_TENSORRT_PLUGIN(name)
     \
5109 static nvinfer1::PluginRegistrar<name> pluginRegistrar##name {}
5110
5111 namespace nvinfer1
5112 {
5115 namespace v_1_0
5116 {
5125 class ILoggerFinder :  public IVersionedInterface
5126 {
5127 public:
5131     InterfaceInfo getInterfaceInfo() const noexceptoverride
5132 {
5133         return {"ILoggerFinder", 1, 0};
5134     }
5135
5143     virtual ILogger* findLogger() = 0;
5144
5145 protected:
5147     ~ILoggerFinder() override = default;
5148 };
5149
5150 } // namespace v_1_0
5151
5152 using ILoggerFinder = v_1_0::ILoggerFinder;
5153
5156 namespace v_1_0
5157 {
5158
5159 class IGpuAsyncAllocator :  public IGpuAllocator
5160 {
5161 public:
5162     IGpuAsyncAllocator() = default;
5163     ~IGpuAsyncAllocator() override = default;
5164
5194     void* allocateAsync(uint64_t const size, uint64_t const alignment, AllocatorFlags const flags,
5195         cudaStream_t /*stream*/) noexcept override = 0;
5196
5222     bool deallocateAsync(void* const memory, cudaStream_t /*stream*/) noexcept override = 0;
5223
5247     TRT_DEPRECATED void* allocate(
5248         uint64_t const size, uint64_t const alignment, AllocatorFlags const flags) noexcept override
5249     {
5250         return allocateAsync(size, alignment, flags, nullptr);
5251     }
5252
5271     TRT_DEPRECATED bool deallocate(void* const memory) noexcept override
5272     {
5273         return deallocateAsync(memory, nullptr);
5274     }
5275
5279     InterfaceInfo getInterfaceInfo() const noexceptoverride
5280 {
5281         return {"IGpuAllocator", 1, 0};
5282     }
5283 };
5284
```

```
5285 class IPluginCreatorV3One :  public IPluginCreatorInterface
5286 {
5287 public:
5291     InterfaceInfo getInterfaceInfo() const noexceptoverride
5292 {
5293         return InterfaceInfo{"PLUGIN CREATOR_V3ONE", 1, 0};
5294     }
5295
5312     virtual IPluginV3* createPlugin(
5313         AsciiChar const* name, PluginFieldCollection const* fc, TensorRTPhase phase) noexcept = 0;
5314
5321     virtual PluginFieldCollection const* getFieldNames() noexcept = 0;
5322
5329     virtual AsciiChar const* getPluginName() const noexcept = 0;
5330
5337     virtual AsciiChar const* getPluginVersion() const noexcept = 0;
5338
5345     virtual AsciiChar const* getPluginNamespace() const noexcept = 0;
5346
5347     IPluginCreatorV3One() = default;
5348     virtual ~IPluginCreatorV3One() = default;
5349
5350 protected:
5351     IPluginCreatorV3One(IPluginCreatorV3One const&) = default;
5352     IPluginCreatorV3One(IPluginCreatorV3One&&) = default;
5353     IPluginCreatorV3One& operator=(IPluginCreatorV3One const&) & = default;
5354     IPluginCreatorV3One& operator=(IPluginCreatorV3One&&) & = default;
5355 };
5356
5357 } // namespace v_1_0
5358
5372 using IGpuAsyncAllocator = v_1_0::IGpuAsyncAllocator;
5373
5382 using IPluginCreatorV3One = v_1_0::IPluginCreatorV3One;
5383
5384 } // namespace nvinfer1
5385
5389 extern "C" TENSORRTAPI int32_t getInferLibMajorVersion() noexcept;
5393 extern "C" TENSORRTAPI int32_t getInferLibMinorVersion() noexcept;
5397 extern "C" TENSORRTAPI int32_t getInferLibPatchVersion() noexcept;
5401 extern "C" TENSORRTAPI int32_t getInferLibBuildVersion() noexcept;
5402
5403 #endif // NV_INFER_RUNTIME_H
```

10.15 NvInferRuntimeBase.h File Reference

```
#include "NvInferVersion.h"
#include <cstddef>
#include <cstdint>
#include <cuda_runtime_api.h>
```

Classes

- struct nvinfer1::impl::EnumMaxImpl< DataType >

  One greater than the maximum value of DataType enum.
- class nvinfer1::Dims64
- class nvinfer1::InterfaceInfo

  Version information associated with a TRT interface.
- struct nvinfer1::impl::EnumMaxImpl< APILanguage >

  One greater than the maximum value of APILanguage enum.
- class nvinfer1::IVersionedInterface

  An Interface class for version control.

- struct nvinfer1::impl::EnumMaxImpl< ErrorCode >

  One greater than the maximum value of ErrorCode enum.
- class nvinfer1::v_1_0::IErrorRecorder
- struct nvinfer1::impl::EnumMaxImpl< TensorIOMode >

  One greater than the maximum value of TensorIOMode enum.

Namespaces

- namespace nvinfer1

  The TensorRT API version 1 namespace.
- namespace nvinfer1::v_1_0

  Forward declare IErrorRecorder and ILogger for use in other interfaces.
- namespace nvinfer1::impl

Macros

- #define TRT_DEPRECATED __attribute__((deprecated))
- #define TRT_DEPRECATED_ENUM
- #define TRT_DEPRECATED_API __attribute__((deprecated, visibility("default")))
- #define TRT_DEPRECATED_BECAUSE(REASON) TRT_DEPRECATED
- #define TRT_NODISCARD

  A stand-in for `[[nodiscard]]` and `[[nodiscard(REASON)]]` that works with older compilers.
- #define TRT_NODISCARD_BECAUSE(REASON)
- #define TENSORRTAPI
- #define TRTNOEXCEPT
- #define NV_TENSORRT_VERSION_INT(major, minor, patch) ((major) *10000L + (minor) *100L + (patch) *1L)
- #define NV_TENSORRT_VERSION NV_TENSORRT_VERSION_INT(NV_TENSORRT_MAJOR, NV_TENSORRT_MINOR, NV_TENSORRT_PATCH)

Typedefs

- using nvinfer1::char_t = char

  char_t is the type used by TensorRT to represent all valid characters.
- using nvinfer1::AsciiChar = char_t
- using nvinfer1::IErrorRecorder = v_1_0::IErrorRecorder
- using nvinfer1::ILogger = v_1_0::ILogger
- using nvinfer1::Dims = Dims64
- using nvinfer1::InterfaceKind = char const *

Enumerations

- enum class nvinfer1::DataType : int32_t {
  nvinfer1::kFLOAT = 0 , nvinfer1::kHALF = 1 , nvinfer1::kINT8 = 2 , nvinfer1::kINT32 = 3 ,
  nvinfer1::kBOOL = 4 , nvinfer1::kUINT8 = 5 , nvinfer1::kFP8 = 6 , nvinfer1::kBF16 = 7 ,
  nvinfer1::kINT64 = 8 , nvinfer1::kINT4 = 9 , nvinfer1::kFP4 = 10 , nvinfer1::kE8M0 = 11 }

  The type of weights and tensors. The datatypes other than kBOOL, kINT32, and kINT64 are "activation datatypes," as they often represent values corresponding to inference results.
- enum class nvinfer1::APILanguage : int32_t { nvinfer1::kCPP = 0 , nvinfer1::kPYTHON = 1 }

  Programming language used in the implementation of a TRT interface.
- enum class nvinfer1::ErrorCode : int32_t {
  nvinfer1::kSUCCESS = 0 , nvinfer1::kUNSPECIFIED_ERROR = 1 , nvinfer1::kINTERNAL_ERROR = 2 ,
  nvinfer1::kINVALID_ARGUMENT = 3 ,
  nvinfer1::kINVALID_CONFIG = 4 , nvinfer1::kFAILED_ALLOCATION = 5 , nvinfer1::kFAILED_INITIALIZATION
  = 6 , nvinfer1::kFAILED_EXECUTION = 7 ,
  nvinfer1::kFAILED_COMPUTATION = 8 , nvinfer1::kINVALID_STATE = 9 , nvinfer1::kUNSUPPORTED_STATE
  = 10 }

  Error codes that can be returned by TensorRT during execution.
- enum class nvinfer1::TensorIOMode : int32_t { nvinfer1::kNONE = 0 , nvinfer1::kINPUT = 1 , nvinfer1::kOUTPUT = 2 }

  Definition of tensor IO Mode.

Functions

- template<typename T >
  constexpr int32_t nvinfer1::EnumMax () noexcept
- TENSORRTAPI int32_t getInferLibVersion () noexcept

  Return the library version number.

10.15.1 Detailed Description

This file contains common definitions, data structures and interfaces shared between the standard and safe runtime.

> Warning
>
> Do not directly include this file. Instead include one of:
>
> - NvInferRuntime.h (for the standard runtime)
> - NvInferPluginUtils.h (for plugin utilities)

10.15.2 Macro Definition Documentation

10.15.2.1 NV_TENSORRT_VERSION

```
#define NV_TENSORRT_VERSION NV_TENSORRT_VERSION_INT(NV_TENSORRT_MAJOR, NV_TENSORRT_MINOR, NV_TENSORRT_PATCH)
```

TensorRT version as a single integer. Usable in preprocessor expressions.

10.15.2.2 NV_TENSORRT_VERSION_INT

```
#define NV_TENSORRT_VERSION_INT(
              major,
              minor,
              patch ) ((major) *10000L + (minor) *100L + (patch) *1L)
```

Construct a single integer denoting TensorRT version. Usable in preprocessor expressions.

10.15.2.3 TENSORRTAPI

```
#define TENSORRTAPI
```

10.15.2.4 TRT_DEPRECATED

```
#define TRT_DEPRECATED __attribute__((deprecated))
```

10.15.2.5 TRT_DEPRECATED_API

```
#define TRT_DEPRECATED_API __attribute__((deprecated, visibility("default")))
```

10.15.2.6 TRT_DEPRECATED_BECAUSE

```
#define TRT_DEPRECATED_BECAUSE(
              REASON ) TRT_DEPRECATED
```

10.15.2.7 TRT_DEPRECATED_ENUM

```
#define TRT_DEPRECATED_ENUM
```

10.15.2.8 TRT_NODISCARD

```
#define TRT_NODISCARD
```

A stand-in for `[[nodiscard]]` and `[[nodiscard(REASON)]]` that works with older compilers.

10.15.2.9 TRT_NODISCARD_BECAUSE

```
#define TRT_NODISCARD_BECAUSE(
              REASON )
```

10.15.2.10 TRTNOEXCEPT

```
#define TRTNOEXCEPT
```

10.15.3 Function Documentation

10.15.3.1 getInferLibVersion()

```
TENSORRTAPI int32_t getInferLibVersion ( )  [noexcept]
```

Return the library version number.

The format is as for TENSORRT_VERSION: (MAJOR $*$ 100 + MINOR) $*$ 100 + PATCH

10.16 NvInferRuntimeBase.h

Go to the documentation of this file.

```
1 /*
2 * SPDX-FileCopyrightText: Copyright (c) 1993-2026 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3 * SPDX-License-Identifier: Apache-2.0
4 *
5 * Licensed under the Apache License, Version 2.0 (the "License");
6 * you may not use this file except in compliance with the License.
7 * You may obtain a copy of the License at
8 *
9 * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17
18 #ifndef NV_INFER_RUNTIME_BASE_H
19 #define NV_INFER_RUNTIME_BASE_H
20
21 #include "NvInferVersion.h"
22 #include <cstddef>
23 #include <cstdint>
24 #include <cuda_runtime_api.h>
25
26 // Items that are marked as deprecated will be removed in a future release.
27 #if __cplusplus >= 201402L
28 #define TRT_DEPRECATED [[deprecated]]
29 #define TRT_DEPRECATED_BECAUSE(REASON) [[deprecated(REASON)]]
30 #define TRT_DEPRECATED_ENUM TRT_DEPRECATED
31 #ifdef _MSC_VER
32 #define TRT_DEPRECATED_API __declspec(dllexport)
33 #else
34 #define TRT_DEPRECATED_API [[deprecated]] __attribute__((visibility("default")))
35 #endif
36 #else
37 #ifdef _MSC_VER
38 #define TRT_DEPRECATED
39 #define TRT_DEPRECATED_ENUM
40 #define TRT_DEPRECATED_API __declspec(dllexport)
41 #else
42 #define TRT_DEPRECATED __attribute__((deprecated))
43 #define TRT_DEPRECATED_ENUM
44 #define TRT_DEPRECATED_API __attribute__((deprecated, visibility("default")))
45 #endif
46 #define TRT_DEPRECATED_BECAUSE(REASON) TRT_DEPRECATED
47 #endif
48
50 #if __cplusplus >= 201907L
51 #define TRT_NODISCARD [[nodiscard]]
52 #define TRT_NODISCARD_BECAUSE(REASON) [[nodiscard(REASON)]]
53 #elif __cplusplus >= 201603L
54 #define TRT_NODISCARD [[nodiscard]]
55 #define TRT_NODISCARD_BECAUSE(REASON) [[nodiscard]]
56 #else
57 #define TRT_NODISCARD
58 #define TRT_NODISCARD_BECAUSE(REASON)
59 #endif
60
61 // Defines which symbols are exported.
62 #ifdef TENSORRT_BUILD_LIB
63 #ifdef _MSC_VER
64 //On Windows, exports are controlled by .def files; not by dllexport.
65 #define TENSORRTAPI
66 #else
67 #define TENSORRTAPI __attribute__((visibility("default")))
68 #endif
69 #else
70 #define TENSORRTAPI
71 #endif
72 #define TRTNOEXCEPT
82 #if !defined(NV_INFER_INTERNAL_INCLUDE)
83 static_assert(false, "Do not directly include this file. Include NvInferRuntime.h or NvInferPluginUtils.h");
84 #endif
85
87
```

```
88 extern "C"
89 {
91     struct cublasContext;
93     struct cudnnContext;
94 }
95
98 #define NV_TENSORRT_VERSION_INT(major, minor, patch) ((major) *10000L + (minor) *100L + (patch) *1L)
99
102 #define NV_TENSORRT_VERSION NV_TENSORRT_VERSION_INT(NV_TENSORRT_MAJOR, NV_TENSORRT_MINOR, NV_TENSORRT_PATCH)
103
109 namespace nvinfer1
110 {
112 using char_t = char;
113
116 using AsciiChar = char_t;
117
119 namespace v_1_0
120 {
121 class IErrorRecorder;
122 class ILogger;
123 } // namespace v_1_0
124 using IErrorRecorder = v_1_0::IErrorRecorder;
125 using ILogger = v_1_0::ILogger;
126
127 namespace impl
128 {
130 template <typename T>
131 struct EnumMaxImpl;
132 } // namespace impl
133
136 template <typename T>
137 constexpr int32_t EnumMax() noexcept
138 {
139     return impl::EnumMaxImpl<T>::kVALUE;
140 }
141
148 enum class DataType :  int32_t
149 {
151     kFLOAT = 0,
152
154     kHALF = 1,
155
157     kINT8 = 2,
158
160     kINT32 = 3,
161
163     kBOOL = 4,
164
177     kUINT8 = 5,
178
181     kFP8 = 6,
182
184     kBF16 = 7,
185
187     kINT64 = 8,
188
190     kINT4 = 9,
191
194     kFP4 = 10,
195
197     kE8M0 = 11,
198 };
199
200 namespace impl
201 {
203 template <>
204 struct EnumMaxImpl<DataType>
205 {
207     static constexpr int32_t kVALUE = 12;
208 };
209 } // namespace impl
210
221 class Dims64
222 {
223 public:
225     static constexpr int32_t MAX_DIMS{8};
226
228     int32_t nbDims;
229
231     int64_t d[MAX_DIMS];
232 };
```

```
233
237 using Dims = Dims64;
238
239 using InterfaceKind = char const*;
240
246 class InterfaceInfo
247 {
248 public:
249     InterfaceKind kind;
250     int32_t major;
251     int32_t minor;
252 };
253
259 enum class APILanguage :  int32_t
260 {
261     kCPP = 0,
262     kPYTHON = 1
263 };
264
265 namespace impl
266 {
268 template <>
269 struct EnumMaxImpl<APILanguage>
270 {
272     static constexpr int32_t kVALUE = 2;
273 };
274 } // namespace impl
275
281 class IVersionedInterface
282 {
283 public:
289     virtual APILanguage getAPILanguage() const noexcept
290     {
291         return APILanguage::kCPP;
292     }
293
297     virtual InterfaceInfo getInterfaceInfo() const noexcept = 0;
298
299     virtual ~IVersionedInterface() noexcept = default;
300
301 protected:
302     IVersionedInterface() = default;
303     IVersionedInterface(IVersionedInterface const&) = default;
304     IVersionedInterface(IVersionedInterface&&) = default;
305     IVersionedInterface& operator=(IVersionedInterface const&) & = default;
306     IVersionedInterface& operator=(IVersionedInterface&&) & = default;
307 };
308
314 enum class ErrorCode :  int32_t
315 {
319     kSUCCESS = 0,
320
324     kUNSPECIFIED_ERROR = 1,
325
330     kINTERNAL_ERROR = 2,
331
336     kINVALID_ARGUMENT = 3,
337
345     kINVALID_CONFIG = 4,
346
352     kFAILED_ALLOCATION = 5,
353
358     kFAILED_INITIALIZATION = 6,
359
366     kFAILED_EXECUTION = 7,
367
375     kFAILED_COMPUTATION = 8,
376
389     kINVALID_STATE = 9,
390
401     kUNSUPPORTED_STATE = 10,
402
403 };
404
405 namespace impl
406 {
408 template <>
409 struct EnumMaxImpl<ErrorCode>
410 {
412     static constexpr int32_t kVALUE = 11;
413 };
```

```
414 } // namespace impl
415
416 namespace v_1_0
417 {
418 class IErrorRecorder :  public IVersionedInterface
419 {
420 public:
424     InterfaceInfo getInterfaceInfo() const noexceptoverride
425 {
426         return InterfaceInfo{"IErrorRecorder", 1, 0};
427     }
428
432     using ErrorDesc = char const*;
433
439     static constexpr size_t kMAX_DESC_LENGTH{127U};
440
444     using RefCount = int32_t;
445
446     IErrorRecorder() = default;
447     ~IErrorRecorder() noexcept override = default;
448
449     // Public API used to retrieve information from the error recorder.
450
474     virtual int32_t getNbErrors() const noexcept = 0;
475
494     virtual ErrorCode getErrorCode(int32_t errorIdx) const noexcept = 0;
495
517     virtual ErrorDesc getErrorDesc(int32_t errorIdx) const noexcept = 0;
518
533     virtual bool hasOverflowed() const noexcept = 0;
534
549     virtual void clear() noexcept = 0;
550
551     // API used by TensorRT to report Error information to the application.
552
579     virtual bool reportError(ErrorCode val, ErrorDesc desc) noexcept = 0;
580
597     virtual RefCount incRefCount() noexcept = 0;
598
615     virtual RefCount decRefCount() noexcept = 0;
616
617 protected:
618     // @cond SuppressDoxyWarnings
619     IErrorRecorder(IErrorRecorder const&) = default;
620     IErrorRecorder(IErrorRecorder&&) = default;
621     IErrorRecorder& operator=(IErrorRecorder const&) & = default;
622     IErrorRecorder& operator=(IErrorRecorder&&) & = default;
623     // @endcond
624 }; // class IErrorRecorder
625 } // namespace v_1_0
626
654 using IErrorRecorder = v_1_0::IErrorRecorder;
655
661 enum class TensorIOMode :  int32_t
662 {
664     kNONE = 0,
665
667     kINPUT = 1,
668
670     kOUTPUT = 2
671 };
672
673 namespace impl
674 {
676 template <>
677 struct EnumMaxImpl<TensorIOMode>
678 {
679     // One greater than the maximum value of TensorIOMode enum
680     static constexpr int32_t kVALUE = 3;
681 };
682 } // namespace impl
683 } // namespace nvinfer1
684
690 extern "C" TENSORRTAPI int32_t getInferLibVersion() noexcept;
691
692 #endif // NV_INFER_RUNTIME_BASE_H
```

10.17 NvInferRuntimeCommon.h File Reference

```
#include "NvInferPluginBase.h"
#include "NvInferRuntimePlugin.h"
```

Classes

- class nvinfer1::IPluginRegistry

  Single registration point for all plugins in an application. It is used to find plugin implementations during engine deserialization. Internally, the plugin registry is considered to be a singleton so all plugins in an application are part of the same global registry. Note that the plugin registry is only supported for plugins of type IPluginV2 and should also have a corresponding IPluginCreator implementation.

Namespaces

- namespace nvinfer1

  The TensorRT API version 1 namespace.

Macros

- #define NV_INFER_INTERNAL_INCLUDE 1

10.17.1 Detailed Description

This file provides the nvinfer1::IPluginRegistry interface, which will be moved to the NvInferRuntime.h header in a future release.

> Warning
>
> This file will be removed in a future release.
>
> Do not directly include this file. Instead include NvInferRuntime.h

10.17.2 Macro Definition Documentation

10.17.2.1 NV_INFER_INTERNAL_INCLUDE

```
#define NV_INFER_INTERNAL_INCLUDE 1
```

10.18 NvInferRuntimeCommon.h

Go to the documentation of this file.

```
1 /*
2 * SPDX-FileCopyrightText: Copyright (c) 1993-2026 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3 * SPDX-License-Identifier: Apache-2.0
4 *
5 * Licensed under the Apache License, Version 2.0 (the "License");
6 * you may not use this file except in compliance with the License.
7 * You may obtain a copy of the License at
8 *
9 * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17
18 #ifndef NV_INFER_RUNTIME_COMMON_H
19 #define NV_INFER_RUNTIME_COMMON_H
20
31 #define NV_INFER_INTERNAL_INCLUDE 1
32 #include "NvInferPluginBase.h"
33 #undef NV_INFER_INTERNAL_INCLUDE
34 #include "NvInferRuntimePlugin.h"
35
36 namespace nvinfer1
37 {
55 class IPluginRegistry
56 {
57 public:
61     using PluginLibraryHandle = void*;
62
63     // @cond SuppressDoxyWarnings
64     IPluginRegistry() = default;
65     IPluginRegistry(IPluginRegistry const&) = delete;
66     IPluginRegistry(IPluginRegistry&&) = delete;
67     IPluginRegistry& operator=(IPluginRegistry const&) & = delete;
68     IPluginRegistry& operator=(IPluginRegistry&&) & = delete;
69     // @endcond
70
71 protected:
72     virtual ~IPluginRegistry() noexcept = 0;
73
74 public:
91     virtual void setErrorRecorder(IErrorRecorder* const recorder) noexcept = 0;
92
108     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
109
118     virtual bool isParentSearchEnabled() const = 0;
119
127     virtual void setParentSearchEnabled(bool const enabled) = 0;
128
137     virtual PluginLibraryHandle loadLibrary(AsciiChar const* pluginPath) noexcept = 0;
138
145     virtual void deregisterLibrary(PluginLibraryHandle handle) noexcept = 0;
146
158     virtual bool registerCreator(IPluginCreatorInterface& creator, AsciiChar const* const pluginNamespace)
    noexcept = 0;
159
170     virtual IPluginCreatorInterface* const* getAllCreators(int32_t* const numCreators) const noexcept = 0;
171
183     virtual IPluginCreatorInterface* getCreator(AsciiChar const* const pluginName, AsciiChar const* const
    pluginVersion,
184         AsciiChar const* const pluginNamespace = "") noexcept = 0;
185
200     virtual bool deregisterCreator(IPluginCreatorInterface const& creator) noexcept = 0;
201
214     virtual IPluginResource* acquirePluginResource(AsciiChar const* key, IPluginResource* resource) noexcept
    = 0;
215
229     virtual int32_t releasePluginResource(AsciiChar const* key) noexcept = 0;
230
246     virtual IPluginCreatorInterface* const* getAllCreatorsRecursive(int32_t* const numCreators) noexcept =
    0;
247 };
248
```

```
249 inline IPluginRegistry::~IPluginRegistry() noexcept = default;
250
251 } // namespace nvinfer1
252
253 #endif /* NV_INFER_RUNTIME_COMMON_H */
```

10.19 NvInferRuntimePlugin.h File Reference

```
#include "NvInferPluginBase.h"
```

Classes

- struct nvinfer1::PluginTensorDesc

  Fields that a plugin might see for an input or output.
- class nvinfer1::IPluginV2

  Plugin class for user-implemented layers.
- class nvinfer1::IPluginV2Ext

  Plugin class for user-implemented layers.
- class nvinfer1::IPluginV2IOExt

  Plugin class for user-implemented layers.
- class nvinfer1::v_1_0::IPluginCreator

Namespaces

- namespace nvinfer1

  The TensorRT API version 1 namespace.
- namespace nvinfer1::v_1_0

  Forward declare IErrorRecorder and ILogger for use in other interfaces.

Macros

- #define NV_INFER_INTERNAL_INCLUDE 1

Typedefs

- using nvinfer1::PluginFormat = TensorFormat

  PluginFormat is reserved for backward compatibility.
- using nvinfer1::IPluginCreator = v_1_0::IPluginCreator

Enumerations

- enum class nvinfer1::PluginVersion : uint8_t {
  nvinfer1::kV2 = 0 , nvinfer1::kV2_EXT = 1 , nvinfer1::kV2_IOEXT = 2 , nvinfer1::kV2_DYNAMICEXT = 3 ,
  nvinfer1::kV2_DYNAMICEXT_PYTHON = kPLUGIN_VERSION_PYTHON_BIT | 3 }
- enum class nvinfer1::PluginCreatorVersion : int32_t { nvinfer1::kV1 = 0 , nvinfer1::kV1_PYTHON = k↩
  PLUGIN_VERSION_PYTHON_BIT }

  Enum to identify version of the plugin creator.

10.19.1 Detailed Description

This file contains common definitions, data structures and interfaces that relate to plugins and are shared between the standard and safe runtime.

> Warning
>
> Do not directly include this file. Instead include NvInferRuntime.h

10.19.2 Macro Definition Documentation

10.19.2.1 NV_INFER_INTERNAL_INCLUDE

```
#define NV_INFER_INTERNAL_INCLUDE 1
```

10.20 NvInferRuntimePlugin.h

Go to the documentation of this file.

```
1 /*
2 * SPDX-FileCopyrightText: Copyright (c) 1993-2026 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3 * SPDX-License-Identifier: Apache-2.0
4 *
5 * Licensed under the Apache License, Version 2.0 (the "License");
6 * you may not use this file except in compliance with the License.
7 * You may obtain a copy of the License at
8 *
9 * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17
18 #ifndef NV_INFER_RUNTIME_PLUGIN_H
19 #define NV_INFER_RUNTIME_PLUGIN_H
20
21 #define NV_INFER_INTERNAL_INCLUDE 1
22 #include "NvInferPluginBase.h"
23 #undef NV_INFER_INTERNAL_INCLUDE
24
33
39 namespace nvinfer1
```

```
40 {
41
42 enum class TensorFormat :  int32_t;
43 namespace v_1_0
44 {
45 class IGpuAllocator;
46 } // namespace v_1_0
47 using IGpuAllocator = v_1_0::IGpuAllocator;
48
54 using PluginFormat = TensorFormat;
55
59 static constexpr int32_t kPLUGIN_VERSION_PYTHON_BIT = 0x40;
60
72 struct PluginTensorDesc
73 {
75     Dims dims;
77     DataType type;
79     TensorFormat format;
81     float scale;
82 };
83
94 enum class PluginVersion :  uint8_t
95 {
97     kV2 TRT_DEPRECATED_ENUM = 0,
99     kV2_EXT TRT_DEPRECATED_ENUM = 1,
101     kV2_IOEXT TRT_DEPRECATED_ENUM = 2,
103     kV2_DYNAMICEXT TRT_DEPRECATED_ENUM = 3,
105     kV2_DYNAMICEXT_PYTHON TRT_DEPRECATED_ENUM = kPLUGIN_VERSION_PYTHON_BIT | 3
106 };
107
116 enum class PluginCreatorVersion :  int32_t
117 {
119     kV1 TRT_DEPRECATED_ENUM = 0,
121     kV1_PYTHON TRT_DEPRECATED_ENUM = kPLUGIN_VERSION_PYTHON_BIT
122 };
123
138 class TRT_DEPRECATED IPluginV2
139 {
140 public:
153     virtual int32_t getTensorRTVersion() const noexcept
154     {
155         return NV_TENSORRT_VERSION;
156     }
157
171     virtual AsciiChar const* getPluginType() const noexcept = 0;
172
186     virtual AsciiChar const* getPluginVersion() const noexcept = 0;
187
201     virtual int32_t getNbOutputs() const noexcept = 0;
202
226     virtual Dims getOutputDimensions(int32_t index, Dims const* inputs, int32_t nbInputDims) noexcept = 0;
227
251     virtual bool supportsFormat(DataType type, PluginFormat format) const noexcept = 0;
252
285     virtual void configureWithFormat(Dims const* inputDims, int32_t nbInputs, Dims const* outputDims, int32_t
    nbOutputs,
286         DataType type, PluginFormat format, int32_t maxBatchSize) noexcept
287         = 0;
288
300     virtual int32_t initialize() noexcept = 0;
301
315     virtual void terminate() noexcept = 0;
316
334     virtual size_t getWorkspaceSize(int32_t maxBatchSize) const noexcept = 0;
335
357     virtual int32_t enqueue(int32_t batchSize, void const* const* inputs, void* const* outputs, void*
    workspace,
358         cudaStream_t stream) noexcept
359         = 0;
360
371     virtual size_t getSerializationSize() const noexcept = 0;
372
386     virtual void serialize(void* buffer) const noexcept = 0;
387
396     virtual void destroy() noexcept = 0;
397
415     virtual IPluginV2* clone() const noexcept = 0;
416
431     virtual void setPluginNamespace(AsciiChar const* pluginNamespace) noexcept = 0;
432
444     virtual AsciiChar const* getPluginNamespace() const noexcept = 0;
```

```
445
446     // @cond SuppressDoxyWarnings
447     IPluginV2() = default;
448     virtual ~IPluginV2() noexcept = default;
449 // @endcond
450
451 protected:
452 // @cond SuppressDoxyWarnings
453     IPluginV2(IPluginV2 const&) = default;
454     IPluginV2(IPluginV2&&) = default;
455     IPluginV2& operator=(IPluginV2 const&) & = default;
456     IPluginV2& operator=(IPluginV2&&) & = default;
457 // @endcond
458 };
459
473 class TRT_DEPRECATED IPluginV2Ext :  public IPluginV2
474 {
475 public:
499     virtual nvinfer1::DataType getOutputDataType(
500         int32_t index, nvinfer1::DataType const* inputTypes, int32_t nbInputs) const noexcept
501         = 0;
502
538     virtual void configurePlugin(Dims const* inputDims, int32_t nbInputs, Dims const* outputDims, int32_t
    nbOutputs,
539         DataType const* inputTypes, DataType const* outputTypes, bool const* inputIsBroadcast,
540         bool const* outputIsBroadcast, PluginFormat floatFormat, int32_t maxBatchSize) noexcept
541         = 0;
542
543     IPluginV2Ext() = default;
544     ~IPluginV2Ext() override = default;
545
578     virtual void attachToContext(
579         cudnnContext* /*cudnn*/, cublasContext* /*cublas*/, IGpuAllocator* /*allocator*/) noexcept
580     {
581     }
582
596     virtual void detachFromContext() noexcept {}
597
612     IPluginV2Ext* clone() const noexcept override = 0;
613
614 protected:
615     // @cond SuppressDoxyWarnings
616     IPluginV2Ext(IPluginV2Ext const&) = default;
617     IPluginV2Ext(IPluginV2Ext&&) = default;
618     IPluginV2Ext& operator=(IPluginV2Ext const&) & = default;
619     IPluginV2Ext& operator=(IPluginV2Ext&&) & = default;
620 // @endcond
621
637     int32_t getTensorRTVersion() const noexceptoverride
638 {
639         return static_cast<int32_t>((static_cast<uint32_t>(PluginVersion::kV2_EXT) << 24U)
640             | (static_cast<uint32_t>(NV_TENSORRT_VERSION) & 0xFFFFFFU));
641     }
642
649     void configureWithFormat(Dims const* /*inputDims*/, int32_t /*nbInputs*/, Dims const* /*outputDims*/,
650         int32_t /*nbOutputs*/, DataType /*type*/, PluginFormat /*format*/, int32_t /*maxBatchSize*/) noexcept
    override
651     {
652     }
653 };
654
667 class TRT_DEPRECATED IPluginV2IOExt :  public IPluginV2Ext
668 {
669 public:
687     virtual void configurePlugin(
688         PluginTensorDesc const* in, int32_t nbInput, PluginTensorDesc const* out, int32_t nbOutput) noexcept
689         = 0;
690
728     virtual bool supportsFormatCombination(
729         int32_t pos, PluginTensorDesc const* inOut, int32_t nbInputs, int32_t nbOutputs) const noexcept
730         = 0;
731
732     // @cond SuppressDoxyWarnings
733     IPluginV2IOExt() = default;
734     ~IPluginV2IOExt() override = default;
735 // @endcond
736
737 protected:
738 // @cond SuppressDoxyWarnings
739     IPluginV2IOExt(IPluginV2IOExt const&) = default;
740     IPluginV2IOExt(IPluginV2IOExt&&) = default;
```

```
741     IPluginV2IOExt& operator=(IPluginV2IOExt const&) & = default;
742     IPluginV2IOExt& operator=(IPluginV2IOExt&&) & = default;
743 // @endcond
744
756     int32_t getTensorRTVersion() const noexceptoverride
757 {
758         return static_cast<int32_t>((static_cast<uint32_t>(PluginVersion::kV2_IOEXT) << 24U)
759             | (static_cast<uint32_t>(NV_TENSORRT_VERSION) & 0xFFFFFFU));
760     }
761
762 private:
763     // Following are obsolete base class methods, and must not be implemented or used.
764
768     void configurePlugin(Dims const*, int32_t, Dims const*, int32_t, DataType const*, DataType const*, bool
    const*,
769         bool const*, PluginFormat, int32_t) noexcept final
770     {
771     }
772
776     bool supportsFormat(DataType, PluginFormat) const noexcept final
777     {
778         return false;
779     }
780 };
781
782 namespace v_1_0
783 {
784 class TRT_DEPRECATED IPluginCreator :  public IPluginCreatorInterface
785 {
786 public:
799     virtual AsciiChar const* getPluginName() const noexcept = 0;
800
813     virtual AsciiChar const* getPluginVersion() const noexcept = 0;
814
826     virtual PluginFieldCollection const* getFieldNames() noexcept = 0;
827
840     virtual IPluginV2* createPlugin(AsciiChar const* name, PluginFieldCollection const* fc) noexcept = 0;
841
857     virtual IPluginV2* deserializePlugin(AsciiChar const* name, void const* serialData, size_t serialLength)
    noexcept
858         = 0;
859
874     virtual void setPluginNamespace(AsciiChar const* pluginNamespace) noexcept = 0;
875
888     virtual AsciiChar const* getPluginNamespace() const noexcept = 0;
889
890     IPluginCreator() = default;
891     ~IPluginCreator() override = default;
892
893 protected:
894     // @cond SuppressDoxyWarnings
895     IPluginCreator(IPluginCreator const&) = default;
896     IPluginCreator(IPluginCreator&&) = default;
897     IPluginCreator& operator=(IPluginCreator const&) & = default;
898     IPluginCreator& operator=(IPluginCreator&&) & = default;
899     // @endcond
900 public:
904     InterfaceInfo getInterfaceInfo() const noexceptoverride
905 {
906         return InterfaceInfo{"PLUGIN CREATOR_V1", 1, 0};
907     }
908 };
909 } // namespace v_1_0
910
921 using IPluginCreator = v_1_0::IPluginCreator;
922
923 } // namespace nvinfer1
924
925 #endif // NV_INFER_RUNTIME_PLUGIN_H
```

10.21 NvInferVersion.h File Reference

Macros

- #define TRT_MAJOR_ENTERPRISE 11

- #define TRT_MINOR_ENTERPRISE 0
- #define TRT_PATCH_ENTERPRISE 1
- #define TRT_BUILD_ENTERPRISE 7
- #define NV_TENSORRT_MAJOR TRT_MAJOR_ENTERPRISE

  TensorRT major version.
- #define NV_TENSORRT_MINOR TRT_MINOR_ENTERPRISE

  TensorRT minor version.
- #define NV_TENSORRT_PATCH TRT_PATCH_ENTERPRISE

  TensorRT patch version.
- #define NV_TENSORRT_BUILD TRT_BUILD_ENTERPRISE

  TensorRT build number.
- #define NV_TENSORRT_LWS_MAJOR 0

  TensorRT LWS major version.
- #define NV_TENSORRT_LWS_MINOR 0

  TensorRT LWS minor version.
- #define NV_TENSORRT_LWS_PATCH 0

  TensorRT LWS patch version.
- #define NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS 0

  An early access release.
- #define NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE 1

  A release candidate.
- #define NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY 2

  A final release.
- #define NV_TENSORRT_RELEASE_TYPE NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY

  TensorRT release type.

10.21.1 Detailed Description

Defines the TensorRT version

10.21.2 Macro Definition Documentation

10.21.2.1 NV_TENSORRT_BUILD

```
#define NV_TENSORRT_BUILD TRT_BUILD_ENTERPRISE
```

TensorRT build number.

10.21.2.2 NV_TENSORRT_LWS_MAJOR

```
#define NV_TENSORRT_LWS_MAJOR 0
```

TensorRT LWS major version.

10.21.2.3 NV_TENSORRT_LWS_MINOR

```
#define NV_TENSORRT_LWS_MINOR 0
```

TensorRT LWS minor version.

10.21.2.4 NV_TENSORRT_LWS_PATCH

```
#define NV_TENSORRT_LWS_PATCH 0
```

TensorRT LWS patch version.

10.21.2.5 NV_TENSORRT_MAJOR

```
#define NV_TENSORRT_MAJOR TRT_MAJOR_ENTERPRISE
```

TensorRT major version.

10.21.2.6 NV_TENSORRT_MINOR

```
#define NV_TENSORRT_MINOR TRT_MINOR_ENTERPRISE
```

TensorRT minor version.

10.21.2.7 NV_TENSORRT_PATCH

```
#define NV_TENSORRT_PATCH TRT_PATCH_ENTERPRISE
```

TensorRT patch version.

10.21.2.8 NV_TENSORRT_RELEASE_TYPE

```
#define NV_TENSORRT_RELEASE_TYPE NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY
```

TensorRT release type.

10.21.2.9 NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS

```
#define NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS 0
```

An early access release.

10.21.2.10 NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY

```
#define NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY 2
```

A final release.

10.21.2.11 NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE

```
#define NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE 1
```

A release candidate.

10.21.2.12 TRT_BUILD_ENTERPRISE

```
#define TRT_BUILD_ENTERPRISE 7
```

10.21.2.13 TRT_MAJOR_ENTERPRISE

```
#define TRT_MAJOR_ENTERPRISE 11
```

10.21.2.14 TRT_MINOR_ENTERPRISE

```
#define TRT_MINOR_ENTERPRISE 0
```

10.21.2.15 TRT_PATCH_ENTERPRISE

```
#define TRT_PATCH_ENTERPRISE 1
```

10.22 NvInferVersion.h

Go to the documentation of this file.

```
1 /*
2 * SPDX-FileCopyrightText: Copyright (c) 1993-2026 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3 * SPDX-License-Identifier: Apache-2.0
4 *
5 * Licensed under the Apache License, Version 2.0 (the "License");
6 * you may not use this file except in compliance with the License.
7 * You may obtain a copy of the License at
8 *
9 * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17
23 #ifndef NV_INFER_VERSION_H
24 #define NV_INFER_VERSION_H
25
26 #define TRT_MAJOR_ENTERPRISE 11
27 #define TRT_MINOR_ENTERPRISE 0
28 #define TRT_PATCH_ENTERPRISE 1
29 #define TRT_BUILD_ENTERPRISE 7
30 #define NV_TENSORRT_MAJOR TRT_MAJOR_ENTERPRISE
31 #define NV_TENSORRT_MINOR TRT_MINOR_ENTERPRISE
32 #define NV_TENSORRT_PATCH TRT_PATCH_ENTERPRISE
33 #define NV_TENSORRT_BUILD TRT_BUILD_ENTERPRISE
34
35 #define NV_TENSORRT_LWS_MAJOR 0
36 #define NV_TENSORRT_LWS_MINOR 0
37 #define NV_TENSORRT_LWS_PATCH 0
38
39 #define NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS 0
40 #define NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE 1
41 #define NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY 2
42
43 #define NV_TENSORRT_RELEASE_TYPE NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY
44
45 #endif // NV_INFER_VERSION_H
```

10.23 NvOnnxConfig.h File Reference

```
#include "NvInfer.h"
```

Classes

- class nvonnxparser::IOnnxConfig

  Configuration Manager Class.

Namespaces

- namespace nvonnxparser

  The TensorRT ONNX parser API namespace.

Functions

- TENSORRTAPI IOnnxConfig * nvonnxparser::createONNXConfig ()

10.23.1 Detailed Description

This is the API file for the Configuration Manager for ONNX Parser for Nvidia TensorRT.

10.24 NvOnnxConfig.h

Go to the documentation of this file.

```
1 /*
2 * SPDX-FileCopyrightText:  Copyright (c) 1993-2026 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3 * SPDX-License-Identifier:  Apache-2.0
4 *
5 * Licensed under the Apache License, Version 2.0 (the "License");
6 * you may not use this file except in compliance with the License.
7 * You may obtain a copy of the License at
8 *
9 * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17
18 #ifndef NV_ONNXCONFIG_H
19 #define NV_ONNXCONFIG_H
20
21 #include "NvInfer.h"
22
23 namespace nvonnxparser
24 {
25
35
41
46 class IOnnxConfig
47 {
48 public:
49     virtual ~IOnnxConfig() noexcept = 0;
55     typedef int32_t Verbosity;
56
65     virtual void setModelDtype(const nvinfer1::DataType) noexcept = 0;
66
74     virtual nvinfer1::DataType getModelDtype() const noexcept = 0;
75
83     virtual char const* getModelFileName() const noexcept = 0;
84
```

```
96      virtual void setModelFileName(char const* onnxFilename) noexcept = 0;
97
105      virtual Verbosity getVerbosityLevel() const noexcept = 0;
106
114      virtual void addVerbosity() noexcept = 0;
115
121      virtual void reduceVerbosity() noexcept = 0;
122
128      virtual void setVerbosityLevel(Verbosity) noexcept = 0;
129
138      virtual char const* getTextFileName() const noexcept = 0;
139
151      virtual void setTextFileName(char const* textFileName) noexcept = 0;
152
161      virtual char const* getFullTextFileName() const noexcept = 0;
162
174      virtual void setFullTextFileName(char const* fullTextFileName) noexcept = 0;
175
183      virtual bool getPrintLayerInfo() const noexcept = 0;
184
190      virtual void setPrintLayerInfo(bool) noexcept = 0;
191
192 }; // class IOnnxConfig
193
194 inline IOnnxConfig::~IOnnxConfig() noexcept = default;
195
196 TENSORRTAPI IOnnxConfig* createONNXConfig();
197
198 } // namespace nvonnxparser
199
200 #endif
```

10.25 NvOnnxParser.h File Reference

```
#include "NvInfer.h"
#include <stddef.h>
```

Classes

- class nvonnxparser::IParserError

  an object containing information about an error
- class nvonnxparser::IParser

  an object for parsing ONNX models into a TensorRT network definition
- class nvonnxparser::IParserRefitter

  An interface designed to refit weights from an ONNX model.

Namespaces

- namespace nvonnxparser

  The TensorRT ONNX parser API namespace.
- namespace nvonnxparser::anonymous_namespace{NvOnnxParser.h}

Macros

- #define NV_ONNX_PARSER_MAJOR 0
- #define NV_ONNX_PARSER_MINOR 1
- #define NV_ONNX_PARSER_PATCH 0

Typedefs

- using nvonnxparser::OnnxParserFlags = uint32_t

  Represents one or more OnnxParserFlag values using binary OR operations, e.g., 1U << OnnxParserFlag::kNATIVE_INSTANCENORM.

Enumerations

- enum class nvonnxparser::ErrorCode : int {
  nvonnxparser::kSUCCESS = 0 , nvonnxparser::kINTERNAL_ERROR = 1 , nvonnxparser::kMEM_ALLOC_FAILED = 2 , nvonnxparser::kMODEL_DESERIALIZE_FAILED = 3 , nvonnxparser::kINVALID_VALUE = 4 , nvonnxparser::kINVALID_GRAPH = 5 , nvonnxparser::kINVALID_NODE = 6 , nvonnxparser::kUNSUPPORTED_GRAPH = 7 , nvonnxparser::kUNSUPPORTED_NODE = 8 , nvonnxparser::kUNSUPPORTED_NODE_ATTR = 9 , nvonnxparser::kUNSUPPORTED_NODE_INPUT = 10 , nvonnxparser::kUNSUPPORTED_NODE_DATATYPE = 11 , nvonnxparser::kUNSUPPORTED_NODE_DYNAMIC = 12 , nvonnxparser::kUNSUPPORTED_NODE_SHAPE = 13 , nvonnxparser::kREFIT_FAILED = 14 }

  The type of error that the parser or refitter may return.
- enum class nvonnxparser::OnnxParserFlag : int32_t {
  nvonnxparser::kNATIVE_INSTANCENORM = 0 , nvonnxparser::kENABLE_UINT8_AND_ASYMMETRIC_QUANTIZATION_I = 1 , nvonnxparser::kREPORT_CAPABILITY_DLA = 2 , nvonnxparser::kENABLE_PLUGIN_OVERRIDE = 3 , nvonnxparser::kADJUST_FOR_DLA = 4 }

Functions

- template<typename T >
  constexpr int32_t nvonnxparser::EnumMax () noexcept=delete
- template<> constexpr int32_t nvonnxparser::EnumMax< ErrorCode > () noexcept

  Specialization. See `nvonnxparser::EnumMax()` for details.
- template<> constexpr int32_t nvonnxparser::EnumMax< OnnxParserFlag > () noexcept

  Specialization. See `nvonnxparser::EnumMax()` for details.
- TENSORRTAPI void * createNvOnnxParser_INTERNAL (void *network, void *logger, int version) noexcept
- TENSORRTAPI void * createNvOnnxParserRefitter_INTERNAL (void *refitter, void *logger, int32_t version) noexcept
- TENSORRTAPI int getNvOnnxParserVersion () noexcept
- IParser * nvonnxparser::anonymous_namespace{NvOnnxParser.h}::createParser (nvinfer1::INetworkDefinition &network, nvinfer1::ILogger &logger) noexcept

  Create a new parser object.
- IParserRefitter * nvonnxparser::anonymous_namespace{NvOnnxParser.h}::createParserRefitter (nvinfer1::IRefitter &refitter, nvinfer1::ILogger &logger) noexcept

  Create a new ONNX refitter object.

10.25.1 Detailed Description

This is the API for the ONNX Parser

10.25.2 Macro Definition Documentation

10.25.2.1 NV_ONNX_PARSER_MAJOR

```
#define NV_ONNX_PARSER_MAJOR 0
```

10.25.2.2 NV_ONNX_PARSER_MINOR

```
#define NV_ONNX_PARSER_MINOR 1
```

10.25.2.3 NV_ONNX_PARSER_PATCH

```
#define NV_ONNX_PARSER_PATCH 0
```

10.25.3 Function Documentation

10.25.3.1 createNvOnnxParser_INTERNAL()

```
TENSORRTAPI void * createNvOnnxParser_INTERNAL (
            void * network,
            void * logger,
            int version )  [noexcept]
```

10.25.3.2 createNvOnnxParserRefitter_INTERNAL()

```
TENSORRTAPI void * createNvOnnxParserRefitter_INTERNAL (
            void * refitter,
            void * logger,
            int32_t version )  [noexcept]
```

10.25.3.3 getNvOnnxParserVersion()

```
TENSORRTAPI int getNvOnnxParserVersion ( )  [noexcept]
```

10.26 NvOnnxParser.h

Go to the documentation of this file.

```
1 /*
2 * SPDX-FileCopyrightText:  Copyright (c) 1993-2025 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3 * SPDX-License-Identifier:  Apache-2.0
4 *
5 * Licensed under the Apache License, Version 2.0 (the "License");
6 * you may not use this file except in compliance with the License.
7 * You may obtain a copy of the License at
8 *
9 * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17
18 #ifndef NV_ONNX_PARSER_H
19 #define NV_ONNX_PARSER_H
20
21 #include "NvInfer.h"
22 #include <stddef.h>
23
29
30 #define NV_ONNX_PARSER_MAJOR 0
31 #define NV_ONNX_PARSER_MINOR 1
32 #define NV_ONNX_PARSER_PATCH 0
33
34 static constexpr int32_t NV_ONNX_PARSER_VERSION
35     = ((NV_ONNX_PARSER_MAJOR * 10000) + (NV_ONNX_PARSER_MINOR * 100) + NV_ONNX_PARSER_PATCH);
36
42 namespace nvonnxparser
43 {
44
50 template <typename T>
51 constexpr int32_t EnumMax() noexcept = delete;
52
58 enum class ErrorCode :  int
59 {
60     kSUCCESS = 0,
61     kINTERNAL_ERROR = 1,
62     kMEM_ALLOC_FAILED = 2,
63     kMODEL_DESERIALIZE_FAILED = 3,
64     kINVALID_VALUE = 4,
65     kINVALID_GRAPH = 5,
66     kINVALID_NODE = 6,
67     kUNSUPPORTED_GRAPH = 7,
68     kUNSUPPORTED_NODE = 8,
69     kUNSUPPORTED_NODE_ATTR = 9,
70     kUNSUPPORTED_NODE_INPUT = 10,
71     kUNSUPPORTED_NODE_DATATYPE = 11,
72     kUNSUPPORTED_NODE_DYNAMIC = 12,
73     kUNSUPPORTED_NODE_SHAPE = 13,
74     kREFIT_FAILED = 14
75 };
76
78 template <>
79 constexpr int32_t EnumMax<ErrorCode>() noexcept
80 {
81     return 14;
82 }
83
90 using OnnxParserFlags
91     = uint32_t;
92
93 enum class OnnxParserFlag :  int32_t
```

```
94 {
99     kNATIVE_INSTANCENORM = 0,
103     kENABLE_UINT8_AND_ASYMMETRIC_QUANTIZATION_DLA = 1,
107     // This flag is set to be OFF by default.
108     kREPORT_CAPABILITY_DLA = 2,
113     kENABLE_PLUGIN_OVERRIDE = 3,
115     kADJUST_FOR_DLA = 4
116 };
117
119 template <>
120 constexpr int32_t EnumMax<OnnxParserFlag>() noexcept
121 {
122     return 4;
123 }
124
130 class IParserError
131 {
132 public:
136     virtual ErrorCode code() const = 0;
140     virtual char const* desc() const = 0;
144     virtual char const* file() const = 0;
148     virtual int line() const = 0;
152     virtual char const* func() const = 0;
156     virtual int node() const = 0;
160     virtual char const* nodeName() const = 0;
164     virtual char const* nodeOperator() const = 0;
170     virtual char const* const* localFunctionStack() const = 0;
174     //              a stack size of 0.
176     virtual int32_t localFunctionStackSize() const = 0;
177
178 protected:
179     virtual ~IParserError() {}
180 };
181
192 class IParser
193 {
194 public:
209     virtual bool parse(
210         void const* serialized_onnx_model, size_t serialized_onnx_model_size, const char* model_path = nullptr)
      noexcept
211         = 0;
212
223     virtual bool parseFromFile(char const* onnxModelFile, int verbosity) noexcept = 0;
224
234     virtual bool supportsOperator(const char* op_name) const noexcept = 0;
235
242     virtual int getNbErrors() const noexcept = 0;
243
249     virtual IParserError const* getError(int index) const noexcept = 0;
250
256     virtual void clearErrors() noexcept = 0;
257
258     virtual ~IParser() noexcept = default;
259
276     virtual char const* const* getUsedVCPluginLibraries(int64_t& nbPluginLibs) const noexcept = 0;
277
289     virtual void setFlags(OnnxParserFlags onnxParserFlags) noexcept = 0;
290
298     virtual OnnxParserFlags getFlags() const noexcept = 0;
299
307     virtual void clearFlag(OnnxParserFlag onnxParserFlag) noexcept = 0;
308
316     virtual void setFlag(OnnxParserFlag onnxParserFlag) noexcept = 0;
317
325     virtual bool getFlag(OnnxParserFlag onnxParserFlag) const noexcept = 0;
326
339     virtual nvinfer1::ITensor const* getLayerOutputTensor(char const* name, int64_t i) noexcept = 0;
340
353     virtual bool supportsModelV2(
354         void const* serializedOnnxModel, size_t serializedOnnxModelSize, char const* modelPath = nullptr)
      noexcept = 0;
355
363     virtual int64_t getNbSubgraphs() noexcept = 0;
364
373     virtual bool isSubgraphSupported(int64_t const index) noexcept = 0;
374
385     virtual int64_t* getSubgraphNodes(int64_t const index, int64_t& subgraphLength) noexcept = 0;
386
403     virtual bool loadModelProto(
404         void const* serializedOnnxModel, size_t serializedOnnxModelSize, char const* modelPath = nullptr)
      noexcept = 0;
```

```
405
428     virtual bool loadInitializer(char const* name, void const* data, size_t size) noexcept = 0;
429
435     virtual bool parseModelProto() noexcept = 0;
436
442     virtual bool setBuilderConfig(const nvinfer1::IBuilderConfig* const builderConfig) noexcept = 0;
443 };
444
452 class IParserRefitter
453 {
454 public:
467     virtual bool refitFromBytes(
468         void const* serializedOnnxModel, size_t serializedOnnxModelSize, char const* modelPath = nullptr)
      noexcept
469         = 0;
470
481     virtual bool refitFromFile(char const* onnxModelFile) noexcept = 0;
482
488     virtual int32_t getNbErrors() const noexcept = 0;
489
495     virtual IParserError const* getError(int32_t index) const noexcept = 0;
496
502     virtual void clearErrors() = 0;
503
504     virtual ~IParserRefitter() noexcept = default;
505
521     virtual bool loadModelProto(
522         void const* serializedOnnxModel, size_t serializedOnnxModelSize, char const* modelPath = nullptr)
      noexcept = 0;
523
545     virtual bool loadInitializer(char const* name, void const* data, size_t size) noexcept = 0;
546
552     virtual bool refitModelProto() noexcept = 0;
553 };
554
555 } // namespace nvonnxparser
556
557 extern "C" TENSORRTAPI void* createNvOnnxParser_INTERNAL(void* network, void* logger, int version) noexcept;
558 extern "C" TENSORRTAPI void* createNvOnnxParserRefitter_INTERNAL(
559     void* refitter, void* logger, int32_t version) noexcept;
560 extern "C" TENSORRTAPI int getNvOnnxParserVersion() noexcept;
561
562 namespace nvonnxparser
563 {
564
565 namespace
566 {
567
583 inline IParser* createParser(nvinfer1::INetworkDefinition& network, nvinfer1::ILogger& logger) noexcept
584 {
585     return static_cast<IParser*>(createNvOnnxParser_INTERNAL(&network, &logger, NV_ONNX_PARSER_VERSION));
586 }
587
597 inline IParserRefitter* createParserRefitter(nvinfer1::IRefitter& refitter, nvinfer1::ILogger& logger)
      noexcept
598 {
599     return static_cast<IParserRefitter*>(
600         createNvOnnxParserRefitter_INTERNAL(&refitter, &logger, NV_ONNX_PARSER_VERSION));
601 }
602
603 } // namespace
604
605 } // namespace nvonnxparser
606
607 #endif // NV_ONNX_PARSER_H
```

10.27 NvInferConsistency.h File Reference

```
#include "NvInferForwardDecl.h"
#include "NvInferPluginBase.h"
#include "NvInferSafeRuntime.h"
#include <vector>
```

Classes

- class nvinfer2::safe::consistency::IConsistencyChecker

  Validates a TRT engine (Serialized binary blob)
- class nvinfer2::safe::consistency::IPluginChecker

  Interface for validating a plugin used in a TensorRT engine.

Namespaces

- namespace nvinfer2
- namespace nvinfer2::safe
- namespace nvinfer2::safe::consistency

Functions

- nvinfer2::safe::ErrorCode nvinfer2::safe::consistency::createConsistencyChecker (IConsistencyChecker ∗&icc, nvinfer2::safe::ISafeRecorder &recorder, void const ∗blob, size_t size, std::vector< std::string > const &pluginBuildLibs={}) noexcept

  Create an instance of an IConsistencyChecker class.

10.28 NvInferConsistency.h

Go to the documentation of this file.

```
1 /*
2 * SPDX-FileCopyrightText:  Copyright (c) 1993-2026 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3 * SPDX-License-Identifier:  Apache-2.0
4 *
5 * Licensed under the Apache License, Version 2.0 (the "License");
6 * you may not use this file except in compliance with the License.
7 * You may obtain a copy of the License at
8 *
9 * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17
18 #ifndef NV_INFER_CONSISTENCY_CHECKER_H
19 #define NV_INFER_CONSISTENCY_CHECKER_H
20 #include "NvInferForwardDecl.h"
21 #include "NvInferPluginBase.h"
22 #include "NvInferSafeRuntime.h"
23 #include <vector>
24
28
29 namespace nvinfer2
30 {
31 namespace safe
32 {
33 // Forward declaration
34 class ISafeRecorder;
35
36 namespace consistency
37 {
38
46 class IConsistencyChecker
47 {
48 public:
```

```
54      virtual bool validate() noexcept = 0;
58      virtual ~IConsistencyChecker() = default;
59
60  protected:
69      IConsistencyChecker() = default;
75      IConsistencyChecker(IConsistencyChecker const& other) = delete;
81      IConsistencyChecker& operator=(IConsistencyChecker const& other) & = delete;
87      IConsistencyChecker(IConsistencyChecker&& other) = delete;
93      IConsistencyChecker& operator=(IConsistencyChecker&& other) & = delete;
94  };
95
104 class IPluginChecker
105 {
106 public:
114     virtual bool validate(std::vector<TensorDescriptor> const& Inputs, std::vector<TensorDescriptor> const&
        Outputs,
115         PluginFieldCollection* fc) noexcept = 0;
116
120     virtual ~IPluginChecker() = default;
121
122 protected:
131     IPluginChecker() = default;
137     IPluginChecker(IPluginChecker const& other) = delete;
143     IPluginChecker& operator=(IPluginChecker const& other) & = delete;
149     IPluginChecker(IPluginChecker&& other) = delete;
155     IPluginChecker& operator=(IPluginChecker&& other) & = delete;
156 };
157
163 extern "C" nvinfer2::safe::ErrorCode createConsistencyChecker(IConsistencyChecker*& icc,
164     nvinfer2::safe::ISafeRecorder& recorder, void const* blob, size_t size,
165     std::vector<std::string> const& pluginBuildLibs = {}) noexcept;
166
167 } // namespace consistency
168
169 } // namespace safe
170 } // namespace nvinfer2
171
172 #endif // NV_INFER_CONSISTENCY_CHECKER_H
```

10.29 NvInferForwardDecl.h File Reference

```
#include "NvInferPluginBase.h"
#include "NvInferRuntimeBase.h"
```

Namespaces

- namespace nvinfer2
- namespace nvinfer2::safe
- namespace nvinfer2::safe::impl

Macros

- #define NV_INFER_INTERNAL_INCLUDE 1

  Allow direct inclusion of TRT Runtime Base header.

Typedefs

- using nvinfer2::safe::Dims = nvinfer1::Dims
- using nvinfer2::safe::IVersionedInterface = nvinfer1::IVersionedInterface
- using nvinfer2::safe::AsciiChar = nvinfer1::AsciiChar
- using nvinfer2::safe::InterfaceInfo = nvinfer1::InterfaceInfo
- using nvinfer2::safe::DataType = nvinfer1::DataType
- using nvinfer2::safe::TensorIOMode = nvinfer1::TensorIOMode
- using nvinfer2::safe::ErrorCode = nvinfer1::ErrorCode
- using nvinfer2::safe::IErrorRecorder = nvinfer1::IErrorRecorder
- using nvinfer2::safe::IPluginResource = nvinfer1::IPluginResource
- using nvinfer2::safe::IPluginCreatorInterface = nvinfer1::IPluginCreatorInterface
- using nvinfer2::safe::IPluginV3 = nvinfer1::IPluginV3
- using nvinfer2::safe::IPluginCapability = nvinfer1::IPluginCapability
- using nvinfer2::safe::PluginField = nvinfer1::PluginField
- using nvinfer2::safe::PluginFieldCollection = nvinfer1::PluginFieldCollection
- using nvinfer2::safe::TensorRTPhase = nvinfer1::TensorRTPhase

Functions

- template<typename T >
  constexpr int32_t nvinfer2::safe::EnumMax () noexcept

  Maximum number of elements in an enumeration type.

10.29.1 Macro Definition Documentation

10.29.1.1 NV_INFER_INTERNAL_INCLUDE

```
#define NV_INFER_INTERNAL_INCLUDE 1
```

Allow direct inclusion of TRT Runtime Base header.

10.30 NvInferForwardDecl.h

Go to the documentation of this file.

```
1 /*
2 * SPDX-FileCopyrightText:  Copyright (c) 1993-2025 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3 * SPDX-License-Identifier:  Apache-2.0
4 *
5 * Licensed under the Apache License, Version 2.0 (the "License");
6 * you may not use this file except in compliance with the License.
7 * You may obtain a copy of the License at
8 *
9 * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
```

```
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17 #ifndef NV_INFER_FORWARD_DECL_H
18 #define NV_INFER_FORWARD_DECL_H
19
21 #define NV_INFER_INTERNAL_INCLUDE 1
22 #pragma GCC diagnostic push
23 #pragma GCC diagnostic ignored "-Wdeprecated-declarations"
24 #include "NvInferPluginBase.h"
25 #include "NvInferRuntimeBase.h"
26 #undef NV_INFER_INTERNAL_INCLUDE
27
28 #pragma GCC diagnostic pop
29
30 namespace nvinfer2
31 {
32 namespace safe
33 {
34 // NvInferRuntimeBase.h classes
35 using Dims = nvinfer1::Dims;
36 using IVersionedInterface = nvinfer1::IVersionedInterface;
37 using AsciiChar = nvinfer1::AsciiChar;
38 using InterfaceInfo = nvinfer1::InterfaceInfo;
39 using DataType = nvinfer1::DataType;
40 using TensorIOMode = nvinfer1::TensorIOMode;
41 using ErrorCode = nvinfer1::ErrorCode;
42 using IErrorRecorder = nvinfer1::IErrorRecorder;
43
44 // NvInferPluginBase.h classes
45 using IPluginResource = nvinfer1::IPluginResource;
46 using IPluginCreatorInterface = nvinfer1::IPluginCreatorInterface;
47 using IPluginV3 = nvinfer1::IPluginV3;
48 using IPluginCapability = nvinfer1::IPluginCapability;
49 using PluginField = nvinfer1::PluginField;
50 using PluginFieldCollection = nvinfer1::PluginFieldCollection;
51 using TensorRTPhase = nvinfer1::TensorRTPhase;
52
53 // Re implement EnumMax in the nvinfer2::safe namespace
54 namespace impl
55 {
57 template <typename T>
58 struct EnumMaxImpl;
59 } // namespace impl
60
62 template <typename T>
63 constexpr int32_t EnumMax() noexcept
64 {
65     return impl::EnumMaxImpl<T>::kVALUE;
66 }
67
68 } // namespace safe
69 } // namespace nvinfer2
70
71 #endif /* NV_INFER_FORWARD_DECL_H */
```

10.31 NvInferSafeMemAllocator.h File Reference

```
#include "NvInferSafeRecorder.h"
#include <cstdint>
```

Classes

- class nvinfer2::safe::ISafeMemAllocator

  Application-implemented class for controlling memory allocation on the GPU/CPU.

Namespaces

- namespace nvinfer2
- namespace nvinfer2::safe

Enumerations

- enum class nvinfer2::safe::MemoryPlacement : uint32_t { nvinfer2::safe::kNONE = 0x0 , nvinfer2::safe::kMANAGED = 0x1 , nvinfer2::safe::kGPU = 0x2 , nvinfer2::safe::kCPU = 0x4 , nvinfer2::safe::kCPU_PINNED = 0x80 }

  Enum to describe the placement of the memory region.
- enum class nvinfer2::safe::MemoryUsage : uint32_t { nvinfer2::safe::kGENERIC = 0x0 , nvinfer2::safe::kIMMUTABLE = 0x3 , nvinfer2::safe::kSCRATCH = 0x5 , nvinfer2::safe::kIOTENSOR = 0x6 }

  Enum to describe the usage of memory region.

Variables

- constexpr uint64_t nvinfer2::safe::kMAXIMUM_ALLOC_SIZE {17179869184U}

  Legacy constant (16 GiB) kept for TensorRT excessiveMemoryDetection test build compatibility.

10.32 NvInferSafeMemAllocator.h

Go to the documentation of this file.

```
1 /*
2 * SPDX-FileCopyrightText:  Copyright (c) 1993-2026 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3 * SPDX-License-Identifier:  Apache-2.0
4 *
5 * Licensed under the Apache License, Version 2.0 (the "License");
6 * you may not use this file except in compliance with the License.
7 * You may obtain a copy of the License at
8 *
9 * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17
18 // Header file for the NVIDIA Safe Runtime Memory Allocator interface.
19 // This file provides the user-implementable interface for the Memory Allocator feature in the NVIDIA Safe
      Runtime API.
20 // It includes the necessary classes, functions, and definitions for creating and managing safe memory
      allocators,
21 // including the ISafeMemAllocator interface.
22 // Users should implement this interface to provide a custom memory allocator that will be used by the Safe
      Runtime
23 // to allocate memory for both CPU and GPU resources.  The allocator should handle memory allocation and
      deallocation
24 // requests from the Safe Runtime, and ensure thread safety and proper memory management.
25 // The implemented allocator will be responsible for allocating memory for various components of the Safe
      Runtime,
26 // including tensors, plugins, and other internal data structures.
27
28 #ifndef NV_INFER_SAFE_MEM_ALLOCATOR_H
29 #define NV_INFER_SAFE_MEM_ALLOCATOR_H
30 #include "NvInferSafeRecorder.h"
31 #include <cstdint>
```

```
32
33 namespace nvinfer2
34 {
35 namespace safe
36 {
37 class ISafeRecorder;
38
42 constexpr uint64_t kMAXIMUM_ALLOC_SIZE{17179869184U};
43
48 enum class MemoryPlacement :  uint32_t
49 {
51     kNONE = 0x0,
53     kMANAGED = 0x1,
55     kGPU = 0x2,
57     kCPU = 0x4,
59     kCPU_PINNED = 0x80
60 };
61
66 enum class MemoryUsage :  uint32_t
67 {
69     kGENERIC = 0x0,
71     kIMMUTABLE = 0x3,
73     kSCRATCH = 0x5,
75     kIOTENSOR = 0x6
76 };
77
85 class ISafeMemAllocator
86 {
87 public:
120     virtual void* allocate(uint64_t const size, uint64_t const alignment, MemoryPlacement const flags,
121         MemoryUsage const usage, ISafeRecorder& recorder) noexcept = 0;
122
128     virtual ~ISafeMemAllocator() = default;
134     ISafeMemAllocator() = default;
135
164     virtual bool deallocate(void* const memory, MemoryPlacement const flags, ISafeRecorder& recorder)
      noexcept = 0;
165
166 protected:
172     ISafeMemAllocator(ISafeMemAllocator const&) = default;
178     ISafeMemAllocator(ISafeMemAllocator&&) = default;
184     ISafeMemAllocator& operator=(ISafeMemAllocator const&) & = default;
190     ISafeMemAllocator& operator=(ISafeMemAllocator&&) & = default;
191 };
192 } // namespace safe
193 } // namespace nvinfer2
194
195 #endif /* NV_INFER_SAFE_MEM_ALLOCATOR_H */
```

10.33 NvInferSafePlugin.h File Reference

```
#include "NvInferForwardDecl.h"
#include "NvInferPluginBase.h"
#include "NvInferSafeMemAllocator.h"
#include "NvInferSafeRecorder.h"
```

Classes

- class nvinfer2::safe::ISafePluginResourceContext

  Interface for plugins to access per context resources provided by TensorRT.
- class nvinfer2::safe::ISafePluginRegistry

  PluginRegistry interface class that contains all the methods user can use to interface with the registry object.
- class nvinfer2::safe::IPluginV3OneSafeCore

  Interface specifying the base plugin capability.

- class nvinfer2::safe::IPluginV3OneSafeBuild

  Interface for plugins used during the TensorRT build phase.
- class nvinfer2::safe::IPluginV3OneSafeBuildMSS

  This class provides build capability and let output shapes be calculated by symbolic expression to support multiple static shape.
- class nvinfer2::safe::IPluginV3OneSafeRuntime

  Runtime component of the pluginV3 system.
- class nvinfer2::safe::ISafePluginCreatorV3One

  The main interface to be implemented by all plugin creator classes.

Namespaces

- namespace nvinfer1

  The TensorRT API version 1 namespace.
- namespace nvinfer2
- namespace nvinfer2::safe

Functions

- nvinfer2::safe::ISafePluginRegistry * nvinfer2::safe::getSafePluginRegistry (nvinfer2::safe::ISafeRecorder &recorder) noexcept

  Get the SafePluginRegistry singleton instance and set the SafeRecorder of the safePluginRegistry.

10.34 NvInferSafePlugin.h

Go to the documentation of this file.

```
1 /*
2 * SPDX-FileCopyrightText:  Copyright (c) 1993-2026 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3 * SPDX-License-Identifier:  Apache-2.0
4 *
5 * Licensed under the Apache License, Version 2.0 (the "License");
6 * you may not use this file except in compliance with the License.
7 * You may obtain a copy of the License at
8 *
9 * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17
18 // Header file for the NVIDIA Safe Runtime PluginV3 interface.
19 // This file provides the user-implementable interface for the PluginV3 feature in the NVIDIA Safe Runtime.
20 // It includes the necessary classes, functions, and definitions for creating and managing safe plugins,
21 // including the ISafePluginResourceContext, ISafePluginRegistry, IPluginV3OneSafeCore,
      IPluginV3OneSafeBuild,
22 // IPluginV3OneSafeRuntime, and ISafePluginCreatorV3One interfaces.
23 // Users should implement these interfaces to create custom plugins that can be used with in the Safe
      Runtime.
24
25 #ifndef NV_INFER_SAFE_PLUGIN_H
26 #define NV_INFER_SAFE_PLUGIN_H
27 #include "NvInferForwardDecl.h"
28 #include "NvInferPluginBase.h"
29 #include "NvInferSafeMemAllocator.h"
30 #include "NvInferSafeRecorder.h"
```

```
31
32 namespace nvinfer1
33 {
34 class DimsExprs;
35 class IExprBuilder;
36 } // namespace nvinfer1
37
38 namespace nvinfer2
39 {
40 namespace safe
41 {
42 // Forward declaration of TensorDescriptor and RuntimeErrorInformation for the plugin interface classes
43 struct RuntimeErrorInformation;
44 struct TensorDescriptor;
45
60 class ISafePluginResourceContext
61 {
62 public:
73     virtual ISafeMemAllocator* getSafeMemAllocator() const noexcept = 0;
74
85     virtual ISafeRecorder* getSafeRecorder() const noexcept = 0;
86
97     virtual RuntimeErrorInformation* getRuntimeErrorInformation() const noexcept = 0;
98
105     virtual ~ISafePluginResourceContext() noexcept = default;
106
107 protected:
114     ISafePluginResourceContext() = default;
121     ISafePluginResourceContext(ISafePluginResourceContext const&) = default;
128     ISafePluginResourceContext(ISafePluginResourceContext&&) = default;
135     ISafePluginResourceContext& operator=(ISafePluginResourceContext const&) & = default;
142     ISafePluginResourceContext& operator=(ISafePluginResourceContext&&) & = default;
143 };
144
147 //
151 class ISafePluginRegistry
152 {
153 public:
178     virtual ErrorCode registerCreator(
179         IPluginCreatorInterface& creator, AsciiChar const* const pluginNamespace, ISafeRecorder& recorder)
      noexcept = 0;
180
202     virtual ErrorCode getAllCreators(
203         IPluginCreatorInterface* const*& creators, int32_t& numCreators) const noexcept = 0;
204
230     virtual ErrorCode getCreator(IPluginCreatorInterface*& creator, AsciiChar const* const pluginName,
231         AsciiChar const* const pluginVersion, AsciiChar const* const pluginNamespace = "") noexcept = 0;
232
258     virtual ErrorCode setSafeRecorder(ISafeRecorder& recorder) noexcept = 0;
259
280     virtual ErrorCode getSafeRecorder(ISafeRecorder*& recorder) const noexcept = 0;
281
301     virtual ErrorCode deregisterCreator(IPluginCreatorInterface const& creator) noexcept = 0;
302
309     ISafePluginRegistry() = default;
315     ISafePluginRegistry(ISafePluginRegistry const&) = delete;
321     ISafePluginRegistry(ISafePluginRegistry&&) = delete;
327     ISafePluginRegistry& operator=(ISafePluginRegistry const&) & = delete;
333     ISafePluginRegistry& operator=(ISafePluginRegistry&&) & = delete;
334
335 protected:
337     virtual ~ISafePluginRegistry() noexcept = default;
338 };
339
345 class IPluginV3OneSafeCore :  public IPluginCapability
346 {
347 public:
353     InterfaceInfo getInterfaceInfo() const noexceptoverride
354 {
355         return InterfaceInfo{"PLUGIN_V3ONE_SAFE_CORE", 1, 0};
356     }
357
359     IPluginV3OneSafeCore(IPluginV3OneSafeCore const&) = default;
360
362     IPluginV3OneSafeCore(IPluginV3OneSafeCore&&) = default;
363
365     IPluginV3OneSafeCore& operator=(IPluginV3OneSafeCore const&) & = default;
366
368     IPluginV3OneSafeCore& operator=(IPluginV3OneSafeCore&&) & = default;
369
371     ~IPluginV3OneSafeCore() noexcept override = default;
```

```
372
383     virtual AsciiChar const* getPluginName() const noexcept = 0;
384
395     virtual AsciiChar const* getPluginVersion() const noexcept = 0;
396
408     virtual AsciiChar const* getPluginNamespace() const noexcept = 0;
409
410 protected:
412     IPluginV3OneSafeCore() = default;
413 };
414
421 class IPluginV3OneSafeBuild :  public IPluginCapability
422 {
423 public:
428     static constexpr int32_t kDEFAULT_FORMAT_COMBINATION_LIMIT = 100;
429
437     InterfaceInfo getInterfaceInfo() const noexceptoverride
438 {
439         return InterfaceInfo{"PLUGIN_V3ONE_SAFE_BUILD", 1, 0};
440     }
441
447     IPluginV3OneSafeBuild() = default;
453     IPluginV3OneSafeBuild(IPluginV3OneSafeBuild&&) = delete;
459     IPluginV3OneSafeBuild& operator=(IPluginV3OneSafeBuild&&) & = delete;
465     ~IPluginV3OneSafeBuild() noexcept override = default;
466
484     virtual int32_t configurePlugin(
485         TensorDescriptor const* in, int32_t nbInputs, TensorDescriptor const* out, int32_t nbOutputs)
    noexcept = 0;
486
503     virtual int32_t getOutputDataTypes(
504         DataType* outputTypes, int32_t nbOutputs, DataType const* inputTypes, int32_t nbInputs) const
    noexcept
505         = 0;
506
527     virtual int32_t getOutputShapes(
528         Dims const* inputs, int32_t nbInputs, Dims* outputs, int32_t nbOutputs) const noexcept = 0;
529
564     virtual bool supportsFormatCombination(
565         int32_t pos, TensorDescriptor const* inOut, int32_t nbInputs, int32_t nbOutputs) noexcept = 0;
566
572     virtual int32_t getNbOutputs() const noexcept = 0;
573
583     virtual size_t getWorkspaceSize(TensorDescriptor const* /*inputs*/, int32_t /*nbInputs*/,
584         TensorDescriptor const* /*outputs*/, int32_t /*nbOutputs*/) const noexcept
585     {
586         return 0U;
587     }
588
620     virtual int32_t getValidTactics(int32_t* /*tactics*/, int32_t /*nbTactics*/) noexcept
621     {
622         return 0;
623     }
624
628     virtual int32_t getNbTactics() noexcept
629     {
630         return 0;
631     }
632
644     virtual char const* getTimingCacheID() noexcept
645     {
646         return nullptr;
647     }
648
652     virtual int32_t getFormatCombinationLimit() noexcept
653     {
654         return kDEFAULT_FORMAT_COMBINATION_LIMIT;
655     }
656
663     virtual char const* getMetadataString() noexcept
664     {
665         return nullptr;
666     }
667
668 protected:
669     IPluginV3OneSafeBuild(IPluginV3OneSafeBuild const&) = default;
670     IPluginV3OneSafeBuild& operator=(IPluginV3OneSafeBuild const&) & = default;
671 };
672
679 class IPluginV3OneSafeBuildMSS :  public IPluginV3OneSafeBuild
680 {
```

```
681 public:
685     InterfaceInfo getInterfaceInfo() const noexceptoverride
686 {
687         return InterfaceInfo{"PLUGIN_V3ONE_SAFE_BUILDMSS", 1, 0};
688     }
689 
690     int32_t getOutputShapes(
691         Dims const* /*inputs*/, int32_t /*nbInputs*/, Dims* /*outputs*/, int32_t /*nbOutputs*/) const
    noexcept override
692     {
693         // This function will not be used, instead, getSymbolicOutputShapes should be used.
694         return 0;
695     }
696 
698     IPluginV3OneSafeBuildMSS(IPluginV3OneSafeBuildMSS const&) = default;
699 
701     IPluginV3OneSafeBuildMSS(IPluginV3OneSafeBuildMSS&&) = delete;
702 
704     IPluginV3OneSafeBuildMSS& operator=(IPluginV3OneSafeBuildMSS const&) & = default;
705 
707     IPluginV3OneSafeBuildMSS& operator=(IPluginV3OneSafeBuildMSS&&) & = delete;
708 
710     ~IPluginV3OneSafeBuildMSS() noexcept override = default;
725     virtual int32_t getSymbolicOutputShapes(nvinfer1::DimsExprs const* inputs, int32_t nbInputs,
726         nvinfer1::DimsExprs* outputs, int32_t nbOutputs, nvinfer1::IExprBuilder& exprBuilder) const noexcept
    = 0;
727 
728 protected:
730     IPluginV3OneSafeBuildMSS() = default;
731 };
732 
735 //
738 class IPluginV3OneSafeRuntime :  public IPluginCapability
739 {
740 public:
746     InterfaceInfo getInterfaceInfo() const noexceptoverride
747 {
748         return InterfaceInfo{"PLUGIN_V3ONE_SAFE_RUNTIME", 1, 0};
749     }
750 
752     IPluginV3OneSafeRuntime(IPluginV3OneSafeRuntime const&) = default;
753 
755     IPluginV3OneSafeRuntime(IPluginV3OneSafeRuntime&&) = default;
756 
758     IPluginV3OneSafeRuntime& operator=(IPluginV3OneSafeRuntime const&) & = default;
759 
761     IPluginV3OneSafeRuntime& operator=(IPluginV3OneSafeRuntime&&) & = default;
762 
764     ~IPluginV3OneSafeRuntime() noexcept override = default;
765 
775     virtual int32_t setTactic(int32_t /*tactic*/) noexcept
776     {
777         return 0;
778     }
779 
795     virtual int32_t enqueue(TensorDescriptor const* inputDesc, TensorDescriptor const* outputDesc,
796         void const* const* inputs, void* const* outputs, void* workspace, cudaStream_t stream) noexcept = 0;
797 
812     virtual int32_t initResource(ISafePluginResourceContext const* context) noexcept = 0;
813 
821     virtual IPluginV3* clone() noexcept = 0;
822 
830 
834     virtual PluginFieldCollection const* getFieldsToSerialize() noexcept = 0;
835 
836 protected:
838     IPluginV3OneSafeRuntime() = default;
839 };
840 
843 //
846 class ISafePluginCreatorV3One :  public IPluginCreatorInterface
847 {
848 public:
854     InterfaceInfo getInterfaceInfo() const noexceptoverride
855 {
856         return InterfaceInfo{"PLUGIN SAFE CREATOR_V3ONE", 1, 0};
857     }
858 
873     virtual IPluginV3* createPlugin(
874         AsciiChar const* name, PluginFieldCollection const* fc, TensorRTPhase phase) noexcept = 0;
875 
```

```
884     virtual PluginFieldCollection const* getFieldNames() noexcept = 0;
885
894     virtual AsciiChar const* getPluginName() const noexcept = 0;
895
904     virtual AsciiChar const* getPluginVersion() const noexcept = 0;
905
914     virtual AsciiChar const* getPluginNamespace() const noexcept = 0;
915
921     virtual ISafeRecorder* getSafeRecorder() const noexcept = 0;
922
928     virtual void setSafeRecorder(ISafeRecorder&) noexcept = 0;
929
930     ISafePluginCreatorV3One() = default;
931     ~ISafePluginCreatorV3One() noexcept override = default;
932
933 protected:
934     ISafePluginCreatorV3One(ISafePluginCreatorV3One const&) = default;
935     ISafePluginCreatorV3One(ISafePluginCreatorV3One&&) = default;
936     ISafePluginCreatorV3One& operator=(ISafePluginCreatorV3One const&) & = default;
937     ISafePluginCreatorV3One& operator=(ISafePluginCreatorV3One&&) & = default;
938 };
939
956 extern "C" nvinfer2::safe::ISafePluginRegistry* getSafePluginRegistry(nvinfer2::safe::ISafeRecorder&
      recorder) noexcept;
957 } // namespace safe
958 } // namespace nvinfer2
959
960 #endif /* NV_INFER_SAFE_PLUGIN_H */
```

10.35 NvInferSafeRecorder.h File Reference

```
#include <cstdint>
#include "NvInferForwardDecl.h"
```

Classes

- struct nvinfer2::safe::impl::EnumMaxImpl< Severity >

  Provides a static constant for the maximum value of the Severity enum.
- class nvinfer2::safe::ISafeRecorder

  Interface for extended recorder which allows error, warn, debug, or info messages to be recorded.
- struct nvinfer2::safe::RuntimeErrorInformation

  Holds information about runtime errors that occur during asynchronous kernel execution.

Namespaces

- namespace nvinfer2
- namespace nvinfer2::safe
- namespace nvinfer2::safe::impl

Typedefs

- using nvinfer2::safe::ErrorDesc = char const *

  Type alias for error description strings.
- using nvinfer2::safe::RefCount = int32_t

  Type alias for reference count.

Enumerations

- enum class nvinfer2::safe::Severity : uint32_t {
  nvinfer2::safe::kINTERNAL_ERROR = 0 , nvinfer2::safe::kERROR = 1 , nvinfer2::safe::kWARNING = 2 ,
  nvinfer2::safe::kINFO = 3 ,
  nvinfer2::safe::kVERBOSE = 4 , nvinfer2::safe::kDEBUG = 5 }

  Enumerates severity levels for messages issued by the message recorder.
- enum class nvinfer2::safe::RuntimeErrorType : uint64_t {
  nvinfer2::safe::kNAN_CONSUMED = 1ULL << 0 , nvinfer2::safe::kINF_CONSUMED = 1ULL << 1 ,
  nvinfer2::safe::kGATHER_OOB = 1ULL << 2 , nvinfer2::safe::kSCATTER_OOB = 1ULL << 3 ,
  nvinfer2::safe::kSCATTER_RACE = 1ULL << 4 , nvinfer2::safe::kDIV_ZERO = 1ULL << 5 }

  Enumerates types of runtime errors that can occur during kernel execution.

10.36 NvInferSafeRecorder.h

Go to the documentation of this file.

```
1 /*
2 * SPDX-FileCopyrightText:  Copyright (c) 1993-2026 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3 * SPDX-License-Identifier:  Apache-2.0
4 *
5 * Licensed under the Apache License, Version 2.0 (the "License");
6 * you may not use this file except in compliance with the License.
7 * You may obtain a copy of the License at
8 *
9 * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17
18 // This file provides the user-implementable interface for the Safe Recorder feature in the NVIDIA Safe
      Runtime API.
19 // The Safe Recorder is a user-defined object that interacts with the internal state of the object it is
      assigned to,
20 // in order to determine information about abnormalities in execution.
21 // The Safe Recorder interface, ISafeRecorder, allows users to define a custom error reporting mechanism that
      receives
22 // both an error enum and a string description of the error.  The error strings are limited to 255 bytes or
      less in
23 // length, including the NULL terminator.  The Safe Recorder is passed along to any class that is created
      from another
24 // class that has an Safe Recorder assigned to it, allowing for hierarchical error reporting.  Users can
      register a
25 // different error recorder or de-register the error recorder for specific objects using accessor functions.
      The Safe
26 // Recorder object implementation must be thread-safe, with all locking and synchronization handled by the
      interface
27 // implementation.  The lifetime of the Safe Recorder object must exceed the lifetime of all TensorRT objects
      that use
28 // it.
29
30 #ifndef NV_INFER_SAFE_RECORDER_H
31 #define NV_INFER_SAFE_RECORDER_H
32
33 #include <cstdint>
34 // Define NV_INFER_SAFE_RECORDER_TYPES_ONLY before including this header to skip
35 // the ISafeRecorder interface and its dependencies (NvInferForwardDecl.h
36 // -> NvInferPluginBase.h / NvInferRuntimeBase.h).
37 #ifndef NV_INFER_SAFE_RECORDER_TYPES_ONLY
38 #include "NvInferForwardDecl.h"
39 #endif
40
41 namespace nvinfer2
42 {
43 namespace safe
44 {
```

```
45
46 #ifndef NV_INFER_SAFE_RECORDER_TYPES_ONLY
47 using ErrorDesc = char const*;
48 using RefCount = int32_t;
49
54 enum class Severity :  uint32_t
55 {
56     kINTERNAL_ERROR = 0,
57     kERROR = 1,
58     kWARNING = 2,
59     kINFO = 3,
60     kVERBOSE = 4,
61     kDEBUG = 5,
62 };
63
64 namespace impl
65 {
67 template <>
68 struct EnumMaxImpl<Severity>
69 {
70     static constexpr int32_t kVALUE = 6;
71 };
72 } // namespace impl
73
81 class ISafeRecorder :  public IErrorRecorder
82 {
83 public:
89     static constexpr size_t kMAX_SAFE_DESC_LENGTH{4095U};
90
103     ISafeRecorder(Severity severity_ = Severity::kERROR, int32_t id_ = -1)
104         :  IErrorRecorder()
105         , mSeverity(severity_)
106         , mId(id_)
107     {
108     }
109
115     ISafeRecorder(ISafeRecorder const&) = delete;
121     ISafeRecorder(ISafeRecorder&&) = delete;
127     ISafeRecorder& operator=(ISafeRecorder const&) & = delete;
133     ISafeRecorder& operator=(ISafeRecorder&&) & = delete;
134
140     ~ISafeRecorder() noexcept override = default;
141
149     virtual void setSeverity(Severity severity_) noexcept
150     {
151         mSeverity = severity_;
152     }
153     // NOLINTBEGIN to avoid clang-tidy requiring [[nodiscard]]
162     virtual Severity getSeverity() const noexcept
163     {
164         return mSeverity;
165     }
166
175     virtual int32_t getId() const noexcept
176     {
177         return mId;
178     }
179     // NOLINTEND
180
190     virtual bool reportWarn(ErrorDesc /*desc*/) noexcept
191     {
192         return false;
193     }
194
204     virtual bool reportDebug(ErrorDesc /*desc*/) noexcept
205     {
206         return false;
207     }
208
218     virtual bool reportInfo(ErrorDesc /*desc*/) noexcept
219     {
220         return false;
221     }
222
232     virtual bool reportVerbose(ErrorDesc /*desc*/) noexcept
233     {
234         return false;
235     }
236
237 protected:
238     Severity mSeverity;
```

```
239     int32_t const mId;
240 };
241 #endif // NV_INFER_SAFE_RECORDER_TYPES_ONLY
242
252 struct RuntimeErrorInformation
253 {
254     uint64_t bitMask{0ULL};
255 };
256
266 enum class RuntimeErrorType :  uint64_t
267 {
268     kNAN_CONSUMED = 1ULL << 0,
269     kINF_CONSUMED = 1ULL << 1,
270     kGATHER_OOB = 1ULL << 2,
271     kSCATTER_OOB = 1ULL << 3,
272     kSCATTER_RACE = 1ULL << 4,
273     kDIV_ZERO = 1ULL << 5,
274 };
275
276 } // namespace safe
277 } // namespace nvinfer2
278
279 #endif /* NV_INFER_SAFE_RECORDER_H */
```

10.37 NvInferSafeRuntime.h File Reference

```
#include "NvInferSafeMemAllocator.h"
#include "NvInferSafePlugin.h"
#include "NvInferSafeRecorder.h"
#include <algorithm>
#include <cuda_fp16.h>
```

Classes

- class nvinfer2::safe::TypedArray

  A standard_layout and trivially_copyable typed array that knows the data type and size it is holding.
- class nvinfer2::safe::PhysicalDims

  Structure to define the physical dimensions of a tensor with support for up to 9 dimensions.
- struct nvinfer2::safe::TensorDescriptor

  A simple record summarizing various properties of a network IO Tensor.
- class nvinfer2::safe::ITRTGraph

  Abstract Interface for a functionally safe graph for executing inference on a built network.

Namespaces

- namespace nvinfer2
- namespace nvinfer2::safe

Typedefs

- using nvinfer2::safe::half_t = __half

Functions

- bool nvinfer2::safe::operator== (PhysicalDims const &d0, PhysicalDims const &d1) noexcept
- bool nvinfer2::safe::operator!= (PhysicalDims const &d0, PhysicalDims const &d1) noexcept
- ErrorCode nvinfer2::safe::createTRTGraph (ITRTGraph *&graph, void const *buffer, int64_t bufferSize, ISafeRecorder &recorder, bool trtManagedScratch=true, ISafeMemAllocator *allocator=nullptr) noexcept

  The C factory function which serves as an entry point to TRT that creates an instance of a ITRTGraph class. It will performs the following operations:
- ErrorCode nvinfer2::safe::destroyTRTGraph (ITRTGraph *&graph) noexcept

  Toplevel graph destructor.

10.38 NvInferSafeRuntime.h

Go to the documentation of this file.

```
1 /*
2 * SPDX-FileCopyrightText: Copyright (c) 1993-2026 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3 * SPDX-License-Identifier: Apache-2.0
4 *
5 * Licensed under the Apache License, Version 2.0 (the "License");
6 * you may not use this file except in compliance with the License.
7 * You may obtain a copy of the License at
8 *
9 * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17
18 // @brief Main header file for the NVIDIA Safe Runtime API.
19 // This file provides the primary interface for users to interact with the NVIDIA Safe Runtime API.
20 // It includes the necessary definitions, classes, and functions for creating and managing safe graphs,
21 // executing inference, and handling errors.
22 // Users should include this header file in their application to access the Safe Runtime API functionality.
23
24 #ifndef NV_INFER_SAFE_RUNTIME_H
25 #define NV_INFER_SAFE_RUNTIME_H
26 #include "NvInferSafeMemAllocator.h"
27 #include "NvInferSafePlugin.h"
28 #include "NvInferSafeRecorder.h"
29 #include <algorithm>
30 #include <cuda_fp16.h>
31
32 namespace nvinfer2
33 {
34 namespace safe
35 {
36 using half_t = __half;
37
50 class TypedArray
51 {
52 public:
53     TypedArray() noexcept
54         : mType(DataType::kFLOAT)
55         , mData(nullptr)
56         , mBufferSize(0U)
57     {
58     }
59     // Shallow copy of TypedArray is allowed
60     TypedArray(TypedArray const&) = default;
61     TypedArray(TypedArray&&) = default;
62     TypedArray& operator=(TypedArray const&) & = default;
63     TypedArray& operator=(TypedArray&&) & = default;
64     ~TypedArray() noexcept = default;
65
67     TypedArray(float* ptr, uint64_t const bufferSize) noexcept
68         : mType(DataType::kFLOAT)
69         , mData(ptr)
```

```
70          , mBufferSize(bufferSize)
71      {
72      }
73
75      TypedArray(half_t* ptr, uint64_t const bufferSize) noexcept
76          :  mType(DataType::kHALF)
77          , mData(ptr)
78          , mBufferSize(bufferSize)
79      {
80      }
81
83      TypedArray(int64_t* ptr, uint64_t const bufferSize) noexcept
84          :  mType(DataType::kINT64)
85          , mData(ptr)
86          , mBufferSize(bufferSize)
87      {
88      }
89
91      TypedArray(int32_t* ptr, uint64_t const bufferSize) noexcept
92          :  mType(DataType::kINT32)
93          , mData(ptr)
94          , mBufferSize(bufferSize)
95      {
96      }
97
99      TypedArray(int8_t* ptr, uint64_t const bufferSize) noexcept
100         :  mType(DataType::kINT8)
101         , mData(ptr)
102         , mBufferSize(bufferSize)
103     {
104     }
105
107     TypedArray(uint8_t* ptr, uint64_t const bufferSize) noexcept
108         :  mType(DataType::kUINT8)
109         , mData(ptr)
110         , mBufferSize(bufferSize)
111     {
112     }
114     TypedArray(bool* ptr, uint64_t const bufferSize) noexcept
115         :  mType(DataType::kBOOL)
116         , mData(ptr)
117         , mBufferSize(bufferSize)
118     {
119     }
120
121     // NOLINTBEGIN to avoid clang-tidy requiring [[nodiscard]]
123     DataType getType() const noexcept
124     {
125         return mType;
126     }
127
130     float* getFloat() const noexcept
131     {
132         if (mType == DataType::kFLOAT)
133         {
134             return static_cast<float*>(mData);
135         }
136         return nullptr;
137     }
138
141     half_t* getHalf() const noexcept
142     {
143         if (mType == DataType::kHALF)
144         {
145             return static_cast<half_t*>(mData);
146         }
147         return nullptr;
148     }
149
152     int64_t* getInt64() const noexcept
153     {
154         if (mType == DataType::kINT64)
155         {
156             return static_cast<int64_t*>(mData);
157         }
158         return nullptr;
159     }
160
163     int32_t* getInt32() const noexcept
164     {
165         if (mType == DataType::kINT32)
```

```
166         {
167             return static_cast<int32_t*>(mData);
168         }
169         return nullptr;
170     }
171
174     int8_t* getInt8() const noexcept
175     {
176         if (mType == DataType::kINT8)
177         {
178             return static_cast<int8_t*>(mData);
179         }
180         return nullptr;
181     }
182
185     uint8_t* getUint8() const noexcept
186     {
187         if (mType == DataType::kUINT8)
188         {
189             return static_cast<uint8_t*>(mData);
190         }
191         return nullptr;
192     }
193
196     bool* getBool() const noexcept
197     {
198         if (mType == DataType::kBOOL)
199         {
200             return static_cast<bool*>(mData);
201         }
202         return nullptr;
203     }
204
206     uint64_t getSize() const noexcept
207     {
208         return mBufferSize;
209     }
210
213     void* getData() const noexcept
214     {
215         return mData;
216     }
217     // NOLINTEND
218
219 private:
220     DataType mType;       // This is the current type of the data.
221     void* mData;          // This is the pointer that holds the data.
222     uint64_t mBufferSize; // This is the size of the buffer in bytes that holds the data.
223 };
224
238 class PhysicalDims
239 {
240 public:
243     static constexpr int32_t MAX_DIMS{9};
244
246     int32_t nbDims;
247
249     int64_t d[MAX_DIMS];
250 };
251
252 inline bool operator==(PhysicalDims const& d0, PhysicalDims const& d1) noexcept
253 {
254     return d0.nbDims == d1.nbDims && (d0.nbDims <= 0 || std::equal(d0.d, d0.d + d0.nbDims, d1.d));
255 }
256
257 inline bool operator!=(PhysicalDims const& d0, PhysicalDims const& d1) noexcept
258 {
259     return !(d0 == d1);
260 }
261
284 struct TensorDescriptor
285 {
287     AsciiChar const* tensorName{nullptr};
290     PhysicalDims shape{-1, {}};
292     PhysicalDims stride{-1, {}};
295     DataType dataType{DataType::kFLOAT};
297     uint64_t bytesPerComponent{0U};
299     uint64_t componentsPerVector{1U};
301     int64_t vectorizedDim{-1};
303     uint64_t sizeInBytes{0U};
305     TensorIOMode ioMode{TensorIOMode::kNONE};
```

```
307     MemoryPlacement memPlacement{MemoryPlacement::kNONE};
309     PhysicalDims strideOrder{-1, {}};
313     Dims userShape{-1, {}};
314 };
315
328 class ITRTGraph
329 {
330 public:
331     ITRTGraph(ITRTGraph const&) = delete;
332     ITRTGraph(ITRTGraph&&) = delete;
333     ITRTGraph& operator=(ITRTGraph const&) & = delete;
334     ITRTGraph& operator=(ITRTGraph&&) & = delete;
335
339     virtual ~ITRTGraph() noexcept = default;
340
359     virtual ErrorCode clone(ITRTGraph*& graph, ISafeRecorder& recorder) noexcept = 0;
360
374     virtual ErrorCode getScratchMemorySize(size_t& size) const noexcept = 0;
375
386     virtual ErrorCode getTRTManagedScratch(bool& flag) const noexcept = 0;
387
403     virtual ErrorCode setScratchMemory(void* memory) noexcept = 0;
404
418     virtual ErrorCode getScratchMemory(void*& memory) noexcept = 0;
419
428     virtual ErrorCode getNbIOTensors(int64_t& nb) const noexcept = 0;
429
439     virtual ErrorCode getIOTensorName(AsciiChar const*& name, size_t const index) const noexcept = 0;
440
453     virtual ErrorCode getIOTensorDescriptor(
454         TensorDescriptor& desc, AsciiChar const* const tensorName) const noexcept = 0;
455
468     virtual ErrorCode getIOTensorDescriptor(TensorDescriptor& desc, int32_t const index) const noexcept = 0;
469
480     virtual ErrorCode setIOTensorAddress(AsciiChar const* const tensorName, TypedArray const& tensor)
    noexcept = 0;
481
492     virtual ErrorCode setIOTensorAddress(int32_t const index, TypedArray const& tensor) noexcept = 0;
493
504     virtual ErrorCode getIOTensorAddress(AsciiChar const* const tensorName, TypedArray& tensor) noexcept =
    0;
505
516     virtual ErrorCode getIOTensorAddress(int32_t const index, TypedArray& tensor) noexcept = 0;
517
527     virtual ErrorCode setInputConsumedEvent(cudaEvent_t event) noexcept = 0;
528
539     virtual ErrorCode getInputConsumedEvent(cudaEvent_t& event) const noexcept = 0;
540
552     virtual ErrorCode getErrorBuffer(RuntimeErrorInformation*& buffer) const noexcept = 0;
553
562     virtual ErrorCode getSafeRecorder(ISafeRecorder*& recorder) const noexcept = 0;
563
574     virtual ErrorCode getNbIOProfiles(int64_t& nb) const noexcept = 0;
575
590     virtual ErrorCode setIOProfile(int64_t profileIndex) noexcept = 0;
591
601     virtual ErrorCode getIOProfile(int64_t& profileIndex) const noexcept = 0;
602
613     virtual ErrorCode getNbAuxStreams(int32_t& nbStreams) const noexcept = 0;
614
638     virtual ErrorCode setAuxStreams(cudaStream_t* auxStreams, int32_t nbStreams) noexcept = 0;
639
652     virtual ErrorCode executeAsync(cudaStream_t stream) noexcept = 0;
653
664     virtual ErrorCode sync() noexcept = 0;
665
666 protected:
667     ITRTGraph() = default;
668 };
669
701 extern "C" ErrorCode createTRTGraph(ITRTGraph*& graph, void const* buffer, int64_t bufferSize,
    ISafeRecorder& recorder,
702     bool trtManagedScratch = true, ISafeMemAllocator* allocator = nullptr) noexcept;
703
721 extern "C" ErrorCode destroyTRTGraph(ITRTGraph*& graph) noexcept;
722
723 } // namespace safe
724 } // namespace nvinfer2
725 #endif /* NV_INFER_SAFE_RUNTIME_H */
```

Index

~IActivationLayer
    nvinfer1::IActivationLayer, 161
~IAssertionLayer
    nvinfer1::IAssertionLayer, 164
~IAttention
    nvinfer1::IAttention, 168
~IAttentionBoundaryLayer
    nvinfer1::IAttentionBoundaryLayer, 183
~IAttentionInputLayer
    nvinfer1::IAttentionInputLayer, 185
~IAttentionOutputLayer
    nvinfer1::IAttentionOutputLayer, 187
~IBuilder
    nvinfer1::IBuilder, 189
~IBuilderConfig
    nvinfer1::IBuilderConfig, 201
~ICastLayer
    nvinfer1::ICastLayer, 224
~IConcatenationLayer
    nvinfer1::IConcatenationLayer, 226
~IConditionLayer
    nvinfer1::IConditionLayer, 228
~IConsistencyChecker
    nvinfer2::safe::consistency::IConsistencyChecker, 229
~IConstantLayer
    nvinfer1::IConstantLayer, 233
~IConvolutionLayer
    nvinfer1::IConvolutionLayer, 236
~ICudaEngine
    nvinfer1::ICudaEngine, 247
~ICumulativeLayer
    nvinfer1::ICumulativeLayer, 269
~IDebugListener
    nvinfer1::v_1_0::IDebugListener, 273
~IDeconvolutionLayer
    nvinfer1::IDeconvolutionLayer, 276
~IDequantizeLayer
    nvinfer1::IDequantizeLayer, 285
~IDimensionExpr
    nvinfer1::IDimensionExpr, 289
~IDistCollectiveLayer
    nvinfer1::IDistCollectiveLayer, 290
~IDynamicQuantizeLayer
    nvinfer1::IDynamicQuantizeLayer, 293
~IEinsumLayer
    nvinfer1::IEinsumLayer, 298
~IElementWiseLayer
    nvinfer1::IElementWiseLayer, 300
~IEngineInspector
    nvinfer1::IEngineInspector, 302
~IErrorRecorder
    nvinfer1::v_1_0::IErrorRecorder, 308
~IExecutionContext
    nvinfer1::IExecutionContext, 317
~IExprBuilder
    nvinfer1::IExprBuilder, 342
~IFillLayer
    nvinfer1::IFillLayer, 346
~IGatherLayer
    nvinfer1::IGatherLayer, 356
~IGpuAllocator
    nvinfer1::v_1_0::IGpuAllocator, 360
~IGpuAsyncAllocator
    nvinfer1::v_1_0::IGpuAsyncAllocator, 366
~IGridSampleLayer
    nvinfer1::IGridSampleLayer, 371
~IHostMemory
    nvinfer1::IHostMemory, 374
~IIdentityLayer
    nvinfer1::IIdentityLayer, 377
~IIfConditional
    nvinfer1::IIfConditional, 378
~IIfConditionalBoundaryLayer
    nvinfer1::IIfConditionalBoundaryLayer, 382
~IIfConditionalInputLayer
    nvinfer1::IIfConditionalInputLayer, 383
~IIfConditionalOutputLayer
    nvinfer1::IIfConditionalOutputLayer, 385
~IIteratorLayer
    nvinfer1::IIteratorLayer, 386
~IKVCacheUpdateLayer
    nvinfer1::IKVCacheUpdateLayer, 389
~ILRNLayer
    nvinfer1::ILRNLayer, 414
~ILayer
    nvinfer1::ILayer, 394
~ILogger
    nvinfer1::v_1_0::ILogger, 402
~ILoggerFinder

nvinfer1::v_1_0::ILoggerFinder, 404
~ILoop
nvinfer1::ILoop, 406
~ILoopBoundaryLayer
nvinfer1::ILoopBoundaryLayer, 409
~ILoopOutputLayer
nvinfer1::ILoopOutputLayer, 411
~IMatrixMultiplyLayer
nvinfer1::IMatrixMultiplyLayer, 418
~IMoELayer
nvinfer1::IMoELayer, 423
~INMSLayer
nvinfer1::INMSLayer, 492
~INetworkDefinition
nvinfer1::INetworkDefinition, 438
~INoCopy
nvinfer1::INoCopy, 496
~INonZeroLayer
nvinfer1::INonZeroLayer, 498
~INormalizationLayer
nvinfer1::INormalizationLayer, 501
~IOneHotLayer
nvinfer1::IOneHotLayer, 506
~IOnnxConfig
nvonnxparser::IOnnxConfig, 508
~IOptimizationProfile
nvinfer1::IOptimizationProfile, 515
~IPaddingLayer
nvinfer1::IPaddingLayer, 524
~IParametricReLULayer
nvinfer1::IParametricReLULayer, 527
~IParser
nvonnxparser::IParser, 529
~IParserError
nvonnxparser::IParserError, 539
~IParserRefitter
nvonnxparser::IParserRefitter, 542
~IPluginChecker
nvinfer2::safe::consistency::IPluginChecker, 547
~IPluginCreator
nvinfer1::v_1_0::IPluginCreator, 551
~IPluginCreatorInterface
nvinfer1::v_1_0::IPluginCreatorInterface, 556
~IPluginCreatorV3One
nvinfer1::v_1_0::IPluginCreatorV3One, 559
~IPluginRegistry
nvinfer1::IPluginRegistry, 563
~IPluginResource
nvinfer1::v_1_0::IPluginResource, 570
~IPluginResourceContext
nvinfer1::IPluginResourceContext, 573
~IPluginV2DynamicExt
nvinfer1::IPluginV2DynamicExt, 587
~IPluginV2Ext
nvinfer1::IPluginV2Ext, 593
~IPluginV2Layer
nvinfer1::IPluginV2Layer, 602
~IPluginV3Layer
nvinfer1::IPluginV3Layer, 605
~IPluginV3OneSafeBuild
nvinfer2::safe::IPluginV3OneSafeBuild, 625
~IPluginV3OneSafeBuildMSS
nvinfer2::safe::IPluginV3OneSafeBuildMSS, 635
~IPluginV3OneSafeCore
nvinfer2::safe::IPluginV3OneSafeCore, 639
~IPluginV3OneSafeRuntime
nvinfer2::safe::IPluginV3OneSafeRuntime, 643
~IPoolingLayer
nvinfer1::IPoolingLayer, 648
~IProfiler
nvinfer1::v_1_0::IProfiler, 655
~IProgressMonitor
nvinfer1::v_1_0::IProgressMonitor, 657
~IQuantizeLayer
nvinfer1::IQuantizeLayer, 662
~IRaggedSoftMaxLayer
nvinfer1::IRaggedSoftMaxLayer, 665
~IRecurrenceLayer
nvinfer1::IRecurrenceLayer, 666
~IReduceLayer
nvinfer1::IReduceLayer, 669
~IRefitter
nvinfer1::IRefitter, 672
~IResizeLayer
nvinfer1::IResizeLayer, 684
~IReverseSequenceLayer
nvinfer1::IReverseSequenceLayer, 692
~IRotaryEmbeddingLayer
nvinfer1::IRotaryEmbeddingLayer, 695
~IRuntime
nvinfer1::IRuntime, 698
~IRuntimeConfig
nvinfer1::IRuntimeConfig, 707
~ISafeMemAllocator
nvinfer2::safe::ISafeMemAllocator, 709
~ISafePluginCreatorV3One
nvinfer2::safe::ISafePluginCreatorV3One, 714
~ISafePluginRegistry
nvinfer2::safe::ISafePluginRegistry, 720
~ISafePluginResourceContext
nvinfer2::safe::ISafePluginResourceContext, 727
~ISafeRecorder
nvinfer2::safe::ISafeRecorder, 734
~IScaleLayer
nvinfer1::IScaleLayer, 742
~IScatterLayer
nvinfer1::IScatterLayer, 747
~ISelectLayer

nvinfer1::ISelectLayer, 749
~ISerializationConfig
nvinfer1::ISerializationConfig, 751
~IShapeLayer
nvinfer1::IShapeLayer, 754
~IShuffleLayer
nvinfer1::IShuffleLayer, 756
~ISliceLayer
nvinfer1::ISliceLayer, 762
~ISoftMaxLayer
nvinfer1::ISoftMaxLayer, 768
~ISqueezeLayer
nvinfer1::ISqueezeLayer, 770
~IStreamReader
nvinfer1::v_1_0::IStreamReader, 773
~IStreamReaderV2
nvinfer1::v_1_0::IStreamReaderV2, 776
~IStreamWriter
nvinfer1::v_1_0::IStreamWriter, 779
~ITRTGraph
nvinfer2::safe::ITRTGraph, 805
~ITensor
nvinfer1::ITensor, 782
~ITimingCache
nvinfer1::ITimingCache, 793
~ITopKLayer
nvinfer1::ITopKLayer, 797
~ITripLimitLayer
nvinfer1::ITripLimitLayer, 802
~IUnaryLayer
nvinfer1::IUnaryLayer, 819
~IUnsqueezeLayer
nvinfer1::IUnsqueezeLayer, 821
~IVersionedInterface
nvinfer1::IVersionedInterface, 824
~TypedArray
nvinfer2::safe::TypedArray, 850

acquirePluginResource
nvinfer1::IPluginRegistry, 563
ActivationType
nvinfer1, 45
addActivation
nvinfer1::INetworkDefinition, 438
addAssertion
nvinfer1::INetworkDefinition, 438
addAttention
nvinfer1::INetworkDefinition, 439
addAttentionV2
nvinfer1::INetworkDefinition, 440
addCast
nvinfer1::INetworkDefinition, 441
addConcatenation
nvinfer1::INetworkDefinition, 441
addConstant
nvinfer1::INetworkDefinition, 442
addConvolutionNd
nvinfer1::INetworkDefinition, 442
addCumulative
nvinfer1::INetworkDefinition, 443
addDeconvolutionNd
nvinfer1::INetworkDefinition, 444
addDequantize
nvinfer1::INetworkDefinition, 445
addDistCollective
nvinfer1::INetworkDefinition, 445
addDynamicQuantize
nvinfer1::INetworkDefinition, 446
addDynamicQuantizeV2
nvinfer1::INetworkDefinition, 447
addEinsum
nvinfer1::INetworkDefinition, 448
addElementWise
nvinfer1::INetworkDefinition, 448
addFill
nvinfer1::INetworkDefinition, 449
addGather
nvinfer1::INetworkDefinition, 450
addGatherV2
nvinfer1::INetworkDefinition, 450
addGridSample
nvinfer1::INetworkDefinition, 451
addIdentity
nvinfer1::INetworkDefinition, 452
addIfConditional
nvinfer1::INetworkDefinition, 452
addInput
nvinfer1::IIfConditional, 379
nvinfer1::INetworkDefinition, 452
addIterator
nvinfer1::ILoop, 406
addKVCacheUpdate
nvinfer1::INetworkDefinition, 453
addLoop
nvinfer1::INetworkDefinition, 454
addLoopOutput
nvinfer1::ILoop, 406
addLRN
nvinfer1::INetworkDefinition, 455
addMatrixMultiply
nvinfer1::INetworkDefinition, 456
addMoE
nvinfer1::INetworkDefinition, 456
addNMS
nvinfer1::INetworkDefinition, 457, 458
addNonZero
nvinfer1::INetworkDefinition, 458, 459
addNormalization

nvinfer1::INetworkDefinition, 459
addNormalizationV2
nvinfer1::INetworkDefinition, 461
addOneHot
nvinfer1::INetworkDefinition, 462
addOptimizationProfile
nvinfer1::IBuilderConfig, 201
addOutput
nvinfer1::IIfConditional, 379
addPaddingNd
nvinfer1::INetworkDefinition, 462
addParametricReLU
nvinfer1::INetworkDefinition, 463
addPluginV2
nvinfer1::INetworkDefinition, 464
addPluginV3
nvinfer1::INetworkDefinition, 464
addPoolingNd
nvinfer1::INetworkDefinition, 465
addQuantize
nvinfer1::INetworkDefinition, 465
addRaggedSoftMax
nvinfer1::INetworkDefinition, 466
addRecurrence
nvinfer1::ILoop, 406
addReduce
nvinfer1::INetworkDefinition, 467
addResize
nvinfer1::INetworkDefinition, 468
addReverseSequence
nvinfer1::INetworkDefinition, 468
addRotaryEmbedding
nvinfer1::INetworkDefinition, 469
addScale
nvinfer1::INetworkDefinition, 469
addScaleNd
nvinfer1::INetworkDefinition, 470
addScatter
nvinfer1::INetworkDefinition, 471
addSelect
nvinfer1::INetworkDefinition, 472
addShape
nvinfer1::INetworkDefinition, 473
addShuffle
nvinfer1::INetworkDefinition, 473
addSlice
nvinfer1::INetworkDefinition, 474
addSoftMax
nvinfer1::INetworkDefinition, 474
addSqueeze
nvinfer1::INetworkDefinition, 475
addTopK
nvinfer1::INetworkDefinition, 475, 476
addTripLimit
nvinfer1::ILoop, 406
addUnary
nvinfer1::INetworkDefinition, 477
addUnsqueeze
nvinfer1::INetworkDefinition, 478
addVerbosity
nvonnxparser::IOnnxConfig, 508
allInputDimensionsSpecified
nvinfer1::IExecutionContext, 317
allocate
nvinfer1::v_1_0::IGpuAllocator, 360
nvinfer1::v_1_0::IGpuAsyncAllocator, 366
nvinfer2::safe::ISafeMemAllocator, 710
allocateAsync
nvinfer1::v_1_0::IGpuAllocator, 361
nvinfer1::v_1_0::IGpuAsyncAllocator, 367
AllocatorFlag
nvinfer1, 46
AllocatorFlags
nvinfer1, 39
anchorsRatioCount
nvinfer1::plugin::RPROIParams, 839
anchorsScaleCount
nvinfer1::plugin::RPROIParams, 839
APILanguage
nvinfer1, 46
areWeightsMarkedRefittable
nvinfer1::INetworkDefinition, 478
AsciiChar
nvinfer1, 39
nvinfer2::safe, 89
aspectRatios
nvinfer1::plugin::GridAnchorParameters, 158
nvinfer1::plugin::PriorBoxParameters, 834
attachToContext
nvinfer1::IPluginV2Ext, 593
nvinfer1::v_1_0::IPluginV3OneRuntime, 620
AttentionIOForm
nvinfer1, 46
AttentionNormalizationOp
nvinfer1, 46

bitMask
nvinfer2::safe::RuntimeErrorInformation, 841
BoundingBoxFormat
nvinfer1, 47
buildEngineWithConfig
nvinfer1::IBuilder, 189
BuilderFlag
nvinfer1, 47
BuilderFlags
nvinfer1, 40
buildSerializedNetwork
nvinfer1::IBuilder, 190, 191

buildSerializedNetworkToStream
nvinfer1::IBuilder, 191
bytesPerComponent
nvinfer2::safe::TensorDescriptor, 844

canRunOnDLA
nvinfer1::IBuilderConfig, 201
CausalMaskKind
nvinfer1, 50
char_t
nvinfer1, 40
child
nvinfer1::plugin::softmaxTree, 842
classes
nvinfer1::plugin::RegionParameters, 837
clear
nvinfer1::v_1_0::IErrorRecorder, 309
clearErrors
nvonnxparser::IParser, 529
nvonnxparser::IParserRefitter, 542
clearFlag
nvinfer1::IBuilderConfig, 201
nvinfer1::ISerializationConfig, 751
nvonnxparser::IParser, 529
clip
nvinfer1::plugin::PriorBoxParameters, 834
clone
nvinfer1::IPluginV2, 576
nvinfer1::IPluginV2DynamicExt, 587
nvinfer1::IPluginV2Ext, 594
nvinfer1::v_1_0::IPluginResource, 571
nvinfer1::v_1_0::IPluginV3, 603
nvinfer2::safe::IPluginV3OneSafeRuntime, 643
nvinfer2::safe::ITRTGraph, 805
code
nvonnxparser::IParserError, 539
CollectiveOperation
nvinfer1, 51
combine
nvinfer1::ITimingCache, 793
componentsPerVector
nvinfer2::safe::TensorDescriptor, 844
configurePlugin
nvinfer1::IPluginV2DynamicExt, 587
nvinfer1::IPluginV2Ext, 594
nvinfer1::IPluginV2IOExt, 599
nvinfer1::v_1_0::IPluginV3OneBuild, 608
nvinfer2::safe::IPluginV3OneSafeBuild, 626
configureWithFormat
nvinfer1::IPluginV2, 577
nvinfer1::IPluginV2Ext, 596
constant
nvinfer1::IExprBuilder, 342
coords
nvinfer1::plugin::RegionParameters, 837
count
nvinfer1::Weights, 855
createBuilderConfig
nvinfer1::IBuilder, 192
createConsistencyChecker
nvinfer2::safe::consistency, 97
createEngineInspector
nvinfer1::ICudaEngine, 247
createExecutionContext
nvinfer1::ICudaEngine, 247, 248
createInferBuilder
nvinfer1::anonymous_namespace{NvInfer.h}, 82
createInferRefitter
nvinfer1::anonymous_namespace{NvInferRuntime.h}, 83
createInferRuntime
nvinfer1::anonymous_namespace{NvInferRuntime.h}, 83
createNetworkV2
nvinfer1::IBuilder, 192
createNvOnnxParser_INTERNAL
NvOnnxParser.h, 995
createNvOnnxParserRefitter_INTERNAL
NvOnnxParser.h, 995
createONNXConfig
nvonnxparser, 101
createOptimizationProfile
nvinfer1::IBuilder, 193
createParser
nvonnxparser::anonymous_namespace{NvOnnxParser.h}, 102
createParserRefitter
nvonnxparser::anonymous_namespace{NvOnnxParser.h}, 103
createPlugin
nvinfer1::v_1_0::IPluginCreator, 551
nvinfer1::v_1_0::IPluginCreatorV3One, 559
nvinfer2::safe::ISafePluginCreatorV3One, 715
createRuntimeConfig
nvinfer1::ICudaEngine, 248
createSerializationConfig
nvinfer1::ICudaEngine, 249
createTimingCache
nvinfer1::IBuilderConfig, 202
createTRTGraph
nvinfer2::safe, 93
CumulativeOperation
nvinfer1, 51

d
nvinfer1::Dims64, 110
nvinfer1::DimsExprs, 111
nvinfer2::safe::PhysicalDims, 827

data
nvinfer1::IHostMemory, 374
nvinfer1::PluginField, 828
nvinfer1::v_1_0::TimingCacheKey, 847
DataType
nvinfer1, 52
nvinfer2::safe, 89
dataType
nvinfer2::safe::TensorDescriptor, 845
deallocate
nvinfer1::v_1_0::IGpuAllocator, 362
nvinfer1::v_1_0::IGpuAsyncAllocator, 368
nvinfer2::safe::ISafeMemAllocator, 711
deallocateAsync
nvinfer1::v_1_0::IGpuAllocator, 362
nvinfer1::v_1_0::IGpuAsyncAllocator, 368
declareSizeTensor
nvinfer1::IExprBuilder, 342
decRefCount
nvinfer1::v_1_0::IErrorRecorder, 309
deregisterCreator
nvinfer1::IPluginRegistry, 564
nvinfer2::safe::ISafePluginRegistry, 720
deregisterLibrary
nvinfer1::IPluginRegistry, 564
desc
nvinfer1::DynamicPluginTensorDesc, 115
nvonnxparser::IParserError, 539
deserializeCudaEngine
nvinfer1::IRuntime, 698, 699
deserializePlugin
nvinfer1::v_1_0::IPluginCreator, 552
destroy
nvinfer1::IPluginV2, 578
destroyTRTGraph
nvinfer2::safe, 95
detachFromContext
nvinfer1::IPluginV2Ext, 596
DeviceType
nvinfer1, 52
DimensionOperation
nvinfer1, 53
Dims, 105
nvinfer1, 40
nvinfer2::safe, 89
dims
nvinfer1::PluginTensorDesc, 832
Dims2
nvinfer1::Dims2, 106
Dims3
nvinfer1::Dims3, 107
Dims4
nvinfer1::Dims4, 109
DimsHW
nvinfer1::DimsHW, 113

ElementWiseOperation
nvinfer1, 53
EngineCapability
nvinfer1, 54
EngineStat
nvinfer1, 54
enqueue
nvinfer1::IPluginV2, 578
nvinfer1::IPluginV2DynamicExt, 588
nvinfer1::v_1_0::IPluginV3OneRuntime, 621
nvinfer2::safe::IPluginV3OneSafeRuntime, 643
enqueueV3
nvinfer1::IExecutionContext, 317
EnumMax
nvinfer1, 81
nvinfer2::safe, 96
nvonnxparser, 101
EnumMax< ErrorCode >
nvonnxparser, 101
EnumMax< OnnxParserFlag >
nvonnxparser, 102
ErrorCode
nvinfer1, 55
nvinfer2::safe, 89
nvonnxparser, 100
ErrorDesc
nvinfer1::v_1_0::IErrorRecorder, 308
nvinfer2::safe, 90
executeAsync
nvinfer2::safe::ITRTGraph, 806
executeV2
nvinfer1::IExecutionContext, 318
ExecutionContextAllocationStrategy
nvinfer1, 56

featureStride
nvinfer1::plugin::RPROIParams, 839
fields
nvinfer1::PluginFieldCollection, 830
file
nvonnxparser::IParserError, 539
FillOperation
nvinfer1, 57
findLogger
nvinfer1::v_1_0::ILoggerFinder, 404
flip
nvinfer1::plugin::PriorBoxParameters, 834
format
nvinfer1::PluginTensorDesc, 832
func
nvonnxparser::IParserError, 539

GatherMode

nvinfer1, 57
getActivationType
nvinfer1::IActivationLayer, 161
nvinfer1::IMoELayer, 423
getAliasedInput
nvinfer1::v_2_0::IPluginV3OneBuild, 615
getAliasedInputTensor
nvinfer1::ICudaEngine, 249
getAlignCorners
nvinfer1::IGridSampleLayer, 371
getAll
nvinfer1::IRefitter, 673
getAllCreators
nvinfer1::IPluginRegistry, 565
nvinfer2::safe::ISafePluginRegistry, 721
getAllCreatorsRecursive
nvinfer1::IPluginRegistry, 565
getAllowedFormats
nvinfer1::ITensor, 783
getAllWeights
nvinfer1::IRefitter, 673
getAlpha
nvinfer1::IActivationLayer, 161
nvinfer1::IFillLayer, 346
nvinfer1::ILRNLayer, 414
getAlphaInt64
nvinfer1::IFillLayer, 346
getAPILanguage
nvinfer1::IVersionedInterface, 824
getAttention
nvinfer1::IAttentionBoundaryLayer, 184
getAverageCountExcludesPadding
nvinfer1::IPoolingLayer, 648
getAvgTimingIterations
nvinfer1::IBuilderConfig, 202
getAxes
nvinfer1::INormalizationLayer, 501
nvinfer1::ISliceLayer, 763
nvinfer1::ISoftMaxLayer, 769
getAxis
nvinfer1::IConcatenationLayer, 226
nvinfer1::IDequantizeLayer, 285
nvinfer1::IDynamicQuantizeLayer, 293
nvinfer1::IIteratorLayer, 386
nvinfer1::ILoopOutputLayer, 411
nvinfer1::IOneHotLayer, 506
nvinfer1::IQuantizeLayer, 662
nvinfer1::IScatterLayer, 747
getBatchAxis
nvinfer1::IReverseSequenceLayer, 692
getBeta
nvinfer1::IActivationLayer, 161
nvinfer1::IFillLayer, 346
nvinfer1::ILRNLayer, 414
getBetaInt64
nvinfer1::IFillLayer, 347
getBiasWeights
nvinfer1::IConvolutionLayer, 236
nvinfer1::IDeconvolutionLayer, 276
getBlendFactor
nvinfer1::IPoolingLayer, 648
getBlockShape
nvinfer1::IDequantizeLayer, 286
nvinfer1::IDynamicQuantizeLayer, 293
nvinfer1::IQuantizeLayer, 662
getBlockSize
nvinfer1::IDynamicQuantizeLayer, 293
getBool
nvinfer2::safe::TypedArray, 852
getBoundingBoxFormat
nvinfer1::INMSLayer, 492
getBroadcastAcrossBatch
nvinfer1::ITensor, 783
getBuilder
nvinfer1::INetworkDefinition, 479
getBuilderOptimizationLevel
nvinfer1::IBuilderConfig, 203
getBuilderPluginRegistry
nvinfer1, 81
getBuilderSafePluginRegistry
nvinfer1, 81
getCacheMode
nvinfer1::IKVCacheUpdateLayer, 389
getCapabilityInterface
nvinfer1::v_1_0::IPluginV3, 604
getCausal
nvinfer1::IAttention, 169
getCausalKind
nvinfer1::IAttention, 169
getChannelAxis
nvinfer1::IScaleLayer, 742
getConditional
nvinfer1::IIfConditionalBoundaryLayer, 382
getConstantValue
nvinfer1::IDimensionExpr, 289
getCoordinateTransformation
nvinfer1::IResizeLayer, 684
getCreator
nvinfer1::IPluginRegistry, 566
nvinfer2::safe::ISafePluginRegistry, 721
getCubicCoeff
nvinfer1::IResizeLayer, 684
getData
nvinfer2::safe::TypedArray, 852
getDebugListener
nvinfer1::IExecutionContext, 318
getDebugState
nvinfer1::IExecutionContext, 318

getDebugSync
nvinfer1::IExecutionContext, 320
getDecomposable
nvinfer1::IAttention, 169
getDefaultDeviceType
nvinfer1::IBuilderConfig, 203
getDeviceMemorySizeForProfileV2
nvinfer1::ICudaEngine, 249
getDeviceMemorySizeV2
nvinfer1::ICudaEngine, 250
getDeviceType
nvinfer1::IBuilderConfig, 203
getDilationNd
nvinfer1::IConvolutionLayer, 236
nvinfer1::IDeconvolutionLayer, 276
getDimensionName
nvinfer1::ITensor, 783
getDimensions
nvinfer1::IConstantLayer, 233
nvinfer1::IFillLayer, 347
nvinfer1::IOptimizationProfile, 515
nvinfer1::ITensor, 784
getDLACore
nvinfer1::IBuilderConfig, 203
nvinfer1::IRuntime, 699
getDynQOutputScaleType
nvinfer1::IMoELayer, 423
getEngine
nvinfer1::IExecutionContext, 320
getEngineCapability
nvinfer1::IBuilderConfig, 204
nvinfer1::ICudaEngine, 250
getEngineHostCodeAllowed
nvinfer1::IRuntime, 700
getEngineInformation
nvinfer1::IEngineInspector, 303
getEngineStat
nvinfer1::ICudaEngine, 250
getEnqueueEmitsProfile
nvinfer1::IExecutionContext, 320
getEpsilon
nvinfer1::INormalizationLayer, 501
getEquation
nvinfer1::IEinsumLayer, 298
getError
nvonnxparser::IParser, 530
nvonnxparser::IParserRefitter, 542
getErrorBuffer
nvinfer2::safe::ITRTGraph, 806
getErrorCode
nvinfer1::v_1_0::IErrorRecorder, 309
getErrorDesc
nvinfer1::v_1_0::IErrorRecorder, 310
getErrorRecorder
nvinfer1::IBuilder, 193
nvinfer1::ICudaEngine, 251
nvinfer1::IEngineInspector, 303
nvinfer1::IExecutionContext, 320
nvinfer1::INetworkDefinition, 479
nvinfer1::IPluginRegistry, 566
nvinfer1::IPluginResourceContext, 574
nvinfer1::IRefitter, 674
nvinfer1::IRuntime, 700
getExcludeOutside
nvinfer1::IResizeLayer, 685
getExclusive
nvinfer1::ICumulativeLayer, 269
getExecutionContext
nvinfer1::IEngineInspector, 304
getExecutionContextAllocationStrategy
nvinfer1::IRuntimeConfig, 707
getExtraMemoryTarget
nvinfer1::IOptimizationProfile, 515
getFieldNames
nvinfer1::v_1_0::IPluginCreator, 552
nvinfer1::v_1_0::IPluginCreatorV3One, 560
nvinfer2::safe::ISafePluginCreatorV3One, 715
getFieldsToSerialize
nvinfer1::v_1_0::IPluginV3OneRuntime, 622
nvinfer2::safe::IPluginV3OneSafeRuntime, 644
getFirstTranspose
nvinfer1::IShuffleLayer, 756
getFlag
nvinfer1::IBuilderConfig, 204
nvinfer1::INetworkDefinition, 479
nvinfer1::ISerializationConfig, 751
nvonnxparser::IParser, 530
getFlags
nvinfer1::IBuilderConfig, 204
nvinfer1::INetworkDefinition, 480
nvinfer1::ISerializationConfig, 752
nvonnxparser::IParser, 530
getFloat
nvinfer2::safe::TypedArray, 852
getFormatCombinationLimit
nvinfer1::v_1_0::IPluginV3OneBuild, 609
nvinfer2::safe::IPluginV3OneSafeBuild, 627
getFullTextFileName
nvonnxparser::IOnnxConfig, 508
getGatherAxis
nvinfer1::IGatherLayer, 356
getGpuAllocator
nvinfer1::IPluginResourceContext, 574
getHalf
nvinfer2::safe::TypedArray, 853
getHardwareCompatibilityLevel
nvinfer1::IBuilderConfig, 205
nvinfer1::ICudaEngine, 251

getId
nvinfer2::safe::ISafeRecorder, 734
getIndicesType
nvinfer1::INMSLayer, 492
nvinfer1::INonZeroLayer, 498
nvinfer1::ITopKLayer, 798
getInferLibBuildVersion
NvInferRuntime.h, 948
getInferLibMajorVersion
NvInferRuntime.h, 948
getInferLibMinorVersion
NvInferRuntime.h, 948
getInferLibPatchVersion
NvInferRuntime.h, 948
getInferLibVersion
NvInferRuntimeBase.h, 976
getInput
nvinfer1::IAttention, 170
nvinfer1::ILayer, 394
nvinfer1::INetworkDefinition, 480
getInputConsumedEvent
nvinfer1::IExecutionContext, 321
nvinfer2::safe::ITRTGraph, 807
getInt32
nvinfer2::safe::TypedArray, 853
getInt64
nvinfer2::safe::TypedArray, 853
getInt8
nvinfer2::safe::TypedArray, 853
getInterfaceInfo
nvinfer1::IVersionedInterface, 824
nvinfer1::v_1_0::IDebugListener, 273
nvinfer1::v_1_0::IErrorRecorder, 311
nvinfer1::v_1_0::IGpuAllocator, 363
nvinfer1::v_1_0::IGpuAsyncAllocator, 369
nvinfer1::v_1_0::ILogger, 402
nvinfer1::v_1_0::ILoggerFinder, 404
nvinfer1::v_1_0::IOutputAllocator, 521
nvinfer1::v_1_0::IPluginCreator, 553
nvinfer1::v_1_0::IPluginCreatorV3One, 560
nvinfer1::v_1_0::IPluginResource, 571
nvinfer1::v_1_0::IPluginV3, 604
nvinfer1::v_1_0::IPluginV3OneBuild, 609
nvinfer1::v_1_0::IPluginV3OneCore, 618
nvinfer1::v_1_0::IPluginV3OneRuntime, 622
nvinfer1::v_1_0::IProfiler, 655
nvinfer1::v_1_0::IProgressMonitor, 658
nvinfer1::v_1_0::IStreamReader, 773
nvinfer1::v_1_0::IStreamReaderV2, 776
nvinfer1::v_1_0::IStreamWriter, 780
nvinfer1::v_2_0::IPluginV3OneBuild, 616
nvinfer2::safe::IPluginV3OneSafeBuild, 627
nvinfer2::safe::IPluginV3OneSafeBuildMSS, 635
nvinfer2::safe::IPluginV3OneSafeCore, 639
nvinfer2::safe::IPluginV3OneSafeRuntime, 644
nvinfer2::safe::ISafePluginCreatorV3One, 716
getInterleaved
nvinfer1::IRotaryEmbeddingLayer, 695
getInterpolationMode
nvinfer1::IGridSampleLayer, 371
getIOProfile
nvinfer2::safe::ITRTGraph, 807
getIOTensorAddress
nvinfer2::safe::ITRTGraph, 808
getIOTensorDescriptor
nvinfer2::safe::ITRTGraph, 809
getIOTensorName
nvinfer1::ICudaEngine, 252
nvinfer2::safe::ITRTGraph, 810
getK
nvinfer1::ILRNLayer, 414
nvinfer1::ITopKLayer, 798
getKeepDimensions
nvinfer1::IReduceLayer, 669
getKernelSizeNd
nvinfer1::IConvolutionLayer, 237
nvinfer1::IDeconvolutionLayer, 276
getKernelWeights
nvinfer1::IConvolutionLayer, 237
nvinfer1::IDeconvolutionLayer, 277
getKeyValueForm
nvinfer1::IAttention, 170
getKeyValueLengths
nvinfer1::IAttention, 170
getL2LimitForTiling
nvinfer1::IBuilderConfig, 205
getLayer
nvinfer1::INetworkDefinition, 481
getLayerInformation
nvinfer1::IEngineInspector, 304
getLayerOutputTensor
nvonnxparser::IParser, 531
getLocation
nvinfer1::ITensor, 784
getLogger
nvinfer1::IBuilder, 193
nvinfer1::IRefitter, 674
nvinfer1::IRuntime, 700
NvInferRuntime.h, 948
getLoop
nvinfer1::ILoopBoundaryLayer, 409
getLoopOutput
nvinfer1::ILoopOutputLayer, 411
getMask
nvinfer1::IAttention, 171
getMaxAuxStreams
nvinfer1::IBuilderConfig, 205
getMaxDLABatchSize

nvinfer1::IBuilder, 194
getMaxNbTactics
nvinfer1::IBuilderConfig, 206
getMaxOutputSize
nvinfer1::IExecutionContext, 321
getMaxThreads
nvinfer1::IBuilder, 194
nvinfer1::IRefitter, 674
nvinfer1::IRuntime, 700
getMemoryPoolLimit
nvinfer1::IBuilderConfig, 206
getMessage
nvinfer1::IAssertionLayer, 164
getMetadata
nvinfer1::IAttention, 171
nvinfer1::ILayer, 396
getMetadataString
nvinfer1::v_1_0::IPluginV3OneBuild, 609
nvinfer2::safe::IPluginV3OneSafeBuild, 627
getMissing
nvinfer1::IRefitter, 674
getMissingWeights
nvinfer1::IRefitter, 675
getMode
nvinfer1::IGatherLayer, 356
nvinfer1::IScaleLayer, 742
nvinfer1::IScatterLayer, 747
nvinfer1::ISliceLayer, 763
getModelDtype
nvonnxparser::IOnnxConfig, 509
getModelFileName
nvonnxparser::IOnnxConfig, 509
getName
nvinfer1::IAttention, 171
nvinfer1::ICudaEngine, 252
nvinfer1::IExecutionContext, 322
nvinfer1::IIfConditional, 380
nvinfer1::ILayer, 396
nvinfer1::ILoop, 407
nvinfer1::INetworkDefinition, 481
nvinfer1::ITensor, 784
getNamedWeights
nvinfer1::IRefitter, 675
getNbAuxStreams
nvinfer1::ICudaEngine, 252
nvinfer2::safe::ITRTGraph, 810
getNbDLACores
nvinfer1::IBuilder, 194
nvinfer1::IRuntime, 701
getNbElementWiseDims
nvinfer1::IGatherLayer, 356
getNbErrors
nvinfer1::v_1_0::IErrorRecorder, 311
nvonnxparser::IParser, 531
nvonnxparser::IParserRefitter, 542
getNbGroups
nvinfer1::IConvolutionLayer, 237
nvinfer1::IDeconvolutionLayer, 277
nvinfer1::INormalizationLayer, 501
getNbInputs
nvinfer1::IAttention, 172
nvinfer1::ILayer, 396
nvinfer1::INetworkDefinition, 481
getNbIOProfiles
nvinfer2::safe::ITRTGraph, 811
getNbIOTensors
nvinfer1::ICudaEngine, 253
nvinfer2::safe::ITRTGraph, 811
getNbLayers
nvinfer1::ICudaEngine, 253
nvinfer1::INetworkDefinition, 482
getNbOptimizationProfiles
nvinfer1::IBuilderConfig, 207
nvinfer1::ICudaEngine, 253
getNbOutputMaps
nvinfer1::IConvolutionLayer, 237
nvinfer1::IDeconvolutionLayer, 277
getNbOutputs
nvinfer1::IAttention, 172
nvinfer1::ILayer, 396
nvinfer1::INetworkDefinition, 482
nvinfer1::IPluginV2, 579
nvinfer1::v_1_0::IPluginV3OneBuild, 609
nvinfer2::safe::IPluginV3OneSafeBuild, 628
getNbPluginsToSerialize
nvinfer1::IBuilderConfig, 207
getNbRanks
nvinfer1::IAttention, 172
nvinfer1::ILayer, 397
getNbShapeValues
nvinfer1::IOptimizationProfile, 515
getNbSubgraphs
nvonnxparser::IParser, 531
getNbTactics
nvinfer1::v_1_0::IPluginV3OneBuild, 610
nvinfer2::safe::IPluginV3OneSafeBuild, 628
getNearestRounding
nvinfer1::IResizeLayer, 685
getNormalizationOperation
nvinfer1::IAttention, 172
getNormalizationQuantizeScale
nvinfer1::IAttention, 173
getNormalizationQuantizeToType
nvinfer1::IAttention, 173
getNvOnnxParserVersion
NvOnnxParser.h, 995
getNvtxVerbosity
nvinfer1::IExecutionContext, 322

getOperation
nvinfer1::ICumulativeLayer, 270
nvinfer1::IElementWiseLayer, 300
nvinfer1::IFillLayer, 347
nvinfer1::IMatrixMultiplyLayer, 418
nvinfer1::IReduceLayer, 669
nvinfer1::ITopKLayer, 798
nvinfer1::IUnaryLayer, 819
getOptimizationProfile
nvinfer1::IExecutionContext, 322
getOutput
nvinfer1::IAttention, 173
nvinfer1::ILayer, 397
nvinfer1::INetworkDefinition, 482
getOutputAllocator
nvinfer1::IExecutionContext, 322
getOutputDataType
nvinfer1::IPluginV2Ext, 596
getOutputDataTypes
nvinfer1::v_1_0::IPluginV3OneBuild, 610
nvinfer2::safe::IPluginV3OneSafeBuild, 628
getOutputDimensions
nvinfer1::IPluginV2, 579
nvinfer1::IPluginV2DynamicExt, 589
nvinfer1::IResizeLayer, 685
getOutputShapes
nvinfer1::v_1_0::IPluginV3OneBuild, 611
nvinfer2::safe::IPluginV3OneSafeBuild, 629
nvinfer2::safe::IPluginV3OneSafeBuildMSS, 635
getOutputTensorAddress
nvinfer1::IExecutionContext, 323
getOutputType
nvinfer1::ILayer, 397
getPaddingMode
nvinfer1::IConvolutionLayer, 238
nvinfer1::IDeconvolutionLayer, 277
nvinfer1::IPoolingLayer, 649
getPaddingNd
nvinfer1::IConvolutionLayer, 238
nvinfer1::IDeconvolutionLayer, 278
nvinfer1::IPoolingLayer, 649
getPersistentCacheLimit
nvinfer1::IExecutionContext, 323
getPlugin
nvinfer1::IPluginV2Layer, 602
nvinfer1::IPluginV3Layer, 606
getPluginName
nvinfer1::v_1_0::IPluginCreator, 553
nvinfer1::v_1_0::IPluginCreatorV3One, 560
nvinfer1::v_1_0::IPluginV3OneCore, 618
nvinfer2::safe::IPluginV3OneSafeCore, 639
nvinfer2::safe::ISafePluginCreatorV3One, 716
getPluginNamespace
nvinfer1::IPluginV2, 580
nvinfer1::v_1_0::IPluginCreator, 553
nvinfer1::v_1_0::IPluginCreatorV3One, 561
nvinfer1::v_1_0::IPluginV3OneCore, 618
nvinfer2::safe::IPluginV3OneSafeCore, 640
nvinfer2::safe::ISafePluginCreatorV3One, 716
getPluginRegistry
nvinfer1::IBuilder, 195
nvinfer1::IRuntime, 701
NvInferRuntime.h, 949
getPluginToSerialize
nvinfer1::IBuilderConfig, 207
getPluginType
nvinfer1::IPluginV2, 580
getPluginVersion
nvinfer1::IPluginV2, 581
nvinfer1::v_1_0::IPluginCreator, 554
nvinfer1::v_1_0::IPluginCreatorV3One, 561
nvinfer1::v_1_0::IPluginV3OneCore, 619
nvinfer2::safe::IPluginV3OneSafeCore, 640
nvinfer2::safe::ISafePluginCreatorV3One, 717
getPoolingType
nvinfer1::IPoolingLayer, 649
getPostPadding
nvinfer1::IConvolutionLayer, 238
nvinfer1::IDeconvolutionLayer, 278
nvinfer1::IPoolingLayer, 650
getPostPaddingNd
nvinfer1::IPaddingLayer, 524
getPower
nvinfer1::IScaleLayer, 743
getPrePadding
nvinfer1::IConvolutionLayer, 238
nvinfer1::IDeconvolutionLayer, 278
nvinfer1::IPoolingLayer, 650
getPrePaddingNd
nvinfer1::IPaddingLayer, 525
getPreviewFeature
nvinfer1::IBuilderConfig, 208
getPrintLayerInfo
nvonnxparser::IOnnxConfig, 509
getProfiler
nvinfer1::IExecutionContext, 323
getProfileShape
nvinfer1::ICudaEngine, 254
getProfileStream
nvinfer1::IBuilderConfig, 208
getProfileTensorValuesV2
nvinfer1::ICudaEngine, 254
getProfilingVerbosity
nvinfer1::IBuilderConfig, 208
nvinfer1::ICudaEngine, 255
getProgressMonitor
nvinfer1::IBuilderConfig, 209
getQuantizationBlockShape

nvinfer1::IMoELayer, 424
getQuantizationToType
nvinfer1::IMoELayer, 424
getQueryForm
nvinfer1::IAttention, 174
getQueryLengths
nvinfer1::IAttention, 174
getReduceAxes
nvinfer1::IReduceLayer, 669
nvinfer1::ITopKLayer, 798
getRemoteAutoTuningConfig
nvinfer1::IBuilderConfig, 209
getReshapeDimensions
nvinfer1::IShuffleLayer, 756
getResizeMode
nvinfer1::IResizeLayer, 685
getReverse
nvinfer1::ICumulativeLayer, 270
nvinfer1::IIteratorLayer, 387
getRotaryEmbeddingDim
nvinfer1::IRotaryEmbeddingLayer, 695
getRuntimeConfig
nvinfer1::IExecutionContext, 324
getRuntimeErrorInformation
nvinfer2::safe::ISafePluginResourceContext, 728
getRuntimePlatform
nvinfer1::IBuilderConfig, 209
getSafeMemAllocator
nvinfer2::safe::ISafePluginResourceContext, 729
getSafePluginRegistry
nvinfer2::safe, 96
getSafeRecorder
nvinfer2::safe::ISafePluginCreatorV3One, 717
nvinfer2::safe::ISafePluginRegistry, 722
nvinfer2::safe::ISafePluginResourceContext, 729
nvinfer2::safe::ITRTGraph, 812
getSampleMode
nvinfer1::IGridSampleLayer, 371
getScale
nvinfer1::IScaleLayer, 743
getScales
nvinfer1::IResizeLayer, 686
getScaleType
nvinfer1::IDynamicQuantizeLayer, 294
getScratchMemory
nvinfer2::safe::ITRTGraph, 812
getScratchMemorySize
nvinfer2::safe::ITRTGraph, 813
getSecondTranspose
nvinfer1::IShuffleLayer, 757
getSelectorForSinglePixel
nvinfer1::IResizeLayer, 686
getSequenceAxis
nvinfer1::IReverseSequenceLayer, 692
getSerializationSize
nvinfer1::IPluginV2, 581
getSeverity
nvinfer2::safe::ISafeRecorder, 735
getShapeValuesV2
nvinfer1::IOptimizationProfile, 516
getShift
nvinfer1::IScaleLayer, 743
getSize
nvinfer1::ISliceLayer, 763
nvinfer2::safe::TypedArray, 853
getStart
nvinfer1::ISliceLayer, 764
getStreamableWeightsSize
nvinfer1::ICudaEngine, 255
getStride
nvinfer1::ISliceLayer, 764
getStrideNd
nvinfer1::IConvolutionLayer, 239
nvinfer1::IDeconvolutionLayer, 278
nvinfer1::IPoolingLayer, 650
getSubgraphNodes
nvonnxparser::IParser, 532
getSwigluParamAlpha
nvinfer1::IMoELayer, 424
getSwigluParamBeta
nvinfer1::IMoELayer, 425
getSwigluParamLimit
nvinfer1::IMoELayer, 425
getSymbolicOutputShapes
nvinfer2::safe::IPluginV3OneSafeBuildMSS, 636
getTacticSources
nvinfer1::IBuilderConfig, 210
nvinfer1::ICudaEngine, 255
getTempfileControlFlags
nvinfer1::IRuntime, 701
getTemporaryDirectory
nvinfer1::IRuntime, 701
getTemporaryStorageAllocator
nvinfer1::IExecutionContext, 324
getTensorAddress
nvinfer1::IExecutionContext, 324
getTensorBytesPerComponent
nvinfer1::ICudaEngine, 256
getTensorComponentsPerElement
nvinfer1::ICudaEngine, 257
getTensorDataType
nvinfer1::ICudaEngine, 258
getTensorFormat
nvinfer1::ICudaEngine, 258, 259
getTensorFormatDesc
nvinfer1::ICudaEngine, 259, 260
getTensorIOMode
nvinfer1::ICudaEngine, 260

getTensorLocation
nvinfer1::ICudaEngine, 261
getTensorRTVersion
nvinfer1::IPluginV2, 581
nvinfer1::IPluginV2DynamicExt, 590
nvinfer1::IPluginV2Ext, 597
nvinfer1::IPluginV2IOExt, 599
getTensorShape
nvinfer1::ICudaEngine, 261
nvinfer1::IExecutionContext, 325
getTensorStrides
nvinfer1::IExecutionContext, 326
getTensorVectorizedDim
nvinfer1::ICudaEngine, 262
getTextFileName
nvonnxparser::IOnnxConfig, 510
getTilingOptimizationLevel
nvinfer1::IBuilderConfig, 210
getTimingCache
nvinfer1::IBuilderConfig, 210
getTimingCacheID
nvinfer1::v_1_0::IPluginV3OneBuild, 611
nvinfer2::safe::IPluginV3OneSafeBuild, 630
getTopKBoxLimit
nvinfer1::INMSLayer, 492
getToType
nvinfer1::ICastLayer, 224
nvinfer1::IDequantizeLayer, 286
nvinfer1::IDynamicQuantizeLayer, 294
nvinfer1::IFillLayer, 348
nvinfer1::IQuantizeLayer, 663
getTripLimit
nvinfer1::ITripLimitLayer, 802
getTRTManagedScratch
nvinfer2::safe::ITRTGraph, 813
getType
nvinfer1::ILayer, 398
nvinfer1::ITensor, 785
nvinfer2::safe::TypedArray, 853
getUint8
nvinfer2::safe::TypedArray, 854
getUnfusedTensorsDebugState
nvinfer1::IExecutionContext, 326
getUpdateForm
nvinfer1::IKVCacheUpdateLayer, 389
getUpdateLengths
nvinfer1::IKVCacheUpdateLayer, 390
getUsedVCPluginLibraries
nvonnxparser::IParser, 532
getValidTactics
nvinfer1::v_1_0::IPluginV3OneBuild, 612
nvinfer2::safe::IPluginV3OneSafeBuild, 630
getVerbosityLevel
nvonnxparser::IOnnxConfig, 510
getWeights
nvinfer1::IConstantLayer, 233
getWeightsLocation
nvinfer1::IRefitter, 676
getWeightsPrototype
nvinfer1::IRefitter, 676
getWeightStreamingAutomaticBudget
nvinfer1::ICudaEngine, 263
getWeightStreamingBudgetV2
nvinfer1::ICudaEngine, 263
getWeightStreamingScratchMemorySize
nvinfer1::ICudaEngine, 264
getWeightsValidation
nvinfer1::IRefitter, 677
getWindowSize
nvinfer1::ILRNLayer, 415
getWindowSizeNd
nvinfer1::IPoolingLayer, 650
getWorkspaceSize
nvinfer1::IPluginV2, 582
nvinfer1::IPluginV2DynamicExt, 590
nvinfer1::v_1_0::IPluginV3OneBuild, 613
nvinfer2::safe::IPluginV3OneSafeBuild, 631
getZeroIsPlaceholder
nvinfer1::IShuffleLayer, 757
group
nvinfer1::plugin::softmaxTree, 842
groupOffset
nvinfer1::plugin::softmaxTree, 842
groups
nvinfer1::plugin::softmaxTree, 842
groupSize
nvinfer1::plugin::softmaxTree, 842

H
nvinfer1::plugin::GridAnchorParameters, 158
h
nvinfer1::DimsHW, 113
half_t
nvinfer2::safe, 90
HardwareCompatibilityLevel
nvinfer1, 58
hasImplicitBatchDimension
nvinfer1::INetworkDefinition, 483
hasOverflowed
nvinfer1::v_1_0::IErrorRecorder, 311

IConsistencyChecker
nvinfer2::safe::consistency::IConsistencyChecker, 229, 230
IDebugListener, 272
nvinfer1, 40
IErrorRecorder, 306
nvinfer1, 40
nvinfer1::v_1_0::IErrorRecorder, 308

nvinfer2::safe, 90
IGpuAllocator, 358
nvinfer1, 40
nvinfer1::v_1_0::IGpuAllocator, 360
IGpuAsyncAllocator, 365
nvinfer1, 41
nvinfer1::v_1_0::IGpuAsyncAllocator, 366
ILogger
nvinfer1, 41
nvinfer1::v_1_0::ILogger, 401
ILoggerFinder
nvinfer1, 41
imgH
nvinfer1::plugin::PriorBoxParameters, 834
imgW
nvinfer1::plugin::PriorBoxParameters, 834
incRefCount
nvinfer1::v_1_0::IErrorRecorder, 312
inferShapes
nvinfer1::IExecutionContext, 326
initialize
nvinfer1::IPluginV2, 582
initLibNvInferPlugins
NvInferPlugin.h, 937
initResource
nvinfer2::safe::IPluginV3OneSafeRuntime, 645
INoCopy
nvinfer1::INoCopy, 496
INonZero, 497
InterfaceInfo
nvinfer2::safe, 90
InterfaceKind
nvinfer1, 41
InterpolationMode
nvinfer1, 58
ioMode
nvinfer2::safe::TensorDescriptor, 845
iouThreshold
nvinfer1::plugin::RPROIParams, 839
IOutputAllocator, 520
nvinfer1, 41
IPluginCapability, 546
nvinfer1, 41
nvinfer2::safe, 90
IPluginChecker
nvinfer2::safe::consistency::IPluginChecker, 547, 548
IPluginCreator, 550
nvinfer1, 41
nvinfer1::v_1_0::IPluginCreator, 551
IPluginCreatorInterface, 555
nvinfer1, 42
nvinfer1::v_1_0::IPluginCreatorInterface, 556
nvinfer2::safe, 90
IPluginCreatorV3One, 557
nvinfer1, 42
nvinfer1::v_1_0::IPluginCreatorV3One, 558, 559
IPluginResource, 569
nvinfer1, 42
nvinfer1::v_1_0::IPluginResource, 570, 571
nvinfer2::safe, 90
IPluginResourceContext
nvinfer1::IPluginResourceContext, 573, 574
IPluginV2Ext
nvinfer1::IPluginV2Ext, 593
IPluginV3, 602
nvinfer1, 42
nvinfer2::safe, 91
IPluginV3OneBuild, 606
nvinfer1, 42
IPluginV3OneBuildV2, 616
nvinfer1, 42
IPluginV3OneCore, 617
nvinfer1, 42
IPluginV3OneRuntime, 619
nvinfer1, 43
IPluginV3OneSafeBuild
nvinfer2::safe::IPluginV3OneSafeBuild, 625, 626
IPluginV3OneSafeBuildMSS
nvinfer2::safe::IPluginV3OneSafeBuildMSS, 634, 635
IPluginV3OneSafeCore
nvinfer2::safe::IPluginV3OneSafeCore, 638, 639
IPluginV3OneSafeRuntime
nvinfer2::safe::IPluginV3OneSafeRuntime, 642, 643
IProfiler, 654
nvinfer1, 43
IProgressMonitor, 656
nvinfer1, 43
nvinfer1::v_1_0::IProgressMonitor, 657
ISafeMemAllocator
nvinfer2::safe::ISafeMemAllocator, 709, 710
ISafePluginCreatorV3One
nvinfer2::safe::ISafePluginCreatorV3One, 714
ISafePluginRegistry
nvinfer2::safe::ISafePluginRegistry, 719, 720
ISafePluginResourceContext
nvinfer2::safe::ISafePluginResourceContext, 727, 728
ISafeRecorder
nvinfer2::safe::ISafeRecorder, 733, 734
isAlphaBetaInt64
nvinfer1::IFillLayer, 348
isConstant
nvinfer1::IDimensionExpr, 289
isDebugTensor
nvinfer1::ICudaEngine, 264
nvinfer1::INetworkDefinition, 483

isDeviceTypeSet
nvinfer1::IBuilderConfig, 211
isExecutionTensor
nvinfer1::ITensor, 785
isNetworkInput
nvinfer1::ITensor, 785
isNetworkOutput
nvinfer1::ITensor, 786
isNetworkSupported
nvinfer1::IBuilder, 195
isParentSearchEnabled
nvinfer1::IPluginRegistry, 566
isRefittable
nvinfer1::ICudaEngine, 265
isShapeInferenceIO
nvinfer1::ICudaEngine, 265
isShapeTensor
nvinfer1::ITensor, 786
isSizeTensor
nvinfer1::IDimensionExpr, 289
isSubgraphSupported
nvonnxparser::IParser, 533
IStreamReader, 772
nvinfer1, 43
nvinfer1::v_1_0::IStreamReader, 773
IStreamReaderV2, 775
nvinfer1, 43
nvinfer1::v_1_0::IStreamReaderV2, 776
IStreamWriter, 778
nvinfer1, 43
nvinfer1::v_1_0::IStreamWriter, 779, 780
isV2
nvinfer1::INormalizationLayer, 502
isValid
nvinfer1::IOptimizationProfile, 516
ITRTGraph
nvinfer2::safe::ITRTGraph, 804, 805
IVersionedInterface
nvinfer1::IVersionedInterface, 824
nvinfer2::safe, 91

kABS
nvinfer1, 80
kACOS
nvinfer1, 80
kACOSH
nvinfer1, 80
kACTIVATION
nvinfer1, 60
kADJUST_FOR_DLA
nvonnxparser, 101
kALIASED_PLUGIN_IO_10_03
nvinfer1, 69
kALIGN_CORNERS
nvinfer1, 71
kALL_GATHER
nvinfer1, 51
kALL_REDUCE
nvinfer1, 51
kALL_TO_ALL
nvinfer1, 51
kALLOW_IN_MEMORY_FILES
nvinfer1, 75
kALLOW_TEMPORARY_FILES
nvinfer1, 75
kAMPERE_PLUS
nvinfer1, 58
kAND
nvinfer1, 54
kANY
nvinfer1, 80
kASIN
nvinfer1, 80
kASINH
nvinfer1, 80
kASSERTION
nvinfer1, 61
kASYMMETRIC
nvinfer1, 71
kATAN
nvinfer1, 80
kATANH
nvinfer1, 80
kATTENTION_INPUT
nvinfer1, 61
kATTENTION_OUTPUT
nvinfer1, 61
kAVERAGE
nvinfer1, 68
kAVG
nvinfer1, 70
kBF16
nvinfer1, 52, 68
kBIAS
nvinfer1, 80
kBOOL
nvinfer1, 52
kBROADCAST
nvinfer1, 51
kBUILD
nvinfer1, 67, 78
kCAST
nvinfer1, 60
kCDHW32
nvinfer1, 77
kCEIL
nvinfer1, 71, 80
kCEIL_DIV

nvinfer1, 53
kCENTER_SIZES
nvinfer1, 47
kCHANNEL
nvinfer1, 73
kCHAR
nvinfer1, 68
kCHW16
nvinfer1, 77
kCHW2
nvinfer1, 76
kCHW32
nvinfer1, 77
kCHW4
nvinfer1, 76
kCLAMP
nvinfer1, 73
kCLIP
nvinfer1, 45
kCONCATENATE
nvinfer1, 61
kCONCATENATION
nvinfer1, 60
kCONDITION
nvinfer1, 61
kCONDITIONAL_INPUT
nvinfer1, 61
kCONDITIONAL_OUTPUT
nvinfer1, 61
kCONSTANT
nvinfer1, 60, 80
kCONVOLUTION
nvinfer1, 60
kCORE
nvinfer1, 67
kCORNER_PAIRS
nvinfer1, 47
kCOS
nvinfer1, 80
kCOSH
nvinfer1, 80
kCOUNT
nvinfer1, 79
kCPP
nvinfer1, 46
kCPU
nvinfer2::safe, 92
kCPU_PINNED
nvinfer2::safe, 92
kCUBIC
nvinfer1, 59
kCUMULATIVE
nvinfer1, 61
kCUR
nvinfer1, 74
kDEBUG
nvinfer1, 47
nvinfer2::safe, 93
kDECONVOLUTION
nvinfer1, 60
kDEFAULT
nvinfer1, 58
kDEFAULT_FORMAT_COMBINATION_LIMIT
nvinfer1::v_1_0::IPluginV3OneBuild, 614
nvinfer2::safe::IPluginV3OneSafeBuild, 633
kDEQUANTIZE
nvinfer1, 61
kDETAILED
nvinfer1, 69
kDEVICE
nvinfer1, 78
kDHWC
nvinfer1, 77
kDHWC8
nvinfer1, 77
kDIMS
nvinfer1, 68
kDIRECT_IO
nvinfer1, 48
kDISABLE_COMPILATION_CACHE
nvinfer1, 48
kDISABLE_TIMING_CACHE
nvinfer1, 47
kDIST_COLLECTIVE
nvinfer1, 61
kDISTRIBUTIVE_INDEPENDENCE
nvinfer1, 50
kDIV
nvinfer1, 54
kDIV_ZERO
nvinfer2::safe, 93
kDLA
nvinfer1, 53
kDLA_GLOBAL_DRAM
nvinfer1, 62
kDLA_HWC4
nvinfer1, 77
kDLA_LINEAR
nvinfer1, 77
kDLA_LOCAL_DRAM
nvinfer1, 62
kDLA_MANAGED_SRAM
nvinfer1, 62
kDLA_STANDALONE
nvinfer1, 54
kDYNAMIC_QUANTIZE
nvinfer1, 61
kE8M0

nvinfer1, 52
kEDGE_MASK_CONVOLUTIONS
nvinfer1, 75
kEDITABLE_TIMING_CACHE
nvinfer1, 50
kEINSUM
nvinfer1, 61
kELEMENT
nvinfer1, 58, 74
kELEMENTWISE
nvinfer1, 60, 73
kELU
nvinfer1, 45
kENABLE_PLUGIN_OVERRIDE
nvonnxparser, 101
kENABLE_UINT8_AND_ASYMMETRIC_QUANTIZATION_DLA
nvonnxparser, 100
kEND
nvinfer1, 74
kEQUAL
nvinfer1, 53, 54
kERF
nvinfer1, 80
kERROR
nvinfer1::v_1_0::ILogger, 401
nvinfer2::safe, 93
kERROR_ON_TIMING_CACHE_MISS
nvinfer1, 48
kEXCLUDE_LEAN_RUNTIME
nvinfer1, 48, 74
kEXCLUDE_WEIGHTS
nvinfer1, 74
kEXP
nvinfer1, 80
kEXPLICIT_ROUND_DOWN
nvinfer1, 66
kEXPLICIT_ROUND_UP
nvinfer1, 66
kFAILED_ALLOCATION
nvinfer1, 55
kFAILED_COMPUTATION
nvinfer1, 56
kFAILED_EXECUTION
nvinfer1, 56
kFAILED_INITIALIZATION
nvinfer1, 55
kFAST
nvinfer1, 78
kFILL
nvinfer1, 60, 73
kFLOAT
nvinfer1, 52
kFLOAT16
nvinfer1, 67
kFLOAT32
nvinfer1, 67
kFLOAT64
nvinfer1, 68
kFLOOR
nvinfer1, 71, 80
kFLOOR_DIV
nvinfer1, 53, 54
kFORMAT_COMBINATION_LIMIT
nvinfer1::IPluginV2DynamicExt, 591
kFORMULA
nvinfer1, 72
kFP4
nvinfer1, 52, 68
kFP8
nvinfer1, 52, 68
kFULL
nvinfer1, 78
kGATHER
nvinfer1, 51, 60
kGATHER_OOB
nvinfer2::safe, 93
kGELU_ERF
nvinfer1, 45
kGELU_TANH
nvinfer1, 45
kGENERIC
nvinfer2::safe, 92
kGPU
nvinfer1, 53
nvinfer2::safe, 92
kGPU_FALLBACK
nvinfer1, 47
kGREATER
nvinfer1, 54
kGRID_SAMPLE
nvinfer1, 61
kHALF
nvinfer1, 52
kHALF_DOWN
nvinfer1, 71
kHALF_PIXEL
nvinfer1, 71
kHALF_UP
nvinfer1, 71
kHARD_SIGMOID
nvinfer1, 45
kHOST
nvinfer1, 78
kHWC
nvinfer1, 77
kHWC16
nvinfer1, 77
kHWC8

nvinfer1, 76
kIDENTITY
nvinfer1, 60
kIMMUTABLE
nvinfer2::safe, 92
kINCLUDE_REFIT
nvinfer1, 74
kind
nvinfer1::InterfaceInfo, 504
kINF_CONSUMED
nvinfer2::safe, 93
kINFO
nvinfer1::v_1_0::ILogger, 401
nvinfer2::safe, 93
kINPUT
nvinfer1, 77
kINT16
nvinfer1, 68
kINT32
nvinfer1, 52, 68
kINT4
nvinfer1, 52, 68
kINT64
nvinfer1, 52, 68
kINT8
nvinfer1, 52, 68
kINTERNAL_ERROR
nvinfer1, 55
nvinfer1::v_1_0::ILogger, 401
nvinfer2::safe, 93
nvonnxparser, 100
kINVALID_ARGUMENT
nvinfer1, 55
kINVALID_CONFIG
nvinfer1, 55
kINVALID_GRAPH
nvonnxparser, 100
kINVALID_NODE
nvonnxparser, 100
kINVALID_STATE
nvinfer1, 56
kINVALID_TACTIC_HASH
nvinfer1::v_1_0::TimingCacheValue, 848
kINVALID_VALUE
nvonnxparser, 100
kIOTENSOR
nvinfer2::safe, 92
kISINF
nvinfer1, 80
kISNAN
nvinfer1, 80
kITERATOR
nvinfer1, 60
kJIT_CONVOLUTIONS
nvinfer1, 75
kJSON
nvinfer1, 59
kKERNEL
nvinfer1, 80
kKVCACHE_UPDATE
nvinfer1, 61
kLAST_VALUE
nvinfer1, 61
kLAYER_NAMES_ONLY
nvinfer1, 69
kLEAKY_RELU
nvinfer1, 45
kLESS
nvinfer1, 53, 54
kLINEAR
nvinfer1, 59, 76
kLINSPACE
nvinfer1, 57
kLOG
nvinfer1, 80
kLOOP_OUTPUT
nvinfer1, 60
kLOWER_RIGHT
nvinfer1, 51
kLRN
nvinfer1, 60
kMANAGED
nvinfer2::safe, 92
kMATRIX_MULTIPLY
nvinfer1, 60
kMAX
nvinfer1, 53, 54, 64, 68, 70, 79
kMAX_AVERAGE_BLEND
nvinfer1, 68
kMAX_DESC_LENGTH
nvinfer1::v_1_0::IErrorRecorder, 313
kMAX_SAFE_DESC_LENGTH
nvinfer2::safe::ISafeRecorder, 739
kMAXIMUM_ALLOC_SIZE
nvinfer2::safe, 97
kMEM_ALLOC_FAILED
nvonnxparser, 100
kMIN
nvinfer1, 53, 54, 64, 70, 79
kMODEL_DESERIALIZE_FAILED
nvonnxparser, 100
kMODERATE
nvinfer1, 78
kMOE
nvinfer1, 61
kMONITOR_MEMORY
nvinfer1, 50
kNAN_CONSUMED

nvinfer2::safe, 93
kNATIVE_INSTANCENORM
nvonnxparser, 100
kND
nvinfer1, 58, 74
kNEAREST
nvinfer1, 59
kNEG
nvinfer1, 80
kNMS
nvinfer1, 61
kNON_ZERO
nvinfer1, 61
kNONE
nvinfer1, 47, 51, 58, 62, 63, 69, 70, 77, 78
nvinfer2::safe, 92
kNORMALIZATION
nvinfer1, 61
kNOT
nvinfer1, 80
kON_PROFILE_CHANGE
nvinfer1, 57
kONE_HOT
nvinfer1, 61
kONELINE
nvinfer1, 59
kOPT
nvinfer1, 64
kOR
nvinfer1, 54
kOUTPUT
nvinfer1, 77
kPACKED_NHD
nvinfer1, 46
kPADDED_BHND
nvinfer1, 46
kPADDING
nvinfer1, 60
kPARAMETRIC_RELU
nvinfer1, 60
kPLUGIN
nvinfer1, 60
kPLUGIN_V2
nvinfer1, 60
kPLUGIN_V3
nvinfer1, 61
kPOOLING
nvinfer1, 60
kPOW
nvinfer1, 54
kPREFER_AOT_PYTHON_PLUGINS
nvinfer1, 64
kPREFER_JIT_PYTHON_PLUGINS
nvinfer1, 64
kPROD
nvinfer1, 53, 54, 70
kPYTHON
nvinfer1, 46
kQUANTIZE
nvinfer1, 60
kRAGGED_SOFTMAX
nvinfer1, 60
kRANDOM_NORMAL
nvinfer1, 57
kRANDOM_UNIFORM
nvinfer1, 57
kRECIP
nvinfer1, 80
kRECURRENCE
nvinfer1, 60
kREDUCE
nvinfer1, 51, 60
kREDUCE_SCATTER
nvinfer1, 51
kREFIT
nvinfer1, 47
kREFIT_FAILED
nvonnxparser, 100
kREFIT_IDENTICAL
nvinfer1, 49
kREFIT_INDIVIDUAL
nvinfer1, 49
kREFLECT
nvinfer1, 73
kRELU
nvinfer1, 45
kREPORT_CAPABILITY_DLA
nvonnxparser, 101
kRESIZABLE
nvinfer1, 46
kRESIZE
nvinfer1, 60
kREVERSE
nvinfer1, 61
kREVERSE_SEQUENCE
nvinfer1, 61
kROTARY_EMBEDDING
nvinfer1, 61
kROUND
nvinfer1, 80
kRUNTIME
nvinfer1, 67, 78
kRUNTIME_ACTIVATION_RESIZE_10_10
nvinfer1, 69
kSAFETY
nvinfer1, 54
kSAFETY_SCOPE
nvinfer1, 48

kSAME_AS_BUILD
nvinfer1, 72
kSAME_COMPUTE_CAPABILITY
nvinfer1, 58
kSAME_LOWER
nvinfer1, 66
kSAME_UPPER
nvinfer1, 66
kSCALE
nvinfer1, 60, 80
kSCALED_TANH
nvinfer1, 45
kSCATTER
nvinfer1, 51, 61
kSCATTER_OOB
nvinfer2::safe, 93
kSCATTER_RACE
nvinfer2::safe, 93
kSCRATCH
nvinfer2::safe, 92
kSELECT
nvinfer1, 60
kSELU
nvinfer1, 45
kSET
nvinfer1, 74
kSHAPE
nvinfer1, 60
kSHIFT
nvinfer1, 80
kSHUFFLE
nvinfer1, 60
kSIGMOID
nvinfer1, 45
kSIGN
nvinfer1, 80
kSILU
nvinfer1, 63
kSIN
nvinfer1, 80
kSINH
nvinfer1, 80
kSLICE
nvinfer1, 60
kSOFTMAX
nvinfer1, 47, 60
kSOFTPLUS
nvinfer1, 45
kSOFTSIGN
nvinfer1, 45
kSPARSE_WEIGHTS
nvinfer1, 48
kSQRT
nvinfer1, 80
kSQUEEZE
nvinfer1, 61
kSTANDARD
nvinfer1, 54
kSTATIC
nvinfer1, 57
kSTRICT_BOUNDS
nvinfer1, 73
kSTRICT_NANS
nvinfer1, 50
kSTRIP_PLAN
nvinfer1, 48
kSTRIPPED_WEIGHTS_SIZE
nvinfer1, 55
kSTRONGLY_TYPED
nvinfer1, 64
kSUB
nvinfer1, 53, 54
kSUCCESS
nvinfer1, 55
nvonnxparser, 100
kSUM
nvinfer1, 52–54, 70
kTACTIC_DRAM
nvinfer1, 63
kTACTIC_SHARED_MEMORY
nvinfer1, 63
kTAN
nvinfer1, 80
kTANH
nvinfer1, 45
kTF32
nvinfer1, 48
kTHRESHOLDED_RELU
nvinfer1, 45
kTOPK
nvinfer1, 60
kTOTAL_WEIGHTS_SIZE
nvinfer1, 55
kTRANSPOSE
nvinfer1, 62
kTRIP_LIMIT
nvinfer1, 60
kUINT8
nvinfer1, 52
kUNARY
nvinfer1, 60
kUNIFORM
nvinfer1, 73
kUNKNOWN
nvinfer1, 68
kUNSPECIFIED_ERROR
nvinfer1, 55
kUNSQUEEZE

nvinfer1, 61
kUNSUPPORTED_GRAPH
nvonnxparser, 100
kUNSUPPORTED_NODE
nvonnxparser, 100
kUNSUPPORTED_NODE_ATTR
nvonnxparser, 100
kUNSUPPORTED_NODE_DATATYPE
nvonnxparser, 100
kUNSUPPORTED_NODE_DYNAMIC
nvonnxparser, 100
kUNSUPPORTED_NODE_INPUT
nvonnxparser, 100
kUNSUPPORTED_NODE_SHAPE
nvonnxparser, 100
kUNSUPPORTED_STATE
nvinfer1, 56
kUPPER
nvinfer1, 72
kUPPER_LEFT
nvinfer1, 51
kUSER_MANAGED
nvinfer1, 57
kV1
nvinfer1, 67
kV1_PYTHON
nvinfer1, 67
kV2
nvinfer1, 68
kV2_DYNAMICEXT
nvinfer1, 68
kV2_DYNAMICEXT_PYTHON
nvinfer1, 68
kV2_EXT
nvinfer1, 68
kV2_IOEXT
nvinfer1, 68
kVALUE
nvinfer1::impl::EnumMaxImpl< ActivationType >, 117
nvinfer1::impl::EnumMaxImpl< AllocatorFlag >, 118
nvinfer1::impl::EnumMaxImpl< APILanguage >, 119
nvinfer1::impl::EnumMaxImpl< AttentionIOForm >, 119
nvinfer1::impl::EnumMaxImpl< AttentionNormalizationOp >, 120
nvinfer1::impl::EnumMaxImpl< BoundingBoxFormat >, 121
nvinfer1::impl::EnumMaxImpl< BuilderFlag >, 121
nvinfer1::impl::EnumMaxImpl< CausalMaskKind >, 122
nvinfer1::impl::EnumMaxImpl< CollectiveOperation >, 123
nvinfer1::impl::EnumMaxImpl< CumulativeOperation >, 124
nvinfer1::impl::EnumMaxImpl< DataType >, 124
nvinfer1::impl::EnumMaxImpl< DeviceType >, 125
nvinfer1::impl::EnumMaxImpl< DimensionOperation >, 126
nvinfer1::impl::EnumMaxImpl< ElementWiseOperation >, 126
nvinfer1::impl::EnumMaxImpl< EngineCapability >, 127
nvinfer1::impl::EnumMaxImpl< EngineStat >, 128
nvinfer1::impl::EnumMaxImpl< ErrorCode >, 129
nvinfer1::impl::EnumMaxImpl< ExecutionContextAllocationStrategy >, 129
nvinfer1::impl::EnumMaxImpl< FillOperation >, 130
nvinfer1::impl::EnumMaxImpl< GatherMode >, 131
nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel >, 132
nvinfer1::impl::EnumMaxImpl< ILogger::Severity >, 132
nvinfer1::impl::EnumMaxImpl< InterpolationMode >, 133
nvinfer1::impl::EnumMaxImpl< KVCacheMode >, 134
nvinfer1::impl::EnumMaxImpl< LayerInformationFormat >, 135
nvinfer1::impl::EnumMaxImpl< LayerType >, 135
nvinfer1::impl::EnumMaxImpl< LoopOutput >, 136
nvinfer1::impl::EnumMaxImpl< MatrixOperation >, 137
nvinfer1::impl::EnumMaxImpl< MemoryPoolType >, 138
nvinfer1::impl::EnumMaxImpl< MoEActType >, 138
nvinfer1::impl::EnumMaxImpl< NetworkDefinitionCreationFlag >, 139
nvinfer1::impl::EnumMaxImpl< OptProfileSelector >, 140
nvinfer1::impl::EnumMaxImpl< PaddingMode >, 140
nvinfer1::impl::EnumMaxImpl< PoolingType >, 141
nvinfer1::impl::EnumMaxImpl< PreviewFeature >, 142
nvinfer1::impl::EnumMaxImpl< ProfilingVerbosity >, 143
nvinfer1::impl::EnumMaxImpl< ReduceOperation >, 143
nvinfer1::impl::EnumMaxImpl< ResizeCoordinate-

Transformation >, 144
nvinfer1::impl::EnumMaxImpl< ResizeRoundMode >, 145
nvinfer1::impl::EnumMaxImpl< ResizeSelector >, 145
nvinfer1::impl::EnumMaxImpl< RuntimePlatform >, 146
nvinfer1::impl::EnumMaxImpl< SampleMode >, 147
nvinfer1::impl::EnumMaxImpl< ScaleMode >, 148
nvinfer1::impl::EnumMaxImpl< ScatterMode >, 148
nvinfer1::impl::EnumMaxImpl< SerializationFlag >, 149
nvinfer1::impl::EnumMaxImpl< TacticSource >, 150
nvinfer1::impl::EnumMaxImpl< TempfileControlFlag >, 151
nvinfer1::impl::EnumMaxImpl< TensorFormat >, 152
nvinfer1::impl::EnumMaxImpl< TensorIOMode >, 153
nvinfer1::impl::EnumMaxImpl< TensorLocation >, 153
nvinfer1::impl::EnumMaxImpl< TilingOptimizationLevel >, 154
nvinfer1::impl::EnumMaxImpl< TopKOperation >, 155
nvinfer1::impl::EnumMaxImpl< TripLimit >, 156
nvinfer1::impl::EnumMaxImpl< UnaryOperation >, 156
nvinfer1::impl::EnumMaxImpl< WeightsRole >, 157
nvinfer2::safe::impl::EnumMaxImpl< Severity >, 150
KVCacheMode
nvinfer1, 59
kVECTOR
nvinfer1, 62
kVERBOSE
nvinfer1::v_1_0::ILogger, 401
nvinfer2::safe, 93
kVERSION_COMPATIBLE
nvinfer1, 48
kWARNING
nvinfer1::v_1_0::ILogger, 401
nvinfer2::safe, 93
kWEIGHT_STREAMING
nvinfer1, 49
kWHILE
nvinfer1, 79
kWINDOWS_AMD64
nvinfer1, 72
kWORKSPACE
nvinfer1, 62
kWRAP
nvinfer1, 73
kXOR
nvinfer1, 54

LayerInformationFormat
nvinfer1, 59
LayerType
nvinfer1, 59
leaf
nvinfer1::plugin::softmaxTree, 843
length
nvinfer1::PluginField, 828
line
nvonnxparser::IParserError, 540
loadInitializer
nvonnxparser::IParser, 533
nvonnxparser::IParserRefitter, 543
loadLibrary
nvinfer1::IPluginRegistry, 567
loadModelProto
nvonnxparser::IParser, 534
nvonnxparser::IParserRefitter, 544
loadRuntime
nvinfer1::IRuntime, 702
localFunctionStack
nvonnxparser::IParserError, 540
localFunctionStackSize
nvonnxparser::IParserError, 540
log
nvinfer1::v_1_0::ILogger, 402
LoopOutput
nvinfer1, 61

major
nvinfer1::InterfaceInfo, 504
markDebug
nvinfer1::INetworkDefinition, 483
markOutput
nvinfer1::INetworkDefinition, 484
markOutputForShapes
nvinfer1::INetworkDefinition, 484
markUnfusedTensorsAsDebugTensors
nvinfer1::INetworkDefinition, 485
markWeightsRefittable
nvinfer1::INetworkDefinition, 485
MatrixOperation
nvinfer1, 61
max
nvinfer1::DynamicPluginTensorDesc, 115
MAX_DIMS
nvinfer1::Dims64, 110
nvinfer2::safe::PhysicalDims, 827
maxSize

nvinfer1::plugin::GridAnchorParameters, 158
nvinfer1::plugin::PriorBoxParameters, 835
mBoundary
nvinfer1::IAttentionBoundaryLayer, 184
nvinfer1::IIfConditionalBoundaryLayer, 382
nvinfer1::ILoopBoundaryLayer, 409
MemoryPlacement
nvinfer2::safe, 92
MemoryPoolType
nvinfer1, 62
MemoryUsage
nvinfer2::safe, 92
memPlacement
nvinfer2::safe::TensorDescriptor, 845
mId
nvinfer2::safe::ISafeRecorder, 740
mImpl
nvinfer1::IActivationLayer, 163
nvinfer1::IAssertionLayer, 165
nvinfer1::IAttention, 182
nvinfer1::IAttentionInputLayer, 186
nvinfer1::IAttentionOutputLayer, 187
nvinfer1::IBuilder, 197
nvinfer1::IBuilderConfig, 223
nvinfer1::ICastLayer, 225
nvinfer1::IConcatenationLayer, 227
nvinfer1::IConditionLayer, 228
nvinfer1::IConstantLayer, 234
nvinfer1::IConvolutionLayer, 244
nvinfer1::ICudaEngine, 268
nvinfer1::ICumulativeLayer, 272
nvinfer1::IDeconvolutionLayer, 283
nvinfer1::IDequantizeLayer, 287
nvinfer1::IDimensionExpr, 289
nvinfer1::IDistCollectiveLayer, 291
nvinfer1::IDynamicQuantizeLayer, 296
nvinfer1::IEinsumLayer, 299
nvinfer1::IElementWiseLayer, 301
nvinfer1::IEngineInspector, 306
nvinfer1::IExecutionContext, 340
nvinfer1::IExprBuilder, 343
nvinfer1::IFillLayer, 353
nvinfer1::IGatherLayer, 358
nvinfer1::IGridSampleLayer, 373
nvinfer1::IHostMemory, 375
nvinfer1::IIdentityLayer, 377
nvinfer1::IIfConditional, 381
nvinfer1::IIfConditionalInputLayer, 384
nvinfer1::IIfConditionalOutputLayer, 385
nvinfer1::IIteratorLayer, 387
nvinfer1::IKVCacheUpdateLayer, 392
nvinfer1::ILoop, 408
nvinfer1::ILoopOutputLayer, 412
nvinfer1::ILRNLayer, 416
nvinfer1::IMatrixMultiplyLayer, 419
nvinfer1::IMoELayer, 432
nvinfer1::INetworkDefinition, 490
nvinfer1::INMSLayer, 494
nvinfer1::INonZeroLayer, 499
nvinfer1::INormalizationLayer, 503
nvinfer1::IOneHotLayer, 506
nvinfer1::IOptimizationProfile, 520
nvinfer1::IPaddingLayer, 526
nvinfer1::IParametricReLULayer, 527
nvinfer1::IPluginV2Layer, 602
nvinfer1::IPluginV3Layer, 606
nvinfer1::IPoolingLayer, 654
nvinfer1::IQuantizeLayer, 664
nvinfer1::IRaggedSoftMaxLayer, 665
nvinfer1::IRecurrenceLayer, 667
nvinfer1::IReduceLayer, 671
nvinfer1::IRefitter, 681
nvinfer1::IResizeLayer, 690
nvinfer1::IReverseSequenceLayer, 693
nvinfer1::IRotaryEmbeddingLayer, 696
nvinfer1::IRuntime, 706
nvinfer1::IRuntimeConfig, 708
nvinfer1::IScaleLayer, 745
nvinfer1::IScatterLayer, 748
nvinfer1::ISelectLayer, 750
nvinfer1::ISerializationConfig, 753
nvinfer1::IShapeLayer, 754
nvinfer1::IShuffleLayer, 760
nvinfer1::ISliceLayer, 767
nvinfer1::ISoftMaxLayer, 769
nvinfer1::ISqueezeLayer, 771
nvinfer1::ITensor, 791
nvinfer1::ITimingCache, 796
nvinfer1::ITopKLayer, 801
nvinfer1::ITripLimitLayer, 802
nvinfer1::IUnaryLayer, 820
nvinfer1::IUnsqueezeLayer, 822
min
nvinfer1::DynamicPluginTensorDesc, 115
minBoxSize
nvinfer1::plugin::RPROIParams, 839
minor
nvinfer1::InterfaceInfo, 504
minSize
nvinfer1::plugin::GridAnchorParameters, 159
nvinfer1::plugin::PriorBoxParameters, 835
mLayer
nvinfer1::ILayer, 400
MoEActType
nvinfer1, 63
mSeverity
nvinfer2::safe::ISafeRecorder, 740

n
nvinfer1::plugin::softmaxTree, 843
name
nvinfer1::plugin::softmaxTree, 843
nvinfer1::PluginField, 829
nbDims
nvinfer1::Dims64, 110
nvinfer1::DimsExprs, 111
nvinfer2::safe::PhysicalDims, 827
nbFields
nvinfer1::PluginFieldCollection, 830
NetworkDefinitionCreationFlag
nvinfer1, 63
NetworkDefinitionCreationFlags
nvinfer1, 43
nmsMaxOut
nvinfer1::plugin::RPROIParams, 839
node
nvonnxparser::IParserError, 540
nodeName
nvonnxparser::IParserError, 540
nodeOperator
nvonnxparser::IParserError, 541
notifyShape
nvinfer1::v_1_0::IOutputAllocator, 521
num
nvinfer1::plugin::RegionParameters, 837
numAspectRatios
nvinfer1::plugin::GridAnchorParameters, 159
nvinfer1::plugin::PriorBoxParameters, 835
numMaxSize
nvinfer1::plugin::PriorBoxParameters, 835
numMinSize
nvinfer1::plugin::PriorBoxParameters, 835
NV_INFER_INTERNAL_INCLUDE
NvInferForwardDecl.h, 1001
NvInferLegacyDims.h, 934
NvInferPluginBase.h, 939
NvInferRuntime.h, 947
NvInferRuntimeCommon.h, 981
NvInferRuntimePlugin.h, 984
NV_ONNX_PARSER_MAJOR
NvOnnxParser.h, 995
NV_ONNX_PARSER_MINOR
NvOnnxParser.h, 995
NV_ONNX_PARSER_PATCH
NvOnnxParser.h, 995
NV_TENSORRT_BUILD
NvInferVersion.h, 988
NV_TENSORRT_LWS_MAJOR
NvInferVersion.h, 988
NV_TENSORRT_LWS_MINOR
NvInferVersion.h, 989
NV_TENSORRT_LWS_PATCH
NvInferVersion.h, 989
NV_TENSORRT_MAJOR
NvInferVersion.h, 989
NV_TENSORRT_MINOR
NvInferVersion.h, 989
NV_TENSORRT_PATCH
NvInferVersion.h, 989
NV_TENSORRT_RELEASE_TYPE
NvInferVersion.h, 989
NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS
NvInferVersion.h, 990
NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY
NvInferVersion.h, 990
NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE
NvInferVersion.h, 990
NV_TENSORRT_VERSION
NvInferRuntimeBase.h, 974
NV_TENSORRT_VERSION_INT
NvInferRuntimeBase.h, 974
NvInfer.h, 857, 865
nvinfer1, 29
ActivationType, 45
AllocatorFlag, 46
AllocatorFlags, 39
APILanguage, 46
AsciiChar, 39
AttentionIOForm, 46
AttentionNormalizationOp, 46
BoundingBoxFormat, 47
BuilderFlag, 47
BuilderFlags, 40
CausalMaskKind, 50
char_t, 40
CollectiveOperation, 51
CumulativeOperation, 51
DataType, 52
DeviceType, 52
DimensionOperation, 53
Dims, 40
ElementWiseOperation, 53
EngineCapability, 54
EngineStat, 54
EnumMax, 81
ErrorCode, 55
ExecutionContextAllocationStrategy, 56
FillOperation, 57
GatherMode, 57
getBuilderPluginRegistry, 81
getBuilderSafePluginRegistry, 81
HardwareCompatibilityLevel, 58
IDebugListener, 40
IErrorRecorder, 40
IGpuAllocator, 40
IGpuAsyncAllocator, 41

ILogger, 41
ILoggerFinder, 41
InterfaceKind, 41
InterpolationMode, 58
IOutputAllocator, 41
IPluginCapability, 41
IPluginCreator, 41
IPluginCreatorInterface, 42
IPluginCreatorV3One, 42
IPluginResource, 42
IPluginV3, 42
IPluginV3OneBuild, 42
IPluginV3OneBuildV2, 42
IPluginV3OneCore, 42
IPluginV3OneRuntime, 43
IProfiler, 43
IProgressMonitor, 43
IStreamReader, 43
IStreamReaderV2, 43
IStreamWriter, 43
kABS, 80
kACOS, 80
kACOSH, 80
kACTIVATION, 60
kALIASED_PLUGIN_IO_10_03, 69
kALIGN_CORNERS, 71
kALL_GATHER, 51
kALL_REDUCE, 51
kALL_TO_ALL, 51
kALLOW_IN_MEMORY_FILES, 75
kALLOW_TEMPORARY_FILES, 75
kAMPERE_PLUS, 58
kAND, 54
kANY, 80
kASIN, 80
kASINH, 80
kASSERTION, 61
kASYMMETRIC, 71
kATAN, 80
kATANH, 80
kATTENTION_INPUT, 61
kATTENTION_OUTPUT, 61
kAVERAGE, 68
kAVG, 70
kBF16, 52, 68
kBIAS, 80
kBOOL, 52
kBROADCAST, 51
kBUILD, 67, 78
kCAST, 60
kCDHW32, 77
kCEIL, 71, 80
kCEIL_DIV, 53
kCENTER_SIZES, 47
kCHANNEL, 73
kCHAR, 68
kCHW16, 77
kCHW2, 76
kCHW32, 77
kCHW4, 76
kCLAMP, 73
kCLIP, 45
kCONCATENATE, 61
kCONCATENATION, 60
kCONDITION, 61
kCONDITIONAL_INPUT, 61
kCONDITIONAL_OUTPUT, 61
kCONSTANT, 60, 80
kCONVOLUTION, 60
kCORE, 67
kCORNER_PAIRS, 47
kCOS, 80
kCOSH, 80
kCOUNT, 79
kCPP, 46
kCUBIC, 59
kCUMULATIVE, 61
kCUR, 74
kDEBUG, 47
kDECONVOLUTION, 60
kDEFAULT, 58
kDEQUANTIZE, 61
kDETAILED, 69
kDEVICE, 78
kDHWC, 77
kDHWC8, 77
kDIMS, 68
kDIRECT_IO, 48
kDISABLE_COMPILATION_CACHE, 48
kDISABLE_TIMING_CACHE, 47
kDIST_COLLECTIVE, 61
kDISTRIBUTIVE_INDEPENDENCE, 50
kDIV, 54
kDLA, 53
kDLA_GLOBAL_DRAM, 62
kDLA_HWC4, 77
kDLA_LINEAR, 77
kDLA_LOCAL_DRAM, 62
kDLA_MANAGED_SRAM, 62
kDLA_STANDALONE, 54
kDYNAMIC_QUANTIZE, 61
kE8M0, 52
kEDGE_MASK_CONVOLUTIONS, 75
kEDITABLE_TIMING_CACHE, 50
kEINSUM, 61
kELEMENT, 58, 74
kELEMENTWISE, 60, 73
kELU, 45

kEND, 74
kEQUAL, 53, 54
kERF, 80
kERROR_ON_TIMING_CACHE_MISS, 48
kEXCLUDE_LEAN_RUNTIME, 48, 74
kEXCLUDE_WEIGHTS, 74
kEXP, 80
kEXPLICIT_ROUND_DOWN, 66
kEXPLICIT_ROUND_UP, 66
kFAILED_ALLOCATION, 55
kFAILED_COMPUTATION, 56
kFAILED_EXECUTION, 56
kFAILED_INITIALIZATION, 55
kFAST, 78
kFILL, 60, 73
kFLOAT, 52
kFLOAT16, 67
kFLOAT32, 67
kFLOAT64, 68
kFLOOR, 71, 80
kFLOOR_DIV, 53, 54
kFORMULA, 72
kFP4, 52, 68
kFP8, 52, 68
kFULL, 78
kGATHER, 51, 60
kGELU_ERF, 45
kGELU_TANH, 45
kGPU, 53
kGPU_FALLBACK, 47
kGREATER, 54
kGRID_SAMPLE, 61
kHALF, 52
kHALF_DOWN, 71
kHALF_PIXEL, 71
kHALF_UP, 71
kHARD_SIGMOID, 45
kHOST, 78
kHWC, 77
kHWC16, 77
kHWC8, 76
kIDENTITY, 60
kINCLUDE_REFIT, 74
kINPUT, 77
kINT16, 68
kINT32, 52, 68
kINT4, 52, 68
kINT64, 52, 68
kINT8, 52, 68
kINTERNAL_ERROR, 55
kINVALID_ARGUMENT, 55
kINVALID_CONFIG, 55
kINVALID_STATE, 56
kISINF, 80
kISNAN, 80
kITERATOR, 60
kJIT_CONVOLUTIONS, 75
kJSON, 59
kKERNEL, 80
kKVCACHE_UPDATE, 61
kLAST_VALUE, 61
kLAYER_NAMES_ONLY, 69
kLEAKY_RELU, 45
kLESS, 53, 54
kLINEAR, 59, 76
kLINSPACE, 57
kLOG, 80
kLOOP_OUTPUT, 60
kLOWER_RIGHT, 51
kLRN, 60
kMATRIX_MULTIPLY, 60
kMAX, 53, 54, 64, 68, 70, 79
kMAX_AVERAGE_BLEND, 68
kMIN, 53, 54, 64, 70, 79
kMODERATE, 78
kMOE, 61
kMONITOR_MEMORY, 50
kND, 58, 74
kNEAREST, 59
kNEG, 80
kNMS, 61
kNON_ZERO, 61
kNONE, 47, 51, 58, 62, 63, 69, 70, 77, 78
kNORMALIZATION, 61
kNOT, 80
kON_PROFILE_CHANGE, 57
kONE_HOT, 61
kONELINE, 59
kOPT, 64
kOR, 54
kOUTPUT, 77
kPACKED_NHD, 46
kPADDED_BHND, 46
kPADDING, 60
kPARAMETRIC_RELU, 60
kPLUGIN, 60
kPLUGIN_V2, 60
kPLUGIN_V3, 61
kPOOLING, 60
kPOW, 54
kPREFER_AOT_PYTHON_PLUGINS, 64
kPREFER_JIT_PYTHON_PLUGINS, 64
kPROD, 53, 54, 70
kPYTHON, 46
kQUANTIZE, 60
kRAGGED_SOFTMAX, 60
kRANDOM_NORMAL, 57
kRANDOM_UNIFORM, 57

kRECIP, 80
kRECURRENCE, 60
kREDUCE, 51, 60
kREDUCE_SCATTER, 51
kREFIT, 47
kREFIT_IDENTICAL, 49
kREFIT_INDIVIDUAL, 49
kREFLECT, 73
kRELU, 45
kRESIZABLE, 46
kRESIZE, 60
kREVERSE, 61
kREVERSE_SEQUENCE, 61
kROTARY_EMBEDDING, 61
kROUND, 80
kRUNTIME, 67, 78
kRUNTIME_ACTIVATION_RESIZE_10_10, 69
kSAFETY, 54
kSAFETY_SCOPE, 48
kSAME_AS_BUILD, 72
kSAME_COMPUTE_CAPABILITY, 58
kSAME_LOWER, 66
kSAME_UPPER, 66
kSCALE, 60, 80
kSCALED_TANH, 45
kSCATTER, 51, 61
kSELECT, 60
kSELU, 45
kSET, 74
kSHAPE, 60
kSHIFT, 80
kSHUFFLE, 60
kSIGMOID, 45
kSIGN, 80
kSILU, 63
kSIN, 80
kSINH, 80
kSLICE, 60
kSOFTMAX, 47, 60
kSOFTPLUS, 45
kSOFTSIGN, 45
kSPARSE_WEIGHTS, 48
kSQRT, 80
kSQUEEZE, 61
kSTANDARD, 54
kSTATIC, 57
kSTRICT_BOUNDS, 73
kSTRICT_NANS, 50
kSTRIP_PLAN, 48
kSTRIPPED_WEIGHTS_SIZE, 55
kSTRONGLY_TYPED, 64
kSUB, 53, 54
kSUCCESS, 55
kSUM, 52–54, 70
kTACTIC_DRAM, 63
kTACTIC_SHARED_MEMORY, 63
kTAN, 80
kTANH, 45
kTF32, 48
kTHRESHOLDED_RELU, 45
kTOPK, 60
kTOTAL_WEIGHTS_SIZE, 55
kTRANSPOSE, 62
kTRIP_LIMIT, 60
kUINT8, 52
kUNARY, 60
kUNIFORM, 73
kUNKNOWN, 68
kUNSPECIFIED_ERROR, 55
kUNSQUEEZE, 61
kUNSUPPORTED_STATE, 56
kUPPER, 72
kUPPER_LEFT, 51
kUSER_MANAGED, 57
kV1, 67
kV1_PYTHON, 67
kV2, 68
kV2_DYNAMICEXT, 68
kV2_DYNAMICEXT_PYTHON, 68
kV2_EXT, 68
kV2_IOEXT, 68
KVCacheMode, 59
kVECTOR, 62
kVERSION_COMPATIBLE, 48
kWEIGHT_STREAMING, 49
kWHILE, 79
kWINDOWS_AMD64, 72
kWORKSPACE, 62
kWRAP, 73
kXOR, 54
LayerInformationFormat, 59
LayerType, 59
LoopOutput, 61
MatrixOperation, 61
MemoryPoolType, 62
MoEActType, 63
NetworkDefinitionCreationFlag, 63
NetworkDefinitionCreationFlags, 43
OptProfileSelector, 64
PaddingMode, 64
PluginCapabilityType, 67
PluginCreatorVersion, 67
PluginFieldType, 67
PluginFormat, 44
PluginVersion, 68
PoolingType, 68
PreviewFeature, 69
ProfilingVerbosity, 69

ReduceOperation, 69
ResizeCoordinateTransformation, 70
ResizeRoundMode, 71
ResizeSelector, 71
RuntimePlatform, 72
SampleMode, 73
ScaleMode, 73
ScatterMode, 73
SeekPosition, 74
SerializationFlag, 74
SerializationFlags, 44
setInternalLibraryPath, 81
TacticSource, 75
TacticSources, 44
TempfileControlFlag, 75
TempfileControlFlags, 44
TensorFormat, 75
TensorFormats, 45
TensorIOMode, 77
TensorLocation, 78
TensorRTPhase, 78
TilingOptimizationLevel, 78
TopKOperation, 79
TripLimit, 79
UnaryOperation, 79
WeightsRole, 80
nvinfer1::anonymous_namespace{NvInfer.h}, 82
createInferBuilder, 82
nvinfer1::anonymous_namespace{NvInferRuntime.h}, 82
createInferRefitter, 83
createInferRuntime, 83
nvinfer1::Dims2, 105
Dims2, 106
nvinfer1::Dims3, 107
Dims3, 107
nvinfer1::Dims4, 108
Dims4, 109
nvinfer1::Dims64, 109
d, 110
MAX_DIMS, 110
nbDims, 110
nvinfer1::DimsExprs, 111
d, 111
nbDims, 111
nvinfer1::DimsHW, 112
DimsHW, 113
h, 113
w, 114
nvinfer1::DynamicPluginTensorDesc, 114
desc, 115
max, 115
min, 115
opt, 115
nvinfer1::IActivationLayer, 160
~IActivationLayer, 161
getActivationType, 161
getAlpha, 161
getBeta, 161
mImpl, 163
setActivationType, 162
setAlpha, 162
setBeta, 162
nvinfer1::IAssertionLayer, 163
~IAssertionLayer, 164
getMessage, 164
mImpl, 165
setMessage, 165
nvinfer1::IAttention, 165
~IAttention, 168
getCausal, 169
getCausalKind, 169
getDecomposable, 169
getInput, 170
getKeyValueForm, 170
getKeyValueLengths, 170
getMask, 171
getMetadata, 171
getName, 171
getNbInputs, 172
getNbOutputs, 172
getNbRanks, 172
getNormalizationOperation, 172
getNormalizationQuantizeScale, 173
getNormalizationQuantizeToType, 173
getOutput, 173
getQueryForm, 174
getQueryLengths, 174
mImpl, 182
setCausal, 174
setCausalKind, 175
setDecomposable, 175
setInput, 176
setKeyValueForm, 176
setKeyValueLengths, 177
setMask, 178
setMetadata, 178
setName, 179
setNbRanks, 179
setNormalizationOperation, 180
setNormalizationQuantizeScale, 180
setNormalizationQuantizeToType, 181
setQueryForm, 181
setQueryLengths, 182
nvinfer1::IAttentionBoundaryLayer, 183
~IAttentionBoundaryLayer, 183
getAttention, 184
mBoundary, 184
nvinfer1::IAttentionInputLayer, 184

~IAttentionInputLayer, 185
mImpl, 186
setInput, 185
nvinfer1::IAttentionOutputLayer, 186
~IAttentionOutputLayer, 187
mImpl, 187
nvinfer1::IBuilder, 188
~IBuilder, 189
buildEngineWithConfig, 189
buildSerializedNetwork, 190, 191
buildSerializedNetworkToStream, 191
createBuilderConfig, 192
createNetworkV2, 192
createOptimizationProfile, 193
getErrorRecorder, 193
getLogger, 193
getMaxDLABatchSize, 194
getMaxThreads, 194
getNbDLACores, 194
getPluginRegistry, 195
isNetworkSupported, 195
mImpl, 197
reset, 196
setErrorRecorder, 196
setGpuAllocator, 196
setMaxThreads, 197
nvinfer1::IBuilderConfig, 198
~IBuilderConfig, 201
addOptimizationProfile, 201
canRunOnDLA, 201
clearFlag, 201
createTimingCache, 202
getAvgTimingIterations, 202
getBuilderOptimizationLevel, 203
getDefaultDeviceType, 203
getDeviceType, 203
getDLACore, 203
getEngineCapability, 204
getFlag, 204
getFlags, 204
getHardwareCompatibilityLevel, 205
getL2LimitForTiling, 205
getMaxAuxStreams, 205
getMaxNbTactics, 206
getMemoryPoolLimit, 206
getNbOptimizationProfiles, 207
getNbPluginsToSerialize, 207
getPluginToSerialize, 207
getPreviewFeature, 208
getProfileStream, 208
getProfilingVerbosity, 208
getProgressMonitor, 209
getRemoteAutoTuningConfig, 209
getRuntimePlatform, 209
getTacticSources, 210
getTilingOptimizationLevel, 210
getTimingCache, 210
isDeviceTypeSet, 211
mImpl, 223
reset, 211
resetDeviceType, 211
setAvgTimingIterations, 211
setBuilderOptimizationLevel, 212
setDefaultDeviceType, 213
setDeviceType, 213
setDLACore, 213
setEngineCapability, 214
setFlag, 214
setFlags, 214
setHardwareCompatibilityLevel, 216
setL2LimitForTiling, 216
setMaxAuxStreams, 217
setMaxNbTactics, 217
setMemoryPoolLimit, 218
setPluginsToSerialize, 218
setPreviewFeature, 219
setProfileStream, 219
setProfilingVerbosity, 220
setProgressMonitor, 220
setRemoteAutoTuningConfig, 220
setRuntimePlatform, 221
setTacticSources, 221
setTilingOptimizationLevel, 222
setTimingCache, 222
nvinfer1::ICastLayer, 223
~ICastLayer, 224
getToType, 224
mImpl, 225
setToType, 224
nvinfer1::IConcatenationLayer, 225
~IConcatenationLayer, 226
getAxis, 226
mImpl, 227
setAxis, 226
nvinfer1::IConditionLayer, 227
~IConditionLayer, 228
mImpl, 228
nvinfer1::IConstantLayer, 232
~IConstantLayer, 233
getDimensions, 233
getWeights, 233
mImpl, 234
setDimensions, 233
setWeights, 234
nvinfer1::IConvolutionLayer, 234
~IConvolutionLayer, 236
getBiasWeights, 236
getDilationNd, 236

getKernelSizeNd, 237
getKernelWeights, 237
getNbGroups, 237
getNbOutputMaps, 237
getPaddingMode, 238
getPaddingNd, 238
getPostPadding, 238
getPrePadding, 238
getStrideNd, 239
mImpl, 244
setBiasWeights, 239
setDilationNd, 239
setInput, 240
setKernelSizeNd, 240
setKernelWeights, 241
setNbGroups, 241
setNbOutputMaps, 241
setPaddingMode, 242
setPaddingNd, 242
setPostPadding, 242
setPrePadding, 243
setStrideNd, 243
nvinfer1::ICudaEngine, 244
~ICudaEngine, 247
createEngineInspector, 247
createExecutionContext, 247, 248
createRuntimeConfig, 248
createSerializationConfig, 249
getAliasedInputTensor, 249
getDeviceMemorySizeForProfileV2, 249
getDeviceMemorySizeV2, 250
getEngineCapability, 250
getEngineStat, 250
getErrorRecorder, 251
getHardwareCompatibilityLevel, 251
getIOTensorName, 252
getName, 252
getNbAuxStreams, 252
getNbIOTensors, 253
getNbLayers, 253
getNbOptimizationProfiles, 253
getProfileShape, 254
getProfileTensorValuesV2, 254
getProfilingVerbosity, 255
getStreamableWeightsSize, 255
getTacticSources, 255
getTensorBytesPerComponent, 256
getTensorComponentsPerElement, 257
getTensorDataType, 258
getTensorFormat, 258, 259
getTensorFormatDesc, 259, 260
getTensorIOMode, 260
getTensorLocation, 261
getTensorShape, 261
getTensorVectorizedDim, 262
getWeightStreamingAutomaticBudget, 263
getWeightStreamingBudgetV2, 263
getWeightStreamingScratchMemorySize, 264
isDebugTensor, 264
isRefittable, 265
isShapeInferenceIO, 265
mImpl, 268
serialize, 265
serializeWithConfig, 266
setErrorRecorder, 266
setWeightStreamingBudgetV2, 267
nvinfer1::ICumulativeLayer, 268
~ICumulativeLayer, 269
getExclusive, 269
getOperation, 270
getReverse, 270
mImpl, 272
setExclusive, 270
setOperation, 271
setReverse, 271
nvinfer1::IDeconvolutionLayer, 274
~IDeconvolutionLayer, 276
getBiasWeights, 276
getDilationNd, 276
getKernelSizeNd, 276
getKernelWeights, 277
getNbGroups, 277
getNbOutputMaps, 277
getPaddingMode, 277
getPaddingNd, 278
getPostPadding, 278
getPrePadding, 278
getStrideNd, 278
mImpl, 283
setBiasWeights, 279
setDilationNd, 279
setInput, 279
setKernelSizeNd, 280
setKernelWeights, 280
setNbGroups, 280
setNbOutputMaps, 281
setPaddingMode, 281
setPaddingNd, 281
setPostPadding, 282
setPrePadding, 282
setStrideNd, 282
nvinfer1::IDequantizeLayer, 283
~IDequantizeLayer, 285
getAxis, 285
getBlockShape, 286
getToType, 286
mImpl, 287
setAxis, 286

setBlockShape, 286
setToType, 287
nvinfer1::IDimensionExpr, 288
~IDimensionExpr, 289
getConstantValue, 289
isConstant, 289
isSizeTensor, 289
mImpl, 289
nvinfer1::IDistCollectiveLayer, 290
~IDistCollectiveLayer, 290
mImpl, 291
nvinfer1::IDynamicQuantizeLayer, 291
~IDynamicQuantizeLayer, 293
getAxis, 293
getBlockShape, 293
getBlockSize, 293
getScaleType, 294
getToType, 294
mImpl, 296
setAxis, 294
setBlockShape, 294
setBlockSize, 295
setInput, 295
setScaleType, 296
setToType, 296
nvinfer1::IEinsumLayer, 297
~IEinsumLayer, 298
getEquation, 298
mImpl, 299
setEquation, 298
nvinfer1::IElementWiseLayer, 299
~IElementWiseLayer, 300
getOperation, 300
mImpl, 301
setOperation, 300
nvinfer1::IEngineInspector, 301
~IEngineInspector, 302
getEngineInformation, 303
getErrorRecorder, 303
getExecutionContext, 304
getLayerInformation, 304
mImpl, 306
setErrorRecorder, 305
setExecutionContext, 305
nvinfer1::IExecutionContext, 314
~IExecutionContext, 317
allInputDimensionsSpecified, 317
enqueueV3, 317
executeV2, 318
getDebugListener, 318
getDebugState, 318
getDebugSync, 320
getEngine, 320
getEnqueueEmitsProfile, 320
getErrorRecorder, 320
getInputConsumedEvent, 321
getMaxOutputSize, 321
getName, 322
getNvtxVerbosity, 322
getOptimizationProfile, 322
getOutputAllocator, 322
getOutputTensorAddress, 323
getPersistentCacheLimit, 323
getProfiler, 323
getRuntimeConfig, 324
getTemporaryStorageAllocator, 324
getTensorAddress, 324
getTensorShape, 325
getTensorStrides, 326
getUnfusedTensorsDebugState, 326
inferShapes, 326
mImpl, 340
reportToProfiler, 327
setAllTensorsDebugState, 327
setAuxStreams, 328
setCommunicator, 329
setDebugListener, 329
setDebugSync, 330
setDeviceMemory, 330
setDeviceMemoryV2, 330
setEnqueueEmitsProfile, 331
setErrorRecorder, 331
setInputConsumedEvent, 332
setInputShape, 332
setInputTensorAddress, 333
setName, 333
setNvtxVerbosity, 334
setOptimizationProfileAsync, 334
setOutputAllocator, 335
setOutputTensorAddress, 336
setPersistentCacheLimit, 336
setProfiler, 337
setTemporaryStorageAllocator, 337
setTensorAddress, 338
setTensorDebugState, 339
setUnfusedTensorsDebugState, 339
updateDeviceMemorySizeForShapes, 340
nvinfer1::IExprBuilder, 341
~IExprBuilder, 342
constant, 342
declareSizeTensor, 342
mImpl, 343
operation, 343
nvinfer1::IFillLayer, 344
~IFillLayer, 346
getAlpha, 346
getAlphaInt64, 346
getBeta, 346

getBetaInt64, 347
getDimensions, 347
getOperation, 347
getToType, 348
isAlphaBetaInt64, 348
mImpl, 353
setAlpha, 348
setAlphaInt64, 349
setBeta, 349
setBetaInt64, 350
setDimensions, 350
setInput, 351
setOperation, 352
setToType, 352
nvinfer1::IGatherLayer, 353
~IGatherLayer, 356
getGatherAxis, 356
getMode, 356
getNbElementWiseDims, 356
mImpl, 358
setGatherAxis, 357
setMode, 357
setNbElementWiseDims, 357
nvinfer1::IGridSampleLayer, 370
~IGridSampleLayer, 371
getAlignCorners, 371
getInterpolationMode, 371
getSampleMode, 371
mImpl, 373
setAlignCorners, 372
setInterpolationMode, 372
setSampleMode, 372
nvinfer1::IHostMemory, 373
~IHostMemory, 374
data, 374
mImpl, 375
size, 374
type, 375
nvinfer1::IIdentityLayer, 375
~IIdentityLayer, 377
mImpl, 377
nvinfer1::IIfConditional, 377
~IIfConditional, 378
addInput, 379
addOutput, 379
getName, 380
mImpl, 381
setCondition, 380
setName, 380
nvinfer1::IIfConditionalBoundaryLayer, 381
~IIfConditionalBoundaryLayer, 382
getConditional, 382
mBoundary, 382
nvinfer1::IIfConditionalInputLayer, 383
~IIfConditionalInputLayer, 383
mImpl, 384
nvinfer1::IIfConditionalOutputLayer, 384
~IIfConditionalOutputLayer, 385
mImpl, 385
nvinfer1::IIteratorLayer, 385
~IIteratorLayer, 386
getAxis, 386
getReverse, 387
mImpl, 387
setAxis, 387
setReverse, 387
nvinfer1::IKVCacheUpdateLayer, 388
~IKVCacheUpdateLayer, 389
getCacheMode, 389
getUpdateForm, 389
getUpdateLengths, 390
mImpl, 392
setCacheMode, 390
setInput, 391
setUpdateForm, 391
setUpdateLengths, 391
nvinfer1::ILayer, 393
~ILayer, 394
getInput, 394
getMetadata, 396
getName, 396
getNbInputs, 396
getNbOutputs, 396
getNbRanks, 397
getOutput, 397
getOutputType, 397
getType, 398
mLayer, 400
setInput, 398
setMetadata, 398
setName, 399
setNbRanks, 399
nvinfer1::ILoop, 405
~ILoop, 406
addIterator, 406
addLoopOutput, 406
addRecurrence, 406
addTripLimit, 406
getName, 407
mImpl, 408
setName, 407
nvinfer1::ILoopBoundaryLayer, 408
~ILoopBoundaryLayer, 409
getLoop, 409
mBoundary, 409
nvinfer1::ILoopOutputLayer, 410
~ILoopOutputLayer, 411
getAxis, 411

getLoopOutput, 411
mImpl, 412
setAxis, 411
setInput, 412
nvinfer1::ILRNLayer, 413
~ILRNLayer, 414
getAlpha, 414
getBeta, 414
getK, 414
getWindowSize, 415
mImpl, 416
setAlpha, 415
setBeta, 415
setK, 415
setWindowSize, 416
nvinfer1::IMatrixMultiplyLayer, 417
~IMatrixMultiplyLayer, 418
getOperation, 418
mImpl, 419
setOperation, 418
nvinfer1::IMoELayer, 419
~IMoELayer, 423
getActivationType, 423
getDynQOutputScaleType, 423
getQuantizationBlockShape, 424
getQuantizationToType, 424
getSwigluParamAlpha, 424
getSwigluParamBeta, 425
getSwigluParamLimit, 425
mImpl, 432
setActivationType, 425
setDynQOutputScaleType, 426
setGatedBiases, 426
setGatedWeights, 426
setInput, 427
setQuantizationBlockShape, 428
setQuantizationDynamicDblQ, 428
setQuantizationStatic, 429
setQuantizationToType, 430
setSwigluParamAlpha, 430
setSwigluParamBeta, 431
setSwigluParamLimit, 431
setSwigluParams, 432
nvinfer1::impl, 83
nvinfer1::impl::EnumMaxImpl< ActivationType >, 117
kVALUE, 117
nvinfer1::impl::EnumMaxImpl< AllocatorFlag >, 117
kVALUE, 118
nvinfer1::impl::EnumMaxImpl< APILanguage >, 118
kVALUE, 119
nvinfer1::impl::EnumMaxImpl< AttentionIOForm >, 119
kVALUE, 119
nvinfer1::impl::EnumMaxImpl< AttentionNormalizationOp >, 120
kVALUE, 120
nvinfer1::impl::EnumMaxImpl< BoundingBoxFormat >, 120
kVALUE, 121
nvinfer1::impl::EnumMaxImpl< BuilderFlag >, 121
kVALUE, 121
nvinfer1::impl::EnumMaxImpl< CausalMaskKind >, 122
kVALUE, 122
nvinfer1::impl::EnumMaxImpl< CollectiveOperation >, 122
kVALUE, 123
nvinfer1::impl::EnumMaxImpl< CumulativeOperation >, 123
kVALUE, 124
nvinfer1::impl::EnumMaxImpl< DataType >, 124
kVALUE, 124
nvinfer1::impl::EnumMaxImpl< DeviceType >, 125
kVALUE, 125
nvinfer1::impl::EnumMaxImpl< DimensionOperation >, 125
kVALUE, 126
nvinfer1::impl::EnumMaxImpl< ElementWiseOperation >, 126
kVALUE, 126
nvinfer1::impl::EnumMaxImpl< EngineCapability >, 127
kVALUE, 127
nvinfer1::impl::EnumMaxImpl< EngineStat >, 128
kVALUE, 128
nvinfer1::impl::EnumMaxImpl< ErrorCode >, 128
kVALUE, 129
nvinfer1::impl::EnumMaxImpl< ExecutionContextAllocationStrategy >, 129
kVALUE, 129
nvinfer1::impl::EnumMaxImpl< FillOperation >, 130
kVALUE, 130
nvinfer1::impl::EnumMaxImpl< GatherMode >, 130
kVALUE, 131
nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel >, 131
kVALUE, 132
nvinfer1::impl::EnumMaxImpl< ILogger::Severity >, 132
kVALUE, 132
nvinfer1::impl::EnumMaxImpl< InterpolationMode >, 133
kVALUE, 133
nvinfer1::impl::EnumMaxImpl< KVCacheMode >, 134
kVALUE, 134
nvinfer1::impl::EnumMaxImpl< LayerInformationFormat >, 134

kVALUE, 135
nvinfer1::impl::EnumMaxImpl< LayerType >, 135
kVALUE, 135
nvinfer1::impl::EnumMaxImpl< LoopOutput >, 136
kVALUE, 136
nvinfer1::impl::EnumMaxImpl< MatrixOperation >, 136
kVALUE, 137
nvinfer1::impl::EnumMaxImpl< MemoryPoolType >, 137
kVALUE, 138
nvinfer1::impl::EnumMaxImpl< MoEActType >, 138
kVALUE, 138
nvinfer1::impl::EnumMaxImpl< NetworkDefinitionCreationFlag >, 139
kVALUE, 139
nvinfer1::impl::EnumMaxImpl< OptProfileSelector >, 139
kVALUE, 140
nvinfer1::impl::EnumMaxImpl< PaddingMode >, 140
kVALUE, 140
nvinfer1::impl::EnumMaxImpl< PoolingType >, 141
kVALUE, 141
nvinfer1::impl::EnumMaxImpl< PreviewFeature >, 141
kVALUE, 142
nvinfer1::impl::EnumMaxImpl< ProfilingVerbosity >, 142
kVALUE, 143
nvinfer1::impl::EnumMaxImpl< ReduceOperation >, 143
kVALUE, 143
nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation >, 144
kVALUE, 144
nvinfer1::impl::EnumMaxImpl< ResizeRoundMode >, 144
kVALUE, 145
nvinfer1::impl::EnumMaxImpl< ResizeSelector >, 145
kVALUE, 145
nvinfer1::impl::EnumMaxImpl< RuntimePlatform >, 146
kVALUE, 146
nvinfer1::impl::EnumMaxImpl< SampleMode >, 146
kVALUE, 147
nvinfer1::impl::EnumMaxImpl< ScaleMode >, 147
kVALUE, 148
nvinfer1::impl::EnumMaxImpl< ScatterMode >, 148
kVALUE, 148
nvinfer1::impl::EnumMaxImpl< SerializationFlag >, 149
kVALUE, 149
nvinfer1::impl::EnumMaxImpl< T >, 116
nvinfer1::impl::EnumMaxImpl< TacticSource >, 150
kVALUE, 150
nvinfer1::impl::EnumMaxImpl< TempfileControlFlag >, 151
kVALUE, 151
nvinfer1::impl::EnumMaxImpl< TensorFormat >, 152
kVALUE, 152
nvinfer1::impl::EnumMaxImpl< TensorIOMode >, 152
kVALUE, 153
nvinfer1::impl::EnumMaxImpl< TensorLocation >, 153
kVALUE, 153
nvinfer1::impl::EnumMaxImpl< TilingOptimizationLevel >, 154
kVALUE, 154
nvinfer1::impl::EnumMaxImpl< TopKOperation >, 154
kVALUE, 155
nvinfer1::impl::EnumMaxImpl< TripLimit >, 155
kVALUE, 156
nvinfer1::impl::EnumMaxImpl< UnaryOperation >, 156
kVALUE, 156
nvinfer1::impl::EnumMaxImpl< WeightsRole >, 157
kVALUE, 157
nvinfer1::INetworkDefinition, 433
~INetworkDefinition, 438
addActivation, 438
addAssertion, 438
addAttention, 439
addAttentionV2, 440
addCast, 441
addConcatenation, 441
addConstant, 442
addConvolutionNd, 442
addCumulative, 443
addDeconvolutionNd, 444
addDequantize, 445
addDistCollective, 445
addDynamicQuantize, 446
addDynamicQuantizeV2, 447
addEinsum, 448
addElementWise, 448
addFill, 449
addGather, 450
addGatherV2, 450
addGridSample, 451
addIdentity, 452
addIfConditional, 452
addInput, 452
addKVCacheUpdate, 453
addLoop, 454
addLRN, 455
addMatrixMultiply, 456
addMoE, 456
addNMS, 457, 458
addNonZero, 458, 459
addNormalization, 459
addNormalizationV2, 461

addOneHot, 462
addPaddingNd, 462
addParametricReLU, 463
addPluginV2, 464
addPluginV3, 464
addPoolingNd, 465
addQuantize, 465
addRaggedSoftMax, 466
addReduce, 467
addResize, 468
addReverseSequence, 468
addRotaryEmbedding, 469
addScale, 469
addScaleNd, 470
addScatter, 471
addSelect, 472
addShape, 473
addShuffle, 473
addSlice, 474
addSoftMax, 474
addSqueeze, 475
addTopK, 475, 476
addUnary, 477
addUnsqueeze, 478
areWeightsMarkedRefittable, 478
getBuilder, 479
getErrorRecorder, 479
getFlag, 479
getFlags, 480
getInput, 480
getLayer, 481
getName, 481
getNbInputs, 481
getNbLayers, 482
getNbOutputs, 482
getOutput, 482
hasImplicitBatchDimension, 483
isDebugTensor, 483
markDebug, 483
markOutput, 484
markOutputForShapes, 484
markUnfusedTensorsAsDebugTensors, 485
markWeightsRefittable, 485
mImpl, 490
removeTensor, 486
setErrorRecorder, 486
setName, 486
setWeightsName, 487
unmarkDebug, 488
unmarkOutput, 488
unmarkOutputForShapes, 488
unmarkUnfusedTensorsAsDebugTensors, 489
unmarkWeightsRefittable, 489
nvinfer1::INMSLayer, 490
~INMSLayer, 492
getBoundingBoxFormat, 492
getIndicesType, 492
getTopKBoxLimit, 492
mImpl, 494
setBoundingBoxFormat, 493
setIndicesType, 493
setInput, 493
setTopKBoxLimit, 494
nvinfer1::INoCopy, 495
~INoCopy, 496
INoCopy, 496
operator=, 497
nvinfer1::INonZeroLayer, 498
~INonZeroLayer, 498
getIndicesType, 498
mImpl, 499
setIndicesType, 499
nvinfer1::INormalizationLayer, 500
~INormalizationLayer, 501
getAxes, 501
getEpsilon, 501
getNbGroups, 501
isV2, 502
mImpl, 503
setAxes, 502
setEpsilon, 502
setNbGroups, 503
nvinfer1::InterfaceInfo, 504
kind, 504
major, 504
minor, 504
nvinfer1::IOneHotLayer, 505
~IOneHotLayer, 506
getAxis, 506
mImpl, 506
setAxis, 506
nvinfer1::IOptimizationProfile, 513
~IOptimizationProfile, 515
getDimensions, 515
getExtraMemoryTarget, 515
getNbShapeValues, 515
getShapeValuesV2, 516
isValid, 516
mImpl, 520
setDimensions, 516
setExtraMemoryTarget, 518
setShapeValuesV2, 518
nvinfer1::IPaddingLayer, 523
~IPaddingLayer, 524
getPostPaddingNd, 524
getPrePaddingNd, 525
mImpl, 526
setPostPaddingNd, 525

setPrePaddingNd, 525
nvinfer1::IParametricReLULayer, 526
~IParametricReLULayer, 527
mImpl, 527
nvinfer1::IPluginRegistry, 562
~IPluginRegistry, 563
acquirePluginResource, 563
deregisterCreator, 564
deregisterLibrary, 564
getAllCreators, 565
getAllCreatorsRecursive, 565
getCreator, 566
getErrorRecorder, 566
isParentSearchEnabled, 566
loadLibrary, 567
PluginLibraryHandle, 563
registerCreator, 567
releasePluginResource, 567
setErrorRecorder, 568
setParentSearchEnabled, 568
nvinfer1::IPluginResourceContext, 572
~IPluginResourceContext, 573
getErrorRecorder, 574
getGpuAllocator, 574
IPluginResourceContext, 573, 574
operator=, 574, 575
nvinfer1::IPluginV2, 575
clone, 576
configureWithFormat, 577
destroy, 578
enqueue, 578
getNbOutputs, 579
getOutputDimensions, 579
getPluginNamespace, 580
getPluginType, 580
getPluginVersion, 581
getSerializationSize, 581
getTensorRTVersion, 581
getWorkspaceSize, 582
initialize, 582
serialize, 583
setPluginNamespace, 583
supportsFormat, 584
terminate, 584
nvinfer1::IPluginV2DynamicExt, 585
~IPluginV2DynamicExt, 587
clone, 587
configurePlugin, 587
enqueue, 588
getOutputDimensions, 589
getTensorRTVersion, 590
getWorkspaceSize, 590
kFORMAT_COMBINATION_LIMIT, 591
supportsFormatCombination, 590
nvinfer1::IPluginV2Ext, 592
~IPluginV2Ext, 593
attachToContext, 593
clone, 594
configurePlugin, 594
configureWithFormat, 596
detachFromContext, 596
getOutputDataType, 596
getTensorRTVersion, 597
IPluginV2Ext, 593
nvinfer1::IPluginV2IOExt, 598
configurePlugin, 599
getTensorRTVersion, 599
supportsFormatCombination, 599
nvinfer1::IPluginV2Layer, 601
~IPluginV2Layer, 602
getPlugin, 602
mImpl, 602
nvinfer1::IPluginV3Layer, 605
~IPluginV3Layer, 605
getPlugin, 606
mImpl, 606
nvinfer1::IPoolingLayer, 646
~IPoolingLayer, 648
getAverageCountExcludesPadding, 648
getBlendFactor, 648
getPaddingMode, 649
getPaddingNd, 649
getPoolingType, 649
getPostPadding, 650
getPrePadding, 650
getStrideNd, 650
getWindowSizeNd, 650
mImpl, 654
setAverageCountExcludesPadding, 651
setBlendFactor, 651
setPaddingMode, 651
setPaddingNd, 652
setPoolingType, 652
setPostPadding, 652
setPrePadding, 653
setStrideNd, 653
setWindowSizeNd, 653
nvinfer1::IQuantizeLayer, 660
~IQuantizeLayer, 662
getAxis, 662
getBlockShape, 662
getToType, 663
mImpl, 664
setAxis, 663
setBlockShape, 663
setToType, 663
nvinfer1::IRaggedSoftMaxLayer, 664
~IRaggedSoftMaxLayer, 665

mImpl, 665
nvinfer1::IRecurrenceLayer, 666
~IRecurrenceLayer, 666
mImpl, 667
setInput, 667
nvinfer1::IReduceLayer, 668
~IReduceLayer, 669
getKeepDimensions, 669
getOperation, 669
getReduceAxes, 669
mImpl, 671
setKeepDimensions, 670
setOperation, 670
setReduceAxes, 670
nvinfer1::IRefitter, 671
~IRefitter, 672
getAll, 673
getAllWeights, 673
getErrorRecorder, 674
getLogger, 674
getMaxThreads, 674
getMissing, 674
getMissingWeights, 675
getNamedWeights, 675
getWeightsLocation, 676
getWeightsPrototype, 676
getWeightsValidation, 677
mImpl, 681
refitCudaEngine, 677
refitCudaEngineAsync, 677
setErrorRecorder, 678
setMaxThreads, 678
setNamedWeights, 679
setWeights, 680
setWeightsValidation, 681
unsetNamedWeights, 681
nvinfer1::IResizeLayer, 682
~IResizeLayer, 684
getCoordinateTransformation, 684
getCubicCoeff, 684
getExcludeOutside, 685
getNearestRounding, 685
getOutputDimensions, 685
getResizeMode, 685
getScales, 686
getSelectorForSinglePixel, 686
mImpl, 690
setCoordinateTransformation, 686
setCubicCoeff, 687
setExcludeOutside, 687
setInput, 687
setNearestRounding, 688
setOutputDimensions, 688
setResizeMode, 689
setScales, 689
setSelectorForSinglePixel, 690
nvinfer1::IReverseSequenceLayer, 691
~IReverseSequenceLayer, 692
getBatchAxis, 692
getSequenceAxis, 692
mImpl, 693
setBatchAxis, 692
setSequenceAxis, 693
nvinfer1::IRotaryEmbeddingLayer, 694
~IRotaryEmbeddingLayer, 695
getInterleaved, 695
getRotaryEmbeddingDim, 695
mImpl, 696
setInput, 695
setInterleaved, 696
setRotaryEmbeddingDim, 696
nvinfer1::IRuntime, 697
~IRuntime, 698
deserializeCudaEngine, 698, 699
getDLACore, 699
getEngineHostCodeAllowed, 700
getErrorRecorder, 700
getLogger, 700
getMaxThreads, 700
getNbDLACores, 701
getPluginRegistry, 701
getTempfileControlFlags, 701
getTemporaryDirectory, 701
loadRuntime, 702
mImpl, 706
setDLACore, 702
setEngineHostCodeAllowed, 703
setErrorRecorder, 703
setGpuAllocator, 704
setMaxThreads, 704
setTempfileControlFlags, 704
setTemporaryDirectory, 705
nvinfer1::IRuntimeConfig, 706
~IRuntimeConfig, 707
getExecutionContextAllocationStrategy, 707
mImpl, 708
setExecutionContextAllocationStrategy, 707
nvinfer1::IScaleLayer, 740
~IScaleLayer, 742
getChannelAxis, 742
getMode, 742
getPower, 743
getScale, 743
getShift, 743
mImpl, 745
setChannelAxis, 743
setMode, 744
setPower, 744

setScale, 744
setShift, 745
nvinfer1::IScatterLayer, 745
~IScatterLayer, 747
getAxis, 747
getMode, 747
mImpl, 748
setAxis, 748
setMode, 748
nvinfer1::ISelectLayer, 749
~ISelectLayer, 749
mImpl, 750
nvinfer1::ISerializationConfig, 750
~ISerializationConfig, 751
clearFlag, 751
getFlag, 751
getFlags, 752
mImpl, 753
setFlag, 752
setFlags, 752
nvinfer1::IShapeLayer, 753
~IShapeLayer, 754
mImpl, 754
nvinfer1::IShuffleLayer, 755
~IShuffleLayer, 756
getFirstTranspose, 756
getReshapeDimensions, 756
getSecondTranspose, 757
getZeroIsPlaceholder, 757
mImpl, 760
setFirstTranspose, 757
setInput, 758
setReshapeDimensions, 758
setSecondTranspose, 759
setZeroIsPlaceholder, 759
nvinfer1::ISliceLayer, 760
~ISliceLayer, 762
getAxes, 763
getMode, 763
getSize, 763
getStart, 764
getStride, 764
mImpl, 767
setAxes, 764
setInput, 765
setMode, 765
setSize, 766
setStart, 766
setStride, 767
nvinfer1::ISoftMaxLayer, 767
~ISoftMaxLayer, 768
getAxes, 769
mImpl, 769
setAxes, 769
nvinfer1::ISqueezeLayer, 770
~ISqueezeLayer, 770
mImpl, 771
setInput, 771
nvinfer1::ITensor, 781
~ITensor, 782
getAllowedFormats, 783
getBroadcastAcrossBatch, 783
getDimensionName, 783
getDimensions, 784
getLocation, 784
getName, 784
getType, 785
isExecutionTensor, 785
isNetworkInput, 785
isNetworkOutput, 786
isShapeTensor, 786
mImpl, 791
setAllowedFormats, 786
setBroadcastAcrossBatch, 788
setDimensionName, 788
setDimensions, 790
setLocation, 790
setName, 791
nvinfer1::ITimingCache, 792
~ITimingCache, 793
combine, 793
mImpl, 796
query, 794
queryKeys, 794
reset, 794
serialize, 795
update, 795
nvinfer1::ITopKLayer, 796
~ITopKLayer, 797
getIndicesType, 798
getK, 798
getOperation, 798
getReduceAxes, 798
mImpl, 801
setIndicesType, 799
setInput, 799
setK, 800
setOperation, 800
setReduceAxes, 800
nvinfer1::ITripLimitLayer, 801
~ITripLimitLayer, 802
getTripLimit, 802
mImpl, 802
nvinfer1::IUnaryLayer, 818
~IUnaryLayer, 819
getOperation, 819
mImpl, 820
setOperation, 819

nvinfer1::IUnsqueezeLayer, 820
~IUnsqueezeLayer, 821
mImpl, 822
setInput, 821
nvinfer1::IVersionedInterface, 822
~IVersionedInterface, 824
getAPILanguage, 824
getInterfaceInfo, 824
IVersionedInterface, 824
operator=, 825
nvinfer1::Permutation, 825
order, 826
nvinfer1::plugin, 85
nvinfer1::plugin::GridAnchorParameters, 157
aspectRatios, 158
H, 158
maxSize, 158
minSize, 159
numAspectRatios, 159
variance, 159
W, 159
nvinfer1::plugin::PriorBoxParameters, 833
aspectRatios, 834
clip, 834
flip, 834
imgH, 834
imgW, 834
maxSize, 835
minSize, 835
numAspectRatios, 835
numMaxSize, 835
numMinSize, 835
offset, 835
stepH, 836
stepW, 836
variance, 836
nvinfer1::plugin::RegionParameters, 836
classes, 837
coords, 837
num, 837
smTree, 837
nvinfer1::plugin::RPROIParams, 838
anchorsRatioCount, 839
anchorsScaleCount, 839
featureStride, 839
iouThreshold, 839
minBoxSize, 839
nmsMaxOut, 839
poolingH, 840
poolingW, 840
preNmsTop, 840
spatialScale, 840
nvinfer1::plugin::softmaxTree, 841
child, 842
group, 842
groupOffset, 842
groups, 842
groupSize, 842
leaf, 843
n, 843
name, 843
parent, 843
nvinfer1::PluginField, 827
data, 828
length, 828
name, 829
PluginField, 828
type, 829
nvinfer1::PluginFieldCollection, 829
fields, 830
nbFields, 830
nvinfer1::PluginRegistrar< T >, 830
PluginRegistrar, 831
nvinfer1::PluginTensorDesc, 831
dims, 832
format, 832
scale, 832
type, 832
nvinfer1::safe, 85
nvinfer1::v_1_0, 85
nvinfer1::v_1_0::IDebugListener, 272
~IDebugListener, 273
getInterfaceInfo, 273
processDebugTensor, 273
nvinfer1::v_1_0::IErrorRecorder, 307
~IErrorRecorder, 308
clear, 309
decRefCount, 309
ErrorDesc, 308
getErrorCode, 309
getErrorDesc, 310
getInterfaceInfo, 311
getNbErrors, 311
hasOverflowed, 311
IErrorRecorder, 308
incRefCount, 312
kMAX_DESC_LENGTH, 313
RefCount, 308
reportError, 312
nvinfer1::v_1_0::IGpuAllocator, 359
~IGpuAllocator, 360
allocate, 360
allocateAsync, 361
deallocate, 362
deallocateAsync, 362
getInterfaceInfo, 363
IGpuAllocator, 360
reallocate, 363

nvinfer1::v_1_0::IGpuAsyncAllocator, 365
~IGpuAsyncAllocator, 366
allocate, 366
allocateAsync, 367
deallocate, 368
deallocateAsync, 368
getInterfaceInfo, 369
IGpuAsyncAllocator, 366
nvinfer1::v_1_0::ILogger, 400
~ILogger, 402
getInterfaceInfo, 402
ILogger, 401
kERROR, 401
kINFO, 401
kINTERNAL_ERROR, 401
kVERBOSE, 401
kWARNING, 401
log, 402
Severity, 401
nvinfer1::v_1_0::ILoggerFinder, 403
~ILoggerFinder, 404
findLogger, 404
getInterfaceInfo, 404
nvinfer1::v_1_0::IOutputAllocator, 520
getInterfaceInfo, 521
notifyShape, 521
reallocateOutput, 522
reallocateOutputAsync, 522
nvinfer1::v_1_0::IPluginCapability, 546
nvinfer1::v_1_0::IPluginCreator, 550
~IPluginCreator, 551
createPlugin, 551
deserializePlugin, 552
getFieldNames, 552
getInterfaceInfo, 553
getPluginName, 553
getPluginNamespace, 553
getPluginVersion, 554
IPluginCreator, 551
setPluginNamespace, 554
nvinfer1::v_1_0::IPluginCreatorInterface, 555
~IPluginCreatorInterface, 556
IPluginCreatorInterface, 556
operator=, 557
nvinfer1::v_1_0::IPluginCreatorV3One, 558
~IPluginCreatorV3One, 559
createPlugin, 559
getFieldNames, 560
getInterfaceInfo, 560
getPluginName, 560
getPluginNamespace, 561
getPluginVersion, 561
IPluginCreatorV3One, 558, 559
operator=, 561
nvinfer1::v_1_0::IPluginResource, 570
~IPluginResource, 570
clone, 571
getInterfaceInfo, 571
IPluginResource, 570, 571
operator=, 571, 572
release, 572
nvinfer1::v_1_0::IPluginV3, 603
clone, 603
getCapabilityInterface, 604
getInterfaceInfo, 604
nvinfer1::v_1_0::IPluginV3OneBuild, 607
configurePlugin, 608
getFormatCombinationLimit, 609
getInterfaceInfo, 609
getMetadataString, 609
getNbOutputs, 609
getNbTactics, 610
getOutputDataTypes, 610
getOutputShapes, 611
getTimingCacheID, 611
getValidTactics, 612
getWorkspaceSize, 613
kDEFAULT_FORMAT_COMBINATION_LIMIT, 614
supportsFormatCombination, 613
nvinfer1::v_1_0::IPluginV3OneCore, 617
getInterfaceInfo, 618
getPluginName, 618
getPluginNamespace, 618
getPluginVersion, 619
nvinfer1::v_1_0::IPluginV3OneRuntime, 620
attachToContext, 620
enqueue, 621
getFieldsToSerialize, 622
getInterfaceInfo, 622
onShapeChange, 622
setTactic, 623
nvinfer1::v_1_0::IProfiler, 655
~IProfiler, 655
getInterfaceInfo, 655
reportLayerTime, 655
nvinfer1::v_1_0::IProgressMonitor, 657
~IProgressMonitor, 657
getInterfaceInfo, 658
IProgressMonitor, 657
phaseFinish, 658
phaseStart, 658
stepComplete, 659
nvinfer1::v_1_0::IStreamReader, 772
~IStreamReader, 773
getInterfaceInfo, 773
IStreamReader, 773
operator=, 774

read, 774
nvinfer1::v_1_0::IStreamReaderV2, 775
~IStreamReaderV2, 776
getInterfaceInfo, 776
IStreamReaderV2, 776
operator=, 777
read, 777
seek, 777
nvinfer1::v_1_0::IStreamWriter, 779
~IStreamWriter, 779
getInterfaceInfo, 780
IStreamWriter, 779, 780
operator=, 780
write, 780
nvinfer1::v_1_0::TimingCacheKey, 847
data, 847
nvinfer1::v_1_0::TimingCacheValue, 847
kINVALID_TACTIC_HASH, 848
tacticHash, 848
timingMSec, 848
nvinfer1::v_2_0, 86
nvinfer1::v_2_0::IPluginV3OneBuild, 615
getAliasedInput, 615
getInterfaceInfo, 616
nvinfer1::Weights, 855
count, 855
type, 856
values, 856
nvinfer2, 87
nvinfer2::safe, 87
AsciiChar, 89
createTRTGraph, 93
DataType, 89
destroyTRTGraph, 95
Dims, 89
EnumMax, 96
ErrorCode, 89
ErrorDesc, 90
getSafePluginRegistry, 96
half_t, 90
IErrorRecorder, 90
InterfaceInfo, 90
IPluginCapability, 90
IPluginCreatorInterface, 90
IPluginResource, 90
IPluginV3, 91
IVersionedInterface, 91
kCPU, 92
kCPU_PINNED, 92
kDEBUG, 93
kDIV_ZERO, 93
kERROR, 93
kGATHER_OOB, 93
kGENERIC, 92
kGPU, 92
kIMMUTABLE, 92
kINF_CONSUMED, 93
kINFO, 93
kINTERNAL_ERROR, 93
kIOTENSOR, 92
kMANAGED, 92
kMAXIMUM_ALLOC_SIZE, 97
kNAN_CONSUMED, 93
kNONE, 92
kSCATTER_OOB, 93
kSCATTER_RACE, 93
kSCRATCH, 92
kVERBOSE, 93
kWARNING, 93
MemoryPlacement, 92
MemoryUsage, 92
operator!=, 96
operator==, 96
PluginField, 91
PluginFieldCollection, 91
RefCount, 91
RuntimeErrorType, 92
Severity, 93
TensorIOMode, 91
TensorRTPhase, 91
nvinfer2::safe::consistency, 97
createConsistencyChecker, 97
nvinfer2::safe::consistency::IConsistencyChecker, 228
~IConsistencyChecker, 229
IConsistencyChecker, 229, 230
operator=, 231
validate, 231
nvinfer2::safe::consistency::IPluginChecker, 547
~IPluginChecker, 547
IPluginChecker, 547, 548
operator=, 549
validate, 549
nvinfer2::safe::impl, 98
nvinfer2::safe::impl::EnumMaxImpl< Severity >, 149
kVALUE, 150
nvinfer2::safe::impl::EnumMaxImpl< T >, 116
nvinfer2::safe::IPluginV3OneSafeBuild, 623
~IPluginV3OneSafeBuild, 625
configurePlugin, 626
getFormatCombinationLimit, 627
getInterfaceInfo, 627
getMetadataString, 627
getNbOutputs, 628
getNbTactics, 628
getOutputDataTypes, 628
getOutputShapes, 629
getTimingCacheID, 630
getValidTactics, 630

getWorkspaceSize, 631
IPluginV3OneSafeBuild, 625, 626
kDEFAULT_FORMAT_COMBINATION_LIMIT, 633
operator=, 631
supportsFormatCombination, 632
nvinfer2::safe::IPluginV3OneSafeBuildMSS, 633
~IPluginV3OneSafeBuildMSS, 635
getInterfaceInfo, 635
getOutputShapes, 635
getSymbolicOutputShapes, 636
IPluginV3OneSafeBuildMSS, 634, 635
operator=, 637
nvinfer2::safe::IPluginV3OneSafeCore, 637
~IPluginV3OneSafeCore, 639
getInterfaceInfo, 639
getPluginName, 639
getPluginNamespace, 640
getPluginVersion, 640
IPluginV3OneSafeCore, 638, 639
operator=, 641
nvinfer2::safe::IPluginV3OneSafeRuntime, 641
~IPluginV3OneSafeRuntime, 643
clone, 643
enqueue, 643
getFieldsToSerialize, 644
getInterfaceInfo, 644
initResource, 645
IPluginV3OneSafeRuntime, 642, 643
operator=, 645
setTactic, 646
nvinfer2::safe::ISafeMemAllocator, 708
~ISafeMemAllocator, 709
allocate, 710
deallocate, 711
ISafeMemAllocator, 709, 710
operator=, 712
nvinfer2::safe::ISafePluginCreatorV3One, 713
~ISafePluginCreatorV3One, 714
createPlugin, 715
getFieldNames, 715
getInterfaceInfo, 716
getPluginName, 716
getPluginNamespace, 716
getPluginVersion, 717
getSafeRecorder, 717
ISafePluginCreatorV3One, 714
operator=, 717
setSafeRecorder, 718
nvinfer2::safe::ISafePluginRegistry, 718
~ISafePluginRegistry, 720
deregisterCreator, 720
getAllCreators, 721
getCreator, 721
getSafeRecorder, 722
ISafePluginRegistry, 719, 720
operator=, 723
registerCreator, 724
setSafeRecorder, 725
nvinfer2::safe::ISafePluginResourceContext, 726
~ISafePluginResourceContext, 727
getRuntimeErrorInformation, 728
getSafeMemAllocator, 729
getSafeRecorder, 729
ISafePluginResourceContext, 727, 728
operator=, 730
nvinfer2::safe::ISafeRecorder, 731
~ISafeRecorder, 734
getId, 734
getSeverity, 735
ISafeRecorder, 733, 734
kMAX_SAFE_DESC_LENGTH, 739
mId, 740
mSeverity, 740
operator=, 735, 736
reportDebug, 736
reportInfo, 737
reportVerbose, 737
reportWarn, 738
setSeverity, 739
nvinfer2::safe::ITRTGraph, 803
~ITRTGraph, 805
clone, 805
executeAsync, 806
getErrorBuffer, 806
getInputConsumedEvent, 807
getIOProfile, 807
getIOTensorAddress, 808
getIOTensorDescriptor, 809
getIOTensorName, 810
getNbAuxStreams, 810
getNbIOProfiles, 811
getNbIOTensors, 811
getSafeRecorder, 812
getScratchMemory, 812
getScratchMemorySize, 813
getTRTManagedScratch, 813
ITRTGraph, 804, 805
operator=, 814
setAuxStreams, 814
setInputConsumedEvent, 815
setIOProfile, 815
setIOTensorAddress, 816
setScratchMemory, 817
sync, 818
nvinfer2::safe::PhysicalDims, 826
d, 827
MAX_DIMS, 827

nbDims, 827
nvinfer2::safe::RuntimeErrorInformation, 841
bitMask, 841
nvinfer2::safe::TensorDescriptor, 843
bytesPerComponent, 844
componentsPerVector, 844
dataType, 845
ioMode, 845
memPlacement, 845
shape, 845
sizeInBytes, 845
stride, 846
strideOrder, 846
tensorName, 846
userShape, 846
vectorizedDim, 846
nvinfer2::safe::TypedArray, 849
∼TypedArray, 850
getBool, 852
getData, 852
getFloat, 852
getHalf, 853
getInt32, 853
getInt64, 853
getInt8, 853
getSize, 853
getType, 853
getUint8, 854
operator=, 854
TypedArray, 850–852
NvInferConsistency.h, 998, 999
NvInferForwardDecl.h, 1000, 1001
NV_INFER_INTERNAL_INCLUDE, 1001
NvInferImpl.h, 916
NvInferLegacyDims.h, 934, 935
NV_INFER_INTERNAL_INCLUDE, 934
NvInferPlugin.h, 936, 937
initLibNvInferPlugins, 937
NvInferPluginBase.h, 937, 939
NV_INFER_INTERNAL_INCLUDE, 939
NvInferPluginUtils.h, 941, 942
NvInferRuntime.h, 943, 949
getInferLibBuildVersion, 948
getInferLibMajorVersion, 948
getInferLibMinorVersion, 948
getInferLibPatchVersion, 948
getLogger, 948
getPluginRegistry, 949
NV_INFER_INTERNAL_INCLUDE, 947
REGISTER_TENSORRT_PLUGIN, 948
NvInferRuntimeBase.h, 972, 977
getInferLibVersion, 976
NV_TENSORRT_VERSION, 974
NV_TENSORRT_VERSION_INT, 974
TENSORRTAPI, 975
TRT_DEPRECATED, 975
TRT_DEPRECATED_API, 975
TRT_DEPRECATED_BECAUSE, 975
TRT_DEPRECATED_ENUM, 975
TRT_NODISCARD, 975
TRT_NODISCARD_BECAUSE, 976
TRTNOEXCEPT, 976
NvInferRuntimeCommon.h, 981, 982
NV_INFER_INTERNAL_INCLUDE, 981
NvInferRuntimePlugin.h, 983, 984
NV_INFER_INTERNAL_INCLUDE, 984
NvInferSafeMemAllocator.h, 1002, 1003
NvInferSafePlugin.h, 1004, 1005
NvInferSafeRecorder.h, 1009, 1010
NvInferSafeRuntime.h, 1012, 1013
NvInferVersion.h, 987, 991
NV_TENSORRT_BUILD, 988
NV_TENSORRT_LWS_MAJOR, 988
NV_TENSORRT_LWS_MINOR, 989
NV_TENSORRT_LWS_PATCH, 989
NV_TENSORRT_MAJOR, 989
NV_TENSORRT_MINOR, 989
NV_TENSORRT_PATCH, 989
NV_TENSORRT_RELEASE_TYPE, 989
NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS, 990
NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY, 990
NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE, 990
TRT_BUILD_ENTERPRISE, 990
TRT_MAJOR_ENTERPRISE, 990
TRT_MINOR_ENTERPRISE, 990
TRT_PATCH_ENTERPRISE, 991
NvOnnxConfig.h, 991, 992
nvonnxparser, 98
createONNXConfig, 101
EnumMax, 101
EnumMax< ErrorCode >, 101
EnumMax< OnnxParserFlag >, 102
ErrorCode, 100
kADJUST_FOR_DLA, 101
kENABLE_PLUGIN_OVERRIDE, 101
kENABLE_UINT8_AND_ASYMMETRIC_QUANTIZATION_DLA, 100
kINTERNAL_ERROR, 100
kINVALID_GRAPH, 100
kINVALID_NODE, 100
kINVALID_VALUE, 100
kMEM_ALLOC_FAILED, 100
kMODEL_DESERIALIZE_FAILED, 100
kNATIVE_INSTANCENORM, 100
kREFIT_FAILED, 100

kREPORT_CAPABILITY_DLA, 101
kSUCCESS, 100
kUNSUPPORTED_GRAPH, 100
kUNSUPPORTED_NODE, 100
kUNSUPPORTED_NODE_ATTR, 100
kUNSUPPORTED_NODE_DATATYPE, 100
kUNSUPPORTED_NODE_DYNAMIC, 100
kUNSUPPORTED_NODE_INPUT, 100
kUNSUPPORTED_NODE_SHAPE, 100
OnnxParserFlag, 100
OnnxParserFlags, 99
NvOnnxParser.h, 993, 996
createNvOnnxParser_INTERNAL, 995
createNvOnnxParserRefitter_INTERNAL, 995
getNvOnnxParserVersion, 995
NV_ONNX_PARSER_MAJOR, 995
NV_ONNX_PARSER_MINOR, 995
NV_ONNX_PARSER_PATCH, 995
nvonnxparser::anonymous_namespace{NvOnnxParser.h}, 102
createParser, 102
createParserRefitter, 103
nvonnxparser::IOnnxConfig, 507
~IOnnxConfig, 508
addVerbosity, 508
getFullTextFileName, 508
getModelDtype, 509
getModelFileName, 509
getPrintLayerInfo, 509
getTextFileName, 510
getVerbosityLevel, 510
reduceVerbosity, 510
setFullTextFileName, 511
setModelDtype, 511
setModelFileName, 511
setPrintLayerInfo, 512
setTextFileName, 512
setVerbosityLevel, 513
Verbosity, 508
nvonnxparser::IParser, 527
~IParser, 529
clearErrors, 529
clearFlag, 529
getError, 530
getFlag, 530
getFlags, 530
getLayerOutputTensor, 531
getNbErrors, 531
getNbSubgraphs, 531
getSubgraphNodes, 532
getUsedVCPluginLibraries, 532
isSubgraphSupported, 533
loadInitializer, 533
loadModelProto, 534
parse, 534
parseFromFile, 535
parseModelProto, 535
setBuilderConfig, 536
setFlag, 536
setFlags, 536
supportsModelV2, 537
supportsOperator, 537
nvonnxparser::IParserError, 538
~IParserError, 539
code, 539
desc, 539
file, 539
func, 539
line, 540
localFunctionStack, 540
localFunctionStackSize, 540
node, 540
nodeName, 540
nodeOperator, 541
nvonnxparser::IParserRefitter, 541
~IParserRefitter, 542
clearErrors, 542
getError, 542
getNbErrors, 542
loadInitializer, 543
loadModelProto, 544
refitFromBytes, 544
refitFromFile, 545
refitModelProto, 545

offset
nvinfer1::plugin::PriorBoxParameters, 835
OnnxParserFlag
nvonnxparser, 100
OnnxParserFlags
nvonnxparser, 99
onShapeChange
nvinfer1::v_1_0::IPluginV3OneRuntime, 622
operation
nvinfer1::IExprBuilder, 343
operator!=
nvinfer2::safe, 96
operator=
nvinfer1::INoCopy, 497
nvinfer1::IPluginResourceContext, 574, 575
nvinfer1::IVersionedInterface, 825
nvinfer1::v_1_0::IPluginCreatorInterface, 557
nvinfer1::v_1_0::IPluginCreatorV3One, 561
nvinfer1::v_1_0::IPluginResource, 571, 572
nvinfer1::v_1_0::IStreamReader, 774
nvinfer1::v_1_0::IStreamReaderV2, 777
nvinfer1::v_1_0::IStreamWriter, 780

nvinfer2::safe::consistency::IConsistencyChecker, 231
nvinfer2::safe::consistency::IPluginChecker, 549
nvinfer2::safe::IPluginV3OneSafeBuild, 631
nvinfer2::safe::IPluginV3OneSafeBuildMSS, 637
nvinfer2::safe::IPluginV3OneSafeCore, 641
nvinfer2::safe::IPluginV3OneSafeRuntime, 645
nvinfer2::safe::ISafeMemAllocator, 712
nvinfer2::safe::ISafePluginCreatorV3One, 717
nvinfer2::safe::ISafePluginRegistry, 723
nvinfer2::safe::ISafePluginResourceContext, 730
nvinfer2::safe::ISafeRecorder, 735, 736
nvinfer2::safe::ITRTGraph, 814
nvinfer2::safe::TypedArray, 854
operator==
nvinfer2::safe, 96
opt
nvinfer1::DynamicPluginTensorDesc, 115
OptProfileSelector
nvinfer1, 64
order
nvinfer1::Permutation, 826

PaddingMode
nvinfer1, 64
parent
nvinfer1::plugin::softmaxTree, 843
parse
nvonnxparser::IParser, 534
parseFromFile
nvonnxparser::IParser, 535
parseModelProto
nvonnxparser::IParser, 535
phaseFinish
nvinfer1::v_1_0::IProgressMonitor, 658
phaseStart
nvinfer1::v_1_0::IProgressMonitor, 658
PluginCapabilityType
nvinfer1, 67
PluginCreatorVersion
nvinfer1, 67
PluginField
nvinfer1::PluginField, 828
nvinfer2::safe, 91
PluginFieldCollection
nvinfer2::safe, 91
PluginFieldType
nvinfer1, 67
PluginFormat
nvinfer1, 44
PluginLibraryHandle
nvinfer1::IPluginRegistry, 563
PluginRegistrar
nvinfer1::PluginRegistrar< T >, 831
PluginVersion, 832
nvinfer1, 68
poolingH
nvinfer1::plugin::RPROIParams, 840
PoolingType
nvinfer1, 68
poolingW
nvinfer1::plugin::RPROIParams, 840
preNmsTop
nvinfer1::plugin::RPROIParams, 840
PreviewFeature
nvinfer1, 69
processDebugTensor
nvinfer1::v_1_0::IDebugListener, 273
ProfilingVerbosity
nvinfer1, 69

query
nvinfer1::ITimingCache, 794
queryKeys
nvinfer1::ITimingCache, 794

read
nvinfer1::v_1_0::IStreamReader, 774
nvinfer1::v_1_0::IStreamReaderV2, 777
reallocate
nvinfer1::v_1_0::IGpuAllocator, 363
reallocateOutput
nvinfer1::v_1_0::IOutputAllocator, 522
reallocateOutputAsync
nvinfer1::v_1_0::IOutputAllocator, 522
ReduceOperation
nvinfer1, 69
reduceVerbosity
nvonnxparser::IOnnxConfig, 510
RefCount
nvinfer1::v_1_0::IErrorRecorder, 308
nvinfer2::safe, 91
refitCudaEngine
nvinfer1::IRefitter, 677
refitCudaEngineAsync
nvinfer1::IRefitter, 677
refitFromBytes
nvonnxparser::IParserRefitter, 544
refitFromFile
nvonnxparser::IParserRefitter, 545
refitModelProto
nvonnxparser::IParserRefitter, 545
REGISTER_TENSORRT_PLUGIN
NvInferRuntime.h, 948
registerCreator
nvinfer1::IPluginRegistry, 567
nvinfer2::safe::ISafePluginRegistry, 724
release
nvinfer1::v_1_0::IPluginResource, 572

releasePluginResource
nvinfer1::IPluginRegistry, 567
removeTensor
nvinfer1::INetworkDefinition, 486
reportDebug
nvinfer2::safe::ISafeRecorder, 736
reportError
nvinfer1::v_1_0::IErrorRecorder, 312
reportInfo
nvinfer2::safe::ISafeRecorder, 737
reportLayerTime
nvinfer1::v_1_0::IProfiler, 655
reportToProfiler
nvinfer1::IExecutionContext, 327
reportVerbose
nvinfer2::safe::ISafeRecorder, 737
reportWarn
nvinfer2::safe::ISafeRecorder, 738
reset
nvinfer1::IBuilder, 196
nvinfer1::IBuilderConfig, 211
nvinfer1::ITimingCache, 794
resetDeviceType
nvinfer1::IBuilderConfig, 211
ResizeCoordinateTransformation
nvinfer1, 70
ResizeRoundMode
nvinfer1, 71
ResizeSelector
nvinfer1, 71
RuntimeErrorType
nvinfer2::safe, 92
RuntimePlatform
nvinfer1, 72

SampleMode
nvinfer1, 73
scale
nvinfer1::PluginTensorDesc, 832
ScaleMode
nvinfer1, 73
ScatterMode
nvinfer1, 73
seek
nvinfer1::v_1_0::IStreamReaderV2, 777
SeekPosition
nvinfer1, 74
SerializationFlag
nvinfer1, 74
SerializationFlags
nvinfer1, 44
serialize
nvinfer1::ICudaEngine, 265
nvinfer1::IPluginV2, 583
nvinfer1::ITimingCache, 795
serializeWithConfig
nvinfer1::ICudaEngine, 266
setActivationType
nvinfer1::IActivationLayer, 162
nvinfer1::IMoELayer, 425
setAlignCorners
nvinfer1::IGridSampleLayer, 372
setAllowedFormats
nvinfer1::ITensor, 786
setAllTensorsDebugState
nvinfer1::IExecutionContext, 327
setAlpha
nvinfer1::IActivationLayer, 162
nvinfer1::IFillLayer, 348
nvinfer1::ILRNLayer, 415
setAlphaInt64
nvinfer1::IFillLayer, 349
setAuxStreams
nvinfer1::IExecutionContext, 328
nvinfer2::safe::ITRTGraph, 814
setAverageCountExcludesPadding
nvinfer1::IPoolingLayer, 651
setAvgTimingIterations
nvinfer1::IBuilderConfig, 211
setAxes
nvinfer1::INormalizationLayer, 502
nvinfer1::ISliceLayer, 764
nvinfer1::ISoftMaxLayer, 769
setAxis
nvinfer1::IConcatenationLayer, 226
nvinfer1::IDequantizeLayer, 286
nvinfer1::IDynamicQuantizeLayer, 294
nvinfer1::IIteratorLayer, 387
nvinfer1::ILoopOutputLayer, 411
nvinfer1::IOneHotLayer, 506
nvinfer1::IQuantizeLayer, 663
nvinfer1::IScatterLayer, 748
setBatchAxis
nvinfer1::IReverseSequenceLayer, 692
setBeta
nvinfer1::IActivationLayer, 162
nvinfer1::IFillLayer, 349
nvinfer1::ILRNLayer, 415
setBetaInt64
nvinfer1::IFillLayer, 350
setBiasWeights
nvinfer1::IConvolutionLayer, 239
nvinfer1::IDeconvolutionLayer, 279
setBlendFactor
nvinfer1::IPoolingLayer, 651
setBlockShape
nvinfer1::IDequantizeLayer, 286
nvinfer1::IDynamicQuantizeLayer, 294

nvinfer1::IQuantizeLayer, 663
setBlockSize
nvinfer1::IDynamicQuantizeLayer, 295
setBoundingBoxFormat
nvinfer1::INMSLayer, 493
setBroadcastAcrossBatch
nvinfer1::ITensor, 788
setBuilderConfig
nvonnxparser::IParser, 536
setBuilderOptimizationLevel
nvinfer1::IBuilderConfig, 212
setCacheMode
nvinfer1::IKVCacheUpdateLayer, 390
setCausal
nvinfer1::IAttention, 174
setCausalKind
nvinfer1::IAttention, 175
setChannelAxis
nvinfer1::IScaleLayer, 743
setCommunicator
nvinfer1::IExecutionContext, 329
setCondition
nvinfer1::IIfConditional, 380
setCoordinateTransformation
nvinfer1::IResizeLayer, 686
setCubicCoeff
nvinfer1::IResizeLayer, 687
setDebugListener
nvinfer1::IExecutionContext, 329
setDebugSync
nvinfer1::IExecutionContext, 330
setDecomposable
nvinfer1::IAttention, 175
setDefaultDeviceType
nvinfer1::IBuilderConfig, 213
setDeviceMemory
nvinfer1::IExecutionContext, 330
setDeviceMemoryV2
nvinfer1::IExecutionContext, 330
setDeviceType
nvinfer1::IBuilderConfig, 213
setDilationNd
nvinfer1::IConvolutionLayer, 239
nvinfer1::IDeconvolutionLayer, 279
setDimensionName
nvinfer1::ITensor, 788
setDimensions
nvinfer1::IConstantLayer, 233
nvinfer1::IFillLayer, 350
nvinfer1::IOptimizationProfile, 516
nvinfer1::ITensor, 790
setDLACore
nvinfer1::IBuilderConfig, 213
nvinfer1::IRuntime, 702
setDynQOutputScaleType
nvinfer1::IMoELayer, 426
setEngineCapability
nvinfer1::IBuilderConfig, 214
setEngineHostCodeAllowed
nvinfer1::IRuntime, 703
setEnqueueEmitsProfile
nvinfer1::IExecutionContext, 331
setEpsilon
nvinfer1::INormalizationLayer, 502
setEquation
nvinfer1::IEinsumLayer, 298
setErrorRecorder
nvinfer1::IBuilder, 196
nvinfer1::ICudaEngine, 266
nvinfer1::IEngineInspector, 305
nvinfer1::IExecutionContext, 331
nvinfer1::INetworkDefinition, 486
nvinfer1::IPluginRegistry, 568
nvinfer1::IRefitter, 678
nvinfer1::IRuntime, 703
setExcludeOutside
nvinfer1::IResizeLayer, 687
setExclusive
nvinfer1::ICumulativeLayer, 270
setExecutionContext
nvinfer1::IEngineInspector, 305
setExecutionContextAllocationStrategy
nvinfer1::IRuntimeConfig, 707
setExtraMemoryTarget
nvinfer1::IOptimizationProfile, 518
setFirstTranspose
nvinfer1::IShuffleLayer, 757
setFlag
nvinfer1::IBuilderConfig, 214
nvinfer1::ISerializationConfig, 752
nvonnxparser::IParser, 536
setFlags
nvinfer1::IBuilderConfig, 214
nvinfer1::ISerializationConfig, 752
nvonnxparser::IParser, 536
setFullTextFileName
nvonnxparser::IOnnxConfig, 511
setGatedBiases
nvinfer1::IMoELayer, 426
setGatedWeights
nvinfer1::IMoELayer, 426
setGatherAxis
nvinfer1::IGatherLayer, 357
setGpuAllocator
nvinfer1::IBuilder, 196
nvinfer1::IRuntime, 704
setHardwareCompatibilityLevel
nvinfer1::IBuilderConfig, 216

setIndicesType
nvinfer1::INMSLayer, 493
nvinfer1::INonZeroLayer, 499
nvinfer1::ITopKLayer, 799
setInput
nvinfer1::IAttention, 176
nvinfer1::IAttentionInputLayer, 185
nvinfer1::IConvolutionLayer, 240
nvinfer1::IDeconvolutionLayer, 279
nvinfer1::IDynamicQuantizeLayer, 295
nvinfer1::IFillLayer, 351
nvinfer1::IKVCacheUpdateLayer, 391
nvinfer1::ILayer, 398
nvinfer1::ILoopOutputLayer, 412
nvinfer1::IMoELayer, 427
nvinfer1::INMSLayer, 493
nvinfer1::IRecurrenceLayer, 667
nvinfer1::IResizeLayer, 687
nvinfer1::IRotaryEmbeddingLayer, 695
nvinfer1::IShuffleLayer, 758
nvinfer1::ISliceLayer, 765
nvinfer1::ISqueezeLayer, 771
nvinfer1::ITopKLayer, 799
nvinfer1::IUnsqueezeLayer, 821
setInputConsumedEvent
nvinfer1::IExecutionContext, 332
nvinfer2::safe::ITRTGraph, 815
setInputShape
nvinfer1::IExecutionContext, 332
setInputTensorAddress
nvinfer1::IExecutionContext, 333
setInterleaved
nvinfer1::IRotaryEmbeddingLayer, 696
setInternalLibraryPath
nvinfer1, 81
setInterpolationMode
nvinfer1::IGridSampleLayer, 372
setIOProfile
nvinfer2::safe::ITRTGraph, 815
setIOTensorAddress
nvinfer2::safe::ITRTGraph, 816
setK
nvinfer1::ILRNLayer, 415
nvinfer1::ITopKLayer, 800
setKeepDimensions
nvinfer1::IReduceLayer, 670
setKernelSizeNd
nvinfer1::IConvolutionLayer, 240
nvinfer1::IDeconvolutionLayer, 280
setKernelWeights
nvinfer1::IConvolutionLayer, 241
nvinfer1::IDeconvolutionLayer, 280
setKeyValueForm
nvinfer1::IAttention, 176
setKeyValueLengths
nvinfer1::IAttention, 177
setL2LimitForTiling
nvinfer1::IBuilderConfig, 216
setLocation
nvinfer1::ITensor, 790
setMask
nvinfer1::IAttention, 178
setMaxAuxStreams
nvinfer1::IBuilderConfig, 217
setMaxNbTactics
nvinfer1::IBuilderConfig, 217
setMaxThreads
nvinfer1::IBuilder, 197
nvinfer1::IRefitter, 678
nvinfer1::IRuntime, 704
setMemoryPoolLimit
nvinfer1::IBuilderConfig, 218
setMessage
nvinfer1::IAssertionLayer, 165
setMetadata
nvinfer1::IAttention, 178
nvinfer1::ILayer, 398
setMode
nvinfer1::IGatherLayer, 357
nvinfer1::IScaleLayer, 744
nvinfer1::IScatterLayer, 748
nvinfer1::ISliceLayer, 765
setModelDtype
nvonnxparser::IOnnxConfig, 511
setModelFileName
nvonnxparser::IOnnxConfig, 511
setName
nvinfer1::IAttention, 179
nvinfer1::IExecutionContext, 333
nvinfer1::IIfConditional, 380
nvinfer1::ILayer, 399
nvinfer1::ILoop, 407
nvinfer1::INetworkDefinition, 486
nvinfer1::ITensor, 791
setNamedWeights
nvinfer1::IRefitter, 679
setNbElementWiseDims
nvinfer1::IGatherLayer, 357
setNbGroups
nvinfer1::IConvolutionLayer, 241
nvinfer1::IDeconvolutionLayer, 280
nvinfer1::INormalizationLayer, 503
setNbOutputMaps
nvinfer1::IConvolutionLayer, 241
nvinfer1::IDeconvolutionLayer, 281
setNbRanks
nvinfer1::IAttention, 179
nvinfer1::ILayer, 399

setNearestRounding
nvinfer1::IResizeLayer, 688
setNormalizationOperation
nvinfer1::IAttention, 180
setNormalizationQuantizeScale
nvinfer1::IAttention, 180
setNormalizationQuantizeToType
nvinfer1::IAttention, 181
setNvtxVerbosity
nvinfer1::IExecutionContext, 334
setOperation
nvinfer1::ICumulativeLayer, 271
nvinfer1::IElementWiseLayer, 300
nvinfer1::IFillLayer, 352
nvinfer1::IMatrixMultiplyLayer, 418
nvinfer1::IReduceLayer, 670
nvinfer1::ITopKLayer, 800
nvinfer1::IUnaryLayer, 819
setOptimizationProfileAsync
nvinfer1::IExecutionContext, 334
setOutputAllocator
nvinfer1::IExecutionContext, 335
setOutputDimensions
nvinfer1::IResizeLayer, 688
setOutputTensorAddress
nvinfer1::IExecutionContext, 336
setPaddingMode
nvinfer1::IConvolutionLayer, 242
nvinfer1::IDeconvolutionLayer, 281
nvinfer1::IPoolingLayer, 651
setPaddingNd
nvinfer1::IConvolutionLayer, 242
nvinfer1::IDeconvolutionLayer, 281
nvinfer1::IPoolingLayer, 652
setParentSearchEnabled
nvinfer1::IPluginRegistry, 568
setPersistentCacheLimit
nvinfer1::IExecutionContext, 336
setPluginNamespace
nvinfer1::IPluginV2, 583
nvinfer1::v_1_0::IPluginCreator, 554
setPluginsToSerialize
nvinfer1::IBuilderConfig, 218
setPoolingType
nvinfer1::IPoolingLayer, 652
setPostPadding
nvinfer1::IConvolutionLayer, 242
nvinfer1::IDeconvolutionLayer, 282
nvinfer1::IPoolingLayer, 652
setPostPaddingNd
nvinfer1::IPaddingLayer, 525
setPower
nvinfer1::IScaleLayer, 744
setPrePadding
nvinfer1::IConvolutionLayer, 243
nvinfer1::IDeconvolutionLayer, 282
nvinfer1::IPoolingLayer, 653
setPrePaddingNd
nvinfer1::IPaddingLayer, 525
setPreviewFeature
nvinfer1::IBuilderConfig, 219
setPrintLayerInfo
nvonnxparser::IOnnxConfig, 512
setProfiler
nvinfer1::IExecutionContext, 337
setProfileStream
nvinfer1::IBuilderConfig, 219
setProfilingVerbosity
nvinfer1::IBuilderConfig, 220
setProgressMonitor
nvinfer1::IBuilderConfig, 220
setQuantizationBlockShape
nvinfer1::IMoELayer, 428
setQuantizationDynamicDblQ
nvinfer1::IMoELayer, 428
setQuantizationStatic
nvinfer1::IMoELayer, 429
setQuantizationToType
nvinfer1::IMoELayer, 430
setQueryForm
nvinfer1::IAttention, 181
setQueryLengths
nvinfer1::IAttention, 182
setReduceAxes
nvinfer1::IReduceLayer, 670
nvinfer1::ITopKLayer, 800
setRemoteAutoTuningConfig
nvinfer1::IBuilderConfig, 220
setReshapeDimensions
nvinfer1::IShuffleLayer, 758
setResizeMode
nvinfer1::IResizeLayer, 689
setReverse
nvinfer1::ICumulativeLayer, 271
nvinfer1::IIteratorLayer, 387
setRotaryEmbeddingDim
nvinfer1::IRotaryEmbeddingLayer, 696
setRuntimePlatform
nvinfer1::IBuilderConfig, 221
setSafeRecorder
nvinfer2::safe::ISafePluginCreatorV3One, 718
nvinfer2::safe::ISafePluginRegistry, 725
setSampleMode
nvinfer1::IGridSampleLayer, 372
setScale
nvinfer1::IScaleLayer, 744
setScales
nvinfer1::IResizeLayer, 689

setScaleType
nvinfer1::IDynamicQuantizeLayer, 296
setScratchMemory
nvinfer2::safe::ITRTGraph, 817
setSecondTranspose
nvinfer1::IShuffleLayer, 759
setSelectorForSinglePixel
nvinfer1::IResizeLayer, 690
setSequenceAxis
nvinfer1::IReverseSequenceLayer, 693
setSeverity
nvinfer2::safe::ISafeRecorder, 739
setShapeValuesV2
nvinfer1::IOptimizationProfile, 518
setShift
nvinfer1::IScaleLayer, 745
setSize
nvinfer1::ISliceLayer, 766
setStart
nvinfer1::ISliceLayer, 766
setStride
nvinfer1::ISliceLayer, 767
setStrideNd
nvinfer1::IConvolutionLayer, 243
nvinfer1::IDeconvolutionLayer, 282
nvinfer1::IPoolingLayer, 653
setSwigluParamAlpha
nvinfer1::IMoELayer, 430
setSwigluParamBeta
nvinfer1::IMoELayer, 431
setSwigluParamLimit
nvinfer1::IMoELayer, 431
setSwigluParams
nvinfer1::IMoELayer, 432
setTactic
nvinfer1::v_1_0::IPluginV3OneRuntime, 623
nvinfer2::safe::IPluginV3OneSafeRuntime, 646
setTacticSources
nvinfer1::IBuilderConfig, 221
setTempfileControlFlags
nvinfer1::IRuntime, 704
setTemporaryDirectory
nvinfer1::IRuntime, 705
setTemporaryStorageAllocator
nvinfer1::IExecutionContext, 337
setTensorAddress
nvinfer1::IExecutionContext, 338
setTensorDebugState
nvinfer1::IExecutionContext, 339
setTextFileName
nvonnxparser::IOnnxConfig, 512
setTilingOptimizationLevel
nvinfer1::IBuilderConfig, 222
setTimingCache
nvinfer1::IBuilderConfig, 222
setTopKBoxLimit
nvinfer1::INMSLayer, 494
setToType
nvinfer1::ICastLayer, 224
nvinfer1::IDequantizeLayer, 287
nvinfer1::IDynamicQuantizeLayer, 296
nvinfer1::IFillLayer, 352
nvinfer1::IQuantizeLayer, 663
setUnfusedTensorsDebugState
nvinfer1::IExecutionContext, 339
setUpdateForm
nvinfer1::IKVCacheUpdateLayer, 391
setUpdateLengths
nvinfer1::IKVCacheUpdateLayer, 391
setVerbosityLevel
nvonnxparser::IOnnxConfig, 513
setWeights
nvinfer1::IConstantLayer, 234
nvinfer1::IRefitter, 680
setWeightsName
nvinfer1::INetworkDefinition, 487
setWeightStreamingBudgetV2
nvinfer1::ICudaEngine, 267
setWeightsValidation
nvinfer1::IRefitter, 681
setWindowSize
nvinfer1::ILRNLayer, 416
setWindowSizeNd
nvinfer1::IPoolingLayer, 653
setZeroIsPlaceholder
nvinfer1::IShuffleLayer, 759
Severity
nvinfer1::v_1_0::ILogger, 401
nvinfer2::safe, 93
shape
nvinfer2::safe::TensorDescriptor, 845
size
nvinfer1::IHostMemory, 374
sizeInBytes
nvinfer2::safe::TensorDescriptor, 845
smTree
nvinfer1::plugin::RegionParameters, 837
spatialScale
nvinfer1::plugin::RPROIParams, 840
stepComplete
nvinfer1::v_1_0::IProgressMonitor, 659
stepH
nvinfer1::plugin::PriorBoxParameters, 836
stepW
nvinfer1::plugin::PriorBoxParameters, 836
stride
nvinfer2::safe::TensorDescriptor, 846
strideOrder

nvinfer2::safe::TensorDescriptor, 846
supportsFormat
nvinfer1::IPluginV2, 584
supportsFormatCombination
nvinfer1::IPluginV2DynamicExt, 590
nvinfer1::IPluginV2IOExt, 599
nvinfer1::v_1_0::IPluginV3OneBuild, 613
nvinfer2::safe::IPluginV3OneSafeBuild, 632
supportsModelV2
nvonnxparser::IParser, 537
supportsOperator
nvonnxparser::IParser, 537
sync
nvinfer2::safe::ITRTGraph, 818

tacticHash
nvinfer1::v_1_0::TimingCacheValue, 848
TacticSource
nvinfer1, 75
TacticSources
nvinfer1, 44
TempfileControlFlag
nvinfer1, 75
TempfileControlFlags
nvinfer1, 44
TensorFormat
nvinfer1, 75
TensorFormats
nvinfer1, 45
TensorIOMode
nvinfer1, 77
nvinfer2::safe, 91
TensorLocation
nvinfer1, 78
tensorName
nvinfer2::safe::TensorDescriptor, 846
TENSORRTAPI
NvInferRuntimeBase.h, 975
TensorRTPhase
nvinfer1, 78
nvinfer2::safe, 91
terminate
nvinfer1::IPluginV2, 584
TilingOptimizationLevel
nvinfer1, 78
timingMSec
nvinfer1::v_1_0::TimingCacheValue, 848
TopKOperation
nvinfer1, 79
TripLimit
nvinfer1, 79
TRT_BUILD_ENTERPRISE
NvInferVersion.h, 990
TRT_DEPRECATED
NvInferRuntimeBase.h, 975
TRT_DEPRECATED_API
NvInferRuntimeBase.h, 975
TRT_DEPRECATED_BECAUSE
NvInferRuntimeBase.h, 975
TRT_DEPRECATED_ENUM
NvInferRuntimeBase.h, 975
TRT_MAJOR_ENTERPRISE
NvInferVersion.h, 990
TRT_MINOR_ENTERPRISE
NvInferVersion.h, 990
TRT_NODISCARD
NvInferRuntimeBase.h, 975
TRT_NODISCARD_BECAUSE
NvInferRuntimeBase.h, 976
TRT_PATCH_ENTERPRISE
NvInferVersion.h, 991
TRTNOEXCEPT
NvInferRuntimeBase.h, 976
type
nvinfer1::IHostMemory, 375
nvinfer1::PluginField, 829
nvinfer1::PluginTensorDesc, 832
nvinfer1::Weights, 856
TypedArray
nvinfer2::safe::TypedArray, 850–852

UnaryOperation
nvinfer1, 79
unmarkDebug
nvinfer1::INetworkDefinition, 488
unmarkOutput
nvinfer1::INetworkDefinition, 488
unmarkOutputForShapes
nvinfer1::INetworkDefinition, 488
unmarkUnfusedTensorsAsDebugTensors
nvinfer1::INetworkDefinition, 489
unmarkWeightsRefittable
nvinfer1::INetworkDefinition, 489
unsetNamedWeights
nvinfer1::IRefitter, 681
update
nvinfer1::ITimingCache, 795
updateDeviceMemorySizeForShapes
nvinfer1::IExecutionContext, 340
userShape
nvinfer2::safe::TensorDescriptor, 846

validate
nvinfer2::safe::consistency::IConsistencyChecker, 231
nvinfer2::safe::consistency::IPluginChecker, 549
Value, 854
values
nvinfer1::Weights, 856

variance
nvinfer1::plugin::GridAnchorParameters, 159
nvinfer1::plugin::PriorBoxParameters, 836
vectorizedDim
nvinfer2::safe::TensorDescriptor, 846
Verbosity
nvonnxparser::IOnnxConfig, 508

W
nvinfer1::plugin::GridAnchorParameters, 159
w
nvinfer1::DimsHW, 114
WeightsRole
nvinfer1, 80
write
nvinfer1::v_1_0::IStreamWriter, 780

NVIDIA TensorRT Standard Python API Documentation

Release 11.0.1

NVIDIA

Jun 23, 2026

OVERVIEW

1 Getting Started with TensorRT **1**
- 1.1 Installation . . . 1
- 1.2 Samples . . . 1
- 1.3 Operator Documentation . . . 1
- 1.4 Installing `cuda-python` . . . 1

2 Core Concepts **3**
- 2.1 TensorRT Workflow . . . 3
- 2.2 Classes Overview . . . 3
  - 2.2.1 Logger . . . 3
  - 2.2.2 Parsers . . . 3
  - 2.2.3 Network . . . 3
  - 2.2.4 Builder . . . 4
  - 2.2.5 Engine and Context . . . 4

3 Writing custom operators with TensorRT Python plugins **5**
- 3.1 Composition of a plugin . . . 5
- 3.2 Example: Circular padding plugin . . . 6
  - 3.2.1 Providing an Ahead-of-Time (AOT) implementation . . . 7
  - 3.2.2 Picking the most performant plugin configuration: Autotuning . . . 11
  - 3.2.3 Adding the plugin to a TensorRT network . . . 11
    - 3.2.3.1 Adding the plugin using TRT Python APIs . . . 11
    - 3.2.3.2 Loading an ONNX model with the custom operator . . . 12
- 3.3 Advanced usage . . . 12
  - 3.3.1 Example: Operators with data-dependent output shapes - Non-zero . . . 12
  - 3.3.2 Example: Implementing in-place custom ops with I/O aliasing . . . 13
  - 3.3.3 Example: Plugins with multiple backends: Using custom tactics . . . 13
- 3.4 FAQ . . . 14
- 3.5 Limitations . . . 15
- 3.6 Additional resources . . . 15

4 Foundational Types **17**
- 4.1 DataType . . . 17
- 4.2 Weights . . . 18
- 4.3 Dims . . . 19
  - 4.3.1 Volume . . . 19
  - 4.3.2 Dims . . . 20
  - 4.3.3 Dims2 . . . 20
  - 4.3.4 DimsHW . . . 20
  - 4.3.5 Dims3 . . . 20

4.3.6 Dims4 . . . 21
4.4 IHostMemory . . . 21

5 Core **23**
5.1 Logger . . . 23
5.2 Profiler . . . 24
5.3 IOptimizationProfile . . . 25
5.4 IBuilderConfig . . . 27
5.5 Builder . . . 35
5.5.1 NetworkDefinitionCreationFlag . . . 35
5.5.2 Builder . . . 36
5.6 ProgressMonitor . . . 39
5.7 Debug Listener . . . 40
5.7.1 IDebugListener . . . 40
5.8 ICudaEngine . . . 40
5.9 IExecutionContext . . . 46
5.10 Runtime . . . 52
5.11 IRuntimeConfig . . . 54
5.12 Refitter . . . 55
5.13 IErrorRecorder . . . 59
5.14 ITimingCache . . . 61
5.15 GPU Allocator . . . 62
5.15.1 AllocatorFlag . . . 62
5.15.2 IGpuAllocator . . . 63
5.16 EngineInspector . . . 65
5.17 ISerializationConfig . . . 65

6 Network **67**
6.1 INetworkDefinition . . . 67
6.2 Layer Base Classes . . . 88
6.2.1 ITensor . . . 88
6.2.2 ILayer . . . 91
6.3 Layers . . . 93
6.3.1 PaddingMode . . . 94
6.3.2 ICastLayer . . . 94
6.3.3 IConvolutionLayer . . . 94
6.3.4 IGridSampleLayer . . . 95
6.3.5 IActivationLayer . . . 96
6.3.6 IPoolingLayer . . . 97
6.3.7 ILRNLayer . . . 97
6.3.8 IScaleLayer . . . 98
6.3.9 ISoftMaxLayer . . . 98
6.3.10 IConcatenationLayer . . . 99
6.3.11 IDeconvolutionLayer . . . 99
6.3.12 IElementWiseLayer . . . 100
6.3.13 IGatherLayer . . . 101
6.3.14 IPluginV2Layer . . . 101
6.3.15 IPluginV3Layer . . . 101
6.3.16 IUnaryLayer . . . 101
6.3.17 IReduceLayer . . . 102
6.3.18 IPaddingLayer . . . 103
6.3.19 IParametricReLULayer . . . 103
6.3.20 ISelectLayer . . . 103
6.3.21 IShuffleLayer . . . 103

6.3.22 ISliceLayer . . . 104
6.3.23 IShapeLayer . . . 106
6.3.24 ITopKLayer . . . 106
6.3.25 IMatrixMultiplyLayer . . . 107
6.3.26 IRaggedSoftMaxLayer . . . 108
6.3.27 IIdentityLayer . . . 108
6.3.28 IConstantLayer . . . 108
6.3.29 IResizeLayer . . . 108
6.3.30 ILoop . . . 110
6.3.30.1 ILoopBoundaryLayer . . . 111
6.3.30.1.1 ITripLimitLayer . . . 111
6.3.30.1.2 IRecurrenceLayer . . . 111
6.3.30.1.3 IIteratorLayer . . . 112
6.3.30.1.4 ILoopOutputLayer . . . 112
6.3.31 IFillLayer . . . 113
6.3.32 IQuantizeLayer . . . 114
6.3.33 IDequantizeLayer . . . 114
6.3.34 IDynamicQuantizeLayer . . . 115
6.3.35 IScatterLayer . . . 116
6.3.36 IIfConditional . . . 116
6.3.37 IConditionLayer . . . 117
6.3.38 IIfConditionalOutputLayer . . . 117
6.3.39 IIfConditionalInputLayer . . . 117
6.3.40 IEinsumLayer . . . 117
6.3.41 IAssertionLayer . . . 118
6.3.42 IOneHotLayer . . . 118
6.3.43 INonZeroLayer . . . 118
6.3.44 INMSLayer . . . 119
6.3.45 IReverseSequenceLayer . . . 120
6.3.46 INormalizationLayer . . . 120
6.3.47 ISqueezeLayer . . . 121
6.3.48 IUnsqueezeLayer . . . 121
6.3.49 IDistCollectiveLayer . . . 122
6.3.50 ICumulativeLayer . . . 122
6.3.51 IAttention . . . 123
6.3.52 IRotaryEmbeddingLayer . . . 124
6.3.53 IKVCacheUpdateLayer . . . 125
6.3.54 IMoELayer . . . 125

7 Plugin **127**
7.1 IPluginCreator . . . 127
7.2 IPluginRegistry . . . 130
7.3 IPluginV2DynamicExt . . . 133
7.4 IPluginV3 . . . 137

8 tensorrt.plugin **147**
8.1 tensorrt.plugin.register . . . 147
8.2 tensorrt.plugin.impl . . . 148
8.3 tensorrt.plugin.aot_impl . . . 149
8.3.1 KernelLaunchParams . . . 151
8.4 tensorrt.plugin.autotune . . . 151
8.4.1 AutoTuneCombination . . . 153
8.5 Shape . . . 155
8.5.1 Shape . . . 155

8.5.2 ShapeExpr 156
8.6 Tensor 157
8.6.1 TensorDesc 157
8.6.2 Tensor 159
8.7 Functions 160

9 Onnx Parser **163**

Index **171**

CHAPTER

ONE

GETTING STARTED WITH TENSORRT

1.1 Installation

For installation instructions, please refer to https://docs.nvidia.com/deeplearning/tensorrt/latest/installing-tensorrt/installing.html

1.2 Samples

For information about samples, please refer to https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html#python_samples_section

1.3 Operator Documentation

For more information, including examples, refer to the TensorRT Operator's Reference documentation.

1.4 Installing `cuda-python`

Although not required by the TensorRT Python API, `cuda-python` is used in several samples. For installation instructions, refer to the CUDA Python Installation documentation.

CHAPTER

TWO

CORE CONCEPTS

2.1 TensorRT Workflow

The general TensorRT workflow consists of 3 steps:

1. Populate a *tensorrt.INetworkDefinition* either with a parser or by using the TensorRT Network API (see *tensorrt.INetworkDefinition* for more details). The *tensorrt.Builder* can be used to generate an empty *tensorrt.INetworkDefinition* .
2. Use the *tensorrt.Builder* to build a *tensorrt.ICudaEngine* using the populated *tensorrt.INetworkDefinition* .
3. Create a *tensorrt.IExecutionContext* from the *tensorrt.ICudaEngine* and use it to perform optimized inference.

2.2 Classes Overview

2.2.1 Logger

Most other TensorRT classes use a logger to report errors, warnings and informative messages. TensorRT provides a basic *tensorrt.Logger* implementation, but you can write your own implementation by deriving from *tensorrt.ILogger* for more advanced functionality.

2.2.2 Parsers

Parsers are used to populate a *tensorrt.INetworkDefinition* from a model trained in a Deep Learning framework.

2.2.3 Network

The *tensorrt.INetworkDefinition* represents a computational graph. In order to populate the network, TensorRT provides a suite of parsers for a variety of Deep Learning frameworks. It is also possible to populate the network manually using the Network API.

2.2.4 Builder

The *tensorrt.Builder* is used to build a *tensorrt.ICudaEngine* . In order to do so, it must be provided a populated *tensorrt.INetworkDefinition* .

2.2.5 Engine and Context

The *tensorrt.ICudaEngine* is the output of the TensorRT optimizer. It is used to generate a *tensorrt.IExecutionContext* that can perform inference.

CHAPTER

THREE

WRITING CUSTOM OPERATORS WITH TENSORRT PYTHON PLUGINS

TensorRT (TRT) offers a wide array of built-in operators that fit most use cases. However, you may wish to introduce a custom operator for a variety of reasons, including:

- Supporting an entirely new operator that TRT does not support out-of-the-box
- Adding an operator with a slightly different behavior to an operator that is already supported

TRT provides support for custom operators through plugins. This guide shows how Python functions that define the behavior of a plugin can be implemented and wrapped such that they can be added to a network as a custom operator.

3.1 Composition of a plugin

Primarily, a plugin definition needs two functions, wrapped with decorators provided by the `tensorrt.plugin` module:

1. *tensorrt.plugin.register()*: Returns shape and type characteristics of output tensors. The function signature also defines the input tensors and any attributes the plugin needs to function.
2. A function defining the plugin computation. This can be either (or both)

- *tensorrt.plugin.impl()*: A Just-in-Time (JIT) compute definition
- *tensorrt.plugin.aot_impl()*: An Ahead-of-Time (AOT) compute definition

It is possible to specify both an AOT and JIT implementation for the same plugin at the same time. However, based on user choice, TensorRT will only use one of these implementations for a given plugin or plugin layer in a single build phase.

Optionally, if a plugin is able to support multiple data type/tensor layout combinations for its I/O, or is able to support multiple backends (e.g. kernels), it can also take advantage of TensorRT's auototuning capabilities to let it pick the most performant configuration of the plugin on the target platform. To enable autotuning, simply define a function wrapped with *tensorrt.plugin.autotune()*.

Let us consider a few examples to understand the specifics of these functions.

3.2 Example: Circular padding plugin

Circular padding is useful for ops like circular convolution. The following image denotes how the original image (red) is circular padded once (green) and twice (blue):



The plugin shall have the following characteristics:

- Input: N-dimensional input
- Attribute(s): m-dimensional parameter `pads` where m is even and $m/2 \leq N$. `pads` denotes the amount of padding to apply before and after each of the $m/2$ last dimensions of the input tensor.
- Output: Padded tensor, which has the same data type as the input. The `i`-th to-last dimension of the output, d^{out}_{N-i-1} can be expressed in terms of the corresponding input dimension, d^{in}_{N-i-1}, as:

$$d^{\text{out}}_{N-i-1} = d^{\text{in}}_{N-i-1} + \text{pads}_{2i} + \text{pads}_{2i+1}$$

Let's capture this information in the `tensorrt.plugin.register` function:

```
import tensorrt.plugin as trtp
import numpy.typing as npt

@trtp.register("example::circ_pad_plugin")
def circ_pad_plugin_desc(
    inp0: trtp.TensorDesc, pads: npt.NDArray[np.int32]
) -> trtp.TensorDesc:
    ndim = inp0.ndim
    out_desc = inp0.like()

    for i in range(np.size(pads) // 2):
        out_desc.shape_expr[ndim - i - 1] += int(
            pads[i * 2] + pads[i * 2 + 1]
        )

    return out_desc
```

The argument `example::circ_pad_plugin` defines the namespace ("example") and name ("circ_pad_plugin") of the plugin. Input arguments annotated with *tensorrt.plugin.TensorDesc* denote the input tensors; all others are interpreted as plugin attributes. Supported attribute types are:

- `int`, `float`, `str`, `bool`, `bytes`. Lists/tuples of these types are not supported.
- 1-D Numpy arrays of the following types: `int8`, `int16`, `int32`, `int64`, `float16`, `float32`, `float64`, `bool`. These must be annotated with `numpy.typing.NDArray[dtype]`, where `dtype` is the expected Numpy data type.

The output signature is a `trt.plugin.TensorDesc` describing the output. To construct the output tensor descriptor, we start with *inp0.like()*, which returns a tensor descriptor with identical shape and type characteristics to `inp0`. Since the output shape is in fact different from the input, symbolic expressions for the output shape is accessed through *tensorrt.plugin.TensorDesc.shape_expr*, to which the expected shape is written.

Now let's define the computation function, decorated with *tensorrt.plugin.impl()*. For simplicity, let's leverage PyTorch's `torch.nn.functional.pad` to do the computation.

```
import tensorrt.plugin as trtp

@trtp.impl("example::circ_pad_plugin")
def circ_pad_plugin_impl(
    inp0: trtp.Tensor,
    pads: npt.NDArray[np.int32],
    outputs: Tuple[trtp.Tensor],
    stream: int
) -> None:
    inp_t = torch.as_tensor(inp0, device="cuda")
    out_t = torch.as_tensor(outputs[0], device="cuda")

    out = torch.nn.functional.pad(inp_t, pads.tolist(), mode="circular")
    out_t.copy_(out)
```

Note that the decorated function receives *tensorrt.plugin.Tensor*s for each input and output. In contrast to `TensorDesc`s, a `Tensor` references an underlying data buffer, directly accessible through *tensorrt.plugin.Tensor.data_ptr()*. When working with Torch and OpenAI Triton kernels, it is easier to use `torch.as_tensor()` to zero-copy construct a `torch.Tensor` corresponding to the `tensorrt.plugin.Tensor`.

3.2.1 Providing an Ahead-of-Time (AOT) implementation

In contrast to a JIT implementation provided by *tensorrt.plugin.impl()*, an AOT implementation provides the following benefits:

1. Building a TRT engine with a JIT computation for a plugin requires both its *tensorrt.plugin.register()* and *tensorrt.plugin.impl()* Python definitions to be present at runtime. With an AOT implementation, the plugin is fully embedded in the engine such that no user-provided definition of the plugin is required at runtime.
2. Python-independence of the plugin layer at runtime. This means that if the TRT engine only consists of AOT plugins, it can be executed on the standard TRT runtime as you would an engine with compiled C++ plugins, for example, through `trtexec`.

tensorrt.plugin.aot_impl() can be used to define the AOT implementation of a plugin. For the above circular padding plugin, let us define an AOT implementation using an OpenAI Triton kernel:

```
import triton
```

(continues on next page)

(continued from previous page)

```
import triton.language as tl

@triton.jit
def circ_pad_kernel(
    # input tensor
    X,
    # extra scalar args in between input and output tensors
    all_pads_0,
    all_pads_2,
    all_pads_4,
    all_pads_6,
    orig_dims_0,
    orig_dims_1,
    orig_dims_2,
    orig_dims_3,
    Y_shape_1,
    Y_shape_2,
    Y_shape_3,
    X_len,
    Y_len,
    # output tensor
    Y,
    BLOCK_SIZE: tl.constexpr,
):
    pid = tl.program_id(0)
    i = pid * BLOCK_SIZE + tl.arange(0, BLOCK_SIZE)

    mask_y = i < Y_len

    i3 = i % Y_shape_3
    i2 = (i // Y_shape_3) % Y_shape_2
    i1 = (i // Y_shape_3 // Y_shape_2) % Y_shape_1
    i0 = i // Y_shape_3 // Y_shape_2 // Y_shape_1

    j0 = (i0 - all_pads_0 + orig_dims_0) % orig_dims_0
    j1 = (i1 - all_pads_2 + orig_dims_1) % orig_dims_1
    j2 = (i2 - all_pads_4 + orig_dims_2) % orig_dims_2
    j3 = (i3 - all_pads_6 + orig_dims_3) % orig_dims_3

    load_idx = (
        orig_dims_3 * orig_dims_2 * orig_dims_1 * j0
        + orig_dims_3 * orig_dims_2 * j1
        + orig_dims_3 * j2
        + j3
    )
    mask_x = load_idx < X_len

    x = tl.load(X + load_idx, mask=mask_x)

    tl.store(Y + i, x, mask=mask_y)

@trtp.aot_impl("example::circ_pad_plugin")
```

(continues on next page)

(continued from previous page)

```
def circ_pad_plugin_aot_impl(
    inp0: trtp.TensorDesc, pads: npt.NDArray[np.int32], outputs: Tuple[trtp.TensorDesc],␣
↪tactic: int
) -> Tuple[Union[str, bytes], Union[str, bytes], trtp.KernelLaunchParams, trtp.SymExprs]:

    assert tactic == 0
    block_size = 256

    type_str = "fp32" if inp0.dtype == trt.float32 else "fp16"

    src = triton.compiler.ASTSource(
        fn=circ_pad_kernel,
        signature={
            "X": f"*{type_str}",
            "all_pads_0": "i32",
            "all_pads_2": "i32",
            "all_pads_4": "i32",
            "all_pads_6": "i32",
            "orig_dims_0": "i32",
            "orig_dims_1": "i32",
            "orig_dims_2": "i32",
            "orig_dims_3": "i32",
            "Y_shape_1": "i32",
            "Y_shape_2": "i32",
            "Y_shape_3": "i32",
            "X_len": "i32",
            "Y_len": "i32",
            "Y": f"*{type_str}",
        },
        constexprs={
            "BLOCK_SIZE": block_size,
        },
    )

    compiled_kernel = triton.compile(src)
    launch_params = trtp.KernelLaunchParams()

    N = inp0.ndim
    all_pads = np.zeros((N * 2,), dtype=np.int32)
    inp_dims = inp0.shape_expr
    out_dims = outputs[0].shape_expr

    for i in range(np.size(pads) // 2):
        all_pads[N * 2 - 2 * i - 2] = pads[i * 2]
        all_pads[N * 2 - 2 * i - 1] = pads[i * 2 + 1]

    # grid dims
    launch_params.grid_x = trtp.cdiv(out_dims.numel(), block_size)
    # block dims
    launch_params.block_x = compiled_kernel.metadata.num_warps * 32
    # shared memory
    launch_params.shared_mem = compiled_kernel.metadata.shared
```

(continues on next page)

(continued from previous page)

```
    all_pads = all_pads.tolist()

    # Representing all int32 scalar inputs as symbolic expressions.
    # These inputs are either constants or derivatives of input/output shapes.
    # The symbolic expressions are resolved after the full shape context becomes␣
↪available at runtime.
    # For the `circ_pad_kernel`, there are 13 such scalar extra-arguments, corresponding␣
↪to the 13 arguments
    # between the mandatory input and output tensors in the triton kernel's function␣
↪signature.
    extra_args = trtp.SymIntExprs.from_tuple(
        [
            trtp.SymInt32(e)
            for e in [
                all_pads[0],
                all_pads[2],
                all_pads[4],
                all_pads[6],
                inp_dims[0],
                inp_dims[1],
                inp_dims[2],
                inp_dims[3],
                out_dims[1],
                out_dims[2],
                out_dims[3],
                inp_dims.numel(),
                out_dims.numel(),
            ]
        ]
    )

    return compiled_kernel.metadata.name, compiled_kernel.asm["ptx"], launch_params,␣
↪extra_args
```

The function decorated with *tensorrt.plugin.aot_impl()* will be invoked for each I/O type/format combination and for each custom tactic advertised with *tensorrt.plugin.autotune()*. For each such combination, it should return the name of the kernel, the compiled form of the kernel, the kernel launch parameters, and any extra scalar arguments to pass to the kernel. Note that

- When dealing with dynamic shapes, concrete I/O dimensions may not be known when *tensorrt.plugin.aot_impl()* is invoked. Therefore, the kernel launch parameters and extra kernel arguments are specified symbolically.
- The kernel should always be written such that the input tensors come first, any scalar args (specified as above) come next, with the output tensors at the end.

3.2.2 Picking the most performant plugin configuration: Autotuning

Let's assume the plugin is able to support both FP32 and FP16 I/O, in linear tensor layouts (tensor formats). If performance is key, and it is uncertain whether FP32 or FP16 would perform better on the target platform, we can define a function decorated with *tensorrt.plugin.autotune()*.

```
@trtp.autotune("example::circ_pad_plugin")
def circ_pad_plugin_autotune(
    inp0: trtp.TensorDesc,
    pads: npt.NDArray[np.int32],
    outputs: Tuple[trtp.TensorDesc],
) -> List[trtp.AutoTuneCombination]:
    return [trtp.AutoTuneCombination("FP32|FP16, FP32|FP16", "LINEAR")]
```

The decorated function must return a list of *tensorrt.plugin.AutoTuneCombination*s. In this case, we define a single combination `AutoTuneCombination("FP32|FP16, FP32|FP16", "LINEAR")`; this indicates that the input and output must be either both FP32 or both FP16, and each have linear formats. When an autotune function is defined, during the engine build, TRT may execute the plugin (invoke its `impl` function) with random inputs, for each of the type/format combinations.

3.2.3 Adding the plugin to a TensorRT network

Now that we have defined the plugin, it can be added to a TensorRT network. Depending on your workflow, you may need to add the plugin directly to a TRT *tensorrt.INetworkDefinition* through the TRT Python API, or you may want the plugin to be added by the TRT ONNX parser when loading an ONNX graph containing the custom operator.

3.2.3.1 Adding the plugin using TRT Python APIs

The *tensorrt.INetworkDefinition.add_plugin()* API can be used to add a plugin to a network definition (an instance of *tensorrt.INetworkDefinition*):

```
input_tensor = network.add_input(name="x", dtype=trt.DataType.FLOAT, shape=x.shape)
plugin_layer = network.add_plugin(trt.plugin.op.example.circ_pad_plugin(input_tensor,␣
↪pads = pads), aot = False)
```

Note that

1. The registered plugin is findable in `tensorrt.plugin.op` under its registered namespace and name.
2. *tensorrt.INetworkDefinition.add_plugin()* takes a boolean argument `aot`, which can be used to choose either the AOT or JIT implementation of a plugin that has both definitions. If a plugin has only one implementation (either AOT or JIT), then omitting the `aot` argument will result in that implementation being picked by TensorRT.

When a plugin has both AOT and JIT implementations defined, it is also possible to apply a global setting to control which implementation is chosen. `tensorrt.NetworkDefinitionCreationFlag.PREFER_AOT_PYTHON_PLUGINS` and `tensorrt.NetworkDefinitionCreationFlag.PREFER_JIT_PYTHON_PLUGINS` are mutually exclusive flags that can be applied to the *tensorrt.INetworkDefinition* as follows:

```
builder = trt.Builder(trt.Logger(trt.Logger.INFO))
# Always choose AOT implementations wherever possible
network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.PREFER_AOT_
↪PYTHON_PLUGINS))
...
```

(continues on next page)

(continued from previous page)

```
# Now the `aot` argument can be omitted (and TRT will choose the AOT implementation)
plugin_layer = network.add_plugin(trt.plugin.op.example.circ_pad_plugin(input_tensor,␣
↪pads = pads))
```

3.2.3.2 Loading an ONNX model with the custom operator

It is possible to load an ONNX model with a custom op node which you need to run through the TRT plugin. To allow the TRT ONNX parser to correctly recognize your plugin as being mapped to the ONNX node(s) of interest, ensure that:

- The `op` property of the ONNX node is exactly the same as your plugin name.
- The node contains a string attribute called `plugin_namespace` with the namespace of your plugin.
- For a plugin with both AOT and JIT implementations, either
  - The node should contain a boolean attribute called `aot` to communicate which implementation should be chosen (`true` for AOT, `false` for JIT), or
  - `NetworkDefinitionCreationFlag.PREFER_AOT_PYTHON_PLUGINS()` or `NetworkDefinitionCreationFlag.PREFER_JIT_PYTHON_PLUGINS()` should be specified (see above for an example)

For example, if using ONNX Graphsurgeon, the custom op node can be constructed as follows:

```
import onnx_graphsurgeon as gs

var_x = gs.Variable(name="x", shape=inp_shape, dtype=np.float32)
var_y = gs.Variable(name="y", dtype=np.float32)

circ_pad_node = gs.Node(
    name="circ_pad_plugin",
    op="circ_pad_plugin",
    inputs=[var_x],
    outputs=[var_y],
    attrs={"pads": pads, "plugin_namespace": "example"},
)
```

3.3 Advanced usage

3.3.1 Example: Operators with data-dependent output shapes - Non-zero

Non-zero is an operation where the indices of the non-zero elements of the input tensor is found. Therefore, it has data-dependent output shapes (DDS), for which typical shape calculations cannot be done with input shapes.

To handle DDS, the extent of each data-dependent output dimension must be expressed in terms of a size tensor, which is a scalar that communicates to TRT an upper-bound and an autotune value for that dimension, in terms of the input shapes. The TRT engine build may be optimized for the autotune value, but the extent of that dimension may stretch up to the upper-bound at runtime.

In this example, we consider a 2D input tensor `inp0`; the output will be an $N \times 2$ tensor (a set of N 2D indices), where N is the number of non-zero indices. At maximum, all elements could be non-zero, and so the upper-bound could be expressed as `upper_bound = inp0.shape_expr[0] * inp0.shape_expr[1]`.

On average, we can expect half of the input to be filled with zero, so a size tensor can be constructed with that as the autotune value:

```
st = trt.plugin.size_tensor(opt = upper_bound // 2, upper_bound = upper_bound)
```

Now we're ready to construct the output shape. *st.expr()* returns a shape expression for the size tensor, so a tensor descriptor for the output shape can be constructed as *trt.plugin.from_shape_expr((st.expr(), 2), dtype=trt.int32)*. TRT requires that any size tensors also be made outputs of the plugin. Putting things together, we arrive at the following:

```
import tensorrt.plugin as trtp

@trtp.register("example::non_zero_plugin")
def non_zero_plugin_desc(
    inp0: trtp.TensorDesc,
) -> Tuple[trtp.TensorDesc, trtp.TensorDesc]:
    upper_bound = inp0.shape_expr[0] * inp0.shape_expr[1]
    st = trtp.size_tensor(upper_bound // 2, upper_bound)
    return trtp.from_shape_expr((st.expr(), 2), dtype=trt.int32), st
```

3.3.2 Example: Implementing in-place custom ops with I/O aliasing

In-place computations can be accomplished with TRT plugins via aliased I/O. i.e. An input that needs to be modified in-place can be represented by an input-output pair, where the output is aliased to the input. For example, consider a simple elementwise addition plugin. If an in-place addition is needed, that can be achieved as below:

```
import tensorrt.plugin as trtp

@trtp.register("sample::elemwise_add_plugin_")
def add_plugin_desc_(inp0: trtp.TensorDesc) -> trtp.TensorDesc:
    return inp0.aliased()
```

Note that *inp0.aliased()* produces an output `TensorDesc` that is aliased to `inp0`.

3.3.3 Example: Plugins with multiple backends: Using custom tactics

It may be the case that there are multiple kernels or libraries available to perform the plugin computation (for the same IO data type/tensor layout combination) but it is not predeterminable which backend would be the fastest on the target platform. Such alternate backends are termed tactics of the plugin. To let TRT figure out the fastest tactic, it is possible to leverage the autotune function.

Let's assume that we have an OpenAI Triton kernel `circ_pad_kernel` which can perform the circular padding operation in the above example. We can ask TRT to pick the fastest between the OpenAI Triton kernel and `torch.nn.functional.pad` as below:

```
import tensorrt.plugin as trtp
from enum import IntEnum

class Tactic(IntEnum):
    TORCH = 1
    TRITON = 2
```

(continues on next page)

(continued from previous page)

```
@trt.plugin.autotune("sample::circ_pad_plugin")
def circ_pad_plugin_autotune(inp0: trtp.TensorDesc, pads: npt.NDArray[np.int32],␣
↪outputs: Tuple[trtp.TensorDesc]) -> List[trtp.AutoTuneCombination]:
    c = trtp.AutoTuneCombination()
    c.pos([0, 1], "FP32|FP16")
    c.tactics([int(Tactic.TORCH), int(Tactic.TRITON)])
    return [c]
```

Note that we're using another way of constructing a `trt.plugin.AutoTuneCombination` here – namely, through *`pos(...)`* to populate the type/format information and *`tactics(...)`* to specify the tactics.

Now, the `impl` function can take an additional `int` argument called `tactic` to use the appropriate backend:

```
@trtp.impl("example::circ_pad_plugin")
def circ_pad_plugin_impl(
    inp0: trtp.Tensor,
    pads: npt.NDArray[np.int32],
    outputs: Tuple[trtp.Tensor],
    stream: int,
    tactic: int
) -> None:
    inp_t = torch.as_tensor(inp0, device="cuda")
    out_t = torch.as_tensor(outputs[0], device="cuda")

    if tactic == Tactic.TORCH:
        out = torch.nn.functional.pad(inp_t, pads.tolist(), mode="circular")
        out_t.copy_(out)
    elif tactic == Tactic.TRITON:
        N = inp0.ndim
        all_pads = np.zeros((N * 2,), dtype=np.int32)
        out_dims = trtp.Shape(tuple(inp0.shape))

        block_size = 256
        num_blocks = tuple(
            [int((np.prod(out_dims) + block_size - 1) // block_size)]
        )

        circ_pad_kernel[num_blocks](inp_t, ..., BLOCK_SIZE=block_size)
```

3.4 FAQ

1. What happened to plugin creators, plugin registration and plugin libraries?

You may have this question if you have previously dealt with class-based TRT plugins. For decorator based plugins, registration is automatically handled by the `tensorrt.plugin` module as long as the `tensorrt.plugin.register` function is defined.

To simulate the experience of plugin libraries, you may define plugins under a common namespace on a separate Python module, and then load the module when that plugin "library" needs to be loaded.

2. How is the serialization/deserialization of plugin attributes handled?

The `tensorrt.plugin` module automatically serializes any plugin attributes included in the `tensorrt.plugin.impl` function signature into the TRT engine. When that engine is loaded, those attributes are deserialized and passed back to the `tensorrt.plugin.impl` function.

Note: If only a subset of the plugin attributes included in `tensorrt.plugin.register` is required for the plugin computation, only include those in `tensorrt.plugin.impl` function signature. This should avoid the unnecessary serialization of data, and therefore produce leaner TRT engines.

3.5 Limitations

- TRT plugins defined in Python result in TRT engines that, when executed, rely on the availability of a Python environment with those plugin definitions. To build Python-independent TRT engines, it is recommended to use the TRT C++ plugin interfaces.
- This guide describes plugin implementations through the `tensorrt.plugin` module, which supports some of the most common use cases for plugin usage. For more advanced use cases, such as passing shape inputs, it is recommended to define plugins as direct implementations of the `tensorrt.IPluginV3` interface.

3.6 Additional resources

- Extending TensorRT with Custom Layers

CHAPTER

FOUR

FOUNDATIONAL TYPES

4.1 DataType

tensorrt.**DataType**

Represents data types.

ivar itemsize
int The size in bytes of this *DataType* .

Members:

FLOAT : 32-bit floating point format.

HALF : IEEE 16-bit floating-point format.

BF16 : Brain float – has an 8 bit exponent and 8 bit significand

INT8 : Signed 8-bit integer representing a quantized floating-point value.

INT32 : Signed 32-bit integer format.

INT64 : Signed 64-bit integer format.

BOOL : 8-bit boolean. 0 = false, 1 = true, other values undefined.

UINT8 :
Unsigned 8-bit integer format. Cannot be used to represent quantized floating-point values. Use the IdentityLayer to convert `uint8` network-level inputs to {`float32`, `float16`} prior to use with other TensorRT layers, or to convert intermediate output before `uint8` network-level outputs from {`float32`, `float16`} to `uint8`. `uint8` conversions are only supported for {`float32`, `float16`}. `uint8` to {`float32`, `float16`} conversion will convert the integer values to equivalent floating point values. {`float32`, `float16`} to `uint8` conversion will convert the floating point values to integer values by truncating towards zero. This conversion has undefined behavior for floating point values outside the range [0.0f, 256.0) after truncation. `uint8` conversions are not supported for {`int8`, `int32`, `bool`}.

FP8 :
Signed 8-bit floating point with 1 sign bit, 4 exponent bits, 3 mantissa bits, and exponent-bias 7.

INT4 : Signed 4-bit integer representing a quantized floating-point value.

FP4 : Signed 4-bit floating point with 1 sign bit, 2 exponent bits and 1 mantissa bits.

E8M0 : Unsigned 8-bit exponent-only floating point.

TensorRT also exposes some short-hand, NumPy-style `DataType` aliases that can be used across the library:

Type	Alias
tensorrt.DataType.FLOAT	tensorrt.float32
tensorrt.DataType.HALF	tensorrt.float16
tensorrt.DataType.INT32	tensorrt.int32
tensorrt.DataType.INT8	tensorrt.int8
tensorrt.DataType.BOOL	tensorrt.bool
tensorrt.DataType.UINT8	tensorrt.uint8
tensorrt.DataType.FP8	tensorrt.fp8
tensorrt.DataType.BF16	tensorrt.bfloat16
tensorrt.DataType.INT64	tensorrt.int64
tensorrt.DataType.INT4	tensorrt.int4
tensorrt.DataType.FP4	tensorrt.fp4
tensorrt.DataType.E8M0	tensorrt.e8m0

tensorrt.**nptype**(*trt_type*)

Returns the numpy-equivalent of a TensorRT *DataType* .

Parameters

trt_type – The TensorRT data type to convert.

Returns

The equivalent numpy type.

4.2 Weights

tensorrt.**WeightsRole**

How a layer uses particular Weights. The power weights of an IScaleLayer are omitted. Refitting those is not supported.

Members:

KERNEL : Kernel for *IConvolutionLayer* or *IDeconvolutionLayer* .

BIAS : Bias for *IConvolutionLayer* or *IDeconvolutionLayer* .

SHIFT : Shift part of *IScaleLayer* .

SCALE : Scale part of *IScaleLayer* .

CONSTANT : Weights for *IConstantLayer* .

ANY : Any other weights role.

class tensorrt.**Weights**(**args*, ***kwargs*)

An array of weights used as a layer parameter. The weights are held by reference until the engine has been built - deep copies are not made automatically.

Variables

- **dtype** – *DataType* The type of the weights.
- **size** – int The number of weights in the array.
- **nbytes** – int Total bytes consumed by the elements of the weights buffer.

Overloaded function.

1. __init__(self: tensorrt.tensorrt.Weights, type: tensorrt.tensorrt.DataType = <DataType.FLOAT: 0>) -> None

   Initializes an empty (0-length) Weights object with the specified type.

   type
   A type to initialize the weights with. Default: `tensorrt.float32`

2. __init__(self: tensorrt.tensorrt.Weights, type: tensorrt.tensorrt.DataType, ptr: typing.SupportsInt, count: typing.SupportsInt) -> None

   Initializes a Weights object with the specified data.

   type
   A type to initialize the weights with.

   ptr
   A pointer to the data.

   count
   The number of weights.

3. __init__(self: tensorrt.tensorrt.Weights, a: numpy.ndarray) -> None

   a
   A numpy array whose values to use. No deep copies are made.

numpy(*self:* tensorrt.tensorrt.Weights) → object

Create a numpy array using the underlying buffer of this weights object. The resulting array is just a view over the existing data, i.e. no deep copy is made.

If the weights cannot be converted to NumPy (e.g. due to unsupported data type), the original weights are returned.

Returns
The NumPy array or the original weights.

4.3 Dims

4.3.1 Volume

tensorrt.**volume**(*iterable*)

Computes the volume of an iterable.

Parameters
iterable – Any python iterable, including a *Dims* object.

Returns
The volume of the iterable. This will return 1 for empty iterables, as a scalar has an empty shape and the volume of a tensor with empty shape is 1.

4.3.2 Dims

class tensorrt.**Dims**(**args*, ***kwargs*)

Structure to define the dimensions of a tensor. *Dims* and all derived classes behave like Python `tuple` s. Furthermore, the TensorRT API can implicitly convert Python iterables to *Dims* objects, so `tuple` or `list` can be used in place of this class.

Overloaded function.

1. __init__(self: tensorrt.tensorrt.Dims) -> None
2. __init__(self: tensorrt.tensorrt.Dims, shape: collections.abc.Sequence[typing.SupportsInt]) -> None

4.3.3 Dims2

class tensorrt.**Dims2**(**args*, ***kwargs*)

Structure to define 2D shape.

Overloaded function.

1. __init__(self: tensorrt.tensorrt.Dims2) -> None
2. __init__(self: tensorrt.tensorrt.Dims2, dim0: typing.SupportsInt, dim1: typing.SupportsInt) -> None
3. __init__(self: tensorrt.tensorrt.Dims2, shape: collections.abc.Sequence[typing.SupportsInt]) -> None

4.3.4 DimsHW

class tensorrt.**DimsHW**(**args*, ***kwargs*)

Structure to define 2D shape with height and width.

Variables

- **h** – `int` The first dimension (height).
- **w** – `int` The second dimension (width).

Overloaded function.

1. __init__(self: tensorrt.tensorrt.DimsHW) -> None
2. __init__(self: tensorrt.tensorrt.DimsHW, h: typing.SupportsInt, w: typing.SupportsInt) -> None
3. __init__(self: tensorrt.tensorrt.DimsHW, shape: collections.abc.Sequence[typing.SupportsInt]) -> None

4.3.5 Dims3

class tensorrt.**Dims3**(**args*, ***kwargs*)

Structure to define 3D shape.

Overloaded function.

1. __init__(self: tensorrt.tensorrt.Dims3) -> None
2. __init__(self: tensorrt.tensorrt.Dims3, dim0: typing.SupportsInt, dim1: typing.SupportsInt, dim2: typing.SupportsInt) -> None
3. __init__(self: tensorrt.tensorrt.Dims3, shape: collections.abc.Sequence[typing.SupportsInt]) -> None

4.3.6 Dims4

class tensorrt.**Dims4**(**args*, ***kwargs*)

Structure to define 4D tensor.

Overloaded function.

1. __init__(self: tensorrt.tensorrt.Dims4) -> None
2. __init__(self: tensorrt.tensorrt.Dims4, dim0: typing.SupportsInt, dim1: typing.SupportsInt, dim2: typing.SupportsInt, dim3: typing.SupportsInt) -> None
3. __init__(self: tensorrt.tensorrt.Dims4, shape: collections.abc.Sequence[typing.SupportsInt]) -> None

4.4 IHostMemory

class tensorrt.**IHostMemory**

Handles library allocated memory that is accessible to the user.

The memory allocated via the host memory object is owned by the library and will be de-allocated when object is destroyed.

This class exposes a buffer interface using Python's buffer protocol.

Variables

- **dtype** – *DataType* The data type of this buffer.
- **nbytes** – int Total bytes consumed by the elements of the buffer.

__buffer__(*flags*, */*)

Return a buffer object that exposes the underlying memory of the object.

__del__(*self:* tensorrt.tensorrt.IHostMemory) → None

__exit__(*exc_type*, *exc_value*, *traceback*)

Context managers are deprecated and have no effect. Objects are automatically freed when the reference count reaches 0.

__init__(**args*, ***kwargs*)

__release_buffer__(*buffer*, */*)

Release the buffer object that exposes the underlying memory of the object.

CHAPTER

FIVE

CORE

5.1 Logger

class tensorrt.**ILogger**(*self:* tensorrt.tensorrt.ILogger)

Abstract base Logger class for the *Builder*, *ICudaEngine* and *Runtime* .

To implement a custom logger, ensure that you explicitly instantiate the base class in `__init__()` :

```
class MyLogger(trt.ILogger):
    def __init__(self):
        trt.ILogger.__init__(self)

    def log(self, severity, msg):
        ... # Your implementation here
```

Parameters
min_severity – The initial minimum severity of this Logger.

Variables
min_severity – `Logger.Severity` This minimum required severity of messages for the logger to log them.

The logger used to create an instance of IBuilder, IRuntime or IRefitter is used for all objects created through that interface. The logger should be valid until all objects created are released.

class Severity(*self:* tensorrt.tensorrt.ILogger.Severity, *value: SupportsInt*)

Indicates the severity of a message. The values in this enum are also accessible in the *ILogger* directly. For example, `tensorrt.ILogger.INFO` corresponds to `tensorrt.ILogger.Severity.INFO` .

Members:

INTERNAL_ERROR :
Represents an internal error. Execution is unrecoverable.

ERROR :
Represents an application error.

WARNING :
Represents an application error that TensorRT has recovered from or fallen back to a default.

INFO :
Represents informational messages.

VERBOSE :
Verbose messages with debugging information.

property name

log(*self:* tensorrt.tensorrt.ILogger, *severity:* tensorrt.tensorrt.ILogger.Severity, *msg: str*) → None

Logs a message to *stderr* . This function must be overriden by a derived class.

Parameters

- **severity** – The severity of the message.
- **msg** – The log message.

class tensorrt.**Logger**(*self: tensorrt.tensorrt.Logger*, *min_severity: tensorrt.tensorrt.ILogger.Severity* = *<Severity.WARNING: 2>*)

Logger for the *Builder*, *ICudaEngine* and *Runtime* .

Parameters
min_severity – The initial minimum severity of this Logger.

Variables
min_severity – `Logger.Severity` This minimum required severity of messages for the logger to log them.

log(*self:* tensorrt.tensorrt.Logger, *severity:* tensorrt.tensorrt.ILogger.Severity, *msg: str*) → None

Logs a message to *stdout*.

Parameters

- **severity** – The severity of the message.
- **msg** – The log message.

5.2 Profiler

class tensorrt.**IProfiler**(*self:* tensorrt.tensorrt.IProfiler)

Abstract base Profiler class.

To implement a custom profiler, ensure that you explicitly instantiate the base class in `__init__()` :

```
class MyProfiler(trt.IProfiler):
    def __init__(self):
        trt.IProfiler.__init__(self)

    def report_layer_time(self, layer_name, ms):
        ... # Your implementation here
```

When this class is added to an *IExecutionContext*, the profiler will be called once per layer for each invocation of *IExecutionContext.execute_v2()*.

It is not recommended to run inference with profiler enabled when the inference execution time is critical since the profiler may affect execution time negatively.

report_layer_time(*self:* tensorrt.tensorrt.IProfiler, *layer_name: str*, *ms: SupportsFloat*) → None

Reports time in milliseconds for each layer. This function must be overriden a derived class.

Parameters

- **layer_name** – The name of the layer, set when constructing the *INetworkDefinition* . If the engine is built with profiling verbosity set to NONE, the layerName is the decimal index of the layer.
- **ms** – The time in milliseconds to execute the layer.

class tensorrt.**Profiler**(*self:* tensorrt.tensorrt.Profiler)

When this class is added to an *IExecutionContext*, the profiler will be called once per layer for each invocation of *IExecutionContext.execute_v2()*.

It is not recommended to run inference with profiler enabled when the inference execution time is critical since the profiler may affect execution time negatively.

report_layer_time(*self:* tensorrt.tensorrt.IProfiler, *layer_name: str*, *ms: SupportsFloat*) → None

Prints time in milliseconds for each layer to stdout.

Parameters

- **layer_name** – The name of the layer, set when constructing the *INetworkDefinition* .
- **ms** – The time in milliseconds to execute the layer.

5.3 IOptimizationProfile

class tensorrt.**IOptimizationProfile**

Optimization profile for dynamic input dimensions and shape tensors.

When building an *ICudaEngine* from an *INetworkDefinition* that has dynamically resizable inputs (at least one input tensor has one or more of its dimensions specified as -1) or shape input tensors, users need to specify at least one optimization profile. Optimization profiles are numbered 0, 1, ...

The first optimization profile that has been defined (with index 0) will be used by the *ICudaEngine* whenever no optimization profile has been selected explicitly. If none of the inputs are dynamic, the default optimization profile will be generated automatically unless it is explicitly provided by the user (this is possible but not required in this case). If more than a single optimization profile is defined, users may set a target how much additional weight space should be maximally allocated to each additional profile (as a fraction of the maximum, unconstrained memory).

Users set optimum input tensor dimensions, as well as minimum and maximum input tensor dimensions. The builder selects the kernels that result in the lowest runtime for the optimum input tensor dimensions, and are valid for all input tensor sizes in the valid range between minimum and maximum dimensions. A runtime error will be raised if the input tensor dimensions fall outside the valid range for this profile. Likewise, users provide minimum, optimum, and maximum values for all shape tensor input values.

IOptimizationProfile implements `__nonzero__()` and `__bool__()` such that evaluating a profile as a `bool` (e.g. `if profile:`) will check whether the optimization profile can be passed to an IBuilderConfig object. This will perform partial validation, by e.g. checking that the maximum dimensions are at least as large as the optimum dimensions, and that the optimum dimensions are always as least as large as the minimum dimensions. Some validation steps require knowledge of the network definition and are deferred to engine build time.

Variables

extra_memory_target – Additional memory that the builder should aim to maximally allocate for this profile, as a fraction of the memory it would use if the user did not impose any constraints on memory. This unconstrained case is the default; it corresponds to `extra_memory_target == 1.0`. If `extra_memory_target == 0.0`, the builder aims to create the new optimization profile without allocating any additional weight memory. Valid inputs lie between 0.0 and 1.0. This parameter is only a hint, and TensorRT does not guarantee that the `extra_memory_target` will be reached. This parameter is ignored for the first (default) optimization profile that is defined.

get_shape(*self:* tensorrt.tensorrt.IOptimizationProfile, *input: str*) → list[*tensorrt.tensorrt.Dims*]

Get the minimum/optimum/maximum dimensions for a dynamic input tensor. If the dimensions have not been previously set via *set_shape()*, return an invalid *Dims* with a length of -1.

Returns
A `List[Dims]` of length 3, containing the minimum, optimum, and maximum shapes, in that order. If the shapes have not been set yet, an empty list is returned.

get_shape_input(*self:* tensorrt.tensorrt.IOptimizationProfile, *input: str*) → list[list[int]]

Get the minimum/optimum/maximum values for a shape input tensor.

Returns
A `List[List[int]]` of length 3, containing the minimum, optimum, and maximum values, in that order. If the values have not been set yet, an empty list is returned.

set_shape(*self:* tensorrt.tensorrt.IOptimizationProfile, *input: str*, *min:* tensorrt.tensorrt.Dims, *opt:* tensorrt.tensorrt.Dims, *max:* tensorrt.tensorrt.Dims) → None

Set the minimum/optimum/maximum dimensions for a dynamic input tensor.

This function must be called for any network input tensor that has dynamic dimensions. If `min`, `opt`, and `max` are the minimum, optimum, and maximum dimensions, and `real_shape` is the shape for this input tensor provided to the *INetworkDefinition* ,then the following conditions must hold:

(1) `len(min) == len(opt) == len(max) == len(real_shape)`

(2) `0 <= min[i] <= opt[i] <= max[i]` for all `i`

(3) if `real_shape[i] != -1`, then `min[i] == opt[i] == max[i] == real_shape[i]`

This function may (but need not be) called for an input tensor that does not have dynamic dimensions. In this case, all shapes must equal `real_shape`.

Parameters

- **input** – The name of the input tensor.
- **min** – The minimum dimensions for this input tensor.
- **opt** – The optimum dimensions for this input tensor.
- **max** – The maximum dimensions for this input tensor.

Raises
`ValueError` if an inconsistency was detected. Note that inputs can be validated only partially; a full validation is performed at engine build time.

set_shape_input(*self:* tensorrt.tensorrt.IOptimizationProfile, *input: str*, *min: collections.abc.Sequence[SupportsInt]*, *opt: collections.abc.Sequence[SupportsInt]*, *max: collections.abc.Sequence[SupportsInt]*) → None

Set the minimum/optimum/maximum values for a shape input tensor.

This function must be called for every input tensor `t` that is a shape tensor (`t.is_shape == True`). This implies that the datatype of `t` is `int64` or `int32`, the rank is either 0 or 1, and the dimensions of `t` are fixed at network definition time. This function must NOT be called for any input tensor that is not a shape tensor.

If `min`, `opt`, and `max` are the minimum, optimum, and maximum values, it must be true that `min[i] <= opt[i] <= max[i]` for all `i`.

Parameters

- **input** – The name of the input tensor.

- **min** – The minimum values for this shape tensor.
- **opt** – The optimum values for this shape tensor.
- **max** – The maximum values for this shape tensor.

Raises
`ValueError` if an inconsistency was detected. Note that inputs can be validated only partially; a full validation is performed at engine build time.

5.4 IBuilderConfig

`tensorrt.`**DeviceType**

Device types that TensorRT can execute on

Members:

GPU : GPU device

DLA : DLA core

`tensorrt.`**ProfilingVerbosity**

Profiling verbosity in NVTX annotations and the engine inspector

Members:

LAYER_NAMES_ONLY : Print only the layer names. This is the default setting.

DETAILED : Print detailed layer information including layer names and layer parameters.

NONE : Do not print any layer information.

`tensorrt.`**TacticSource**

Tactic sources that can provide tactics for TensorRT.

Members:

EDGE_MASK_CONVOLUTIONS :
Enables convolution tactics implemented with edge mask tables. These tactics tradeoff memory for performance by consuming additional memory space proportional to the input size. Enabled by default.

JIT_CONVOLUTIONS :
Enables convolution tactics implemented with source-code JIT fusion. The engine building time may increase when this is enabled. Enabled by default.

`tensorrt.`**EngineCapability**

List of supported engine capability flows.
The EngineCapability determines the restrictions of a network during build time and what runtime it targets. EngineCapability.STANDARD does not provide any restrictions on functionality and the resulting serialized engine can be executed with TensorRT's standard runtime APIs in the nvinfer1 namespace. EngineCapability.SAFETY provides a restricted subset of network operations that are safety certified and the resulting serialized engine can be executed with TensorRT's safe runtime APIs in the *nvinfer2::safe* namespace. EngineCapability.DLA_STANDALONE provides a restricted subset of network operations that are DLA compatible and the resulting serialized engine can be executed using standalone DLA runtime APIs. See sampleCudla for an example of integrating cuDLA APIs with TensorRT APIs.

Members:

STANDARD : Standard: TensorRT flow without targeting the standard runtime. This flow supports both DeviceType::kGPU and DeviceType::kDLA.

SAFETY : Safety: TensorRT flow with restrictions targeting the safety runtime. See safety documentation for list of supported layers and formats. This flow supports only DeviceType::kGPU.

DLA_STANDALONE : DLA Standalone: TensorRT flow with restrictions targeting external, to TensorRT, DLA runtimes. See DLA documentation for list of supported layers and formats. This flow supports only DeviceType::kDLA.

tensorrt.**BuilderFlag**

Valid modes that the builder can enable when creating an engine from a network definition.

Members:

DEBUG : Enable debugging of layers via synchronizing after every layer

GPU_FALLBACK : Enable layers marked to execute on GPU if layer cannot execute on DLA

REFIT : Enable building a refittable engine

DISABLE_TIMING_CACHE : Disable reuse of timing information across identical layers.

EDITABLE_TIMING_CACHE : Enable the editable timing cache.

TF32 : Allow (but not require) computations on tensors of type DataType.FLOAT to use TF32. TF32 computes inner products by rounding the inputs to 10-bit mantissas before multiplying, but accumulates the sum using 23-bit mantissas. Enabled by default.

SPARSE_WEIGHTS : Allow the builder to examine weights and use optimized functions when weights have suitable sparsity.

SAFETY_SCOPE
: [Change the allowed parameters in the EngineCapability.STANDARD flow to match the restrictions that EngineCapability.SAFETY check against for DeviceType.GPU and EngineCapability.DLA_STANDALONE check against the DeviceType.DLA case. This flag is forced to true if EngineCapability.SAFETY at build time if it is unset.] [DEPRECATED] Deprecated in TensorRT 10.16. In EngineCapability.STANDARD flow, safety restrictions are no longer supported. In EngineCapability.SAFETY and EngineCapability.DLA_STANDALONE flows, restrictions are enforced natively. This flag is retained for API compatibility but is ignored.

DIRECT_IO
: [Require that no reformats be inserted between a layer and a network I/O tensor for which `ITensor.allowed_formats` was set. Build fails if a reformat is required for functional correctness.] [DEPRECATED] Deprecated in TensorRT 10.7.)

VERSION_COMPATIBLE : Restrict to lean runtime operators to provide version forward compatibility for the plan files.

EXCLUDE_LEAN_RUNTIME : Exclude lean runtime from the plan.

ERROR_ON_TIMING_CACHE_MISS : Emit error when a tactic being timed is not present in the timing cache.

DISABLE_COMPILATION_CACHE : Disable caching JIT compilation results during engine build.

STRIP_PLAN : Strip the refittable weights from the engine plan file.

REFIT_IDENTICAL : Create a refittable engine using identical weights. Different weights during refits yield unpredictable behavior.

WEIGHT_STREAMING : Enable building with the ability to stream varying amounts of weights during Runtime. This decreases GPU memory of TRT at the expense of performance.

REFIT_INDIVIDUAL : Create a refittable engine and allows the users to specify which weights are refittable and which are not.

STRICT_NANS : Disable floating-point optimizations: 0*x => 0, x-x => 0, or x/x => 1. These identities are not true when x is a NaN or Inf, and thus might hide propagation or generation of NaNs.

MONITOR_MEMORY : Enable memory monitor during build time.

DISTRIBUTIVE_INDEPENDENCE : Enable distributive independence.

tensorrt.**PreviewFeature**

List of Preview Features that can be enabled. Preview Features have been fully tested but are not yet as stable as other features in TensorRT.
They are provided as opt-in features for at least one release.

Members:

ALIASED_PLUGIN_IO_10_03 :
Allows plugin I/O to be aliased when using IPluginV3OneBuildV2.

RUNTIME_ACTIVATION_RESIZE_10_10 :
Allow update_device_memory_size_for_shapes to resize runner internal activation memory by changing the allocation algorithm. Using this feature can reduce runtime memory requirement when the actual input tensor shapes are smaller than the maximum input tensor dimensions.

tensorrt.**MemoryPoolType**

The type for memory pools used by TensorRT.

Members:

WORKSPACE :
WORKSPACE is used by TensorRT to store intermediate buffers within an operation. This defaults to max device memory. Set to a smaller value to restrict tactics that use over the threshold en masse.

DLA_MANAGED_SRAM :
DLA_MANAGED_SRAM is a fast software managed RAM used by DLA to communicate within a layer. The size of this pool must be at least 4 KiB and must be a power of 2. This defaults to 1 MiB. Orin has capacity of 1 MiB per core.

DLA_LOCAL_DRAM :
DLA_LOCAL_DRAM is host RAM used by DLA to share intermediate tensor data across operations. The size of this pool must be at least 4 KiB and must be a power of 2. This defaults to 1 GiB.

DLA_GLOBAL_DRAM :
DLA_GLOBAL_DRAM is host RAM used by DLA to store weights and metadata for execution. The size of this pool must be at least 4 KiB and must be a power of 2. This defaults to 512 MiB.

TACTIC_DRAM :
TACTIC_DRAM is the host DRAM used by the optimizer to run tactics. On embedded devices, where host and device memory are unified, this includes all device memory required by TensorRT to build the network up to the point of each memory allocation. This defaults to 75% of totalGlobalMem as reported by cudaGetDeviceProperties when cudaGetDeviceProperties.embedded is true, and 100% otherwise.

TACTIC_SHARED_MEMORY :
TACTIC_SHARED_MEMORY defines the maximum shared memory size utilized for driver reserved and executing the backend CUDA kernel implementation. Adjust this value to restrict tactics that exceed the specified threshold en masse. The default value is device max capability. This value must be less than 1GiB.

Updating this flag will override the shared memory limit set by ref HardwareCompatibilityLevel, which defaults to 48KiB.

tensorrt.**HardwareCompatibilityLevel**

Describes requirements of compatibility with GPU architectures other than that of the GPU on which the engine was built.

Note that compatibility with future hardware depends on CUDA forward compatibility support.

Members:

NONE :

Do not require hardware compatibility with GPU architectures other than that of the GPU on which the engine was built.

AMPERE_PLUS :

Require that the engine is compatible with Ampere and newer GPUs. This will limit the combined usage of driver reserved and backend kernel max shared memory to 48KiB, may reduce the number of available tactics for each layer, and may prevent some fusions from occurring. Thus this can decrease the performance, especially for tf32 models. This option will disable cuDNN, cuBLAS, and cuBLASLt as tactic sources. This option is only supported for engines built on NVIDIA Ampere and later GPUs.

SAME_COMPUTE_CAPABILITY :

Require that the engine is compatible with GPUs that have the same Compute Capability version as the one it was built on. This may decrease the performance compared to an engine with no compatibility. This option will disable cuDNN, cuBLAS, and cuBLASLt as tactic sources. This option is only supported for engines built on NVIDIA Turing and later GPUs.

tensorrt.**TilingOptimizationLevel**

Describes the optimization level of tiling strategies. A higher level allows TensorRT to spend more time searching for better tiling strategy.

Members:

NONE :

Do not apply any tiling strategy.

FAST :

Use a fast algorithm and heuristic based strategy. Slightly increases engine build time.

MODERATE :

Increase search space and use a mixed heuristic/profiling strategy. Moderately increases engine build time.

FULL :

Increase search space even wider. Significantly increases engine build time.

class tensorrt.**IBuilderConfig**

Variables

- **avg_timing_iterations** – int The number of averaging iterations used when timing layers. When timing layers, the builder minimizes over a set of average times for layer execution. This parameter controls the number of iterations used in averaging. By default the number of averaging iterations is 1.
- **flags** – int The build mode flags to turn on builder options for this network. The flags are listed in the BuilderFlags enum. The flags set configuration options to build the network. This should be in integer consisting of one or more *BuilderFlag* s, combined via binary OR. For example, `1 << BuilderFlag.FP16 | 1 << BuilderFlag.DEBUG`.

- **profile_stream** – int The handle for the CUDA stream that is used to profile this network.
- **num_optimization_profiles** – int The number of optimization profiles.
- **default_device_type** – *tensorrt.DeviceType* The default DeviceType to be used by the Builder.
- **DLA_core** – int The DLA core that the engine executes on. Must be between 0 and N-1 where N is the number of available DLA cores.
- **profiling_verbosity** – Profiling verbosity in NVTX annotations.
- **engine_capability** – The desired engine capability. See *EngineCapability* for details.
- **builder_optimization_level** – The builder optimization level which TensorRT should build the engine at. Setting a higher optimization level allows TensorRT to spend longer engine building time searching for more optimization options. The resulting engine may have better performance compared to an engine built with a lower optimization level. The default optimization level is 3. Valid values include integers from 0 to the maximum optimization level, which is currently 5. Setting it to be greater than the maximum level results in identical behavior to the maximum level.
- **max_num_tactics** – The maximum number of tactics to time when there is a choice of tactics. Setting a larger number allows TensorRT to spend longer engine building time searching for more optimization options. The resulting engine may have better performance compared to an engine built with a smaller number of tactics. Valid values include integers from -1 to the maximum 32-bit integer. Default value -1 indicates that TensorRT can decide the number of tactics based on its own heuristic.
- **hardware_compatibility_level** – Hardware compatibility allows an engine compatible with GPU architectures other than that of the GPU on which the engine was built.
- **plugins_to_serialize** – The plugin libraries to be serialized with forward-compatible engines.
- **max_aux_streams** – The maximum number of auxiliary streams that TRT is allowed to use. If the network contains operators that can run in parallel, TRT can execute them using auxiliary streams in addition to the one provided to the IExecutionContext::enqueueV3() call. The default maximum number of auxiliary streams is determined by the heuristics in TensorRT on whether enabling multi-stream would improve the performance. This behavior can be overridden by calling this API to set the maximum number of auxiliary streams explicitly. Set this to 0 to enforce single-stream inference. The resulting engine may use fewer auxiliary streams than the maximum if the network does not contain enough parallelism or if TensorRT determines that using more auxiliary streams does not help improve the performance. Allowing more auxiliary streams does not always give better performance since there will be synchronizations overhead between streams. Using CUDA graphs at runtime can help reduce the overhead caused by cross-stream synchronizations. Using more auxiliary leads to more memory usage at runtime since some activation memory blocks will not be able to be reused.
- **progress_monitor** – The *IProgressMonitor* to use.
- **tiling_optimization_level** – The optimization level of tiling strategies. A Higher level allows TensorRT to spend more time searching for better optimization strategy.
- **l2_limit_for_tiling** – The target L2 cache usage for tiling optimization.
- **remote_auto_tuning_config** – The config string to be used during remote auto-tuning. Remote auto-tuning is only enabled for engines built with EngineCapability.SAFETY.

Below are the descriptions about each builder optimization level:

- Level 0: This enables the fastest compilation by disabling dynamic kernel generation and selecting the first tactic that succeeds in execution. This will also not respect a timing cache.
- Level 1: Available tactics are sorted by heuristics, but only the top are tested to select the best. If a dynamic kernel is generated its compile optimization is low.
- Level 2: Available tactics are sorted by heuristics, but only the fastest tactics are tested to select the best.
- Level 3: Apply heuristics to see if a static precompiled kernel is applicable or if a new one has to be compiled dynamically.
- Level 4: Always compiles a dynamic kernel.
- Level 5: Always compiles a dynamic kernel and compares it to static kernels.

__del__(*self:* tensorrt.tensorrt.IBuilderConfig) → None

__exit__(*exc_type*, *exc_value*, *traceback*)

Context managers are deprecated and have no effect. Objects are automatically freed when the reference count reaches 0.

__init__(**args*, ***kwargs*)

add_optimization_profile(*self:* tensorrt.tensorrt.IBuilderConfig, *profile:* tensorrt.tensorrt.IOptimizationProfile) → int

Add an optimization profile.

This function must be called at least once if the network has dynamic or shape input tensors.

Parameters
profile – The new optimization profile, which must satisfy `bool(profile) == True`

Returns
The index of the optimization profile (starting from 0) if the input is valid, or -1 if the input is not valid.

can_run_on_DLA(*self:* tensorrt.tensorrt.IBuilderConfig, *layer:* tensorrt.tensorrt.ILayer) → bool

Check if the layer can run on DLA.

Parameters
layer – The layer to check

Returns
A *bool* indicating whether the layer can run on DLA

clear_flag(*self:* tensorrt.tensorrt.IBuilderConfig, *flag: tensorrt.tensorrt.BuilderFlag*) → None

Clears the builder mode flag from the enabled flags.

Parameters
flag – The flag to clear.

create_timing_cache(*self:* tensorrt.tensorrt.IBuilderConfig, *serialized_timing_cache: collections.abc.Buffer*) → *tensorrt.tensorrt.ITimingCache*

Create timing cache

Create *ITimingCache* instance from serialized raw data. The created timing cache doesn't belong to a specific builder config. It can be shared by multiple builder instances

Parameters
serialized_timing_cache – The serialized timing cache. If an empty cache is provided (i.e. `b""`), a new cache will be created.

Returns
The created *ITimingCache* object.

get_device_type(*self:* tensorrt.tensorrt.IBuilderConfig, *layer:* tensorrt.tensorrt.ILayer) → tensorrt.tensorrt.DeviceType

Get the device that the layer executes on.

Parameters
layer – The layer to get the DeviceType for

Returns
The DeviceType of the layer

get_flag(*self:* tensorrt.tensorrt.IBuilderConfig, *flag: tensorrt.tensorrt.BuilderFlag*) → bool

Check if a build mode flag is set.

Parameters
flag – The flag to check.

Returns
A *bool* indicating whether the flag is set.

get_memory_pool_limit(*self:* tensorrt.tensorrt.IBuilderConfig, *pool: tensorrt.tensorrt.MemoryPoolType*) → int

Retrieve the memory size limit of the corresponding pool in bytes. If *set_memory_pool_limit()* for the pool has not been called, this returns the default value used by TensorRT. This default value is not necessarily the maximum possible value for that configuration.

Parameters
pool – The memory pool to get the limit for.

Returns
The size of the memory limit, in bytes, for the corresponding pool.

get_preview_feature(*self:* tensorrt.tensorrt.IBuilderConfig, *feature: tensorrt.tensorrt.PreviewFeature*) → bool

Check if a preview feature is enabled.

Parameters
feature – the feature to query

Returns
true if the feature is enabled, false otherwise

get_tactic_sources(*self:* tensorrt.tensorrt.IBuilderConfig) → int

Get the tactic sources currently set in the engine build configuration.

get_timing_cache(*self:* tensorrt.tensorrt.IBuilderConfig) → *tensorrt.tensorrt.ITimingCache*

Get the timing cache from current IBuilderConfig

Returns
The timing cache used in current IBuilderConfig, or *None* if no timing cache is set.

is_device_type_set(*self:* tensorrt.tensorrt.IBuilderConfig, *layer:* tensorrt.tensorrt.ILayer) → bool

Check if the DeviceType for a layer is explicitly set.

Parameters
layer – The layer to check for DeviceType

Returns
True if DeviceType is not default, False otherwise

reset(*self:* tensorrt.tensorrt.IBuilderConfig) → None

Resets the builder configuration to defaults. When initializing a builder config object, we can call this function.

reset_device_type(*self:* tensorrt.tensorrt.IBuilderConfig, *layer:* tensorrt.tensorrt.ILayer) → None

Reset the DeviceType for the given layer.

Parameters
layer – The layer to reset the DeviceType for

set_device_type(*self:* tensorrt.tensorrt.IBuilderConfig, *layer:* tensorrt.tensorrt.ILayer, *device_type: tensorrt.tensorrt.DeviceType*) → None

Set the device that this layer must execute on. If DeviceType is not set or is reset, TensorRT will use the default DeviceType set in the builder.

The DeviceType for a layer must be compatible with the safety flow (if specified). For example a layer cannot be marked for DLA execution while the builder is configured for SAFE_GPU.

Parameters

- **layer** – The layer to set the DeviceType of
- **device_type** – The DeviceType the layer must execute on

set_flag(*self:* tensorrt.tensorrt.IBuilderConfig, *flag: tensorrt.tensorrt.BuilderFlag*) → None

Add the input builder mode flag to the already enabled flags.

Parameters
flag – The flag to set.

set_memory_pool_limit(*self:* tensorrt.tensorrt.IBuilderConfig, *pool: tensorrt.tensorrt.MemoryPoolType*, *pool_size: SupportsInt*) → None

Set the memory size for the memory pool.

TensorRT layers access different memory pools depending on the operation. This function sets in the *IBuilderConfig* the size limit, specified by pool_size, for the corresponding memory pool, specified by pool. TensorRT will build a plan file that is constrained by these limits or report which constraint caused the failure.

If the size of the pool, specified by pool_size, fails to meet the size requirements for the pool, this function does nothing and emits the recoverable error, ErrorCode.INVALID_ARGUMENT, to the registered *IErrorRecorder* .

If the size of the pool is larger than the maximum possible value for the configuration, this function does nothing and emits ErrorCode.UNSUPPORTED_STATE.

If the pool does not exist on the requested device type when building the network, a warning is emitted to the logger, and the memory pool value is ignored.

Refer to MemoryPoolType to see the size requirements for each pool.

Parameters

- **pool** – The memory pool to limit the available memory for.
- **pool_size** – The size of the pool in bytes.

set_preview_feature(*self:* tensorrt.tensorrt.IBuilderConfig, *feature: tensorrt.tensorrt.PreviewFeature*, *enable: bool*) → None

Enable or disable a specific preview feature.

Allows enabling or disabling experimental features, which are not enabled by default in the current release. Preview Features have been fully tested but are not yet as stable as other features in TensorRT. They are provided as opt-in features for at least one release.

Refer to PreviewFeature for additional information, and a list of the available features.

Parameters

- **feature** – the feature to enable
- **enable** – whether to enable or disable

set_tactic_sources(*self:* tensorrt.tensorrt.IBuilderConfig, *tactic_sources: SupportsInt*) → bool

Set tactic sources.

This bitset controls which tactic sources TensorRT is allowed to use for tactic selection.

Multiple tactic sources may be combined with a bitwise OR operation. For example, to enable edge mask convolutions and JIT convolutions as tactic sources, use a value of: `1 << int(trt.TacticSource.EDGE_MASK_CONVOLUTIONS) | 1 << int(trt.TacticSource.JIT_CONVOLUTIONS)`

Parameters
tactic_sources – The tactic sources to set

Returns
A *bool* indicating whether the tactic sources in the build configuration were updated. The tactic sources in the build configuration will not be updated if the provided value is invalid.

set_timing_cache(*self:* tensorrt.tensorrt.IBuilderConfig, *cache:* tensorrt.tensorrt.ITimingCache, *ignore_mismatch: bool*) → bool

Attach a timing cache to IBuilderConfig

The timing cache has verification header to make sure the provided cache can be used in current environment. A failure will be reported if the CUDA device property in the provided cache is different from current environment. `bool(ignore_mismatch) == True` skips strict verification and allows loading cache created from a different device. The cache must not be destroyed until after the engine is built.

Parameters

- **cache** – The timing cache to be used
- **ignore_mismatch** – Whether or not allow using a cache that contains different CUDA device property

Returns
A *BOOL* indicating whether the operation is done successfully.

5.5 Builder

5.5.1 NetworkDefinitionCreationFlag

tensorrt.**NetworkDefinitionCreationFlag**

List of immutable network properties expressed at network creation time.

Members:

STRONGLY_TYPED : [DEPRECATED] Deprecated in TensorRT 11.0. Strongly typed mode is always enabled. This flag is retained for API compatibility but is ignored.

PREFER_AOT_PYTHON_PLUGINS : If set, for a Python plugin with both AOT and JIT implementations, the AOT implementation will be used.

PREFER_JIT_PYTHON_PLUGINS : If set, for a Python plugin with both AOT and JIT implementations, the JIT implementation will be used.

5.5.2 Builder

class tensorrt.**Builder**(*self:* tensorrt.tensorrt.Builder, *logger:* tensorrt.tensorrt.ILogger)

Builds an *ICudaEngine* from a *INetworkDefinition* .

Variables

- **max_DLA_batch_size** – int The maximum batch size DLA can support. For any tensor the total volume of index dimensions combined(dimensions other than CHW) with the requested batch size should not exceed the value returned by this function.
- **num_DLA_cores** – int The number of DLA engines available to this builder.
- **error_recorder** – *IErrorRecorder* Application-implemented error reporting interface for TensorRT objects.
- **gpu_allocator** – *IGpuAllocator* The GPU allocator to be used by the *Builder* . All GPU memory acquired will use this allocator. If set to None, the default allocator will be used.
- **logger** – *ILogger* The logger provided when creating the refitter.
- **max_threads** – int The maximum thread that can be used by the *Builder*.

Parameters

logger – The logger to use.

__del__(*self:* tensorrt.tensorrt.Builder) → None

__exit__(*exc_type*, *exc_value*, *traceback*)

Context managers are deprecated and have no effect. Objects are automatically freed when the reference count reaches 0.

__init__(*self:* tensorrt.tensorrt.Builder, *logger:* tensorrt.tensorrt.ILogger) → None

Parameters

logger – The logger to use.

build_engine_with_config(*self:* tensorrt.tensorrt.Builder, *network:* tensorrt.tensorrt.INetworkDefinition, *config:* tensorrt.tensorrt.IBuilderConfig) → *tensorrt.tensorrt.ICudaEngine*

Builds a network for the given *INetworkDefinition* and *IBuilderConfig* .

This function allows building a network and creating an engine.

Parameters

- **network** – Network definition.
- **config** – Builder configuration.

Returns

A pointer to a *ICudaEngine* object that contains a built engine.

build_serialized_network(**args*, ***kwargs*)

Overloaded function.

1. build_serialized_network(self: tensorrt.tensorrt.Builder, network: tensorrt.tensorrt.INetworkDefinition, config: tensorrt.tensorrt.IBuilderConfig) -> tensorrt.tensorrt.IHostMemory

   Builds and serializes a network for the given *INetworkDefinition* and *IBuilderConfig* .

   This function allows building and serialization of a network without creating an engine.

   arg network
   Network definition.

   arg config
   Builder configuration.

   returns
   A pointer to a *IHostMemory* object that contains a serialized network.

2. build_serialized_network(self: tensorrt.tensorrt.Builder, network: tensorrt.tensorrt.INetworkDefinition, config: tensorrt.tensorrt.IBuilderConfig, kernel_text: object) -> tensorrt.tensorrt.IHostMemory

   Builds and serializes a network for the given *INetworkDefinition* and *IBuilderConfig* .

   This function allows building and serialization of a network without creating an engine.

   arg network
   Network definition.

   arg config
   Builder configuration.

   returns
   A pointer to a *IHostMemory* object that contains a serialized network.

build_serialized_network_to_stream(*self:* tensorrt.tensorrt.Builder, *network:* tensorrt.tensorrt.INetworkDefinition, *config:* tensorrt.tensorrt.IBuilderConfig, *writer:* *tensorrt.tensorrt.IStreamWriter*) → bool

Builds and serializes a network for the given *INetworkDefinition* and *IBuilderConfig* and write the serialized network to a writer stream.

This function allows building and serialization of a network without creating an engine.

Parameters

- **network** – Network definition.
- **config** – Builder configuration.
- **writer** – Output stream writer.

Returns
True if build succeed, otherwise False.

create_builder_config(*self:* tensorrt.tensorrt.Builder) → *tensorrt.tensorrt.IBuilderConfig*

Create a builder configuration object.

See *IBuilderConfig*

create_network(*self:* tensorrt.tensorrt.Builder, *flags: SupportsInt = 0*) → *tensorrt.tensorrt.INetworkDefinition*

Create a *INetworkDefinition* object.

Parameters
flags – *NetworkDefinitionCreationFlag* s combined using bitwise OR.

Returns
An empty TensorRT *INetworkDefinition* .

create_optimization_profile(*self:* tensorrt.tensorrt.Builder) → *tensorrt.tensorrt.IOptimizationProfile*

Create a new optimization profile.

If the network has any dynamic input tensors, the appropriate calls to *IOptimizationProfile.set_shape()* must be made. Likewise, if there are any shape input tensors, the appropriate calls to *IOptimizationProfile.set_shape_input()* are required.

See *IOptimizationProfile*

get_plugin_registry(*self:* tensorrt.tensorrt.Builder) → *tensorrt.tensorrt.IPluginRegistry*

Get the local plugin registry that can be used by the builder.

Returns
The local plugin registry that can be used by the builder.

is_network_supported(*self:* tensorrt.tensorrt.Builder, *network:* tensorrt.tensorrt.INetworkDefinition, *config:* tensorrt.tensorrt.IBuilderConfig) → bool

Checks that a network is within the scope of the *IBuilderConfig* settings.

Parameters

- **network** – The network definition to check for configuration compliance.
- **config** – The configuration of the builder to use when checking the network.

Given an *INetworkDefinition* and an *IBuilderConfig* , check if the network falls within the constraints of the builder configuration based on the *EngineCapability* , *BuilderFlag* , and *DeviceType* .

Returns
`True` if network is within the scope of the restrictions specified by the builder config, `False` otherwise. This function reports the conditions that are violated to the registered `ErrorRecorder` .

NOTE: A `True` return value does not guarantee that engine building will succeed, as backends may reject it for
reasons not detectable with this fast validation. To definitively check whether a network can be built with a given config, use *build_engine_with_config()* or *build_serialized_network()* (depending on the engine capability).

NOTE: This function will synchronize the cuda stream returned by `config.profile_stream` before returning.

reset(*self:* tensorrt.tensorrt.Builder) → None

Resets the builder state to default values.

5.6 ProgressMonitor

class tensorrt.**IProgressMonitor**(*self:* tensorrt.tensorrt.IProgressMonitor)

Application-implemented progress reporting interface for TensorRT.

The IProgressMonitor is a user-defined object that TensorRT uses to report back when an internal algorithm has started or finished a phase to help provide feedback on the progress of the optimizer.

The IProgressMonitor will trigger its start function when a phase is entered and will trigger its finish function when that phase is exited. Each phase consists of one or more steps. When each step is completed, the step_complete function is triggered. This will allow an application using the builder to communicate progress relative to when the optimization step is expected to complete.

The implementation of IProgressMonitor must be thread-safe so that it can be called from multiple internal threads. The lifetime of the IProgressMonitor must exceed the lifetime of all TensorRT objects that use it.

phase_finish(*self:* tensorrt.tensorrt.IProgressMonitor, *phase_name: str*) → None

Signal that a phase of the optimizer has finished.

Parameters
phase_name – The name of the phase that has finished.

The phase_finish function signals to the application that the phase is complete. This function may be called before all steps in the range [0, num_steps) have been reported to step_complete. This scenario can be triggered by error handling, internal optimizations, or when step_complete returns False to request cancellation of the build.

phase_start(*self:* tensorrt.tensorrt.IProgressMonitor, *phase_name: str*, *parent_phase: str*, *num_steps: SupportsInt*) → None

Signal that a phase of the optimizer has started.

Parameters

- **phase_name** – The name of this phase for tracking purposes.
- **parent_phase** – The parent phase that this phase belongs to, None if there is no parent.
- **num_steps** – The number of steps that are involved in this phase.

The phase_start function signals to the application that the current phase is beginning, and that it has a certain number of steps to perform. If phase_parent is None, then the phase_start is beginning an independent phase, and if phase_parent is specified, then the current phase, specified by phase_name, is within the scope of the parent phase. num_steps will always be a positive number. The phase_start function implies that the first step is being executed. TensorRT will signal when each step is complete.

Phase names are human readable English strings which are unique within a single phase hierarchy but which can be reused once the previous instance has completed. Phase names and their hierarchies may change between versions of TensorRT.

step_complete(*self:* tensorrt.tensorrt.IProgressMonitor, *phase_name: str*, *step: SupportsInt*) → bool

Signal that a step of an optimizer phase has finished.

Parameters

- **phase_name** – The name of the innermost phase being executed.
- **step** – The step number that was completed.

The step_complete function signals to the application that TensorRT has finished the current step for the phase `phase_name` , and will move on to the next step if there is one. The application can return False

for TensorRT to exit the build early. The step value will increase on subsequent calls in the range [0, num_steps).

Returns
True to continue to the next step or False to stop the build.

5.7 Debug Listener

5.7.1 IDebugListener

class tensorrt.**IDebugListener**(*self:* tensorrt.tensorrt.IDebugListener)

A user-implemented class for notification when value of a debug tensor is updated.

__init__(*self:* tensorrt.tensorrt.IDebugListener) → None

process_debug_tensor(*self:* tensorrt.tensorrt.IDebugListener, *addr: typing_extensions.CapsuleType*, *location: tensorrt.TensorLocation*, *type: tensorrt.tensorrt.DataType*, *shape:* tensorrt.tensorrt.Dims, *name: str*, *stream: SupportsInt*) → None

User implemented callback function that is called when value of a debug tensor is updated and the debug state of the tensor is set to true. Content in the given address is only guaranteed to be valid for the duration of the callback.

Parameters

- **location** – TensorLocation of the tensor
- **addr** – pointer to buffer
- **type** – data Type of the tensor
- **shape** – shape of the tensor
- **name** – name of the tensor
- **stream** – Cuda stream object

Returns
True on success, False otherwise.

5.8 ICudaEngine

tensorrt.**TensorIOMode**

IO tensor modes for TensorRT.

Members:

NONE : Tensor is not an input or output.

INPUT : Tensor is input to the engine.

OUTPUT : Tensor is output to the engine.

class tensorrt.**ICudaEngine**

An *ICudaEngine* for executing inference on a built network.

The engine can be indexed with [] . When indexed in this way with an integer, it will return the corresponding binding name. When indexed with a string, it will return the corresponding binding index.

Variables

- **num_io_tensors** – int The number of IO tensors.
- **num_layers** – int The number of layers in the network. The number of layers in the network is not necessarily the number in the original *INetworkDefinition*, as layers may be combined or eliminated as the *ICudaEngine* is optimized. This value can be useful when building per-layer tables, such as when aggregating profiling data over a number of executions.
- **max_workspace_size** – int The amount of workspace the *ICudaEngine* uses. The workspace size will be no greater than the value provided to the *Builder* when the *ICudaEngine* was built, and will typically be smaller. Workspace will be allocated for each *IExecutionContext* .
- **device_memory_size_v2** – int The amount of device memory required by an *IExecutionContext*. The return value depends on the weight streaming budget if enabled.
- **refittable** – bool Whether the engine can be refit.
- **name** – str The name of the network associated with the engine. The name is set during network creation and is retrieved after building or deserialization.
- **num_optimization_profiles** – int The number of optimization profiles defined for this engine. This is always at least 1.
- **error_recorder** – *IErrorRecorder* Application-implemented error reporting interface for TensorRT objects.
- **engine_capability** – *EngineCapability* The engine capability. See *EngineCapability* for details.
- **tactic_sources** – int The tactic sources required by this engine.
- **profiling_verbosity** – The profiling verbosity the builder config was set to when the engine was built.
- **hardware_compatibility_level** – The hardware compatibility level of the engine.
- **num_aux_streams** – Read-only. The number of auxiliary streams used by this engine, which will be less than or equal to the maximum allowed number of auxiliary streams by setting builder_config.max_aux_streams when the engine is built.
- **streamable_weights_size** – Returns the size of the streamable weights in the engine. This may not include all the weights.
- **weight_streaming_budget_v2** – Set and get the current weight streaming budget for inference. The budget may be set any non-negative value. A value of 0 streams the most weights. Values equal to streamable_weights_size (default) or larger will disable weight streaming.
- **weight_streaming_scratch_memory_size** – The amount of scratch memory required by a TensorRT ExecutionContext to perform inference. This value may change based on the current weight streaming budget. Please use the V2 memory APIs, engine.device_memory_size_v2 and ExecutionContext.set_device_memory() to provide memory which includes the current weight streaming scratch memory. Not specifying these APIs or using the V1 APIs will not include this memory, so TensorRT will resort to allocating itself.

__del__(*self:* tensorrt.tensorrt.ICudaEngine) → None

__exit__(*exc_type*, *exc_value*, *traceback*)

Context managers are deprecated and have no effect. Objects are automatically freed when the reference count reaches 0.

__getitem__(*self:* tensorrt.tensorrt.ICudaEngine, *arg0: SupportsInt*) → str

__init__(**args*, ***kwargs*)

create_engine_inspector(*self:* tensorrt.tensorrt.ICudaEngine) → *tensorrt.tensorrt.EngineInspector*

Create an `IEngineInspector` which prints out the layer information of an engine or an execution context.

Returns
The `IEngineInspector`.

create_execution_context(**args*, ***kwargs*)

Overloaded function.

1. create_execution_context(self: tensorrt.tensorrt.ICudaEngine, strategy: tensorrt.tensorrt.ExecutionContextAllocationStrategy = <ExecutionContextAllocationStrategy.STATIC: 0>) -> tensorrt.tensorrt.IExecutionContext

   Create an *IExecutionContext* and specify the device memory allocation strategy.

   returns
   The newly created *IExecutionContext* .

2. create_execution_context(self: tensorrt.tensorrt.ICudaEngine, runtime_config: tensorrt.tensorrt.IRuntimeConfig = None) -> tensorrt.tensorrt.IExecutionContext

   Create an *IExecutionContext* and specify the device memory allocation strategy.

   returns
   The newly created *IExecutionContext* .

create_runtime_config(*self:* tensorrt.tensorrt.ICudaEngine) → *tensorrt.tensorrt.IRuntimeConfig*

Create a runtime configuration.

Returns
The newly created *IRuntimeConfig* .

create_serialization_config(*self:* tensorrt.tensorrt.ICudaEngine) → *tensorrt.tensorrt.ISerializationConfig*

Create a serialization configuration object.

get_aliased_input_tensor(*self:* tensorrt.tensorrt.ICudaEngine, *name: str*) → str

Determine if an output tensor is aliased with an input tensor.

For certain operations like KVCacheUpdate, the output tensor may share the same memory buffer as the input tensor to avoid unnecessary copies. This method returns the name of the input tensor that shares memory with the given output tensor name. If the given tensor name is not an output tensor or the output does not alias with any input, `None` should be returned.

Parameters
name – The output tensor name.

Returns
The name of the input tensor that is aliased with this output.

get_device_memory_size_for_profile_v2(*self:* tensorrt.tensorrt.ICudaEngine, *profile_index: SupportsInt*) → int

Return the device memory size required for a certain profile.

The return value will change depending on the following API calls 1. setWeightStreamingBudgetV2

Parameters
profile_index – The index of the profile.

get_engine_stat(*self: tensorrt.tensorrt.ICudaEngine*, *stat: tensorrt.tensorrt.EngineStat = <EngineStat.TOTAL_WEIGHTS_SIZE: 0>*) → int

Return the engine statistics specified by the given enum value. If STRIPPED_WEIGHTS_SIZE is passed to query a normal engine, this function will return -1 to indicate invalid enum value.

Parameters
stat – The engine statistic kind to get.

get_tensor_bytes_per_component(**args*, ***kwargs*)

Overloaded function.

1. get_tensor_bytes_per_component(self: tensorrt.tensorrt.ICudaEngine, name: str) -> int

   Return the number of bytes per component of an element.

   The vector component size is returned if *get_tensor_vectorized_dim()* != -1.

   arg name
   The tensor name.

2. get_tensor_bytes_per_component(self: tensorrt.tensorrt.ICudaEngine, name: str, profile_index: typing.SupportsInt) -> int

   Return the number of bytes per component of an element.

   The vector component size is returned if *get_tensor_vectorized_dim()* != -1.

   arg name
   The tensor name.

get_tensor_components_per_element(**args*, ***kwargs*)

Overloaded function.

1. get_tensor_components_per_element(self: tensorrt.tensorrt.ICudaEngine, name: str) -> int

   Return the number of components included in one element.

   The number of elements in the vectors is returned if *get_tensor_vectorized_dim()* != -1.

   arg name
   The tensor name.

2. get_tensor_components_per_element(self: tensorrt.tensorrt.ICudaEngine, name: str, profile_index: typing.SupportsInt) -> int

   Return the number of components included in one element.

   The number of elements in the vectors is returned if *get_tensor_vectorized_dim()* != -1.

   arg name
   The tensor name.

get_tensor_dtype(*self:* tensorrt.tensorrt.ICudaEngine, *name: str*) → tensorrt.tensorrt.DataType

Return the required data type for a buffer from its tensor name.

Parameters
name – The tensor name.

get_tensor_format(**args*, ***kwargs*)

Overloaded function.

1. get_tensor_format(self: tensorrt.tensorrt.ICudaEngine, name: str) -> tensorrt.tensorrt.TensorFormat

   Return the tensor format.

   arg name
   The tensor name.

2. get_tensor_format(self: tensorrt.tensorrt.ICudaEngine, name: str, profile_index: typing.SupportsInt) -> tensorrt.tensorrt.TensorFormat

   Return the tensor format.

   arg name
   The tensor name.

get_tensor_format_desc(**args*, ***kwargs*)

Overloaded function.

1. get_tensor_format_desc(self: tensorrt.tensorrt.ICudaEngine, name: str) -> str

   Return the human readable description of the tensor format.

   The description includes the order, vectorization, data type, strides, etc. For example:

   Example 1: CHW + FP32
   "Row major linear FP32 format"
   Example 2: CHW2 + FP16
   "Two wide channel vectorized row major FP16 format"
   Example 3: HWC8 + FP16 + Line Stride = 32
   "Channel major FP16 format where C % 8 == 0 and H Stride % 32 == 0"

   arg name
   The tensor name.

2. get_tensor_format_desc(self: tensorrt.tensorrt.ICudaEngine, name: str, profile_index: typing.SupportsInt) -> str

   Return the human readable description of the tensor format.

   The description includes the order, vectorization, data type, strides, etc. For example:

   Example 1: CHW + FP32
   "Row major linear FP32 format"
   Example 2: CHW2 + FP16
   "Two wide channel vectorized row major FP16 format"
   Example 3: HWC8 + FP16 + Line Stride = 32
   "Channel major FP16 format where C % 8 == 0 and H Stride % 32 == 0"

arg name
The tensor name.

get_tensor_location(*self:* tensorrt.tensorrt.ICudaEngine, *name: str*) → tensorrt.tensorrt.TensorLocation

Determine whether an input or output tensor must be on GPU or CPU.

Parameters
name – The tensor name.

get_tensor_mode(*self:* tensorrt.tensorrt.ICudaEngine, *name: str*) → tensorrt.tensorrt.TensorIOMode

Determine whether a tensor is an input or output tensor.

Parameters
name – The tensor name.

get_tensor_name(*self:* tensorrt.tensorrt.ICudaEngine, *index: SupportsInt*) → str

Return the name of an input or output tensor.

Parameters
index – The tensor index.

get_tensor_profile_shape(*self:* tensorrt.tensorrt.ICudaEngine, *name: str*, *profile_index: SupportsInt*) → list[*tensorrt.tensorrt.Dims*]

Get the minimum/optimum/maximum dimensions for a particular tensor under an optimization profile.

Parameters

- **name** – The tensor name.
- **profile_index** – The index of the profile.

get_tensor_profile_values(*self:* tensorrt.tensorrt.ICudaEngine, *name: SupportsInt*, *profile_index: str*) → list[list[int]]

Get minimum/optimum/maximum values for an input shape binding under an optimization profile. If the specified binding is not an input shape binding, an exception is raised.

Parameters

- **name** – The tensor name.
- **profile_index** – The index of the profile.

Returns
A `List[List[int]]` of length 3, containing the minimum, optimum, and maximum values, in that order. If the values have not been set yet, an empty list is returned.

get_tensor_shape(*self:* tensorrt.tensorrt.ICudaEngine, *name: str*) → *tensorrt.tensorrt.Dims*

Return the shape of an input or output tensor.

Parameters
name – The tensor name.

get_tensor_vectorized_dim(**args*, ***kwargs*)

Overloaded function.

1. get_tensor_vectorized_dim(self: tensorrt.tensorrt.ICudaEngine, name: str) -> int

   Return the dimension index that the buffer is vectorized.

   Specifically -1 is returned if scalars per vector is 1.

   arg name
   The tensor name.

2. get_tensor_vectorized_dim(self: tensorrt.tensorrt.ICudaEngine, name: str, profile_index: typing.SupportsInt) -> int

   Return the dimension index that the buffer is vectorized.

   Specifically -1 is returned if scalars per vector is 1.

   arg name
   The tensor name.

get_weight_streaming_automatic_budget(*self:* tensorrt.tensorrt.ICudaEngine) → int

Get an automatic weight streaming budget based on available device memory. This value may change between TensorRT major and minor versions. Please use CudaEngine.weight_streaming_budget_v2 to set the returned budget.

is_debug_tensor(*self:* tensorrt.tensorrt.ICudaEngine, *name: str*) → bool

Determine whether the given name corresponds to a debug tensor.

Parameters
name – The tensor name.

is_shape_inference_io(*self:* tensorrt.tensorrt.ICudaEngine, *name: str*) → bool

Determine whether a tensor is read or written by infer_shapes.

Parameters
name – The tensor name.

serialize(*self:* tensorrt.tensorrt.ICudaEngine) → *tensorrt.tensorrt.IHostMemory*

Serialize the engine to a stream.

Returns
An *IHostMemory* object containing the serialized *ICudaEngine* .

serialize_with_config(*self:* tensorrt.tensorrt.ICudaEngine, *arg0:* tensorrt.tensorrt.ISerializationConfig) → *tensorrt.tensorrt.IHostMemory*

Serialize the network to a stream.

5.9 IExecutionContext

class tensorrt.**IOutputAllocator**(*self:* tensorrt.tensorrt.IOutputAllocator)

Application-implemented class for controlling output tensor allocation.

To implement a custom output allocator, ensure that you explicitly instantiate the base class in *__init__()* :

```
class MyOutputAllocator(trt.IOutputAllocator):
    def __init__(self):
        trt.IOutputAllocator.__init__(self)

    def reallocate_output(self, tensor_name, memory, size, alignment):
        ... # Your implementation here

    def reallocate_output_async(self, tensor_name, memory, size, alignment, stream):
        ... # Your implementation here

    def notify_shape(self, tensor_name, shape):
        ... # Your implementation here
```

__init__(*self:* tensorrt.tensorrt.IOutputAllocator) → None

class tensorrt.**IExecutionContext**

Context for executing inference using an *ICudaEngine* . Multiple *IExecutionContext* s may exist for one *ICudaEngine* instance, allowing the same *ICudaEngine* to be used for the execution of multiple batches simultaneously.

Variables

- **debug_sync** – bool The debug sync flag. If this flag is set to true, the *ICudaEngine* will log the successful execution for each kernel during execute_v2().
- **profiler** – *IProfiler* The profiler in use by this *IExecutionContext* .
- **engine** – *ICudaEngine* The associated *ICudaEngine* .
- **name** – str The name of the *IExecutionContext* .
- **device_memory** – capsule The device memory for use by this execution context. The memory must be aligned with cuda memory alignment property (using `cuda.cudart.cudaGetDeviceProperties()`), and its size must be large enough for performing inference with the given network inputs. `engine.device_memory_size()` and `engine.get_device_memory_size_for_profile()` report upper bounds of the size. Setting memory to nullptr is acceptable if the reported size is 0. If using *execute_async_v3()* to run the network, the memory is in use from the invocation of *execute_async_v3()* until network execution is complete. If using *execute_v2()*, it is in use until *execute_v2()* returns. Releasing or otherwise using the memory for other purposes, including using it in another execution context running in parallel, during this time will result in undefined behavior.
- **active_optimization_profile** – int The active optimization profile for the context. The selected profile will be used in subsequent calls to *execute_v2()*. Profile 0 is selected by default. This is a readonly property and active optimization profile can be changed with *set_optimization_profile_async()*. Changing this value will invalidate all dynamic bindings for the current execution context, so that they have to be set again using *set_input_shape()* before calling either *execute_v2()*.
- **all_binding_shapes_specified** – bool Whether all dynamic dimensions of input tensors have been specified by calling *set_input_shape()* . Trivially true if network has no dynamically shaped input tensors. Does not work with name-base interfaces eg. *set_input_shape()*. Use *infer_shapes()* instead.
- **error_recorder** – *IErrorRecorder* Application-implemented error reporting interface for TensorRT objects.
- **enqueue_emits_profile** – bool Whether enqueue emits layer timing to the profiler. The default value is `True`. If set to `False`, enqueue will be asynchronous if there is a profiler attached. An extra method `IExecutionContext::report_to_profiler()` needs to be called to obtain the profiling data and report to the profiler attached.
- **persistent_cache_limit** – The maximum size of persistent L2 cache that this execution context may use for activation caching. Activation caching is not supported on all architectures - see "How TensorRT uses Memory" in the developer guide for details. The default is 0 Bytes.
- **nvtx_verbosity** – The NVTX verbosity of the execution context. Building with DETAILED verbosity will generally increase latency in enqueueV3(). Call this method to select NVTX verbosity in this execution context at runtime. The default is the verbosity with which the engine was built, and the verbosity may not be raised above that level. This function does not affect how IEngineInspector interacts with the engine.

- **temporary_allocator** – *IGpuAllocator* The GPU allocator used for internal temporary storage.

__del__(*self:* tensorrt.tensorrt.IExecutionContext) → None

__exit__(*exc_type*, *exc_value*, *traceback*)

Context managers are deprecated and have no effect. Objects are automatically freed when the reference count reaches 0.

__init__(**args*, ***kwargs*)

execute_async_v3(*self:* tensorrt.tensorrt.IExecutionContext, *stream_handle: SupportsInt*) → bool

Asynchronously execute inference.

Modifying or releasing memory that has been registered for the tensors before stream synchronization or the event passed to *set_input_consumed_event()* has been triggered results in undefined behavior.

Input tensors can be released after the *set_input_consumed_event()* whereas output tensors require stream synchronization.

Parameters
stream_handle – The cuda stream on which the inference kernels will be enqueued. Using default stream may lead to performance issues due to additional cudaDeviceSynchronize() calls by TensorRT to ensure correct synchronizations. Please use non-default stream instead.

execute_v2(*self:* tensorrt.tensorrt.IExecutionContext, *bindings: collections.abc.Sequence[SupportsInt]*) → bool

Synchronously execute inference on a batch. This method requires a array of input and output buffers.

Parameters
bindings – A list of integers representing input and output buffer addresses for the network.

Returns
True if execution succeeded.

get_debug_listener(*self:* tensorrt.tensorrt.IExecutionContext) → *tensorrt.tensorrt.IDebugListener*

Get debug listener for execution context.

Returns
The *IDebugListener* of the execution context.

get_debug_state(*self:* tensorrt.tensorrt.IExecutionContext, *name: str*) → bool

Get the debug state of the tensor.

Parameters
name – The name of the tensor.

get_input_consumed_event(*self:* tensorrt.tensorrt.IExecutionContext) → int

Return the event associated with consuming the input tensors.

get_max_output_size(*self:* tensorrt.tensorrt.IExecutionContext, *name: str*) → int

Return the upper bound on an output tensor's size, in bytes, based on the current optimization profile.

If the profile or input shapes are not yet set, or the provided name does not map to an output, returns -1.

Parameters
name – The tensor name.

get_output_allocator(*self:* tensorrt.tensorrt.IExecutionContext, *name: str*) → *tensorrt.tensorrt.IOutputAllocator*

Return the output allocator associated with given output tensor, or `None` if the provided name does not map to an output tensor.

Parameters
name – The tensor name.

`get_runtime_config`(*self:* tensorrt.tensorrt.IExecutionContext) → *tensorrt.tensorrt.IRuntimeConfig*

Get the runtime configuration. From the execution context.

Returns
The runtime configuration.

`get_tensor_address`(*self:* tensorrt.tensorrt.IExecutionContext, *name: str*) → int

Get memory address for the given input or output tensor.

Parameters
name – The tensor name.

`get_tensor_shape`(*self:* tensorrt.tensorrt.IExecutionContext, *name: str*) → *tensorrt.tensorrt.Dims*

Return the shape of the given input or output tensor.

Parameters
name – The tensor name.

`get_tensor_strides`(*self:* tensorrt.tensorrt.IExecutionContext, *name: str*) → *tensorrt.tensorrt.Dims*

Return the strides of the buffer for the given tensor name.

Note that strides can be different for different execution contexts with dynamic shapes.

Parameters
name – The tensor name.

`infer_shapes`(*self:* tensorrt.tensorrt.IExecutionContext) → list[str]

Infer shapes and return the names of any tensors that are insufficiently specified.

An input tensor is insufficiently specified if either of the following is true:

- It has dynamic dimensions and its runtime dimensions have not yet been specified via *set_input_shape()* .
- is_shape_inference_io(t) is True and the tensor's address has not yet been set.

Returns
A `List[str]` indicating the names of any tensors which have not been sufficiently specified, or an empty list on success.

Raises
RuntimeError if shape inference fails due to reasons other than insufficiently specified tensors.

`report_to_profiler`(*self:* tensorrt.tensorrt.IExecutionContext) → bool

Calculate layer timing info for the current optimization profile in IExecutionContext and update the profiler after one iteration of inference launch.

If the enqueue_emits_profiler flag was set to true, the enqueue function will calculate layer timing implicitly if a profiler is provided. There is no need to call this function. If the enqueue_emits_profiler flag was set to false, the enqueue function will record the CUDA event timers if a profiler is provided. But it will not perform the layer timing calculation. This function needs to be called explicitly to calculate layer timing for the previous inference launch.

In the CUDA graph launch scenario, it will record the same set of CUDA events as in regular enqueue functions if the graph is captured from an *IExecutionContext* with profiler enabled. This function needs to be called after graph launch to report the layer timing info to the profiler.

Profiling CUDA graphs is only available from CUDA 11.1 onwards.

Returns
`True` if the call succeeded, else `False` (e.g. profiler not provided, in CUDA graph capture mode, etc.)

set_all_tensors_debug_state(*self:* tensorrt.tensorrt.IExecutionContext, *flag: bool*) → bool

Turn the debug state of all debug tensors on or off.

Parameters
flag – True if turning on debug state of tensor. False if turning off.

set_aux_streams(*self:* tensorrt.tensorrt.IExecutionContext, *aux_streams: collections.abc.Sequence[SupportsInt]*) → None

Set the auxiliary streams that TensorRT should launch kernels on in the next execute_async_v3() call.

If set, TensorRT will launch the kernels that are supposed to run on the auxiliary streams using the streams provided by the user with this API. If this API is not called before the execute_async_v3() call, then TensorRT will use the auxiliary streams created by TensorRT internally.

TensorRT will always insert event synchronizations between the main stream provided via execute_async_v3() call and the auxiliary streams:

- At the beginning of the execute_async_v3() call, TensorRT will make sure that all the auxiliary streams wait on the activities on the main stream.
- At the end of the execute_async_v3() call, TensorRT will make sure that the main stream wait on the activities on all the auxiliary streams.

The provided auxiliary streams must not be the default stream and must all be different to avoid deadlocks.

Parameters
aux_streams – A list of cuda streams. If the length of the list is greater than engine.num_aux_streams, then only the first "engine.num_aux_streams" streams will be used. If the length is less than engine.num_aux_streams, such as an empty list, then TensorRT will use the provided streams for the first few auxiliary streams, and will create additional streams internally for the rest of the auxiliary streams.

set_communicator(*self:* tensorrt.tensorrt.IExecutionContext, *communicator: typing_extensions.CapsuleType*) → bool

Set the NCCL communicator for the execution context.

Parameters
communicator – A pointer to the NCCL communicator that is used by the execution context.

The communicator must be uniform across all multi-device instances or undefined behavior occurs.

Returns
True if the communicator was set successfully, False otherwise.

set_debug_listener(*self:* tensorrt.tensorrt.IExecutionContext, *listener:* tensorrt.tensorrt.IDebugListener) → bool

Set debug listener for execution context.

Parameters
listener – The *IDebugListener*.

set_device_memory(*self:* tensorrt.tensorrt.IExecutionContext, *memory: SupportsInt*, *size: SupportsInt*) → None

The device memory for use by this *IExecutionContext* .

Parameters

- **memory** – 256-byte aligned device memory.
- **size** – Size of the provided memory. This must be at least as large as CudaEngine.get_device_memory_size_v2

If using `enqueue_v3()`, it is in use until `enqueue_v3()` returns. Releasing or otherwise using the memory for other purposes during this time will result in undefined behavior. This includes using the same memory for a parallel execution context.

set_input_consumed_event(*self:* tensorrt.tensorrt.IExecutionContext, *event: SupportsInt*) → bool

Mark all input tensors as consumed.

Parameters
event – The cuda event that is triggered after all input tensors have been consumed.

set_input_shape(**args*, ***kwargs*)

Overloaded function.

1. set_input_shape(self: tensorrt.tensorrt.IExecutionContext, name: str, shape: tuple) -> bool

   Set shape for the given input tensor.

   arg name
   The input tensor name.

   arg shape
   The input tensor shape.

2. set_input_shape(self: tensorrt.tensorrt.IExecutionContext, name: str, shape: list) -> bool

   Set shape for the given input tensor.

   arg name
   The input tensor name.

   arg shape
   The input tensor shape.

3. set_input_shape(self: tensorrt.tensorrt.IExecutionContext, name: str, shape: tensorrt.tensorrt.Dims) -> bool

   Set shape for the given input tensor.

   arg name
   The input tensor name.

   arg shape
   The input tensor shape.

set_optimization_profile_async(*self:* tensorrt.tensorrt.IExecutionContext, *profile_index: SupportsInt*, *stream_handle: SupportsInt*) → bool

Set the optimization profile with async semantics

Parameters

- **profile_index** – The index of the optimization profile

- **stream_handle** – cuda stream on which the work to switch optimization profile can be enqueued

When an optimization profile is switched via this API, TensorRT may require that data is copied via cudaMemcpyAsync. It is the application's responsibility to guarantee that synchronization between the profile sync stream and the enqueue stream occurs.

Returns
`True` if the optimization profile was set successfully

set_output_allocator(*self:* tensorrt.tensorrt.IExecutionContext, *name: str*, *output_allocator:* tensorrt.tensorrt.IOutputAllocator) → bool

Set output allocator to use for the given output tensor.

Pass `None` to unset the output allocator.

The allocator is called by *execute_async_v3()*.

Parameters

- **name** – The tensor name.
- **output_allocator** – The output allocator.

set_tensor_address(*self:* tensorrt.tensorrt.IExecutionContext, *name: str*, *memory: SupportsInt*) → bool

Set memory address for the given input or output tensor.

Parameters

- **name** – The tensor name.
- **memory** – The memory address.

set_tensor_debug_state(*self:* tensorrt.tensorrt.IExecutionContext, *name: str*, *flag: bool*) → bool

Turn the debug state of a tensor on or off. The Tensor must have been marked as a debug tensor during build time.

Parameters

- **name** – The name of the target tensor.
- **flag** – True if turning on debug state of tensor. False if turning off.

update_device_memory_size_for_shapes(*self:* tensorrt.tensorrt.IExecutionContext) → int

Recompute the internal activation buffer sizes based on the current input shapes, and return the total amount of memory required.

Users can allocate the device memory based on the size returned and provided the memory to TRT with an assignment to IExecutionContext.device_memory. Must specify all input shapes and the optimization profile to use before calling this function, otherwise the partition will be invalidated.

5.10 Runtime

tensorrt.**TempfileControlFlag**

Flags used to control TensorRT's behavior when creating executable temporary files.

On some platforms the TensorRT runtime may need to create files in a temporary directory or use platform-specific APIs to create files in-memory to load temporary DLLs that implement runtime code. These flags allow the application to explicitly control TensorRT's use of these files. This will preclude the use of certain TensorRT APIs for deserializing and loading lean runtimes.

These should be treated as bit offsets, e.g. in order to allow in-memory files for a given IRuntime:

```
runtime.tempfile_control_flags |= (1 << int(TempfileControlFlag.ALLOW_IN_MEMORY_
↪FILES))
```

Members:

ALLOW_IN_MEMORY_FILES : Allow creating and loading files in-memory (or unnamed files).

ALLOW_TEMPORARY_FILES : Allow creating and loading named files in a temporary directory on the filesystem.

class tensorrt.**Runtime**(*self:* tensorrt.tensorrt.Runtime, *logger:* tensorrt.tensorrt.ILogger)

Allows a serialized *ICudaEngine* to be deserialized.

Variables

- **error_recorder** – *IErrorRecorder* Application-implemented error reporting interface for TensorRT objects.
- **gpu_allocator** – *IGpuAllocator* The GPU allocator to be used by the *Runtime* . All GPU memory acquired will use this allocator. If set to None, the default allocator will be used (Default: cudaMalloc/cudaFree).
- **DLA_core** – int The DLA core that the engine executes on. Must be between 0 and N-1 where N is the number of available DLA cores.
- **num_DLA_cores** – int The number of DLA engines available to this builder.
- **logger** – *ILogger* The logger provided when creating the refitter.
- **max_threads** – int The maximum thread that can be used by the *Runtime*.
- **temporary_directory** – str The temporary directory to use when loading executable code for engines. If set to None (the default), TensorRT will attempt to find a suitable directory for use using platform-specific heuristics: - On UNIX/Linux platforms, TensorRT will first try the TMPDIR environment variable, then fall back to /tmp - On Windows, TensorRT will try the TEMP environment variable.
- **tempfile_control_flags** – int Flags which control whether TensorRT is allowed to create in-memory or temporary files. See *TempfileControlFlag* for details.
- **engine_host_code_allowed** – bool Whether this runtime is allowed to deserialize engines that contain host executable code (Default: False).

Parameters

logger – The logger to use.

__del__(*self:* tensorrt.tensorrt.Runtime) → None

__exit__(*exc_type*, *exc_value*, *traceback*)

Context managers are deprecated and have no effect. Objects are automatically freed when the reference count reaches 0.

__init__(*self:* tensorrt.tensorrt.Runtime, *logger:* tensorrt.tensorrt.ILogger) → None

Parameters

logger – The logger to use.

deserialize_cuda_engine(**args*, ***kwargs*)

Overloaded function.

1. deserialize_cuda_engine(self: tensorrt.tensorrt.Runtime, serialized_engine: collections.abc.Buffer) -> tensorrt.tensorrt.ICudaEngine

   Deserialize an *ICudaEngine* from host memory.

   arg serialized_engine
   The `buffer` that holds the serialized *ICudaEngine*.

   returns
   The *ICudaEngine*, or None if it could not be deserialized.

2. deserialize_cuda_engine(self: tensorrt.tensorrt.Runtime, stream_reader_v2: tensorrt.tensorrt.IStreamReaderV2) -> tensorrt.tensorrt.ICudaEngine

   Deserialize an *ICudaEngine* from a stream reader v2.

   arg stream_reader
   The `PyStreamReaderV2` that will read the serialized *ICudaEngine*. This enables deserialization from a file directly, with possible benefits to performance.

   returns
   The *ICudaEngine*, or None if it could not be deserialized.

get_plugin_registry(*self:* tensorrt.tensorrt.Runtime) → *tensorrt.tensorrt.IPluginRegistry*

Get the local plugin registry that can be used by the runtime.

Returns
The local plugin registry that can be used by the runtime.

load_runtime(*self:* tensorrt.tensorrt.Runtime, *path: str*) → *tensorrt.tensorrt.Runtime*

Load IRuntime from the file.

This method loads a runtime library from a shared library file. The runtime can then be used to execute a plan file built with BuilderFlag.VERSION_COMPATIBLE and BuilderFlag.EXCLUDE_LEAN_RUNTIME both set and built with the same version of TensorRT as the loaded runtime library.

Variables
path – Path to the runtime lean library.

Returns
The `IRuntime`, or None if it could not be loaded.

5.11 IRuntimeConfig

tensorrt.**ExecutionContextAllocationStrategy**

Different memory allocation behaviors for IExecutionContext.

Members:

STATIC : Default static allocation with the maximum size across all profiles.

ON_PROFILE_CHANGE : Reallocate for a profile when it's selected.

USER_MANAGED : The user supplies custom allocation to the execution context.

class tensorrt.**IRuntimeConfig**

A runtime configuration for an *ICudaEngine* .

__init__(**args*, ***kwargs*)

get_execution_context_allocation_strategy(*self:* tensorrt.tensorrt.IRuntimeConfig) → tensorrt.tensorrt.ExecutionContextAllocationStrategy

Get the execution context allocation strategy.

Returns
The execution context allocation strategy.

set_execution_context_allocation_strategy(*self: tensorrt.tensorrt.IRuntimeConfig*, *strategy: tensorrt.tensorrt.ExecutionContextAllocationStrategy = <ExecutionContextAllocationStrategy.STATIC: 0>*) → None

Set the execution context allocation strategy.

Parameters
strategy – The execution context allocation strategy.

5.12 Refitter

class tensorrt.**Refitter**(*self:* tensorrt.tensorrt.Refitter, *engine:* tensorrt.tensorrt.ICudaEngine, *logger:* tensorrt.tensorrt.ILogger)

Updates weights in an *ICudaEngine* .

Variables

- **error_recorder** – *IErrorRecorder* Application-implemented error reporting interface for TensorRT objects.
- **logger** – *ILogger* The logger provided when creating the refitter.
- **max_threads** – int The maximum thread that can be used by the *Refitter*.
- **weights_validation** – bool The flag to indicate whether to validate weights in the refitting process.

Parameters

- **engine** – The engine to refit.
- **logger** – The logger to use.

__del__(*self:* tensorrt.tensorrt.Refitter) → None

__init__(*self:* tensorrt.tensorrt.Refitter, *engine:* tensorrt.tensorrt.ICudaEngine, *logger:* tensorrt.tensorrt.ILogger) → None

Parameters

- **engine** – The engine to refit.
- **logger** – The logger to use.

get_all(*self:* tensorrt.tensorrt.Refitter) → tuple[list[str], list[tensorrt.tensorrt.WeightsRole]]

Get description of all weights that could be refitted.

Returns
The names of layers with refittable weights, and the roles of those weights.

get_all_weights(*self:* tensorrt.tensorrt.Refitter) → list[str]

Get names of all weights that could be refitted.

Returns
The names of refittable weights.

get_missing(*self:* tensorrt.tensorrt.Refitter) → tuple[list[str], list[tensorrt.tensorrt.WeightsRole]]

Get description of missing weights.

For example, if some Weights have been set, but the engine was optimized in a way that combines weights, any unsupplied Weights in the combination are considered missing.

Returns
The names of layers with missing weights, and the roles of those weights.

get_missing_weights(*self:* tensorrt.tensorrt.Refitter) → list[str]

Get names of missing weights.

For example, if some Weights have been set, but the engine was optimized in a way that combines weights, any unsupplied Weights in the combination are considered missing.

Returns
The names of missing weights, empty string for unnamed weights.

get_named_weights(*self:* tensorrt.tensorrt.Refitter, *weights_name: str*) → *tensorrt.tensorrt.Weights*

Get weights associated with the given name.

If the weights were never set, returns null weights and reports an error to the refitter errorRecorder.

Parameters
weights_name – The name of the weights to be refitted.

Returns
Weights associated with the given name.

get_weights_location(*self:* tensorrt.tensorrt.Refitter, *weights_name: str*) → tensorrt.tensorrt.TensorLocation

Get location for the weights associated with the given name.

If the weights were never set, returns TensorLocation.HOST and reports an error to the refitter errorRecorder.

Parameters
weights_name – The name of the weights to be refitted.

Returns
Location for the weights associated with the given name.

get_weights_prototype(*self:* tensorrt.tensorrt.Refitter, *weights_name: str*) → *tensorrt.tensorrt.Weights*

Get the weights prototype associated with the given name.

The dtype and size of weights prototype is the same as weights used for engine building. The size of the weights prototype is -1 when the name of the weights is None or does not correspond to any refittable weights.

Parameters
weights_name – The name of the weights to be refitted.

Returns
weights prototype associated with the given name.

refit_cuda_engine(*self:* tensorrt.tensorrt.Refitter) → bool

Refits associated engine.

If `False` is returned, a subset of weights may have been refitted.

The behavior is undefined if the engine has pending enqueued work. Provided weights on CPU or GPU can be unset and released, or updated after refit_cuda_engine returns.

IExecutionContexts associated with the engine remain valid for use afterwards. There is no need to set the same weights repeatedly for multiple refit calls as the weights memory can be updated directly instead.

Returns
`True` on success, or `False` if new weights validation fails or get_missing_weights() != 0 before the call.

refit_cuda_engine_async(*self:* tensorrt.tensorrt.Refitter, *stream_handle: SupportsInt*) → bool

Enqueue weights refitting of the associated engine on the given stream.

If `False` is returned, a subset of weights may have been refitted.

The behavior is undefined if the engine has pending enqueued work on a different stream from the provided one. Provided weights on CPU can be unset and released, or updated after refit_cuda_engine_async returns. Freeing or updating of the provided weights on GPU can be enqueued on the same stream after refit_cuda_engine_async returns.

IExecutionContexts associated with the engine remain valid for use afterwards. There is no need to set the same weights repeatedly for multiple refit calls as the weights memory can be updated directly instead. The weights updating task should use the the same stream as the one used for the refit call.

Parameters
stream – The stream to enqueue the weights updating task.

Returns
`True` on success, or `False` if new weights validation fails or get_missing_weights() != 0 before the call.

set_named_weights(**args*, ***kwargs*)

Overloaded function.

1. set_named_weights(self: tensorrt.tensorrt.Refitter, name: str, weights: tensorrt.tensorrt.Weights) -> bool

   Specify new weights of given name. Possible reasons for rejection are:

   - The name of weights is empty or does not correspond to any refittable weights.
   - The size of the weights is inconsistent with the size returned from calling *get_weights_prototype()* with the same name.
   - The dtype of the weights is inconsistent with the dtype returned from calling *get_weights_prototype()* with the same name.

   Modifying the weights before *refit_cuda_engine()* or *refit_cuda_engine_async()* returns will result in undefined behavior.

   arg name
   The name of the weights to be refitted.

   arg weights
   The new weights to associate with the name.

   returns
   `True` on success, or `False` if new weights are rejected.

2. set_named_weights(self: tensorrt.tensorrt.Refitter, name: str, weights: tensorrt.tensorrt.Weights, location: tensorrt.tensorrt.TensorLocation) -> bool

   Specify new weights on a specified device of given name. Possible reasons for rejection are:

   - The name of weights is empty or does not correspond to any refittable weights.
   - The size of the weights is inconsistent with the size returned from calling *get_weights_prototype()* with the same name.
   - The dtype of the weights is inconsistent with the dtype returned from calling *get_weights_prototype()* with the same name.

   It is allowed to provide some weights on CPU and others on GPU. Modifying the weights before *refit_cuda_engine()* or *refit_cuda_engine_async()* returns will result in undefined behavior.

   arg name
   The name of the weights to be refitted.

   arg weights
   The new weights on the specified device.

   arg location
   The location (host vs. device) of the new weights.

   returns
   `True` on success, or `False` if new weights are rejected.

set_weights(*self:* tensorrt.tensorrt.Refitter, *layer_name: str*, *role: tensorrt.tensorrt.WeightsRole*, *weights:* tensorrt.tensorrt.Weights) → bool

Specify new weights for a layer of given name. Possible reasons for rejection are:

- There is no such layer by that name.
- The layer does not have weights with the specified role.
- The size of weights is inconsistent with the layer's original specification.

Modifying the weights before *refit_cuda_engine()* or *refit_cuda_engine_async()* returns will result in undefined behavior.

Parameters

- **layer_name** – The name of the layer.
- **role** – The role of the weights. See *WeightsRole* for more information.
- **weights** – The weights to refit with.

Returns
`True` on success, or `False` if new weights are rejected.

unset_named_weights(*self:* tensorrt.tensorrt.Refitter, *weights_name: str*) → bool

Unset weights associated with the given name.

Unset weights before releasing them.

Parameters
weights_name – The name of the weights to be refitted.

Returns
`False` if the weights were never set, returns `True` otherwise.

5.13 IErrorRecorder

tensorrt.**ErrorCodeTRT**

Error codes that can be returned by TensorRT during execution.

Members:

SUCCESS : Execution completed successfully.

UNSPECIFIED_ERROR :
An error that does not fall into any other category. This error is included for forward compatibility.

INTERNAL_ERROR : A non-recoverable TensorRT error occurred.

INVALID_ARGUMENT :
An argument passed to the function is invalid in isolation. This is a violation of the API contract.

INVALID_CONFIG :
An error occurred when comparing the state of an argument relative to other arguments. For example, the dimensions for concat differ between two tensors outside of the channel dimension. This error is triggered when an argument is correct in isolation, but not relative to other arguments. This is to help to distinguish from the simple errors from the more complex errors. This is a violation of the API contract.

FAILED_ALLOCATION :
An error occurred when performing an allocation of memory on the host or the device. A memory allocation error is normally fatal, but in the case where the application provided its own memory allocation routine, it is possible to increase the pool of available memory and resume execution.

FAILED_INITIALIZATION :
One, or more, of the components that TensorRT relies on did not initialize correctly. This is a system setup issue.

FAILED_EXECUTION :
An error occurred during execution that caused TensorRT to end prematurely, either an asynchronous error, user cancellation, or other execution errors reported by CUDA/DLA. In a dynamic system, the data can be thrown away and the next frame can be processed or execution can be retried. This is either an execution error or a memory error.

FAILED_COMPUTATION :
An error occurred during execution that caused the data to become corrupted, but execution finished. Examples of this error are NaN squashing or integer overflow. In a dynamic system, the data can be thrown away and the next frame can be processed or execution can be retried. This is either a data corruption error, an input error, or a range error.

INVALID_STATE :
TensorRT was put into a bad state by incorrect sequence of function calls. An example of an invalid state is specifying a layer to be DLA only without GPU fallback, and that layer is not supported by DLA. This can occur in situations where a service is optimistically executing networks for multiple different configurations without checking proper error configurations, and instead throwing away bad configurations caught by TensorRT. This is a violation of the API contract, but can be recoverable.

Example of a recovery: GPU fallback is disabled and conv layer with large filter(63x63) is specified to run on DLA. This will fail due to DLA not supporting the large kernel size. This can be recovered by either turning on GPU fallback or setting the layer to run on the GPU.

UNSUPPORTED_STATE :
An error occurred due to the network not being supported on the device due to constraints of

the hardware or system. An example is running a unsafe layer in a safety certified context, or a resource requirement for the current network is greater than the capabilities of the target device. The network is otherwise correct, but the network and hardware combination is problematic. This can be recoverable. Examples: * Scratch space requests larger than available device memory and can be recovered by increasing allowed workspace size. * Tensor size exceeds the maximum element count and can be recovered by reducing the maximum batch size.

class tensorrt.**IErrorRecorder**(*self:* tensorrt.tensorrt.IErrorRecorder)

Reference counted application-implemented error reporting interface for TensorRT objects.

The error reporting mechanism is a user defined object that interacts with the internal state of the object that it is assigned to in order to determine information about abnormalities in execution. The error recorder gets both an error enum that is more descriptive than pass/fail and also a description that gives more detail on the exact failure modes. In the safety context, the error strings are all limited to 128 characters in length. The ErrorRecorder gets passed along to any class that is created from another class that has an ErrorRecorder assigned to it. For example, assigning an ErrorRecorder to an Builder allows all INetwork's, ILayer's, and ITensor's to use the same error recorder. For functions that have their own ErrorRecorder accessor functions. This allows registering a different error recorder or de-registering of the error recorder for that specific object.

The ErrorRecorder object implementation must be thread safe if the same ErrorRecorder is passed to different interface objects being executed in parallel in different threads. All locking and synchronization is pushed to the interface implementation and TensorRT does not hold any synchronization primitives when accessing the interface functions.

clear(*self:* tensorrt.tensorrt.IErrorRecorder) → None

Clear the error stack on the error recorder.

Removes all the tracked errors by the error recorder. This function must guarantee that after this function is called, and as long as no error occurs, *num_errors* will be zero.

get_error_code(*self:* tensorrt.tensorrt.IErrorRecorder, *arg0: SupportsInt*) → tensorrt.tensorrt.ErrorCodeTRT

Returns the ErrorCode enumeration.

The error_idx specifies what error code from 0 to *num_errors*-1 that the application wants to analyze and return the error code enum.

Parameters
error_idx – A 32bit integer that indexes into the error array.

Returns
Returns the enum corresponding to error_idx.

get_error_desc(*self:* tensorrt.tensorrt.IErrorRecorder, *arg0: SupportsInt*) → str

Returns description of the error.

For the error specified by the idx value, return description of the error. In the safety context there is a constant length requirement to remove any dynamic memory allocations and the error message may be truncated. The format of the error description is "<EnumAsStr> - <Description>".

Parameters
error_idx – A 32bit integer that indexes into the error array.

Returns
Returns description of the error.

has_overflowed(*self:* tensorrt.tensorrt.IErrorRecorder) → bool

Determine if the error stack has overflowed.

In the case when the number of errors is large, this function is used to query if one or more errors have been dropped due to lack of storage capacity. This is especially important in the automotive safety case where the internal error handling mechanisms cannot allocate memory.

Returns
True if errors have been dropped due to overflowing the error stack.

num_errors(*self:* tensorrt.tensorrt.IErrorRecorder) → int

Return the number of errors

Determines the number of errors that occurred between the current point in execution and the last time that the clear() was executed. Due to the possibility of asynchronous errors occuring, a TensorRT API can return correct results, but still register errors with the Error Recorder. The value of getNbErrors must monotonically increases until clear() is called.

Returns
Returns the number of errors detected, or 0 if there are no errors.

report_error(*self:* tensorrt.tensorrt.IErrorRecorder, *arg0: tensorrt.tensorrt.ErrorCodeTRT*, *arg1: str*) → bool

Clear the error stack on the error recorder.

Report an error to the user that has a given value and human readable description. The function returns false if processing can continue, which implies that the reported error is not fatal. This does not guarantee that processing continues, but provides a hint to TensorRT.

Parameters

- **val** – The error code enum that is being reported.
- **desc** – The description of the error.

Returns
True if the error is determined to be fatal and processing of the current function must end.

5.14 ITimingCache

class tensorrt.**ITimingCache**

Class to handle tactic timing info collected from builder.

combine(*self:* tensorrt.tensorrt.ITimingCache, *input_cache:* tensorrt.tensorrt.ITimingCache, *ignore_mismatch: bool*) → bool

Combine input timing cache into local instance.

Append entries in input cache to local cache. Conflicting entries will be skipped. The input cache must be generated by a TensorRT build of exact same version, otherwise combine will be skipped and return false. `bool(ignore_mismatch) == True` if combining a timing cache created from a different device.

Parameters

- **input_cache** – The input timing cache
- **ignore_mismatch** – Whether or not to allow cache verification header mismatch

Returns
A *bool* indicating whether the combine operation is done successfully.

query(*self:* tensorrt.tensorrt.ITimingCache, *key: tensorrt.tensorrt.TimingCacheKey*) → tensorrt.tensorrt.TimingCacheValue

Query value in a cache entry.

If the key exists, write the value out, otherwise return an invalid value.

Parameters
key – The query key.

Cache cache
value if the key exists, otherwise an invalid value.

:returns A `TimingCacheValue` object.

queryKeys(*self:* tensorrt.tensorrt.ITimingCache) → list[tensorrt.tensorrt.TimingCacheKey]

Query cache keys from Timing Cache.

If an error occurs, a RuntimeError will be raised.

:returns A list containing the cache keys.

reset(*self:* tensorrt.tensorrt.ITimingCache) → bool

Empty the timing cache

Returns
A *bool* indicating whether the reset operation is done successfully.

serialize(*self:* tensorrt.tensorrt.ITimingCache) → *tensorrt.tensorrt.IHostMemory*

Serialize a timing cache to a *IHostMemory* object.

Returns
An *IHostMemory* object that contains a serialized timing cache.

update(*self:* tensorrt.tensorrt.ITimingCache, *key: tensorrt.tensorrt.TimingCacheKey*, *value: tensorrt.tensorrt.TimingCacheValue*) → bool

Update values in a cache entry.

Update the value of the given cache key. If the key does not exist, return False. If the key exists and the new tactic timing is NaN, delete the cache entry and return True. If tactic timing is not NaN and the new value is valid, override the cache value and return True. False is returned when the new value is invalid. If this layer cannot use the new tactic, build errors will be reported when building the next engine.

Parameters

- **key** – The key to the entry to be updated.
- **value** – New cache value.

:returns True if update succeeds, otherwise False.

5.15 GPU Allocator

5.15.1 AllocatorFlag

tensorrt.**AllocatorFlag**

Members:

RESIZABLE : TensorRT may call realloc() on this allocation

5.15.2 IGpuAllocator

class tensorrt.**IGpuAllocator**(*self:* tensorrt.tensorrt.IGpuAllocator)

Application-implemented class for controlling allocation on the GPU.

To implement a custom allocator, ensure that you explicitly instantiate the base class in *__init__()* :

```
class MyAllocator(trt.IGpuAllocator):
    def __init__(self):
        trt.IGpuAllocator.__init__(self)

    ...
```

Note that all methods below (allocate, reallocate, deallocate, allocate_async, deallocate_async) must be overridden in the custom allocator, or else pybind11 would not be able to call the method from a custom allocator.

__init__(*self:* tensorrt.tensorrt.IGpuAllocator) → None

allocate(*self:* tensorrt.tensorrt.IGpuAllocator, *size: SupportsInt*, *alignment: SupportsInt*, *flags: SupportsInt*) → typing_extensions.CapsuleType

[DEPRECATED] Deprecated in TensorRT 10.0. Please use allocate_async instead. A callback implemented by the application to handle acquisition of GPU memory. If an allocation request of size 0 is made, `None` should be returned.

If an allocation request cannot be satisfied, `None` should be returned.

Parameters

- **size** – The size of the memory required.
- **alignment** – The required alignment of memory. Alignment will be zero or a power of 2 not exceeding the alignment guaranteed by cudaMalloc. Thus this allocator can be safely implemented with cudaMalloc/cudaFree. An alignment value of zero indicates any alignment is acceptable.
- **flags** – Allocation flags. See *AllocatorFlag*

Returns
The address of the allocated memory

allocate_async(*self:* tensorrt.tensorrt.IGpuAllocator, *size: SupportsInt*, *alignment: SupportsInt*, *flags: SupportsInt*, *stream: SupportsInt*) → typing_extensions.CapsuleType

A callback implemented by the application to handle acquisition of GPU memory asynchronously. This is just a wrapper around a syncronous method allocate. For the asynchronous allocation please use the corresponding IGpuAsyncAllocator class. If an allocation request of size 0 is made, `None` should be returned.

If an allocation request cannot be satisfied, `None` should be returned.

Parameters

- **size** – The size of the memory required.
- **alignment** – The required alignment of memory. Alignment will be zero or a power of 2 not exceeding the alignment guaranteed by cudaMalloc. Thus this allocator can be safely implemented with cudaMalloc/cudaFree. An alignment value of zero indicates any alignment is acceptable.
- **flags** – Allocation flags. See *AllocatorFlag*
- **stream** – CUDA stream

Returns
The address of the allocated memory

deallocate(*self:* tensorrt.tensorrt.IGpuAllocator, *memory: typing_extensions.CapsuleType*) → bool

[DEPRECATED] Deprecated in TensorRT 10.0. Please use dealocate_async instead; A callback implemented by the application to handle release of GPU memory.

TensorRT may pass a 0 to this function if it was previously returned by `allocate()`.

Parameters
memory – The memory address of the memory to release.

Returns
True if the acquired memory is released successfully.

deallocate_async(*self:* tensorrt.tensorrt.IGpuAllocator, *memory: typing_extensions.CapsuleType*, *stream: SupportsInt*) → bool

A callback implemented by the application to handle release of GPU memory asynchronously. This is just a wrapper around a syncronous method deallocate. For the asynchronous deallocation please use the corresponding IGpuAsyncAllocator class.

TensorRT may pass a 0 to this function if it was previously returned by `allocate()`.

Parameters

- **memory** – The memory address of the memory to release.
- **stream** – CUDA stream

Returns
True if the acquired memory is released successfully.

reallocate(*self:* tensorrt.tensorrt.IGpuAllocator, *address: typing_extensions.CapsuleType*, *alignment: SupportsInt*, *new_size: SupportsInt*) → typing_extensions.CapsuleType

A callback implemented by the application to resize an existing allocation.

Only allocations which were allocated with AllocatorFlag.RESIZABLE will be resized.

Options are one of: - resize in place leaving min(old_size, new_size) bytes unchanged and return the original address - move min(old_size, new_size) bytes to a new location of sufficient size and return its address - return None, to indicate that the request could not be fulfilled.

If None is returned, TensorRT will assume that resize() is not implemented, and that the allocation at address is still valid.

This method is made available for use cases where delegating the resize strategy to the application provides an opportunity to improve memory management. One possible implementation is to allocate a large virtual device buffer and progressively commit physical memory with cuMemMap. CU_MEM_ALLOC_GRANULARITY_RECOMMENDED is suggested in this case.

TensorRT may call realloc to increase the buffer by relatively small amounts.

Parameters

- **address** – the address of the original allocation.
- **alignment** – The alignment used by the original allocation.
- **new_size** – The new memory size required.

Returns
The address of the reallocated memory

5.16 EngineInspector

class tensorrt.**EngineInspector**

An engine inspector which prints out the layer information of an engine or an execution context.

The amount of printed information depends on the profiling verbosity setting of the builder config when the engine is built. By default, the profiling verbosity is set to ProfilingVerbosity.LAYER_NAMES_ONLY, and only layer names will be printed. If the profiling verbosity is set to ProfilingVerbosity.DETAILED, layer names and layer parameters will be printed. If the profiling verbosity is set to ProfilingVerbosity.NONE, no layer information will be printed.

Variables

- **execution_context** – *IExecutionContext* Set or get context currently being inspected.
- **error_recorder** – *IErrorRecorder* Application-implemented error reporting interface for TensorRT objects.

__init__(**args*, ***kwargs*)

get_engine_information(*self:* tensorrt.tensorrt.EngineInspector, *format:* *tensorrt.tensorrt.LayerInformationFormat*) → str

Get a string describing the information about all the layers in the current engine or the execution context.

Parameters

format – LayerInformationFormat The format the layer information should be printed in.

Returns

A string describing the information about all the layers in the current engine or the execution context.

get_layer_information(*self:* tensorrt.tensorrt.EngineInspector, *layer_index: SupportsInt*, *format:* *tensorrt.tensorrt.LayerInformationFormat*) → str

Get a string describing the information about a specific layer in the current engine or the execution context.

Parameters

- **layer_index** – The index of the layer. It must lie in [0, engine.num_layers].
- **format** – LayerInformationFormat The format the layer information should be printed in.

Returns

A string describing the information about a specific layer in the current engine or the execution context.

5.17 ISerializationConfig

tensorrt.**SerializationFlag**

Valid flags that can be use to creating binary file from engine.

Members:

EXCLUDE_WEIGHTS : Exclude weights that can be refitted.

EXCLUDE_LEAN_RUNTIME : Exclude lean runtime from the plan.

INCLUDE_REFIT : Remain refittable if originally so.

class tensorrt.**ISerializationConfig**

Class to hold properties for configuring an engine to serialize the binary.

__init__(**args*, ***kwargs*)

clear_flag(*self:* tensorrt.tensorrt.ISerializationConfig, *flag: tensorrt.tensorrt.SerializationFlag*) → bool

Clears the serialization flag from the config.

get_flag(*self:* tensorrt.tensorrt.ISerializationConfig, *flag: tensorrt.tensorrt.SerializationFlag*) → bool

Check if a serialization flag is set.

set_flag(*self:* tensorrt.tensorrt.ISerializationConfig, *flag: tensorrt.tensorrt.SerializationFlag*) → bool

Add the input serialization flag to the already enabled flags.

CHAPTER

SIX

NETWORK

6.1 INetworkDefinition

class tensorrt.**INetworkDefinition**

Represents a TensorRT Network from which the Builder can build an Engine

Variables

- **num_layers** – int The number of layers in the network.
- **num_inputs** – int The number of inputs of the network.
- **num_outputs** – int The number of outputs of the network.
- **name** – str The name of the network. This is used so that it can be associated with a built engine. The name must be at most 128 characters in length. TensorRT makes no use of this string except storing it as part of the engine so that it may be retrieved at runtime. A name unique to the builder will be generated by default.
- **has_implicit_batch_dimension** – bool [DEPRECATED] Deprecated in TensorRT 10.0. Always flase since the implicit batch dimensions support has been removed.
- **error_recorder** – *IErrorRecorder* Application-implemented error reporting interface for TensorRT objects.

Flags

int
A bitset of the NetworkDefinitionCreationFlag s set for this network.

__del__(*self:* tensorrt.tensorrt.INetworkDefinition) → None

__exit__(*exc_type*, *exc_value*, *traceback*)

Context managers are deprecated and have no effect. Objects are automatically freed when the reference count reaches 0.

__getitem__(*self:* tensorrt.tensorrt.INetworkDefinition, *arg0: SupportsInt*) → *tensorrt.tensorrt.ILayer*

__init__(**args*, ***kwargs*)

__len__(*self:* tensorrt.tensorrt.INetworkDefinition) → int

add_activation(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *type: tensorrt.tensorrt.ActivationType*) → *tensorrt.tensorrt.IActivationLayer*

Add an activation layer to the network. See *IActivationLayer* for more information.

Parameters

- **input** – The input tensor to the layer.
- **type** – The type of activation function to apply.

Returns
The new activation layer, or None if it could not be created.

add_assertion(*self:* tensorrt.tensorrt.INetworkDefinition, *condition:* tensorrt.tensorrt.ITensor, *message:* *str*) → *tensorrt.tensorrt.IAssertionLayer*

Add a assertion layer. See *IAssertionLayer* for more information.

Parameters

- **condition** – The condition tensor to the layer.
- **message** – The message to print if the assertion fails.

Returns
The new assertion layer, or None if it could not be created.

add_attention(*self:* tensorrt.tensorrt.INetworkDefinition, *query:* tensorrt.tensorrt.ITensor, *key:* tensorrt.tensorrt.ITensor, *value:* tensorrt.tensorrt.ITensor, *norm_op:* *tensorrt.tensorrt.AttentionNormalizationOp*, *causal:* *bool*) → *tensorrt.tensorrt.IAttention*

Add an attention to the network. Deprecated, use add_attention_v2 instead. See *IAttention* for more information.

Parameters

- **query** – The 4d query input tensor to the attention.
- **key** – The 4d key input tensor to the attention.
- **value** – The 4d value input tensor to the attention.
- **normOp** – The normalization operation to perform.
- **causal** – The boolean that specifies whether an attention will run causal inference.

Returns
The new Attention, or None if it could not be created.

add_attention_v2(*self:* tensorrt.tensorrt.INetworkDefinition, *query:* tensorrt.tensorrt.ITensor, *key:* tensorrt.tensorrt.ITensor, *value:* tensorrt.tensorrt.ITensor, *norm_op:* *tensorrt.tensorrt.AttentionNormalizationOp*, *causal_kind:* *tensorrt.tensorrt.CausalMaskKind*) → *tensorrt.tensorrt.IAttention*

Add an attention to the network. See *IAttention* for more information.

Parameters

- **query** – The 4d query input tensor to the attention.
- **key** – The 4d key input tensor to the attention.
- **value** – The 4d value input tensor to the attention.
- **normOp** – The normalization operation to perform.
- **causal_kind** – The CausalMaskKind that specifies the causal mask alignment orientation.

Returns
The new Attention, or None if it could not be created.

add_cast(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *to_type:* *tensorrt.tensorrt.DataType*) → *tensorrt.tensorrt.ICastLayer*

Add a cast layer. See `ICastLayer` for more information.

Parameters

- **input** – The input tensor to the layer.
- **to_type** – The data type the output tensor should be cast into.

Returns
The new cast layer, or `None` if it could not be created.

add_concatenation(*self:* tensorrt.tensorrt.INetworkDefinition, *inputs:* *collections.abc.Sequence[*tensorrt.tensorrt.ITensor*]*) → *tensorrt.tensorrt.IConcatenationLayer*

Add a concatenation layer to the network. Note that all tensors must have the same dimension except for the Channel dimension. See `IConcatenationLayer` for more information.

Parameters
inputs – The input tensors to the layer.

Returns
The new concatenation layer, or `None` if it could not be created.

add_constant(*self:* tensorrt.tensorrt.INetworkDefinition, *shape:* tensorrt.tensorrt.Dims, *weights:* tensorrt.tensorrt.Weights) → *tensorrt.tensorrt.IConstantLayer*

Add a constant layer to the network. See `IConstantLayer` for more information.

Parameters

- **shape** – The shape of the constant.
- **weights** – The constant value, represented as weights.

Returns
The new constant layer, or `None` if it could not be created.

add_convolution_nd(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *num_output_maps: SupportsInt*, *kernel_shape:* tensorrt.tensorrt.Dims, *kernel:* tensorrt.tensorrt.Weights, *bias:* tensorrt.tensorrt.Weights = *None*) → *tensorrt.tensorrt.IConvolutionLayer*

Add a multi-dimension convolution layer to the network. See `IConvolutionLayer` for more information.

Parameters

- **input** – The input tensor to the convolution.
- **num_output_maps** – The number of output feature maps for the convolution.
- **kernel_shape** – The dimensions of the convolution kernel.
- **kernel** – The kernel weights for the convolution.
- **bias** – The optional bias weights for the convolution.

Returns
The new convolution layer, or `None` if it could not be created.

add_cumulative(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *axis:* tensorrt.tensorrt.ITensor, *op: tensorrt.tensorrt.CumulativeOperation*, *exclusive: bool*, *reverse: bool*) → *tensorrt.tensorrt.ICumulativeLayer*

Add a cumulative layer to the network. See `ICumulativeLayer` for more information.

Parameters

- **input** – The input tensor to the layer.
- **axis** – The axis tensor to apply the cumulative operation on. Currently, it must be a build-time constant 0-D shape tensor.
- **op** – The reduction operation to perform.
- **exclusive** – The boolean that specifies whether it is an exclusive cumulative or inclusive cumulative.
- **reverse** – The boolean that specifies whether the cumulative should be applied backward.

Returns
The new cumulative layer, or `None` if it could not be created.

add_deconvolution_nd(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *num_output_maps: SupportsInt*, *kernel_shape:* tensorrt.tensorrt.Dims, *kernel:* tensorrt.tensorrt.Weights, *bias:* tensorrt.tensorrt.Weights = *None*) → *tensorrt.tensorrt.IDeconvolutionLayer*

Add a multi-dimension deconvolution layer to the network. See *IDeconvolutionLayer* for more information.

Parameters

- **input** – The input tensor to the layer.
- **num_output_maps** – The number of output feature maps.
- **kernel_shape** – The dimensions of the convolution kernel.
- **kernel** – The kernel weights for the convolution.
- **bias** – The optional bias weights for the convolution.

Returns
The new deconvolution layer, or `None` if it could not be created.

add_dequantize(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *scale:* tensorrt.tensorrt.ITensor, *output_type: tensorrt.tensorrt.DataType*) → *tensorrt.tensorrt.IDequantizeLayer*

Add a dequantization layer to the network. See *IDequantizeLayer* for more information.

Parameters

- **input** – A tensor to dequantize.
- **scale** – A tensor with the scale coefficients.
- **output_type** – The datatype of the output tensor.

Returns
The new dequantization layer, or `None` if it could not be created.

add_dist_collective(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *dist_collective_op: tensorrt.tensorrt.CollectiveOperation*, *reduce_op: object*, *root: object*, *groups: object*) → *tensorrt.tensorrt.IDistCollectiveLayer*

Add a dist collective layer to the network. See *IDistCollectiveLayer* for more information.

Parameters

- **input** – The input tensor to the layer.
- **dist_collective_op** – The collective operation to perform.

- **reduce_op** – The reduction operation to perform when `dist_collective_op` is `CollectiveOperation.ALL_REDUCE`, or `CollectiveOperation.REDUCE`, or `CollectiveOperation.REDUCE_SCATTER`.
- **root** – The root rank of the collective operation.
- **group_size** – The size of the groups array.
- **groups** – The groups to perform the collective operation on.

Returns
The new dist collective layer, or `None` if it could not be created.

add_dynamic_quantize(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *axis: SupportsInt*, *block_size: SupportsInt*, *output_type: tensorrt.tensorrt.DataType*, *scale_type: tensorrt.tensorrt.DataType*) → *tensorrt.tensorrt.IDynamicQuantizeLayer*

Add a dynamic quantization layer to the network. See *IDynamicQuantizeLayer* for more information.

Parameters

- **input** – A tensor to quantize.
- **axis** – The axis that is sliced into blocks.
- **block_size** – The number of elements that are quantized using a shared scale factor.
- **output_type** – The data type of the quantized output tensor.
- **scale_type** – The data type of the scale factor used for quantizing the input data.

Returns
The new DynamicQuantization layer, or `None` if it could not be created.

add_dynamic_quantize_v2(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *block_shape:* tensorrt.tensorrt.Dims, *output_type: tensorrt.tensorrt.DataType*, *scale_type: tensorrt.tensorrt.DataType*) → *tensorrt.tensorrt.IDynamicQuantizeLayer*

Add a dynamic quantization layer to the network. See *IDynamicQuantizeLayer* for more information.

Parameters

- **input** – A tensor to quantize.
- **block_shape** – The shape of the block.
- **output_type** – The data type of the quantized output tensor.
- **scale_type** – The data type of the scale factor used for quantizing the input data.

Returns
The new DynamicQuantization layer, or `None` if it could not be created.

add_einsum(*self:* tensorrt.tensorrt.INetworkDefinition, *inputs: collections.abc.Sequence[*tensorrt.tensorrt.ITensor*]*, *equation: str*) → *tensorrt.tensorrt.IEinsumLayer*

Adds an Einsum layer to the network. See *IEinsumLayer* for more information.

Parameters

- **inputs** – The input tensors to the layer.
- **equation** – The Einsum equation of the layer.

Returns
the new Einsum layer, or `None` if it could not be created.

`add_elementwise`(*self:* tensorrt.tensorrt.INetworkDefinition, *input1:* tensorrt.tensorrt.ITensor, *input2:* tensorrt.tensorrt.ITensor, *op: tensorrt.tensorrt.ElementWiseOperation*) → *tensorrt.tensorrt.IElementWiseLayer*

Add an elementwise layer to the network. See `IElementWiseLayer` for more information.

Parameters

- **input1** – The first input tensor to the layer.
- **input2** – The second input tensor to the layer.
- **op** – The binary operation that the layer applies.

The input tensors must have the same number of dimensions. For each dimension, their lengths must match, or one of them must be one. In the latter case, the tensor is broadcast along that axis.

The output tensor has the same number of dimensions as the inputs. For each dimension, its length is the maximum of the lengths of the corresponding input dimension.

Returns
The new element-wise layer, or `None` if it could not be created.

`add_fill`(*self: tensorrt.tensorrt.INetworkDefinition*, *shape: tensorrt.tensorrt.Dims*, *op: tensorrt.tensorrt.FillOperation*, *output_type: tensorrt.tensorrt.DataType = <DataType.FLOAT: 0>*) → *tensorrt.tensorrt.IFillLayer*

Add a fill layer. See `IFillLayer` for more information.

Parameters

- **dimensions** – The output tensor dimensions.
- **op** – The fill operation that the layer applies.
- **output_type** – The datatype of the output tensor. Default value tensorrt.float32.

Returns
The new fill layer, or `None` if it could not be created.

`add_gather`(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *indices:* tensorrt.tensorrt.ITensor, *axis: SupportsInt*) → *tensorrt.tensorrt.IGatherLayer*

Add a gather layer to the network. See `IGatherLayer` for more information.

Parameters

- **input** – The tensor to gather values from.
- **indices** – The tensor to get indices from to populate the output tensor.
- **axis** – The non-batch dimension axis in the data tensor to gather on.

Returns
The new gather layer, or `None` if it could not be created.

`add_gather_v2`(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *indices:* tensorrt.tensorrt.ITensor, *mode: tensorrt.tensorrt.GatherMode*) → *tensorrt.tensorrt.IGatherLayer*

Add a gather layer to the network. See `IGatherLayer` for more information.

Parameters

- **input** – The tensor to gather values from.

- **indices** – The tensor to get indices from to populate the output tensor.
- **mode** – The gather mode.

Returns
The new gather layer, or None if it could not be created.

add_grid_sample(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *grid:* tensorrt.tensorrt.ITensor) → *tensorrt.tensorrt.IGridSampleLayer*

Creates a GridSample layer with a trt.InterpolationMode.LINEAR, unaligned corners, and trt.SampleMode.FILL for 4d-shape input tensors. See *IGridSampleLayer* for more information.

Parameters

- **input** – The input tensor to the layer.
- **grid** – The grid tensor to the layer.

Variables

- **interpolation_mode** – class:*InterpolationMode* The interpolation mode to use in the layer. Default is LINEAR.
- **align_corners** – class:*bool* the align mode to use in the layer. Default is False.
- **padding_mode** – *SampleMode* The padding mode to use in the layer. Default is FILL.

Returns
The new grid sample layer, or None if it could not be created.

add_identity(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor) → *tensorrt.tensorrt.IIdentityLayer*

Add an identity layer. See *IIdentityLayer* for more information.

Parameters
input – The input tensor to the layer.

Returns
The new identity layer, or None if it could not be created.

add_if_conditional(*self:* tensorrt.tensorrt.INetworkDefinition) → *tensorrt.tensorrt.IIfConditional*

Adds an if-conditional to the network, which provides a way to specify subgraphs that will be conditionally executed using lazy evaluation. See *IIfConditional* for more information.

Returns
The new if-condtional, or None if it could not be created.

add_input(*self:* tensorrt.tensorrt.INetworkDefinition, *name: str*, *dtype: tensorrt.tensorrt.DataType*, *shape:* tensorrt.tensorrt.Dims) → *tensorrt.tensorrt.ITensor*

Adds an input to the network.

Parameters

- **name** – The name of the tensor. Each input and output tensor must have a unique name.
- **dtype** – The data type of the tensor.
- **shape** – The dimensions of the tensor.

Returns
The newly added Tensor.

add_kv_cache_update(*self:* tensorrt.tensorrt.INetworkDefinition, *cache:* tensorrt.tensorrt.ITensor, *update:* tensorrt.tensorrt.ITensor, *write_indices:* tensorrt.tensorrt.ITensor, *cache_mode:* *tensorrt.tensorrt.KVCacheMode*) → *tensorrt.tensorrt.IKVCacheUpdateLayer*

Add a KVCacheUpdate layer to the network. See `IKVCacheUpdateLayer` for more information.

Parameters

- **cache** – The key/value cache tensor for the layer. The user is responsible for properly allocating and binding the tensor memory.
- **update** – The newly updated key/value tensor for the layer.
- **write_indices** – The write indices tensor for key/value cache updates.
- **cache_mode** – The mode of the KVCacheUpdate layer. For TensorRT 10.15, only *LINEAR* mode is supported.

Returns
The new KVCacheUpdate layer, or `None` if it could not be created.

add_loop(*self:* tensorrt.tensorrt.INetworkDefinition) → *tensorrt.tensorrt.ILoop*

Adds a loop to the network, which provides a way to specify a recurrent subgraph. See `ILoop` for more information.

Returns
The new loop layer, or `None` if it could not be created.

add_lrn(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *window: SupportsInt*, *alpha: SupportsFloat*, *beta: SupportsFloat*, *k: SupportsFloat*) → *tensorrt.tensorrt.ILRNLayer*

Add a LRN layer to the network. See `ILRNLayer` for more information.

Parameters

- **input** – The input tensor to the layer.
- **window** – The size of the window.
- **alpha** – The alpha value for the LRN computation.
- **beta** – The beta value for the LRN computation.
- **k** – The k value for the LRN computation.

Returns
The new LRN layer, or `None` if it could not be created.

add_matrix_multiply(*self:* tensorrt.tensorrt.INetworkDefinition, *input0:* tensorrt.tensorrt.ITensor, *op0:* *tensorrt.tensorrt.MatrixOperation*, *input1:* tensorrt.tensorrt.ITensor, *op1:* *tensorrt.tensorrt.MatrixOperation*) → *tensorrt.tensorrt.IMatrixMultiplyLayer*

Add a matrix multiply layer to the network. See `IMatrixMultiplyLayer` for more information.

Parameters

- **input0** – The first input tensor (commonly A).
- **op0** – Whether to treat input0 as matrices, transposed matrices, or vectors.
- **input1** – The second input tensor (commonly B).
- **op1** – Whether to treat input1 as matrices, transposed matrices, or vectors.

Returns
The new matrix multiply layer, or `None` if it could not be created.

add_moe(*self:* tensorrt.tensorrt.INetworkDefinition, *hidden_states:* tensorrt.tensorrt.ITensor, *selected_experts_for_tokens:* tensorrt.tensorrt.ITensor, *scores_for_selected_experts:* tensorrt.tensorrt.ITensor) → *tensorrt.tensorrt.IMoELayer*

Add a MoE layer to the network. See *IMoELayer* for more information.

Parameters

- **hidden_states** – The hidden states tensor input to the MoE layer.
- **selected_experts_for_tokens** – The tensor containing expert indices selected for each token.
- **scores_for_selected_experts** – The tensor containing scores computed for the selected experts.

Returns
The new MoE layer, or `None` if it could not be created.

add_nms(**args*, ***kwargs*)

Overloaded function.

1. add_nms(self: tensorrt.tensorrt.INetworkDefinition, boxes: tensorrt.tensorrt.ITensor, scores: tensorrt.tensorrt.ITensor, max_output_boxes_per_class: tensorrt.tensorrt.ITensor) -> tensorrt.tensorrt.INMSLayer

   Add a non-maximum suppression layer to the network. See *INMSLayer* for more information.

   arg boxes
   The input boxes tensor to the layer.

   arg scores
   The input scores tensor to the layer.

   arg max_output_boxes_per_class
   The maxOutputBoxesPerClass tensor to the layer.

   ivar bounding_box_format
   `BoundingBoxFormat` The bounding box format used by the layer. Default is CORNER_PAIRS.

   ivar topk_box_limit
   `int` The maximum number of filtered boxes considered for selection per batch item. Default is 2000 for SM 5.3 and 6.2 devices, and 5000 otherwise. The TopK box limit must be less than or equal to {2000 for SM 5.3 and 6.2 devices, 5000 otherwise}.

   arg indices_type
   The datatype of the output indices tensor. Specifying indices_type is optional (default value tensorrt.int32).

   returns
   The new NMS layer, or `None` if it could not be created.

2. add_nms(self: tensorrt.tensorrt.INetworkDefinition, boxes: tensorrt.tensorrt.ITensor, scores: tensorrt.tensorrt.ITensor, max_output_boxes_per_class: tensorrt.tensorrt.ITensor, indices_type: tensorrt.tensorrt.DataType) -> tensorrt.tensorrt.INMSLayer

   Add a non-maximum suppression layer to the network. See *INMSLayer* for more information.

   arg boxes
   The input boxes tensor to the layer.

   arg scores
   The input scores tensor to the layer.

arg max_output_boxes_per_class
The maxOutputBoxesPerClass tensor to the layer.

ivar bounding_box_format
`BoundingBoxFormat` The bounding box format used by the layer. Default is CORNER_PAIRS.

ivar topk_box_limit
`int` The maximum number of filtered boxes considered for selection per batch item. Default is 2000 for SM 5.3 and 6.2 devices, and 5000 otherwise. The TopK box limit must be less than or equal to {2000 for SM 5.3 and 6.2 devices, 5000 otherwise}.

arg indices_type
The datatype of the output indices tensor. Specifying indices_type is optional (default value tensorrt.int32).

returns
The new NMS layer, or `None` if it could not be created.

`add_non_zero`(**args*, ***kwargs*)

Overloaded function.

1. add_non_zero(self: tensorrt.tensorrt.INetworkDefinition, input: tensorrt.tensorrt.ITensor) -> tensorrt.tensorrt.INonZeroLayer

   Adds an NonZero layer to the network. See *INonZeroLayer* for more information.

   arg input
   The input tensor to the layer.

   arg indices_type
   The datatype of the output indices tensor. Specifying indices_type is optional (default value tensorrt.int32).

   returns
   the new NonZero layer, or `None` if it could not be created.

2. add_non_zero(self: tensorrt.tensorrt.INetworkDefinition, input: tensorrt.tensorrt.ITensor, indices_type: tensorrt.tensorrt.DataType) -> tensorrt.tensorrt.INonZeroLayer

   Adds an NonZero layer to the network. See *INonZeroLayer* for more information.

   arg input
   The input tensor to the layer.

   arg indices_type
   The datatype of the output indices tensor. Specifying indices_type is optional (default value tensorrt.int32).

   returns
   the new NonZero layer, or `None` if it could not be created.

`add_normalization`(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *scale:* tensorrt.tensorrt.ITensor, *bias:* tensorrt.tensorrt.ITensor, *axesMask: SupportsInt*) → *tensorrt.tensorrt.INormalizationLayer*

[DEPRECATED] Deprecated in TensorRT 10.15. Superseded by add_normalization_v2.

Adds a Normalization layer to the network. See `Normalization` for more information.

Parameters

- **input** – The input tensor to the layer.

- **scale** – The scale tensor used to scale the normalized output.
- **bias** – The bias tensor used to scale the normalized output.
- **axesMask** – The axes on which to perform mean calculations. The bit in position i of bitmask axes corresponds to explicit dimension i of the result. E.g., the least significant bit corresponds to the first explicit dimension and the next to least significant bit corresponds to the second explicit dimension.

The normalization layer works by performing normalization of the tensor input on the specified axesMask. The result is then scaled by multiplying with scale and adding bias.

The shape of scale and bias must be the same, and must have the same rank and be unidirectionally broadcastable to the shape of input. Given a 4D NCHW input tensor, the expected shapes for scale and bias are: * [1, C, 1, 1] for InstanceNormalization * [1, G, 1, 1] for GroupNormalization. Use *INetworkDefinition.add_normalization_v2()* instead if [1, C, 1, 1] shapes for scale and bias are required.

Returns
the new Normalization layer, or `None` if it could not be created.

add_normalization_v2(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *scale:* tensorrt.tensorrt.ITensor, *bias:* tensorrt.tensorrt.ITensor, *axesMask: SupportsInt*) → *tensorrt.tensorrt.INormalizationLayer*

Adds a Normalization layer to the network. See `Normalization` for more information.

Parameters

- **input** – The input tensor to the layer.
- **scale** – The scale tensor used to scale the normalized output.
- **bias** – The bias tensor used to scale the normalized output.
- **axesMask** – The axes on which to perform mean calculations. The bit in position i of bitmask axes corresponds to explicit dimension i of the result. E.g., the least significant bit corresponds to the first explicit dimension and the next to least significant bit corresponds to the second explicit dimension.

The normalization layer works by performing normalization of the tensor input on the specified axesMask. The result is then scaled by multiplying with scale and adding bias.

The shapes of scale and bias must be the same, and must have the same rank and be unidirectionally broadcastable to the shape of input. In the case of InstanceNorm or GroupNorm, the shapes of scale and bias are expected to be [1, C, 1, 1] in the case of a 4D NCHW input tensor.

Returns
the new Normalization layer, or `None` if it could not be created.

add_one_hot(*self:* tensorrt.tensorrt.INetworkDefinition, *indices:* tensorrt.tensorrt.ITensor, *values:* tensorrt.tensorrt.ITensor, *depth:* tensorrt.tensorrt.ITensor, *axis: SupportsInt*) → *tensorrt.tensorrt.IOneHotLayer*

Add a OneHot layer to the network. See *IOneHotLayer* for more information.

Parameters

- **indices** – The tensor to get indices from to populate the output tensor.
- **values** – The tensor to get off (cold) value and on (hot) value
- **depth** – The tensor to get depth (number of classes) of one-hot encoding
- **axis** – The axis to append the one-hot encoding to

Returns
The new OneHot layer, or `None` if it could not be created.

add_padding_nd(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *pre_padding:* tensorrt.tensorrt.Dims, *post_padding:* tensorrt.tensorrt.Dims) → *tensorrt.tensorrt.IPaddingLayer*

Add a multi-dimensional padding layer to the network. See *IPaddingLayer* for more information.

Parameters

- **input** – The input tensor to the layer.
- **pre_padding** – The padding to apply to the start of the tensor.
- **post_padding** – The padding to apply to the end of the tensor.

Returns
The new padding layer, or `None` if it could not be created.

add_parametric_relu(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *slopes:* tensorrt.tensorrt.ITensor) → *tensorrt.tensorrt.IParametricReLULayer*

Add a parametric ReLU layer. See *IParametricReLULayer* for more information.

Parameters

- **input** – The input tensor to the layer.
- **slopes** – The slopes tensor (input elements are multiplied with the slopes where the input is negative).

Returns
The new parametric ReLU layer, or `None` if it could not be created.

add_plugin(**args*, ***kwargs*)

Overloaded function.

1. add_plugin(self: tensorrt.tensorrt.INetworkDefinition, tuple: tuple) -> tensorrt.tensorrt.IPluginV3Layer

   Add a plugin layer to the network using an *IPluginV3* interface. See *IPluginV3* for more information.

   arg inputs
   The input tensors to the layer.

   arg shape_inputs
   The shape input tensors to the layer.

   arg plugin
   The layer plugin.

   returns
   The new plugin layer, or `None` if it could not be created.

2. add_plugin(self: tensorrt.tensorrt.INetworkDefinition, func: collections.abc.Callable) -> tensorrt.tensorrt.IPluginV3Layer

3. add_plugin(self: tensorrt.tensorrt.INetworkDefinition, func: collections.abc.Callable, aot: bool) -> tensorrt.tensorrt.IPluginV3Layer

add_plugin_v2(*self:* tensorrt.tensorrt.INetworkDefinition, *inputs:* *collections.abc.Sequence[*tensorrt.tensorrt.ITensor*]*, *plugin: tensorrt.tensorrt.IPluginV2*) → *tensorrt.tensorrt.IPluginV2Layer*

Add a plugin layer to the network using an `IPluginV2` interface. See `IPluginV2` for more information.

Parameters

- **inputs** – The input tensors to the layer.
- **plugin** – The layer plugin.

Returns
The new plugin layer, or `None` if it could not be created.

add_plugin_v3(*self:* tensorrt.tensorrt.INetworkDefinition, *inputs:* *collections.abc.Sequence[*tensorrt.tensorrt.ITensor*]*, *shape_inputs:* *collections.abc.Sequence[*tensorrt.tensorrt.ITensor*]*, *plugin:* tensorrt.tensorrt.IPluginV3) → *tensorrt.tensorrt.IPluginV3Layer*

Add a plugin layer to the network using an *IPluginV3* interface. See *IPluginV3* for more information.

Parameters

- **inputs** – The input tensors to the layer.
- **shape_inputs** – The shape input tensors to the layer.
- **plugin** – The layer plugin.

Returns
The new plugin layer, or `None` if it could not be created.

add_pooling_nd(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *type:* *tensorrt.tensorrt.PoolingType*, *window_size:* tensorrt.tensorrt.Dims) → *tensorrt.tensorrt.IPoolingLayer*

Add a multi-dimension pooling layer to the network. See *IPoolingLayer* for more information.

Parameters

- **input** – The input tensor to the layer.
- **type** – The type of pooling to apply.
- **window_size** – The size of the pooling window.

Returns
The new pooling layer, or `None` if it could not be created.

add_quantize(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *scale:* tensorrt.tensorrt.ITensor, *output_type: tensorrt.tensorrt.DataType*) → *tensorrt.tensorrt.IQuantizeLayer*

Add a quantization layer to the network. See *IQuantizeLayer* for more information.

Parameters

- **input** – A tensor to quantize.
- **scale** – A tensor with the scale coefficients.
- **output_type** – The datatype of the output tensor.

Returns
The new quantization layer, or `None` if it could not be created.

add_ragged_softmax(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *bounds:* tensorrt.tensorrt.ITensor) → *tensorrt.tensorrt.IRaggedSoftMaxLayer*

Add a ragged softmax layer to the network. See *IRaggedSoftMaxLayer* for more information.

Parameters

- **input** – The ZxS input tensor.
- **bounds** – The Zx1 bounds tensor.

Returns
The new ragged softmax layer, or None if it could not be created.

add_reduce(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *op:* *tensorrt.tensorrt.ReduceOperation*, *axes: SupportsInt*, *keep_dims: bool*) → *tensorrt.tensorrt.IReduceLayer*

Add a reduce layer to the network. See *IReduceLayer* for more information.

Parameters

- **input** – The input tensor to the layer.
- **op** – The reduction operation to perform.
- **axes** – The reduction dimensions. The bit in position i of bitmask axes corresponds to explicit dimension i of the result. E.g., the least significant bit corresponds to the first explicit dimension and the next to least significant bit corresponds to the second explicit dimension.
- **keep_dims** – The boolean that specifies whether or not to keep the reduced dimensions in the output of the layer.

Returns
The new reduce layer, or None if it could not be created.

add_resize(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor) → *tensorrt.tensorrt.IResizeLayer*

Add a resize layer. See *IResizeLayer* for more information.

Parameters
input – The input tensor to the layer.

Returns
The new resize layer, or None if it could not be created.

add_reverse_sequence(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *sequence_lens:* tensorrt.tensorrt.ITensor) → *tensorrt.tensorrt.IReverseSequenceLayer*

Adds a ReverseSequence layer to the network. See *IReverseSequenceLayer* for more information.

Parameters

- **input** – The input tensor to the layer.
- **sequence_lens** – 1D tensor specifying lengths of sequences to reverse in a batch. The length of sequence_lens must be equal to the size of the dimension in input specified by batch_axis.

Returns
the new ReverseSequence layer, or None if it could not be created.

add_rotary_embedding(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *cos_cache:* tensorrt.tensorrt.ITensor, *sin_cache:* tensorrt.tensorrt.ITensor, *interleaved: bool*, *rotary_embedding_dim: SupportsInt*) → *tensorrt.tensorrt.IRotaryEmbeddingLayer*

Add a RotaryEmbedding layer to the network. See *IRotaryEmbeddingLayer* for more information.

Parameters

- **input** – The input activation tensor to the layer.
- **cos_cache** – The cosine cache tensor for use in RoPE computation.
- **sin_cache** – The sine cache tensor for use in RoPE computation.
- **interleaved** – Whether the input tensor is in interleaved format, i.e., whether the 2-d vectors rotated are taken from adjacent 2 elements in the hidden dimension.
- **rotary_embedding_dim** – The hidden dimension that participates in RoPE.

An optional input, position_ids, can be provided using `set_input()` with index 3. If provided, it is used to index into cos_cache and sin_cache.

Returns
The new RotaryEmbedding layer, or `None` if it could not be created.

add_scale(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *mode:* *tensorrt.tensorrt.ScaleMode*, *shift:* tensorrt.tensorrt.Weights = *None*, *scale:* tensorrt.tensorrt.Weights = *None*, *power:* tensorrt.tensorrt.Weights = *None*) → *tensorrt.tensorrt.IScaleLayer*

Add a scale layer to the network. See *IScaleLayer* for more information.

Parameters

- **input** – The input tensor to the layer. This tensor is required to have a minimum of 3 dimensions.
- **mode** – The scaling mode.
- **shift** – The shift value.
- **scale** – The scale value.
- **power** – The power value.

If the weights are available, then the size of weights are dependent on the ScaleMode. For UNIFORM, the number of weights is equal to 1. For CHANNEL, the number of weights is equal to the channel dimension. For ELEMENTWISE, the number of weights is equal to the volume of the input.

Returns
The new scale layer, or `None` if it could not be created.

add_scale_nd(*self: tensorrt.tensorrt.INetworkDefinition*, *input: tensorrt.tensorrt.ITensor*, *mode:* *tensorrt.tensorrt.ScaleMode*, *shift: tensorrt.tensorrt.Weights = None*, *scale:* *tensorrt.tensorrt.Weights = None*, *power: tensorrt.tensorrt.Weights = None*, *channel_axis:* *typing.SupportsInt*) → *tensorrt.tensorrt.IScaleLayer*

Add a multi-dimension scale layer to the network. See *IScaleLayer* for more information.

Parameters

- **input** – The input tensor to the layer. This tensor is required to have a minimum of 3 dimensions.
- **mode** – The scaling mode.
- **shift** – The shift value.
- **scale** – The scale value.
- **power** – The power value.
- **channel_axis** – The channel dimension axis.

If the weights are available, then the size of weights are dependent on the ScaleMode. For UNIFORM, the number of weights is equal to 1. For CHANNEL, the number of weights is equal to the channel dimension. For ELEMENTWISE, the number of weights is equal to the volume of the input.

Returns
The new scale layer, or `None` if it could not be created.

`add_scatter`(*self:* tensorrt.tensorrt.INetworkDefinition, *data:* tensorrt.tensorrt.ITensor, *indices:* tensorrt.tensorrt.ITensor, *updates:* tensorrt.tensorrt.ITensor, *mode:* *tensorrt.tensorrt.ScatterMode*) → *tensorrt.tensorrt.IScatterLayer*

Add a scatter layer to the network. See `IScatterLayer` for more information.

Parameters

- **`data`** – The tensor to get default values from.
- **`indices`** – The tensor to get indices from to populate the output tensor.
- **`updates`** – The tensor to get values from to populate the output tensor.
- **`mode`** – operation mode see IScatterLayer for more info

Returns
The new Scatter layer, or `None` if it could not be created.

`add_select`(*self:* tensorrt.tensorrt.INetworkDefinition, *condition:* tensorrt.tensorrt.ITensor, *then_input:* tensorrt.tensorrt.ITensor, *else_input:* tensorrt.tensorrt.ITensor) → *tensorrt.tensorrt.ISelectLayer*

Add a select layer. See `ISelectLayer` for more information.

Parameters

- **`condition`** – The condition tensor to the layer.
- **`then_input`** – The then input tensor to the layer.
- **`else_input`** – The else input tensor to the layer.

Returns
The new select layer, or `None` if it could not be created.

`add_shape`(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor) → *tensorrt.tensorrt.IShapeLayer*

Add a shape layer to the network. See `IShapeLayer` for more information.

Parameters
`input` – The input tensor to the layer.

Returns
The new shape layer, or `None` if it could not be created.

`add_shuffle`(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor) → *tensorrt.tensorrt.IShuffleLayer*

Add a shuffle layer to the network. See `IShuffleLayer` for more information.

Parameters
`input` – The input tensor to the layer.

Returns
The new shuffle layer, or `None` if it could not be created.

`add_slice`(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *start:* tensorrt.tensorrt.Dims, *shape:* tensorrt.tensorrt.Dims, *stride:* tensorrt.tensorrt.Dims) → *tensorrt.tensorrt.ISliceLayer*

Add a slice layer to the network. See *ISliceLayer* for more information.

Parameters

- **input** – The input tensor to the layer.
- **start** – The start offset.
- **shape** – The output shape.
- **stride** – The slicing stride. Positive, negative, zero stride values, and combinations of them in different dimensions are allowed.

Returns
The new slice layer, or `None` if it could not be created.

add_softmax(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor) → *tensorrt.tensorrt.ISoftMaxLayer*

Add a softmax layer to the network. See *ISoftMaxLayer* for more information.

Parameters
input – The input tensor to the layer.

Returns
The new softmax layer, or `None` if it could not be created.

add_squeeze(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *axes:* tensorrt.tensorrt.ITensor) → *tensorrt.tensorrt.ISqueezeLayer*

Adds a Squeeze layer to the network. See *ISqueezeLayer* for more information.

Parameters

- **input** – The input tensor to the layer.
- **axes** – The tensor containing axes to remove. Must be resolvable to a constant Int32 or Int64 1D shape tensor.

Returns
the new Squeeze layer, or `None` if it could not be created.

add_topk(**args*, ***kwargs*)

Overloaded function.

1. add_topk(self: tensorrt.tensorrt.INetworkDefinition, input: tensorrt.tensorrt.ITensor, op: tensorrt.tensorrt.TopKOperation, k: typing.SupportsInt, axes: typing.SupportsInt) -> tensorrt.tensorrt.ITopKLayer

   Add a TopK layer to the network. See *ITopKLayer* for more information.

   The TopK layer has two outputs of the same dimensions. The first contains data values, the second contains index positions for the values. Output values are sorted, largest first for operation `TopKOperation.MAX` and smallest first for operation `TopKOperation.MIN` .

   Currently only values of K up to 3840 are supported.

   arg input
   The input tensor to the layer.

   arg op
   Operation to perform.

   arg k
   Number of elements to keep.

arg axes
The reduction dimensions. The bit in position i of bitmask axes corresponds to explicit dimension i of the result. E.g., the least significant bit corresponds to the first explicit dimension and the next to least significant bit corresponds to the second explicit dimension. Currently axes must specify exactly one dimension, and it must be one of the last four dimensions.

arg indices_type
The datatype of the output indices tensor. Specifying indices_type is optional (default value tensorrt.int32).

returns
The new TopK layer, or `None` if it could not be created.

2. add_topk(self: tensorrt.tensorrt.INetworkDefinition, input: tensorrt.tensorrt.ITensor, op: tensorrt.tensorrt.TopKOperation, k: typing.SupportsInt, axes: typing.SupportsInt, indices_type: tensorrt.tensorrt.DataType) -> tensorrt.tensorrt.ITopKLayer

   Add a TopK layer to the network. See *ITopKLayer* for more information.

   The TopK layer has two outputs of the same dimensions. The first contains data values, the second contains index positions for the values. Output values are sorted, largest first for operation `TopKOperation.MAX` and smallest first for operation `TopKOperation.MIN` .

   Currently only values of K up to 3840 are supported.

   arg input
   The input tensor to the layer.

   arg op
   Operation to perform.

   arg k
   Number of elements to keep.

   arg axes
   The reduction dimensions. The bit in position i of bitmask axes corresponds to explicit dimension i of the result. E.g., the least significant bit corresponds to the first explicit dimension and the next to least significant bit corresponds to the second explicit dimension. Currently axes must specify exactly one dimension, and it must be one of the last four dimensions.

   arg indices_type
   The datatype of the output indices tensor. Specifying indices_type is optional (default value tensorrt.int32).

   returns
   The new TopK layer, or `None` if it could not be created.

add_unary(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *op:* *tensorrt.tensorrt.UnaryOperation*) → *tensorrt.tensorrt.IUnaryLayer*

Add a unary layer to the network. See *IUnaryLayer* for more information.

Parameters

- **input** – The input tensor to the layer.
- **op** – The operation to apply.

Returns
The new unary layer, or `None` if it could not be created.

add_unsqueeze(*self:* tensorrt.tensorrt.INetworkDefinition, *input:* tensorrt.tensorrt.ITensor, *axes:* tensorrt.tensorrt.ITensor) → *tensorrt.tensorrt.IUnsqueezeLayer*

Adds an Unsqueeze layer to the network. See *IUnsqueezeLayer* for more information.

Parameters

- **input** – The input tensor to the layer.
- **axes** – The tensor containing axes to add. Must be resolvable to a constant Int32 or Int64 1D shape tensor.

Returns
the new Unsqueeze layer, or None if it could not be created.

are_weights_marked_refittable(*self:* tensorrt.tensorrt.INetworkDefinition, *name: str*) → bool

Whether the weight has been marked as refittable.

Parameters
name – The name of the weights to check.

property builder

The builder from which this INetworkDefinition was created.

See IBuilder for more information.

get_flag(*self:* tensorrt.tensorrt.INetworkDefinition, *flag: tensorrt.NetworkDefinitionCreationFlag*) → bool

Returns true if the specified NetworkDefinitionCreationFlag is set.

Parameters
flag – The NetworkDefinitionCreationFlag .

Returns
Whether the flag is set.

get_input(*self:* tensorrt.tensorrt.INetworkDefinition, *index: SupportsInt*) → *tensorrt.tensorrt.ITensor*

Get the input tensor specified by the given index.

Parameters
index – The index of the input tensor.

Returns
The tensor, or None if it is out of range.

get_layer(*self:* tensorrt.tensorrt.INetworkDefinition, *index: SupportsInt*) → *tensorrt.tensorrt.ILayer*

Get the layer specified by the given index.

Parameters
index – The index of the layer.

Returns
The layer, or None if it is out of range.

get_output(*self:* tensorrt.tensorrt.INetworkDefinition, *index: SupportsInt*) → *tensorrt.tensorrt.ITensor*

Get the output tensor specified by the given index.

Parameters
index – The index of the output tensor.

Returns
The tensor, or None if it is out of range.

is_debug_tensor(*self:* tensorrt.tensorrt.INetworkDefinition, *tensor:* tensorrt.tensorrt.ITensor) → bool

Check if a tensor is marked as debug.

Parameters
tensor – The tensor to be checked.

mark_debug(*self:* tensorrt.tensorrt.INetworkDefinition, *tensor:* tensorrt.tensorrt.ITensor) → bool

Mark a tensor as a debug tensor in the network.

Parameters
tensor – The tensor to be marked as debug tensor.

Returns
True on success, False otherwise.

mark_output(*self:* tensorrt.tensorrt.INetworkDefinition, *tensor:* tensorrt.tensorrt.ITensor) → None

Mark a tensor as an output.

Parameters
tensor – The tensor to mark.

mark_output_for_shapes(*self:* tensorrt.tensorrt.INetworkDefinition, *tensor:* tensorrt.tensorrt.ITensor) → bool

Enable tensor's value to be computed by `IExecutionContext.get_shape_binding()`.

Parameters
tensor – The tensor to unmark as an output tensor. The tensor must be of type `int32` and have no more than one dimension.

Returns
`True` if successful, `False` if tensor is already marked as an output.

mark_unfused_tensors_as_debug_tensors(*self:* tensorrt.tensorrt.INetworkDefinition) → bool

Mark unfused tensors as debug tensors.

Debug tensors can be optionally emitted at runtime. Tensors that are fused by the optimizer will not be emitted. Tensors marked this way will not prevent fusion like mark_debug() does, thus preserving performance.

Tensors marked this way cannot be detected by is_debug_tensor(). DebugListener can only get internal tensor names instead of the original tensor names in the NetworkDefinition for tensors marked this way. But the names correspond to the names obtained by IEngineInspector. There is no guarantee that all unfused tensors are marked.

Returns
True if tensors were successfully marked (or were already marked), false otherwise.

mark_weights_refittable(*self:* tensorrt.tensorrt.INetworkDefinition, *name: str*) → bool

Mark a weight as refittable.

Parameters
name – The weight to mark.

remove_tensor(*self:* tensorrt.tensorrt.INetworkDefinition, *tensor:* tensorrt.tensorrt.ITensor) → None

Remove a tensor from the network.

Parameters
tensor – The tensor to remove

It is illegal to remove a tensor that is the input or output of a layer. if this method is called with such a tensor, a warning will be emitted on the log and the call will be ignored.

set_weights_name(*self:* tensorrt.tensorrt.INetworkDefinition, *weights:* tensorrt.tensorrt.Weights, *name: str*) → bool

Associate a name with all current uses of the given weights.

The name must be set after the Weights are used in the network. Lookup is associative. The name applies to all Weights with matching type, value pointer, and count. If Weights with a matching value pointer, but different type or count exists in the network, an error message is issued, the name is rejected, and return false. If the name has already been used for other weights, return false. None causes the weights to become unnamed, i.e. clears any previous name.

Parameters

- **weights** – The weights to be named.
- **name** – The name to associate with the weights.

Returns
true on success.

unmark_debug(*self:* tensorrt.tensorrt.INetworkDefinition, *tensor:* tensorrt.tensorrt.ITensor) → bool

Unmark a tensor as a debug tensor in the network.

Parameters
tensor – The tensor to be unmarked as debug tensor.

Returns
True on success, False otherwise.

unmark_output(*self:* tensorrt.tensorrt.INetworkDefinition, *tensor:* tensorrt.tensorrt.ITensor) → None

Unmark a tensor as a network output.

Parameters
tensor – The tensor to unmark as an output tensor.

unmark_output_for_shapes(*self:* tensorrt.tensorrt.INetworkDefinition, *tensor:* tensorrt.tensorrt.ITensor) → bool

Undo *mark_output_for_shapes()* .

Parameters
tensor – The tensor to unmark as an output tensor.

Returns
`True` if successful, `False` if tensor is not marked as an output.

unmark_unfused_tensors_as_debug_tensors(*self:* tensorrt.tensorrt.INetworkDefinition) → bool

Undo the marking of unfused tensor as debug tensors.

This has no effect on tensors marked by mark_debug().

Returns
True if tensor successfully unmarked (or was already unmarked), false otherwise.

unmark_weights_refittable(*self:* tensorrt.tensorrt.INetworkDefinition, *name: str*) → bool

Unmark a weight as refittable.

Parameters
name – The weight to unmark.

6.2 Layer Base Classes

6.2.1 ITensor

tensorrt.**TensorLocation**

The physical location of the data.

Members:

DEVICE : Data is stored on the device.

HOST : Data is stored on the host.

tensorrt.**TensorFormat**

Format of the input/output tensors.

This enum is used by both plugins and network I/O tensors.

For more information about data formats, see the topic "Data Format Description" located in the TensorRT Developer Guide (https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html).

Members:

LINEAR :
Row major linear format.

For a tensor with dimensions {N, C, H, W}, the W axis always has unit stride, and the stride of every other axis is at least the product of the next dimension times the next stride. the strides are the same as for a C array with dimensions [N][C][H][W].

CHW2 :
Two wide channel vectorized row major format.

This format is bound to FP16 and BF16. It is only available for dimensions >= 3.

For a tensor with dimensions {N, C, H, W}, the memory layout is equivalent to a C array with dimensions [N][(C+1)/2][H][W][2], with the tensor coordinates (n, c, h, w) mapping to array subscript [n][c/2][h][w][c%2].

HWC8 :
Eight channel format where C is padded to a multiple of 8.

This format is bound to FP16 and BF16. It is only available for dimensions >= 3.

For a tensor with dimensions {N, C, H, W}, the memory layout is equivalent to the array with dimensions [N][H][W][(C+7)/8*8], with the tensor coordinates (n, c, h, w) mapping to array subscript [n][h][w][c].

CHW4 :
Four wide channel vectorized row major format. This format is bound to INT8. It is only available for dimensions >= 3.

For a tensor with dimensions {N, C, H, W}, the memory layout is equivalent to a C array with dimensions [N][(C+3)/4][H][W][4], with the tensor coordinates (n, c, h, w) mapping to array subscript [n][c/4][h][w][c%4].

CHW16 :
Sixteen wide channel vectorized row major format.

This format is only supported by DLA and requires FP16. It is only available for dimensions >= 3.

For a tensor with dimensions {N, C, H, W}, the memory layout is equivalent to a C array with dimensions [N][(C+15)/16][H][W][16], with the tensor coordinates (n, c, h, w) mapping to array subscript [n][c/16][h][w][c%16].

CHW32 :
Thirty-two wide channel vectorized row major format.

This format is only available for dimensions >= 3.

For a tensor with dimensions {N, C, H, W}, the memory layout is equivalent to a C array with dimensions [N][(C+31)/32][H][W][32], with the tensor coordinates (n, c, h, w) mapping to array subscript [n][c/32][h][w][c%32].

DHWC8 :
Eight channel format where C is padded to a multiple of 8.

This format is bound to FP16 and BF16, and it is only available for dimensions >= 4.

For a tensor with dimensions {N, C, D, H, W}, the memory layout is equivalent to an array with dimensions [N][D][H][W][(C+7)/8*8], with the tensor coordinates (n, c, d, h, w) mapping to array subscript [n][d][h][w][c].

CDHW32 :
Thirty-two wide channel vectorized row major format with 3 spatial dimensions.

This format is bound to FP16 and INT8. It is only available for dimensions >= 4.

For a tensor with dimensions {N, C, D, H, W}, the memory layout is equivalent to a C array with dimensions [N][(C+31)/32][D][H][W][32], with the tensor coordinates (n, d, c, h, w) mapping to array subscript [n][c/32][d][h][w][c%32].

HWC :
Non-vectorized channel-last format. This format is bound to FP32, FP16, INT8, INT64 and BF16, and is only available for dimensions >= 3.

DLA_LINEAR :
DLA planar format. Row major format. The stride for stepping along the H axis is rounded up to 64 bytes.

This format is bound to FP16/Int8 and is only available for dimensions >= 3.

For a tensor with dimensions {N, C, H, W}, the memory layout is equivalent to a C array with dimensions [N][C][H][roundUp(W, 64/elementSize)] where elementSize is 2 for FP16 and 1 for Int8, with the tensor coordinates (n, c, h, w) mapping to array subscript [n][c][h][w].

DLA_HWC4 :
DLA image format. channel-last format. C can only be 1, 3, 4. If C == 3 it will be rounded to 4. The stride for stepping along the H axis is rounded up to 32 bytes.

This format is bound to FP16/Int8 and is only available for dimensions >= 3.

For a tensor with dimensions {N, C, H, W}, with C' is 1, 4, 4 when C is 1, 3, 4 respectively, the memory layout is equivalent to a C array with dimensions [N][H][roundUp(W, 32/C'/elementSize)][C'] where elementSize is 2 for FP16 and 1 for Int8, C' is the rounded C. The tensor coordinates (n, c, h, w) maps to array subscript [n][h][w][c].

HWC16 :
Sixteen channel format where C is padded to a multiple of 16. This format is bound to FP16/INT8/FP8. It is only available for dimensions >= 3.

For a tensor with dimensions {N, C, H, W}, the memory layout is equivalent to the array with dimensions [N][H][W][(C+15)/16*16], with the tensor coordinates (n, c, h, w) mapping to array subscript [n][h][w][c].

DHWC :
Non-vectorized channel-last format. This format is bound to FP32. It is only available for dimensions >= 4.

class tensorrt.ITensor

A tensor in an *INetworkDefinition* .

Variables

- **name** – str The tensor name. For a network input, the name is assigned by the application. For tensors which are layer outputs, a default name is assigned consisting of the layer name followed by the index of the output in brackets. Each network input and output tensor must have a unique name.
- **shape** – *Dims* The shape of a tensor. For a network input the shape is assigned by the application. For a network output it is computed based on the layer parameters and the inputs to the layer. If a tensor size or a parameter is modified in the network, the shape of all dependent tensors will be recomputed. This call is only legal for network input tensors, since the shape of layer output tensors are inferred based on layer inputs and parameters.
- **dtype** – *DataType* The data type of a tensor. The type is unchanged if the type is invalid for the given tensor.
- **broadcast_across_batch** – bool [DEPRECATED] Deprecated in TensorRT 10.0. Always false since the implicit batch dimensions support has been removed.
- **location** – *TensorLocation* The storage location of a tensor.
- **is_network_input** – bool Whether the tensor is a network input.
- **is_network_output** – bool Whether the tensor is a network output.
- **is_shape** – bool Whether the tensor is a shape tensor.
- **allowed_formats** – int32 The allowed set of TensorFormat candidates. This should be an integer consisting of one or more *TensorFormat* s, combined via bitwise OR after bit shifting. For example, `1 << int(TensorFormat.CHW4) | 1 << int(TensorFormat.CHW32)`.

get_dimension_name(*self:* tensorrt.tensorrt.ITensor, *index: SupportsInt*) → str

Get the name of an input dimension.

Parameters
index – index of the dimension.

Returns
name of the dimension, or null if dimension is unnamed.

set_dimension_name(*self:* tensorrt.tensorrt.ITensor, *index: SupportsInt*, *name: str*) → None

Name a dimension of an input tensor.

Associate a runtime dimension of an input tensor with a symbolic name. Dimensions with the same non-empty name must be equal at runtime. Knowing this equality for runtime dimensions may help the TensorRT optimizer. Both runtime and build-time dimensions can be named. If the function is called again, with the same index, it will overwrite the previous name. If None is passed as name, it will clear the name of the dimension.

For example, setDimensionName(0, "n") associates the symbolic name "n" with the leading dimension.

Parameters

- **index** – index of the dimension.

- **name** – name of the dimension.

6.2.2 ILayer

tensorrt.**LayerType**

Type of Layer

Members:

CONVOLUTION : Convolution layer

GRID_SAMPLE : Grid sample layer

NMS : NMS layer

ACTIVATION : Activation layer

POOLING : Pooling layer

LRN : LRN layer

SCALE : Scale layer

SOFTMAX : Softmax layer

DECONVOLUTION : Deconvolution layer

CONCATENATION : Concatenation layer

ELEMENTWISE : Elementwise layer

PLUGIN : Plugin layer

UNARY : Unary layer

PADDING : Padding layer

SHUFFLE : Shuffle layer

REDUCE : Reduce layer

TOPK : TopK layer

GATHER : Gather layer

MATRIX_MULTIPLY : Matrix multiply layer

RAGGED_SOFTMAX : Ragged softmax layer

CONSTANT : Constant layer

IDENTITY : Identity layer

CAST : Cast layer

PLUGIN_V2 : PluginV2 layer

SLICE : Slice layer

SHAPE : Shape layer

PARAMETRIC_RELU : Parametric ReLU layer

RESIZE : Resize layer

TRIP_LIMIT : Loop Trip limit layer

RECURRENCE : Loop Recurrence layer

ITERATOR : Loop Iterator layer

LOOP_OUTPUT : Loop output layer

SELECT : Select layer

ASSERTION : Assertion layer

FILL : Fill layer

QUANTIZE : Quantize layer

DEQUANTIZE : Dequantize layer

CONDITION : If-conditional Condition layer

CONDITIONAL_INPUT : If-conditional input layer

CONDITIONAL_OUTPUT : If-conditional output layer

SCATTER : Scatter layer

EINSUM : Einsum layer

ONE_HOT : OneHot layer

NON_ZERO : NonZero layer

REVERSE_SEQUENCE : ReverseSequence layer

NORMALIZATION : Normalization layer

PLUGIN_V3 : PluginV3 layer

SQUEEZE : Squeeze layer

UNSQUEEZE : Unsqueeze layer

CUMULATIVE : Cumulative layer

DYNAMIC_QUANTIZE : DynamicQuantize layer

ATTENTION_INPUT : Attention input layer

ATTENTION_OUTPUT : Attention output layer

ROTARY_EMBEDDING : Rotary Embedding layer

KV_CACHE_UPDATE : KVCacheUpdate layer

MOE : MoE layer

DIST_COLLECTIVE : DistCollective layer

class tensorrt.**ILayer**

Base class for all layer classes in an *INetworkDefinition* .

Variables

- **name** – str The name of the layer.
- **metadata** – str The per-layer metadata.
- **num_ranks** – int The number of ranks for multi-device execution (default: 1).
- **type** – *LayerType* The type of the layer.
- **num_inputs** – int The number of inputs of the layer.
- **num_outputs** – int The number of outputs of the layer.

get_input(*self:* tensorrt.tensorrt.ILayer, *index: SupportsInt*) → *tensorrt.tensorrt.ITensor*

Get the layer input corresponding to the given index.

Parameters
index – The index of the input tensor.

Returns
The input tensor, or `None` if the index is out of range.

get_output(*self:* tensorrt.tensorrt.ILayer, *index: SupportsInt*) → *tensorrt.tensorrt.ITensor*

Get the layer output corresponding to the given index.

Parameters
index – The index of the output tensor.

Returns
The output tensor, or `None` if the index is out of range.

get_output_type(*self:* tensorrt.tensorrt.ILayer, *index: SupportsInt*) → tensorrt.tensorrt.DataType

Get the output type of the layer.

Parameters
index – The index of the output tensor.

Returns
The output precision. Default : DataType.FLOAT.

property num_ranks

`int` The number of ranks for multi-device execution.

Currently, setting num_ranks > 1 via ILayer is only allowed for IDistCollectiveLayer, which uses it to determine output shape for kALL_GATHER and kREDUCE_SCATTER operations.

For attention layers, use IAttention.num_ranks instead.

Default value is 1.

set_input(*self:* tensorrt.tensorrt.ILayer, *index: SupportsInt*, *tensor:* tensorrt.tensorrt.ITensor) → None

Set the layer input corresponding to the given index.

Parameters

- **index** – The index of the input tensor.
- **tensor** – The input tensor.

6.3 Layers

For more information, including examples, refer to the TensorRT Operator's Reference documentation.

Note that layer weight properties may be represented as NumPy arrays or Weights objects depending on whether the underlying datatype is supported by NumPy. Explicitly construct a Weights object from the property if you want a consistent type:

```
conv_layer = network.add_convolution_nd(...)
bias_weights = trt.Weights(conv_layer.bias)
```

6.3.1 PaddingMode

tensorrt.**PaddingMode**

Enumerates types of padding available in convolution, deconvolution and pooling layers.
Padding mode takes precedence if both `padding_mode` and `pre_padding` are set.

EXPLICIT* corresponds to explicit padding.

SAME* implicitly calculates padding such that the output dimensions are the same as the input dimensions. For convolution and pooling, output dimensions are determined by ceil(input dimensions, stride).

CAFFE* corresponds to symmetric padding.

Members:

EXPLICIT_ROUND_DOWN : Use explicit padding, rounding the output size down

EXPLICIT_ROUND_UP : Use explicit padding, rounding the output size up

SAME_UPPER : Use SAME padding, with `pre_padding` <= `post_padding`

SAME_LOWER : Use SAME padding, with `pre_padding` >= `post_padding`

6.3.2 ICastLayer

class tensorrt.**ICastLayer**

A layer that represents the cast function.

This layer casts the element of a given input tensor to a specified data type and returns an output tensor of the same shape in the converted type.

Conversions between all types except FP8 is supported.

Variables
to_type – *DataType* The specified data type of the output tensor.

6.3.3 IConvolutionLayer

class tensorrt.**IConvolutionLayer**

A convolution layer in an *INetworkDefinition* .

This layer performs a correlation operation between 3 or 4 dimensional filter with a 4 or 5 dimensional tensor to produce another 4 or 5 dimensional tensor.

An optional bias argument is supported, which adds a per-channel constant to each value in the output.

Variables

- **num_output_maps** – `int` The number of output maps for the convolution.
- **pre_padding** – *DimsHW* The pre-padding. The start of input will be zero-padded by this number of elements in the height and width directions. Default: (0, 0)
- **post_padding** – *DimsHW* The post-padding. The end of input will be zero-padded by this number of elements in the height and width directions. Default: (0, 0)

- **padding_mode** – *PaddingMode* The padding mode. Padding mode takes precedence if both `IConvolutionLayer.padding_mode` and either `IConvolutionLayer.pre_padding` or `IConvolutionLayer.post_padding` are set.
- **num_groups** – `int` The number of groups for a convolution. The input tensor channels are divided into this many groups, and a convolution is executed for each group, using a filter per group. The results of the group convolutions are concatenated to form the output. **Note** When using groups in int8 mode, the size of the groups (i.e. the channel count divided by the group count) must be a multiple of 4 for both input and output. Default: 1.
- **kernel** – *Weights* The kernel weights for the convolution. The weights are specified as a contiguous array in *GKCRS* order, where *G* is the number of groups, *K* the number of output feature maps, *C* the number of input channels, and *R* and *S* are the height and width of the filter.
- **bias** – *Weights* The bias weights for the convolution. Bias is optional. To omit bias, set this to an empty *Weights* object. The bias is applied per-channel, so the number of weights (if non-zero) must be equal to the number of output feature maps.
- **kernel_size_nd** – *Dims* The multi-dimension kernel size of the convolution.
- **stride_nd** – *Dims* The multi-dimension stride of the convolution. Default: (1, ..., 1)
- **padding_nd** – *Dims* The multi-dimension padding of the convolution. The input will be zero-padded by this number of elements in each dimension. If the padding is asymmetric, this value corresponds to the pre-padding. Default: (0, ..., 0)
- **dilation_nd** – *Dims* The multi-dimension dilation for the convolution. Default: (1, ..., 1)

6.3.4 IGridSampleLayer

`tensorrt.`**InterpolationMode**

Various modes of interpolation, used in resize and grid_sample layers.

Members:

NEAREST : 1D, 2D, and 3D nearest neighbor interpolation.

LINEAR : Supports linear, bilinear, trilinear interpolation.

CUBIC : Supports bicubic interpolation.

`tensorrt.`**SampleMode**

Controls how ISliceLayer and IGridSample handles out of bounds coordinates

Members:

STRICT_BOUNDS : Fail with error when the coordinates are out of bounds.

WRAP : Coordinates wrap around periodically.

CLAMP : Out of bounds indices are clamped to bounds

FILL : Use fill input value when coordinates are out of bounds.

REFLECT : Coordinates reflect.

class `tensorrt.`**IGridSampleLayer**

A grid sample layer in an *INetworkDefinition* .

This layer uses an input tensor and a grid tensor to produce an interpolated output tensor. The input and grid tensors must shape tensors of rank 4. The only supported *SampleMode* s are trt.samplemode.CLAMP, trt.samplemode.FILL, and trt.samplemode.REFLECT.

Variables

- **interpolation_mode** – class:*InterpolationMode* The interpolation type to use. Defaults to LINEAR.
- **align_corners** – class:*bool* the align mode to use. Defaults to False.
- **sample_mode** – *SampleMode* The sample mode to use. Defaults to FILL.

6.3.5 IActivationLayer

tensorrt.**ActivationType**

The type of activation to perform.

Members:

RELU : Rectified Linear activation

SIGMOID : Sigmoid activation

TANH : Hyperbolic Tangent activation

LEAKY_RELU : Leaky Relu activation: f(x) = x if x >= 0, f(x) = alpha * x if x < 0

ELU : Elu activation: f(x) = x if x >= 0, f(x) = alpha * (exp(x) - 1) if x < 0

SELU : Selu activation: f(x) = beta * x if x > 0, f(x) = beta * (alpha * exp(x) - alpha) if x <= 0

SOFTSIGN : Softsign activation: f(x) = x / (1 + abs(x))

SOFTPLUS : Softplus activation: f(x) = alpha * log(exp(beta * x) + 1)

CLIP : Clip activation: f(x) = max(alpha, min(beta, x))

HARD_SIGMOID : Hard sigmoid activation: f(x) = max(0, min(1, alpha * x + beta))

SCALED_TANH : Scaled Tanh activation: f(x) = alpha * tanh(beta * x)

THRESHOLDED_RELU : Thresholded Relu activation: f(x) = x if x > alpha, f(x) = 0 if x <= alpha

GELU_ERF : GELU erf activation: 0.5 * x * (1 + erf(sqrt(0.5) * x))

GELU_TANH : GELU tanh activation: 0.5 * x * (1 + tanh(sqrt(2/pi) * (0.044715F * pow(x, 3) + x)))

class tensorrt.**IActivationLayer**

An Activation layer in an *INetworkDefinition* . This layer applies a per-element activation function to its input. The output has the same shape as the input.

Variables

- **type** – *ActivationType* The type of activation to be performed.
- **alpha** – `float` The alpha parameter that is used by some parametric activations (LEAKY_RELU, ELU, SELU, SOFTPLUS, CLIP, HARD_SIGMOID, SCALED_TANH). Other activations ignore this parameter.
- **beta** – `float` The beta parameter that is used by some parametric activations (SELU, SOFTPLUS, CLIP, HARD_SIGMOID, SCALED_TANH). Other activations ignore this parameter.

6.3.6 IPoolingLayer

tensorrt.**PoolingType**

The type of pooling to perform in a pooling layer.

Members:

MAX : Maximum over elements

AVERAGE : Average over elements. If the tensor is padded, the count includes the padding

MAX_AVERAGE_BLEND : Blending between the max pooling and average pooling: *(1-blendFactor)*maxPool + blendFactor*avgPool*

class tensorrt.**IPoolingLayer**

A Pooling layer in an *INetworkDefinition* . The layer applies a reduction operation within a window over the input.

Variables

- **type** – *PoolingType* The type of pooling to be performed.
- **pre_padding** – *DimsHW* The pre-padding. The start of input will be zero-padded by this number of elements in the height and width directions. Default: (0, 0)
- **post_padding** – *DimsHW* The post-padding. The end of input will be zero-padded by this number of elements in the height and width directions. Default: (0, 0)
- **padding_mode** – *PaddingMode* The padding mode. Padding mode takes precedence if both `IPoolingLayer.padding_mode` and either `IPoolingLayer.pre_padding` or `IPoolingLayer.post_padding` are set.
- **blend_factor** – `float` The blending factor for the max_average_blend mode: $max_average_blendPool = (1 - blendFactor) * maxPool + blendFactor * avgPool$. `blend_factor` is a user value in [0,1] with the default value of 0.0. This value only applies for the `PoolingType.MAX_AVERAGE_BLEND` mode.
- **average_count_excludes_padding** – `bool` Whether average pooling uses as a denominator the overlap area between the window and the unpadded input. If this is not set, the denominator is the overlap between the pooling window and the padded input. Default: True
- **window_size_nd** – *Dims* The multi-dimension window size for pooling.
- **stride_nd** – *Dims* The multi-dimension stride for pooling. Default: (1, ..., 1)
- **padding_nd** – *Dims* The multi-dimension padding for pooling. Default: (0, ..., 0)

6.3.7 ILRNLayer

class tensorrt.**ILRNLayer**

A LRN layer in an *INetworkDefinition* . The output size is the same as the input size.

Variables

- **window_size** – `int` The LRN window size. The window size must be odd and in the range of [1, 15].
- **alpha** – `float` The LRN alpha value. The valid range is [-1e20, 1e20].
- **beta** – `float` The LRN beta value. The valid range is [0.01, 1e5f].
- **k** – `float` The LRN K value. The valid range is [1e-5, 1e10].

6.3.8 IScaleLayer

tensorrt.**ScaleMode**

Controls how scale is applied in a Scale layer.

Members:

UNIFORM : Identical coefficients across all elements of the tensor.

CHANNEL : Per-channel coefficients. The channel dimension is assumed to be the third to last dimension.

ELEMENTWISE : Elementwise coefficients.

class tensorrt.**IScaleLayer**

A Scale layer in an *INetworkDefinition* .

This layer applies a per-element computation to its input:

$output = (input * scale + shift)^{power}$

The coefficients can be applied on a per-tensor, per-channel, or per-element basis.

Note If the number of weights is 0, then a default value is used for shift, power, and scale. The default shift is 0, the default power is 1, and the default scale is 1.

The output size is the same as the input size.

Note The input tensor for this layer is required to have a minimum of 3 dimensions.

Variables

- **mode** – *ScaleMode* The scale mode.
- **shift** – *Weights* The shift value.
- **scale** – *Weights* The scale value.
- **power** – *Weights* The power value.
- **channel_axis** – int The channel axis.

6.3.9 ISoftMaxLayer

class tensorrt.**ISoftMaxLayer**

A Softmax layer in an *INetworkDefinition* .

This layer applies a per-channel softmax to its input.

The output size is the same as the input size.

Variables

axes – int The axis along which softmax is computed. Currently, only one axis can be set.

The axis is specified by setting the bit corresponding to the axis to 1, as a bit mask.

For example, consider an NCHW tensor as input (three non-batch dimensions).

By default, softmax is performed on the axis which is the number of axes minus three. It is 0 if there are fewer than 3 non-batch axes. For example, if the input is NCHW, the default axis is C. If the input is NHW, then the default axis is H.

Bit 0 corresponds to the N dimension boolean.

Bit 1 corresponds to the C dimension boolean.
Bit 2 corresponds to the H dimension boolean.
Bit 3 corresponds to the W dimension boolean.
By default, softmax is performed on the axis which is the number of axes minus three. It is 0 if there are fewer than 3 axes. For example, if the input is NCHW, the default axis is C. If the input is NHW, then the default axis is N.

For example, to perform softmax on axis R of a NPQRCHW input, set bit 3.

The following constraints must be satisfied to execute this layer on DLA:

- Axis must be one of the channel or spatial dimensions.
- There are two classes of supported input sizes:
  - Non-axis, non-batch dimensions are all 1 and the axis dimension is at most 8192. This is the recommended case for using softmax since it is the most accurate.
  - At least one non-axis, non-batch dimension greater than 1 and the axis dimension is at most 1024. Note that in this case, there may be some approximation error as the axis dimension size approaches the upper bound. See the TensorRT Developer Guide for more details on the approximation error.

6.3.10 IConcatenationLayer

class tensorrt.**IConcatenationLayer**

A concatenation layer in an *INetworkDefinition* .

The output channel size is the sum of the channel sizes of the inputs. The other output sizes are the same as the other input sizes, which must all match.

Variables

axis – int The axis along which concatenation occurs. The default axis is the number of tensor dimensions minus three, or zero if the tensor has fewer than three dimensions. For example, for a tensor with dimensions NCHW, it is C.

6.3.11 IDeconvolutionLayer

class tensorrt.**IDeconvolutionLayer**

A deconvolution layer in an *INetworkDefinition* .

Variables

- **num_output_maps** – int The number of output feature maps for the deconvolution.
- **pre_padding** – *DimsHW* The pre-padding. The start of input will be zero-padded by this number of elements in the height and width directions. Default: (0, 0)
- **post_padding** – *DimsHW* The post-padding. The end of input will be zero-padded by this number of elements in the height and width directions. Default: (0, 0)
- **padding_mode** – *PaddingMode* The padding mode. Padding mode takes precedence if both IDeconvolutionLayer.padding_mode and either IDeconvolutionLayer.pre_padding or IDeconvolutionLayer.post_padding are set.

- **num_groups** – int The number of groups for a deconvolution. The input tensor channels are divided into this many groups, and a deconvolution is executed for each group, using a filter per group. The results of the group convolutions are concatenated to form the output. **Note** When using groups in int8 mode, the size of the groups (i.e. the channel count divided by the group count) must be a multiple of 4 for both input and output. Default: 1
- **kernel** – *Weights* The kernel weights for the deconvolution. The weights are specified as a contiguous array in *CKRS* order, where *C* the number of input channels, *K* the number of output feature maps, and *R* and *S* are the height and width of the filter.
- **bias** – *Weights* The bias weights for the deconvolution. Bias is optional. To omit bias, set this to an empty *Weights* object. The bias is applied per-feature-map, so the number of weights (if non-zero) must be equal to the number of output feature maps.
- **kernel_size_nd** – *Dims* The multi-dimension kernel size of the convolution.
- **stride_nd** – *Dims* The multi-dimension stride of the deconvolution. Default: (1, . . . , 1)
- **padding_nd** – *Dims* The multi-dimension padding of the deconvolution. The input will be zero-padded by this number of elements in each dimension. Padding is symmetric. Default: (0, . . . , 0)

6.3.12 IElementWiseLayer

tensorrt.**ElementWiseOperation**

The binary operations that may be performed by an ElementWise layer.

Members:

SUM : Sum of the two elements

PROD : Product of the two elements

MAX : Max of the two elements

MIN : Min of the two elements

SUB : Subtract the second element from the first

DIV : Divide the first element by the second

POW : The first element to the power of the second element

FLOOR_DIV : Floor division of the first element by the second

AND : Logical AND of two elements

OR : Logical OR of two elements

XOR : Logical XOR of two elements

EQUAL : Check if two elements are equal

GREATER : Check if element in first tensor is greater than corresponding element in second tensor

LESS : Check if element in first tensor is less than corresponding element in second tensor

class tensorrt.**IElementWiseLayer**

A elementwise layer in an *INetworkDefinition* .

This layer applies a per-element binary operation between corresponding elements of two tensors.

The input dimensions of the two input tensors must be equal, and the output tensor is the same size as each input.

Variables
op – *ElementWiseOperation* The binary operation for the layer.

6.3.13 IGatherLayer

class tensorrt.**IGatherLayer**

A gather layer in an *INetworkDefinition* .

Variables

- **axis** – int The non-batch dimension axis to gather on. The axis must be less than the number of non-batch dimensions in the data input.
- **num_elementwise_dims** – int The number of leading dimensions of indices tensor to be handled elementwise. For *GatherMode.DEFAULT*, it can be 0 or 1. For *GatherMode::kND*, it can be between 0 and one less than rank(data). For *GatherMode::kELEMENT*, it must be 0.
- **mode** – GatherMode The gather mode.

6.3.14 IPluginV2Layer

class tensorrt.**IPluginV2Layer**

A plugin layer in an *INetworkDefinition* .

Variables
plugin – IPluginV2 The plugin for the layer.

6.3.15 IPluginV3Layer

class tensorrt.**IPluginV3Layer**

A plugin layer in an *INetworkDefinition* .

Variables
plugin – *IPluginV3* The plugin for the layer.

6.3.16 IUnaryLayer

tensorrt.**UnaryOperation**

The unary operations that may be performed by a Unary layer.

Members:

EXP : Exponentiation

LOG : Log (base e)

SQRT : Square root

RECIP : Reciprocal

ABS : Absolute value

NEG : Negation

SIN : Sine

COS : Cosine

TAN : Tangent

SINH : Hyperbolic sine

COSH : Hyperbolic cosine

ASIN : Inverse sine

ACOS : Inverse cosine

ATAN : Inverse tangent

ASINH : Inverse hyperbolic sine

ACOSH : Inverse hyperbolic cosine

ATANH : Inverse hyperbolic tangent

CEIL : Ceiling

FLOOR : Floor

ERF : Gauss error function

NOT : Not

SIGN : Sign. If input > 0, output 1; if input < 0, output -1; if input == 0, output 0.

ROUND : Round to nearest even for floating-point data type.

ISINF : Return true if the input value equals +/- infinity for floating-point data type.

ISNAN : Return true if the input value equals NaN for floating-point data type.

class tensorrt.**IUnaryLayer**

A unary layer in an *INetworkDefinition* .

Variables

op – *UnaryOperation* The unary operation for the layer. When running this layer on DLA, only `UnaryOperation.ABS` is supported.

6.3.17 IReduceLayer

tensorrt.**ReduceOperation**

The reduce operations that may be performed by a Reduce layer

Members:

SUM : Sum of the elements

PROD : Product of the elements

MAX : Maximum of the elements

MIN : Minimum of the elements

AVG : Average of the elements

NONE : No reduction

class tensorrt.**IReduceLayer**

A reduce layer in an *INetworkDefinition* .

Variables

- **op** – *ReduceOperation* The reduce operation for the layer.
- **axes** – int The axes over which to reduce.
- **keep_dims** – bool Specifies whether or not to keep the reduced dimensions for the layer.

6.3.18 IPaddingLayer

class tensorrt.**IPaddingLayer**

A padding layer in an *INetworkDefinition* .

Variables

- **pre_padding_nd** – *Dims* The padding that is applied at the start of the tensor. Negative padding results in trimming the edge by the specified amount. Only 2 dimensions currently supported.
- **post_padding_nd** – *Dims* The padding that is applied at the end of the tensor. Negative padding results in trimming the edge by the specified amount. Only 2 dimensions currently supported.

6.3.19 IParametricReLULayer

class tensorrt.**IParametricReLULayer**

A parametric ReLU layer in an *INetworkDefinition* .

This layer applies a parametric ReLU activation to an input tensor (first input), with slopes taken from a slopes tensor (second input). This can be viewed as a leaky ReLU operation where the negative slope differs from element to element (and can in fact be learned).

The slopes tensor must be unidirectional broadcastable to the input tensor: the rank of the two tensors must be the same, and all dimensions of the slopes tensor must either equal the input tensor or be 1. The output tensor has the same shape as the input tensor.

6.3.20 ISelectLayer

class tensorrt.**ISelectLayer**

A select layer in an *INetworkDefinition* .

This layer implements an element-wise ternary conditional operation. Wherever `condition` is `True`, elements are taken from the first input, and wherever `condition` is `False`, elements are taken from the second input.

6.3.21 IShuffleLayer

class tensorrt.**Permutation**(**args*, ***kwargs*)

The elements of the permutation. The permutation is applied as outputDimensionIndex = permutation[inputDimensionIndex], so to permute from CHW order to HWC order, the required permutation is [1, 2, 0], and to permute from HWC to CHW, the required permutation is [2, 0, 1].

It supports iteration and indexing and is implicitly convertible to/from Python iterables (like `tuple` or `list`). Therefore, you can use those classes in place of *Permutation* .

Overloaded function.

1. __init__(self: tensorrt.tensorrt.Permutation) -> None

2. __init__(self: tensorrt.tensorrt.Permutation, arg0: collections.abc.Sequence[typing.SupportsInt]) -> None

class tensorrt.**IShuffleLayer**

A shuffle layer in an *INetworkDefinition* .

This class shuffles data by applying in sequence: a transpose operation, a reshape operation and a second transpose operation. The dimension types of the output are those of the reshape dimension.

Variables

- **first_transpose** – *Permutation* The permutation applied by the first transpose operation. Default: Identity Permutation
- **reshape_dims** – *Dims* The reshaped dimensions. Two special values can be used as dimensions. Value 0 copies the corresponding dimension from input. This special value can be used more than once in the dimensions. If number of reshape dimensions is less than input, 0s are resolved by aligning the most significant dimensions of input. Value -1 infers that particular dimension by looking at input and rest of the reshape dimensions. Note that only a maximum of one dimension is permitted to be specified as -1. The product of the new dimensions must be equal to the product of the old.
- **second_transpose** – *Permutation* The permutation applied by the second transpose operation. Default: Identity Permutation
- **zero_is_placeholder** – bool The meaning of 0 in reshape dimensions. If true, then a 0 in the reshape dimensions denotes copying the corresponding dimension from the first input tensor. If false, then a 0 in the reshape dimensions denotes a zero-length dimension.

set_input(*self:* tensorrt.tensorrt.ILayer, *index: SupportsInt*, *tensor:* tensorrt.tensorrt.ITensor) → None

Sets the input tensor for the given index. The index must be 0 for a static shuffle layer. A static shuffle layer is converted to a dynamic shuffle layer by calling *set_input()* with an index 1. A dynamic shuffle layer cannot be converted back to a static shuffle layer.

For a dynamic shuffle layer, the values 0 and 1 are valid. The indices in the dynamic case are as follows:

Index	Description
0	Data or Shape tensor to be shuffled.
1	The dimensions for the reshape operation, as a 1D int32 shape tensor.

If this function is called with a value 1, then num_inputs changes from 1 to 2.

Parameters

- **index** – The index of the input tensor.
- **tensor** – The input tensor.

6.3.22 ISliceLayer

class tensorrt.**ISliceLayer**

A slice layer in an *INetworkDefinition* .

The slice layer has two variants, static and dynamic. Static slice specifies the start, size, and stride dimensions at layer creation time via *Dims* and can use the get/set accessor functions of the *ISliceLayer* . Dynamic slice specifies one or more of start, size, stride, or axes as ITensor`s, by using :func:`ILayer.set_input to add a second, third, fourth, or sixth input respectively. The corresponding *Dims* are used if an input is missing or null.

An application can determine if the *ISliceLayer* has a dynamic output shape based on whether the size or axes input is present and non-null.

The slice layer selects for each dimension a start location from within the input tensor, and copies elements to the output tensor using the specified stride across the input tensor. Start, size, and stride tensors must be 1-D integer-typed shape tensors if not specified via *Dims* .

An example of using slice on a tensor: input = {{0, 2, 4}, {1, 3, 5}} start = {1, 0} size = {1, 2} stride = {1, 2} output = {{1, 5}}

If axes is provided then starts, ends, and strides must have the same length as axes and specifies a subset of dimensions to slice. If axes is not provided, starts, ends, and strides must be of the same length as the rank of the input tensor.

An example of using slice on a tensor with axes specified: input = {{0, 2, 4}, {1, 3, 5}} start = {1} size = {2} stride = {1} axes = {1} output = {{2, 4}, {3, 5}}

When the sampleMode is `SampleMode.CLAMP` or `SampleMode.REFLECT` , for each input dimension, if its size is 0 then the corresponding output dimension must be 0 too.

When the sampleMode is `SampleMode.FILL`, the fifth input to the slice layer is used to determine the value to fill in out-of-bound indices. It is an error to specify the fifth input in any other sample mode.

A slice layer can produce a shape tensor if the following conditions are met:

- `start`, `size`, and `stride` are build time constants, either as static *Dims* or as constant input tensors.
- `axes`, if provided, is a build time constant, either as static *Dims* or as a constant input tensor.
- The number of elements in the output tensor does not exceed 2 * `Dims.MAX_DIMS` .

The input tensor is a shape tensor if the output is a shape tensor.

The following constraints must be satisfied to execute this layer on DLA: * `start`, `size`, and `stride` are build time constants, either as static *Dims* or as constant input tensors. * `axes`, if provided, is a build time constant, either as static *Dims* or as a constant input tensor. * sampleMode is `SampleMode.DEFAULT` , `SampleMode.WRAP` , or `SampleMode.FILL` . * Strides are 1 for all dimensions. * Slicing is not performed on the first dimension. * The input tensor has four dimensions. * For `SliceMode.FILL` , the fill value input is a scalar output of an *IConstantLayer* with value 0 that is not consumed by any other layer.

Variables

- **start** – *Dims* The start offset.
- **shape** – *Dims* The output dimensions.
- **stride** – *Dims* The slicing stride.
- **mode** – *SampleMode* Controls how *ISliceLayer* handles out of bounds coordinates.
- **axes** – *Dims* The axes that starts, sizes, and strides correspond to.

set_input(*self:* tensorrt.tensorrt.ILayer, *index: SupportsInt*, *tensor:* tensorrt.tensorrt.ITensor) → None

Sets the input tensor for the given index. The index must be 0 or 4 for a static slice layer. A static slice layer is converted to a dynamic slice layer by calling *set_input()* with an index between 1 and 3. A dynamic slice layer cannot be converted back to a static slice layer.

The indices are as follows:

Index	Description
0	Data or Shape tensor to be sliced.
1	The start tensor to begin slicing, N-dimensional for Data, and 1-D for Shape.
2	The size tensor of the resulting slice, N-dimensional for Data, and 1-D for Shape.
3	The stride of the slicing operation, N-dimensional for Data, and 1-D for Shape.
4	Value for the `SampleMode.FILL` slice mode. Disallowed for other modes.
5	The axes tensor indicating the axes that starts, sizes, and strides correspond to. Must be a 1-D tensor.

If this function is called with a value greater than 0, then `num_inputs` changes from 1 to index + 1.

Parameters

- **index** – The index of the input tensor.
- **tensor** – The input tensor.

6.3.23 IShapeLayer

class tensorrt.**IShapeLayer**

A shape layer in an *INetworkDefinition* . Used for getting the shape of a tensor. This class sets the output to a one-dimensional tensor with the dimensions of the input tensor.

For example, if the input is a four-dimensional tensor (of any type) with dimensions [2,3,5,7], the output tensor is a one-dimensional `int64` tensor of length 4 containing the sequence 2, 3, 5, 7.

6.3.24 ITopKLayer

tensorrt.**TopKOperation**

The operations that may be performed by a TopK layer

Members:

MAX : Maximum of the elements

MIN : Minimum of the elements

class tensorrt.**ITopKLayer**

A TopK layer in an *INetworkDefinition* .

Variables

- **op** – *TopKOperation* The operation for the layer.
- **k** – *TopKOperation* the k value for the layer. Currently only values up to 3840 are supported. Use the set_input() method with index 1 to pass in dynamic k as a tensor.
- **axes** – *TopKOperation* The axes along which to reduce.
- **indices_type** – *DataType* The specified data type of the output indices tensor. Must be tensorrt.int32 or tensorrt.int64.

set_input(*self:* tensorrt.tensorrt.ILayer, *index: SupportsInt*, *tensor:* tensorrt.tensorrt.ITensor) → None

Sets the input tensor for the given index. The index must be 0 or 1 for a TopK layer.

The indices are as follows:

Index	Description
0	Input data tensor.
1	**A scalar Int32 tensor containing a positive value corresponding to the number** of top elements to retrieve. Values larger than 3840 will result in a runtime error. If provided, this will override the static k value in calculations.

Parameters

- **index** – The index of the input tensor.
- **tensor** – The input tensor.

6.3.25 IMatrixMultiplyLayer

tensorrt.**MatrixOperation**

The matrix operations that may be performed by a Matrix layer

Members:

NONE :

TRANSPOSE : Transpose each matrix

VECTOR : Treat operand as collection of vectors

class tensorrt.**IMatrixMultiplyLayer**

A matrix multiply layer in an *INetworkDefinition* .

Let A be op(getInput(0)) and B be op(getInput(1)) where op(x) denotes the corresponding MatrixOperation.

When A and B are matrices or vectors, computes the inner product A * B:

matrix * matrix -> matrix
matrix * vector -> vector
vector * matrix -> vector
vector * vector -> scalar

Inputs of higher rank are treated as collections of matrices or vectors. The output will be a corresponding collection of matrices, vectors, or scalars.

Variables

- **op0** – *MatrixOperation* How to treat the first input.
- **op1** – *MatrixOperation* How to treat the second input.

6.3.26 IRaggedSoftMaxLayer

class tensorrt.**IRaggedSoftMaxLayer**

A ragged softmax layer in an *INetworkDefinition* .

This layer takes a ZxS input tensor and an additional Zx1 bounds tensor holding the lengths of the Z sequences.

This layer computes a softmax across each of the Z sequences.

The output tensor is of the same size as the input tensor.

6.3.27 IIdentityLayer

class tensorrt.**IIdentityLayer**

A layer that represents the identity function.

If tensor precision is explicitly specified, it can be used to transform from one precision to another.

Other than conversions between the same type (`float32` -> `float32` for example), the only valid conversions are:

(`float32` | `float16` | `int32` | `bool`) -> (`float32` | `float16` | `int32` | `bool`)

(`float32` | `float16`) -> `uint8`

`uint8` -> (`float32` | `float16`)

6.3.28 IConstantLayer

class tensorrt.**IConstantLayer**

A constant layer in an *INetworkDefinition* .

Note: This layer does not support boolean and uint8 types.

Variables

- **weights** – *Weights* The weights for the layer.
- **shape** – *Dims* The shape of the layer.

6.3.29 IResizeLayer

class tensorrt.**IResizeLayer**

A resize layer in an *INetworkDefinition* .

Resize layer can be used for resizing a N-D tensor.

Resize layer currently supports the following configurations:

- InterpolationMode.NEAREST - resizes innermost *m* dimensions of N-D, where 0 < m <= min(3, N) and N > 0.
- InterpolationMode.LINEAR - resizes innermost *m* dimensions of N-D, where 0 < m <= min(3, N) and N > 0.
- InterpolationMode.CUBIC - resizes innermost *2* dimensions of N-D, N >= 2.

Default resize mode is InterpolationMode.NEAREST.

Resize layer provides two ways to resize tensor dimensions:

- **Set output dimensions directly. It can be done for static as well as dynamic resize layer.**
  Static resize layer requires output dimensions to be known at build-time. Dynamic resize layer requires output dimensions to be set as one of the input tensors.
- **Set scales for resize. Each output dimension is calculated as floor(input dimension * scale).**
  Only static resize layer allows setting scales where the scales are known at build-time.

If executing this layer on DLA, the following combinations of parameters are supported:

- In NEAREST mode:
  - (ResizeCoordinateTransformation.ASYMMETRIC, ResizeSelector.FORMULA, ResizeRoundMode.FLOOR)
  - (ResizeCoordinateTransformation.HALF_PIXEL, ResizeSelector.FORMULA, ResizeRoundMode.HALF_DOWN)
  - (ResizeCoordinateTransformation.HALF_PIXEL, ResizeSelector.FORMULA, ResizeRoundMode.HALF_UP)
- In LINEAR and CUBIC mode:
  - (ResizeCoordinateTransformation.HALF_PIXEL, ResizeSelector.FORMULA)
  - (ResizeCoordinateTransformation.HALF_PIXEL, ResizeSelector.UPPER)

Variables

- **shape** – *Dims* The output dimensions. Must to equal to input dimensions size.
- **scales** – `List[float]` List of resize scales. If executing this layer on DLA, there are three restrictions: 1. `len(scales)` has to be exactly 4. 2. The first two elements in scales need to be exactly 1 (for unchanged batch and channel dimensions). 3. The last two elements in scales, representing the scale values along height and width dimensions, respectively, need to be integer values in the range of [1, 32] for NEAREST mode and [1, 4] for LINEAR. Example of DLA-supported scales: [1, 1, 2, 2].
- **resize_mode** – *InterpolationMode* Resize mode can be Linear, Cubic or Nearest.
- **coordinate_transformation** – `ResizeCoordinateTransformationDoc` Supported resize coordinate transformation modes are ALIGN_CORNERS, ASYMMETRIC and HALF_PIXEL.
- **selector_for_single_pixel** – `ResizeSelector` Supported resize selector modes are FORMULA and UPPER.
- **nearest_rounding** – `ResizeRoundMode` Supported resize Round modes are HALF_UP, HALF_DOWN, FLOOR and CEIL.
- **exclude_outside** – `int` If set to 1, the weight of sampling locations outside the input tensor will be set to 0, and the weight will be renormalized so that their sum is 1.0.
- **cubic_coeff** – `float` coefficient 'a' used in cubic interpolation.

set_input(*self:* tensorrt.tensorrt.ILayer, *index: SupportsInt*, *tensor:* tensorrt.tensorrt.ITensor) → None

Sets the input tensor for the given index.

If index == 1 and num_inputs == 1, num_inputs changes to 2. Once such additional input is set, resize layer works in dynamic mode. When index == 1 and num_inputs == 1, the output dimensions are used from the input tensor, overriding the dimensions supplied by *shape*.

Parameters

- **index** – The index of the input tensor.
- **tensor** – The input tensor.

6.3.30 ILoop

class tensorrt.**ILoop**

Helper for creating a recurrent subgraph.

Variables
name – The name of the loop. The name is used in error diagnostics.

add_iterator(*self:* tensorrt.tensorrt.ILoop, *tensor:* tensorrt.tensorrt.ITensor, *axis: SupportsInt = 0*, *reverse: bool = False*) → *tensorrt.tensorrt.IIteratorLayer*

Return layer that subscripts tensor by loop iteration.

For reverse=false, this is equivalent to add_gather(tensor, I, 0) where I is a scalar tensor containing the loop iteration number. For reverse=true, this is equivalent to add_gather(tensor, M-1-I, 0) where M is the trip count computed from TripLimits of kind COUNT.

Parameters

- **tensor** – The tensor to iterate over.
- **axis** – The axis along which to iterate.
- **reverse** – Whether to iterate in the reverse direction.

Returns
The *IIteratorLayer* , or None if it could not be created.

add_loop_output(*self:* tensorrt.tensorrt.ILoop, *tensor:* tensorrt.tensorrt.ITensor, *kind: tensorrt.tensorrt.LoopOutput*, *axis: SupportsInt = 0*) → *tensorrt.tensorrt.ILoopOutputLayer*

Make an output for this loop, based on the given tensor.

If kind is CONCATENATE or REVERSE, a second input specifying the concatenation dimension must be added via method *ILoopOutputLayer.set_input()* .

Parameters

- **kind** – The kind of loop output. See *LoopOutput*
- **axis** – The axis for concatenation (if using kind of CONCATENATE or REVERSE).

Returns
The added *ILoopOutputLayer* , or None if it could not be created.

add_recurrence(*self:* tensorrt.tensorrt.ILoop, *initial_value:* tensorrt.tensorrt.ITensor) → *tensorrt.tensorrt.IRecurrenceLayer*

Create a recurrence layer for this loop with initial_value as its first input.

Parameters
initial_value – The initial value of the recurrence layer.

Returns
The added *IRecurrenceLayer* , or None if it could not be created.

add_trip_limit(*self:* tensorrt.tensorrt.ILoop, *tensor:* tensorrt.tensorrt.ITensor, *kind:* *tensorrt.tensorrt.TripLimit*) → *tensorrt.tensorrt.ITripLimitLayer*

Add a trip-count limiter, based on the given tensor.

There may be at most one `COUNT` and one `WHILE` limiter for a loop. When both trip limits exist, the loop exits when the count is reached or condition is falsified. It is an error to not add at least one trip limiter.

For `WHILE`, the input tensor must be the output of a subgraph that contains only layers that are not *ITripLimitLayer* , *IIteratorLayer* or *ILoopOutputLayer* . Any *IRecurrenceLayer* s in the subgraph must belong to the same loop as the *ITripLimitLayer* . A trivial example of this rule is that the input to the `WHILE` is the output of an *IRecurrenceLayer* for the same loop.

Parameters

- **tensor** – The input tensor. Must be available before the loop starts.
- **kind** – The kind of trip limit. See *TripLimit*

Returns
The added *ITripLimitLayer* , or `None` if it could not be created.

6.3.30.1 ILoopBoundaryLayer

class tensorrt.**ILoopBoundaryLayer**

Variables
loop – *ILoop* associated with this boundary layer.

6.3.30.1.1 ITripLimitLayer

tensorrt.**TripLimit**

Describes kinds of trip limits.

Members:

COUNT : Tensor is a scalar of type `int32` that contains the trip count.

WHILE : Tensor is a scalar of type `bool`. Loop terminates when its value is false.

class tensorrt.**ITripLimitLayer**

Variables
kind – The kind of trip limit. See *TripLimit*

6.3.30.1.2 IRecurrenceLayer

class tensorrt.**IRecurrenceLayer**

set_input(*self:* tensorrt.tensorrt.ILayer, *index: SupportsInt*, *tensor:* tensorrt.tensorrt.ITensor) → None

Set the first or second input. If index==1 and the number of inputs is one, the input is appended. The first input specifies the initial output value, and must come from outside the loop. The second input specifies the next output value, and must come from inside the loop. The two inputs must have the same dimensions.

Parameters

- **index** – The index of the input to set.
- **tensor** – The input tensor.

6.3.30.1.3 IIteratorLayer

class tensorrt.**IIteratorLayer**

Variables

- **axis** – The axis to iterate over
- **reverse** – For reverse=false, the layer is equivalent to add_gather(tensor, I, 0) where I is a scalar tensor containing the loop iteration number. For reverse=true, the layer is equivalent to add_gather(tensor, M-1-I, 0) where M is the trip count computed from TripLimits of kind COUNT. The default is reverse=false.

6.3.30.1.4 ILoopOutputLayer

tensorrt.**LoopOutput**

Describes kinds of loop outputs.

Members:

LAST_VALUE : Output value is value of tensor for last iteration.

CONCATENATE : Output value is concatenation of values of tensor for each iteration, in forward order.

REVERSE : Output value is concatenation of values of tensor for each iteration, in reverse order.

class tensorrt.**ILoopOutputLayer**

An *ILoopOutputLayer* is the sole way to get output from a loop.

The first input tensor must be defined inside the loop; the output tensor is outside the loop. The second input tensor, if present, must be defined outside the loop.

If kind is LAST_VALUE, a single input must be provided.

If kind is CONCATENATE or REVERSE, a second input must be provided. The second input must be a scalar "shape tensor", defined before the loop commences, that specifies the concatenation length of the output.

The output tensor has j more dimensions than the input tensor, where j == 0 if kind is LAST_VALUE j == 1 if kind is CONCATENATE or REVERSE.

Variables

- **axis** – The contenation axis. Ignored if kind is LAST_VALUE. For example, if the input tensor has dimensions [b,c,d], and kind is CONCATENATE, the output has four dimensions. Let a be the value of the second input. axis=0 causes the output to have dimensions [a,b,c,d]. axis=1 causes the output to have dimensions [b,a,c,d]. axis=2 causes the output to have dimensions [b,c,a,d]. axis=3 causes the output to have dimensions [b,c,d,a]. Default is axis is 0.
- **kind** – The kind of loop output. See *LoopOutput*

set_input(*self:* tensorrt.tensorrt.ILayer, *index: SupportsInt*, *tensor:* tensorrt.tensorrt.ITensor) → None

Like *ILayer.set_input()*, but additionally works if index==1, num_inputs`==1, in which case :attr:`num_inputs changes to 2.

6.3.31 IFillLayer

tensorrt.**FillOperation**

The tensor fill operations that may performed by an Fill layer.

Members:

LINSPACE : Generate evenly spaced numbers over a specified interval

RANDOM_UNIFORM : Generate a tensor with random values drawn from a uniform distribution

RANDOM_NORMAL : Generate a tensor with random values drawn from a normal distribution

class tensorrt.**IFillLayer**

A fill layer in an *INetworkDefinition* .

The data type of the output tensor can be specified by `to_type`. Supported output types for each fill operation is as follows.

Operation	to_type
kLINSPACE	int32, int64, float32
kRANDOM_UNIFORM	float16, float32
kRANDOM_NORMAL	float16, float32

Variables

to_type – *DataType* The specified data type of the output tensor. Defaults to tensorrt.float32.

is_alpha_beta_int64(*self:* tensorrt.tensorrt.IFillLayer) → bool

set_input(*self:* tensorrt.tensorrt.ILayer, *index: SupportsInt*, *tensor:* tensorrt.tensorrt.ITensor) → None

replace an input of this layer with a specific tensor.

Index	Description for kLINSPACE
0	Shape tensor, represents the output tensor's dimensions.
1	Start, a scalar, represents the start value.
2	Delta, a 1D tensor, length equals to shape tensor's nbDims, represents the delta value for each dimension.

Index	Description for kRANDOM_UNIFORM
0	Shape tensor, represents the output tensor's dimensions.
1	Minimum, a scalar, represents the minimum random value.
2	Maximum, a scalar, represents the maximal random value.

Index	Description for kRANDOM_NORMAL
0	Shape tensor, represents the output tensor's dimensions.
1	Mean, a scalar, represents the mean of the normal distribution.
2	Scale, a scalar, represents the standard deviation of the normal distribution.

Parameters

- **index** – the index of the input to modify.
- **tensor** – the input tensor.

6.3.32 IQuantizeLayer

class tensorrt.**IQuantizeLayer**

A Quantize layer in an *INetworkDefinition* .

This layer accepts a floating-point data input tensor, and uses the scale and zeroPt inputs to

quantize the data to an 8-bit signed integer according to:

$output = clamp(round(input/scale) + zeroPt)$

Rounding type is rounding-to-nearest ties-to-even (https://en.wikipedia.org/wiki/Rounding#Round_half_to_even).

Clamping is in the range [-128, 127].

The first input (index 0) is the tensor to be quantized. The second (index 1) and third (index 2) are the scale and zero point respectively. Each of scale and zeroPt must be either a scalar, or a 1D tensor.

The zeroPt tensor is optional, and if not set, will be assumed to be zero. Its data type must be tensorrt.int8. zeroPt must only contain zero-valued coefficients, because only symmetric quantization is supported. The scale value must be either a scalar for per-tensor quantization, or a 1D tensor for per-axis quantization. The size of the 1-D scale tensor must match the size of the quantization axis. The size of the scale must match the size of the zeroPt.

The subgraph which terminates with the scale tensor must be a build-time constant. The same restrictions apply to the zeroPt. The output type, if constrained, must be constrained to tensorrt.int8 or tensorrt.fp8. The input type, if constrained, must be constrained to tensorrt.float32, tensorrt.float16 or tensorrt.bfloat16. The output size is the same as the input size.

IQuantizeLayer supports tensorrt.float32, tensorrt.float16 and tensorrt.bfloat16 precision and will default to tensorrt.float32 precision during instantiation. IQuantizeLayer supports tensorrt.int8, tensorrt.float8, tensorrt.int4 and tensorrt.fp4 output.

Variables

- **axis** – int The axis along which quantization occurs. The quantization axis is in reference to the input tensor's dimensions.
- **to_type** – *DataType* The specified data type of the output tensor. Must be tensorrt.int8 or tensorrt.float8.

6.3.33 IDequantizeLayer

class tensorrt.**IDequantizeLayer**

A Dequantize layer in an *INetworkDefinition* .

This layer accepts a signed 8-bit integer input tensor, and uses the configured scale and zeroPt inputs to dequantize the input according to: $output = (input - zeroPt) * scale$

The first input (index 0) is the tensor to be quantized. The second (index 1) and third (index 2) are the scale and zero point respectively. Each of scale and zeroPt must be either a scalar, or a 1D tensor.

The zeroPt tensor is optional, and if not set, will be assumed to be zero. Its data type must be tensorrt.int8. zeroPt must only contain zero-valued coefficients, because only symmetric quantization is supported. The scale value

must be either a scalar for per-tensor quantization, or a 1D tensor for per-axis quantization. The size of the 1-D scale tensor must match the size of the quantization axis. The size of the scale must match the size of the zeroPt.

The subgraph which terminates with the scale tensor must be a build-time constant. The same restrictions apply to the zeroPt. The output type, if constrained, must be constrained to tensorrt.int8 or tensorrt.fp8. The input type, if constrained, must be constrained to tensorrt.float32, tensorrt.float16 or tensorrt.bfloat16. The output size is the same as the input size.

IDequantizeLayer supports tensorrt.int8, tensorrt.float8, tensorrt.int4 and tensorrt.fp4 precision and will default to tensorrt.int8 precision during instantiation. IDequantizeLayer supports tensorrt.float32, tensorrt.float16 and tensorrt.bfloat16 output.

Variables

- **axis** – `int` The axis along which dequantization occurs. The dequantization axis is in reference to the input tensor's dimensions.
- **to_type** – *DataType* The specified data type of the output tensor. Must be tensorrt.float32 or tensorrt.float16.

6.3.34 IDynamicQuantizeLayer

class tensorrt.**IDynamicQuantizeLayer**

A DynamicQuantize layer in an *INetworkDefinition* .

This layer performs dynamic block quantization of its input tensor and outputs the quantized data and the computed block scale-factors. The size of the blocked axis must be divisible by the block size.

The first input (index 0) is the tensor to be quantized. Its data type must be one of DataType::kFLOAT, DataType::kHALF, or DataType::kBF16. Currently only 2D and 3D inputs are supported.

The second input (index 1) is the double quantization scale factor. It is a scalar scale factor used to quantize the computed block scales-factors.

Variables

- **axis** – `int` The axis that is sliced into blocks. The axis must be the last dimension or the second to last dimension.
- **block_size** – `int` The number of elements that are quantized using a shared scale factor. Supports block sizes of 16 with NVFP4 quantization and 32 with MXFP8 quantization.
- **output_type** – *DataType* The data type of the quantized output tensor, must be either DataType::kFP4 (NVFP4 quantization) or DataType::kFP8 (MXFP8 quantization).
- **scale_type** – *DataType* The data type of the scale factor used for quantizing the input data, must be DataType::kFP8 (NVFP4 quantization) or DataType::kE8M0 (MXFP8 quantization).

6.3.35 IScatterLayer

class tensorrt.**IScatterLayer**

A Scatter layer as in *INetworkDefinition*. :ivar axis: axis to scatter on when using Scatter Element mode (ignored in ND mode) :ivar mode: `ScatterMode` The operation mode of the scatter.

6.3.36 IIfConditional

class tensorrt.**IIfConditional**

Helper for constructing conditionally-executed subgraphs.

An If-conditional conditionally executes (lazy evaluation) part of the network according to the following pseudo-code:

```
If condition is true Then:
    output = trueSubgraph(trueInputs);
Else:
    output = falseSubgraph(falseInputs);
Emit output
```

Condition is a 0D boolean tensor (representing a scalar). trueSubgraph represents a network subgraph that is executed when condition is evaluated to True. falseSubgraph represents a network subgraph that is executed when condition is evaluated to False.

The following constraints apply to If-conditionals: - Both the trueSubgraph and falseSubgraph must be defined. - The number of output tensors in both subgraphs is the same. - The type and shape of each output tensor from the true/false subgraphs are the same, except that the shapes are allowed to differ if the condition is a build-time constant.

add_input(*self:* tensorrt.tensorrt.IIfConditional, *input:* tensorrt.tensorrt.ITensor) → *tensorrt.tensorrt.IIfConditionalInputLayer*

Make an input for this if-conditional, based on the given tensor.

Parameters

input – An input to the conditional that can be used by either or both of the conditional's subgraphs.

add_output(*self:* tensorrt.tensorrt.IIfConditional, *true_subgraph_output:* tensorrt.tensorrt.ITensor, *false_subgraph_output:* tensorrt.tensorrt.ITensor) → *tensorrt.tensorrt.IIfConditionalOutputLayer*

Make an output for this if-conditional, based on the given tensors.

Each output layer of the if-conditional represents a single output of either the true-subgraph or the false-subgraph of the if-conditional, depending on which subgraph was executed.

Parameters

- **true_subgraph_output** – The output of the subgraph executed when this conditional's condition input evaluates to true.
- **false_subgraph_output** – The output of the subgraph executed when this conditional's condition input evaluates to false.

Returns

The *IIfConditionalOutputLayer* , or `None` if it could not be created.

set_condition(*self:* tensorrt.tensorrt.IIfConditional, *condition:* tensorrt.tensorrt.ITensor) → *tensorrt.tensorrt.IConditionLayer*

Set the condition tensor for this If-Conditional construct.

The `condition` tensor must be a 0D data tensor (scalar) with type `bool`.

Parameters
condition – The condition tensor that will determine which subgraph to execute.

Returns
The *IConditionLayer* , or `None` if it could not be created.

6.3.37 IConditionLayer

class tensorrt.**IConditionLayer**

Describes the boolean condition of an if-conditional.

6.3.38 IIfConditionalOutputLayer

class tensorrt.**IIfConditionalOutputLayer**

Describes kinds of if-conditional outputs.

6.3.39 IIfConditionalInputLayer

class tensorrt.**IIfConditionalInputLayer**

Describes kinds of if-conditional inputs.

6.3.40 IEinsumLayer

class tensorrt.**IEinsumLayer**

An Einsum layer in an *INetworkDefinition* .

This layer implements a summation over the elements of the inputs along dimensions specified by the equation parameter, based on the Einstein summation convention. The layer can have one or more inputs of rank >= 0. All the inputs must be of same data type. This layer supports all TensorRT data types except `bool`. There is one output tensor of the same type as the input tensors. The shape of output tensor is determined by the equation.

The equation specifies ASCII lower-case letters for each dimension in the inputs in the same order as the dimensions, separated by comma for each input. The dimensions labeled with the same subscript must match or be broadcastable. Repeated subscript labels in one input take the diagonal. Repeating a label across multiple inputs means that those axes will be multiplied. Omitting a label from the output means values along those axes will be summed. In implicit mode, the indices which appear once in the expression will be part of the output in increasing alphabetical order. In explicit mode, the output can be controlled by specifying output subscript labels by adding an arrow ('->') followed by subscripts for the output. For example, "ij,jk->ik" is equivalent to "ij,jk". Ellipsis ('...') can be used in place of subscripts to broadcast the dimensions. See the TensorRT Developer Guide for more details on equation syntax.

Many common operations can be expressed using the Einsum equation. For example: Matrix Transpose: ij->ji Sum: ij-> Matrix-Matrix Multiplication: ik,kj->ij Dot Product: i,i-> Matrix-Vector Multiplication: ik,k->i Batch Matrix Multiplication: ijk,ikl->ijl Batch Diagonal: ...ii->...i

Note that TensorRT does not support ellipsis or diagonal operations.

Variables
equation – `str` The Einsum equation of the layer. The equation is a comma-separated list of subscript labels, where each label refers to a dimension of the corresponding tensor.

6.3.41 IAssertionLayer

class tensorrt.**IAssertionLayer**

An assertion layer in an *INetworkDefinition* .

This layer implements assertions. The input must be a boolean shape tensor. If any element of it is `False`, a build-time or run-time error occurs. Asserting equality of input dimensions may help the optimizer.

Variables
message – `string` Message to print if the assertion fails.

6.3.42 IOneHotLayer

class tensorrt.**IOneHotLayer**

A OneHot layer in a network definition.

The OneHot layer has three input tensors: Indices, Values, and Depth, one output tensor, Output, and an axis attribute. :ivar indices: is an Int32 tensor that determines which locations in Output to set as on_value. :ivar values: is a two-element (rank=1) tensor that consists of [off_value, on_value] :ivar depth: is an Int32 shape tensor of rank 0, which contains the depth (number of classes) of the one-hot encoding. The depth tensor must be a build-time constant, and its value should be positive. :returns: a tensor with rank = rank(indices)+1, where the added dimension contains the one-hot encoding. :param axis: specifies to which dimension of the output one-hot encoding is added.

The data types of Output shall be equal to the Values data type. The output is computed by copying off_values to all output elements, then setting on_value on the indices specified by the indices tensor.

when axis = 0: output[indices[i, j, k], i, j, k] = on_value for all i, j, k and off_value otherwise.

when axis = -1: output[i, j, k, indices[i, j, k]] = on_value for all i, j, k and off_value otherwise.

6.3.43 INonZeroLayer

class tensorrt.**INonZeroLayer**

A NonZero layer in an *INetworkDefinition* .

Computes the indices of the input tensor where the value is non-zero. The returned indices are in row-major order.

The output shape is always *{D, C}*, where *D* is the number of dimensions of the input and *C* is the number of non-zero values.

Variables
indices_type – *DataType* The specified data type of the output indices tensor. Must be tensorrt.int32 or tensorrt.int64.

6.3.44 INMSLayer

class tensorrt.**INMSLayer**

A non-maximum suppression layer in an *INetworkDefinition* .

Boxes: The input boxes tensor to the layer. This tensor contains the input bounding boxes. It is a linear tensor of type `float32` or `float16`. It has shape [batchSize, numInputBoundingBoxes, numClasses, 4] if the boxes are per class, or [batchSize, numInputBoundingBoxes, 4] if the same boxes are to be used for each class.

Scores: The input scores tensor to the layer. This tensor contains the per-box scores. It is a linear tensor of the same type as the boxes tensor. It has shape [batchSize, numInputBoundingBoxes, numClasses].

MaxOutputBoxesPerClass: The input maxOutputBoxesPerClass tensor to the layer. This tensor contains the maximum number of output boxes per batch item per class. It is a scalar (0D tensor) of type `int32`.

IoUThreshold is the maximum IoU for selected boxes. It is a scalar (0D tensor) of type `float32` in the range [0.0, 1.0]. It is an optional input with default 0.0. Use *set_input()* to add this optional tensor.

ScoreThreshold is the value that a box score must exceed in order to be selected. It is a scalar (0D tensor) of type `float32`. It is an optional input with default 0.0. Use *set_input()* to add this optional tensor.

The SelectedIndices output tensor contains the indices of the selected boxes. It is a linear tensor of type `int32` or `int64`. It has shape [NumOutputBoxes, 3].] Each row contains a (batchIndex, classIndex, boxIndex) tuple. The output boxes are sorted in order of increasing batchIndex and then in order of decreasing score within each batchIndex. For each batchIndex, the ordering of output boxes with the same score is unspecified. If MaxOutputBoxesPerClass is a constant input, the maximum number of output boxes is batchSize * numClasses * min(numInputBoundingBoxes, MaxOutputBoxesPerClass). Otherwise, the maximum number of output boxes is batchSize * numClasses * numInputBoundingBoxes. The maximum number of output boxes is used to determine the upper-bound on allocated memory for this output tensor.

The NumOutputBoxes output tensor contains the number of output boxes in selectedIndices. It is a scalar (0D tensor) of type `int32`.

The NMS algorithm iterates through a set of bounding boxes and their confidence scores, in decreasing order of score. Boxes are selected if their score is above a given threshold, and their intersection-over-union (IoU) with previously selected boxes is less than or equal to a given threshold. This layer implements NMS per batch item and per class.

For each batch item, the ordering of candidate bounding boxes with the same score is unspecified.

Variables

- **bounding_box_format** – `BoundingBoxFormat` The bounding box format used by the layer. Default is CORNER_PAIRS.
- **topk_box_limit** – `int` The maximum number of filtered boxes considered for selection. Default is 2000 for SM 5.3 and 6.2 devices, and 5000 otherwise. The TopK box limit must be less than or equal to {2000 for SM 5.3 and 6.2 devices, 5000 otherwise}.
- **indices_type** – *DataType* The specified data type of the output indices tensor. Must be tensorrt.int32 or tensorrt.int64.

set_input(*self:* tensorrt.tensorrt.ILayer, *index: SupportsInt*, *tensor:* tensorrt.tensorrt.ITensor) → None

Sets the input tensor for the given index. The indices are as follows:

Index	Description
0	The required Boxes tensor.
1	The required Scores tensor.
2	The required MaxOutputBoxesPerClass tensor.
3	The optional IoUThreshold tensor.
4	The optional ScoreThreshold tensor.

If this function is called for an index greater or equal to `num_inputs`, then afterwards `num_inputs` returns index + 1, and any missing intervening inputs are set to null. Note that only optional inputs can be missing.

Parameters

- **index** – The index of the input tensor.
- **tensor** – The input tensor.

6.3.45 IReverseSequenceLayer

class tensorrt.**IReverseSequenceLayer**

A ReverseSequence layer in an *INetworkDefinition* .

This layer performs batch-wise reversal, which slices the input tensor along the axis `batch_axis`. For the `i`-th slice, the operation reverses the first `N` elements, specified by the corresponding `i`-th value in `sequence_lens`, along `sequence_axis` and keeps the remaining elements unchanged. The output tensor will have the same shape as the input tensor.

Variables

- **batch_axis** – `int` The batch axis. Default: 1.
- **sequence_axis** – `int` The sequence axis. Default: 0.

6.3.46 INormalizationLayer

class tensorrt.**INormalizationLayer**

A Normalization layer in an *INetworkDefinition* .

The normalization layer performs the following operation:

X - input Tensor Y - output Tensor S - scale Tensor B - bias Tensor

Y = (X - Mean(X, axes)) / Sqrt(Variance(X) + epsilon) * S + B

Where Mean(X, axes) is a reduction over a set of axes, and Variance(X) = Mean((X - Mean(X, axes)) ^ 2, axes).

Variables

- **epsilon** – `float` The epsilon value used for the normalization calculation. Default: 1e-5F.
- **axes** – `int` The reduction axes for the normalization calculation.
- **num_groups** – `int` The number of groups to split the channels into for the normalization calculation. Default: 1.

6.3.47 ISqueezeLayer

class tensorrt.**ISqueezeLayer**

A Squeeze layer in an *INetworkDefinition* .

This layer represents a squeeze operation, removing unit dimensions of the input tensor on a set of axes.

Axes must be resolvable to a constant Int32 or Int64 1D shape tensor. Values in axes must be unique and in the range of [-r, r-1], where r is the rank of the input tensor. For each axis value, the corresponding dimension in the input tensor must be one.

set_input(*self:* tensorrt.tensorrt.ILayer, *index: SupportsInt*, *tensor:* tensorrt.tensorrt.ITensor) → None

Sets the input tensor for the given index. The index must be 0 or 1 for a Squeeze layer.

The indices are as follows:

Index	Description
0	Input data tensor.
1	The axes to remove. Must be resolvable to a constant Int32 or Int64 1D shape tensor.

Parameters

- **index** – The index of the input tensor.
- **tensor** – The input tensor.

6.3.48 IUnsqueezeLayer

class tensorrt.**IUnsqueezeLayer**

An Unsqueeze layer in an *INetworkDefinition* .

This layer represents an unsqueeze operation, which reshapes the input tensor by inserting unit-length dimensions at specified axes of the output.

Axes must be resolvable to a constant Int32 or Int64 shape tensor. Values in axes must be unique and in the range of [-r_final, r_final-1], where r_final is the sum of rank(input) and len(axes).

r_final must be less than Dims.MAX_DIMS.

set_input(*self:* tensorrt.tensorrt.ILayer, *index: SupportsInt*, *tensor:* tensorrt.tensorrt.ITensor) → None

Sets the input tensor for the given index. The index must be 0 or 1 for an Unsqueeze layer.

The indices are as follows:

Index	Description
0	Input data tensor.
1	The axes to add. Must be resolvable to a constant Int32 or Int64 1D shape tensor.

Parameters

- **index** – The index of the input tensor.
- **tensor** – The input tensor.

6.3.49 IDistCollectiveLayer

tensorrt.**CollectiveOperation**

The collective operations that may be performed by a DistCollective layer

Members:

ALL_REDUCE : All reduce collective operation

ALL_GATHER : All gather collective operation

BROADCAST : Broadcast collective operation

REDUCE : Reduce collective operation

REDUCE_SCATTER : Reduce scatter collective operation

ALL_TO_ALL : All-to-all collective operation

GATHER : Gather collective operation

SCATTER : Scatter collective operation

class tensorrt.**IDistCollectiveLayer**

A dist collective layer in an *INetworkDefinition* .

6.3.50 ICumulativeLayer

class tensorrt.**ICumulativeLayer**

A cumulative layer in an *INetworkDefinition* .

This layer represents a cumulative operation across a tensor.

It computes successive reductions across an axis of a tensor. The output always has the same shape as the input.

If the reduction operation is summation, then this is also known as prefix-sum or cumulative sum.

The operation has forward vs. reverse variants, and inclusive vs. exclusive variants.

For example, let the input be a vector x of length n and the output be vector y. Then y[j] = sum(x[...]) where ... denotes a sequence of indices from this list:

- inclusive + forward: 0..j
- inclusive + reverse: j..n-1
- exclusive + forward: 0..j-1
- exclusive + reverse: j+1..n-1

For multidimensional tensors, the cumulative applies across a specified axis. For example, given a 2D input, a forward inclusive cumulative across axis 0 generates cumulative sums within each column.

Variables

- **op** – CumulativeOperation The cumulative operation for the layer.
- **exclusive** – bool Specifies whether it is an exclusive cumulative or inclusive cumulative.
- **reverse** – bool Specifies whether the cumulative operation should be applied backward.

6.3.51 IAttention

class tensorrt.**IAttention**

An attention in a *INetworkDefinition* .

Variables

- **mask** – *ITensor* The mask tensor for attention. Cannot be set together with causal attention.
- **norm_op** – AttentionNormalizationOp The normalization operation for the attention layer. Default to AttentionNormalizationOp::kSOFTMAX.
- **decomposable** – bool Specifies whether decomposition into primitive ops is allowed when no attention fusion is supported. Default to False.
- **causal** – bool (Deprecated) Specifies whether the attention will run a causal inference. Cannot be used together with mask. Superseded by causal_kind.
- **causal_kind** – CausalMaskKind The causal mask alignment orientation for the attention. Cannot be used together with mask. Default to CausalMaskKind.NONE.
- **name** – str The name of the attention.
- **metadata** – str The metadata of the attention.
- **normalization_quantize_scale** – *ITensor* The quantization scale for the attention normalization output.
- **normalization_quantize_to_type** – *DataType* The datatype the attention normalization is quantized to.
- **num_inputs** – int The number of inputs of the attention.
- **num_outputs** – int The number of outputs of the attention.
- **num_ranks** – int The number of ranks for multi-device attention execution (default: 1).
- **query_form** – AttentionIOForm The layout of the query tensor. Default is AttentionIOForm.PADDED_BHND.
- **key_value_form** – AttentionIOForm The layout of the key and value tensors. Default is AttentionIOForm.PADDED_BHND.
- **query_lengths** – *ITensor* Optional 1D INT32 tensor of cumulative query token counts with shape [batch_size + 1]. Must be set when query_form is PACKED_NHD. Ignored when query_form is PADDED_BHND. Set to None to clear.
- **key_value_lengths** – *ITensor* Optional 1D INT32 tensor specifying key-value lengths. When key_value_form is PADDED_BHND, a per-batch lengths tensor of shape [batch_size]. When key_value_form is PACKED_NHD, cumulative token counts of shape [batch_size + 1] (required). Set to None to clear.

get_input(*self:* tensorrt.tensorrt.IAttention, *index: SupportsInt*) → *tensorrt.tensorrt.ITensor*

Get the input tensor specified by the given index.

Parameters
index – The index of the input tensor.

Returns
The tensor, or None if it is out of range.

get_output(*self:* tensorrt.tensorrt.IAttention, *index: SupportsInt*) → *tensorrt.tensorrt.ITensor*

Get the output tensor specified by the given index.

Parameters
index – The index of the output tensor.

Returns
The tensor, or None if it is out of range.

property num_ranks

int The number of ranks for multi-device attention execution.

When num_ranks > 1, this hints attention to perform multi-device attention.

Default value is 1.

set_input(*self:* tensorrt.tensorrt.IAttention, *index: SupportsInt*, *tensor:* tensorrt.tensorrt.ITensor) → bool

Set the input tensor specified by the given index.

The indices are as follows:

Index	Description
0	query.
1	key.
2	value.

Parameters

- **index** – The index of the input tensor. query:0, key:1, value:2
- **tensor** – The input tensor.

6.3.52 IRotaryEmbeddingLayer

class tensorrt.**IRotaryEmbeddingLayer**

A RotaryEmbedding layer in *INetworkDefinition*.

Variables

- **interleaved** – bool Specifies whether the input tensor is in interleaved format, i.e., whether the 2-d vectors rotated are taken from adjacent 2 elements in the hidden dimension.
- **rotary_embedding_dim** – int Specifies the hidden dimension that participates in RoPE.

set_input(*self:* tensorrt.tensorrt.ILayer, *index: SupportsInt*, *tensor:* tensorrt.tensorrt.ITensor) → None

Set the input tensor specified by the given index.

Parameters

- **index** – The index of the input tensor.
- **tensor** – The input tensor.

The indices are as follows:

Input 0 is the input activation tensor. Input 1 is the cosine cache tensor. Input 2 is the sine cache tensor. Input 3 (optional) is the positionIds tensor, which is used for indexing into the cosine and sine caches.

6.3.53 IKVCacheUpdateLayer

class tensorrt.**IKVCacheUpdateLayer**

A KVCacheUpdate layer in a *INetworkDefinition* .

This layer caches Key (K) or Value (V) tensors for reuse in subsequent attention computations. Users provide newly computed K/V values, and the layer will output the updated K/V cache. The write_indices input specifies where to write K/V updates for each sequence in the batch. Separate KVCacheUpdate layers should be used for K and V.

Variables

- **cache_mode** – `KVCacheMode` The mode of the KVCacheUpdate layer.
- **update_form** – `AttentionIOForm` The layout of the update tensor. Default is AttentionIOForm.PADDED_BHND. When set to PACKED_NHD, the update tensor has shape [total_tokens, num_heads, head_dim] and update_lengths must be provided.
- **update_lengths** – *ITensor* Optional 1D INT32 tensor of cumulative token counts with shape [batch_size + 1]. Only valid when update_form is PACKED_NHD (required in that case; ignored when update_form is PADDED_BHND). Set to None to clear.

set_input(*self:* tensorrt.tensorrt.ILayer, *index: SupportsInt*, *tensor:* tensorrt.tensorrt.ITensor) → None

Sets the input tensor specified by the given index.

The indices are as follows:

Index	Description
0	cache.
1	update.
2	write_indices.
3	update_lengths (optional; use the update_lengths property instead).

Parameters

- **index** – The index of the input tensor.
- **tensor** – The input tensor.

6.3.54 IMoELayer

class tensorrt.**IMoELayer**

A MoE layer in *INetworkDefinition*.

Variables

- **activation_type** – `MoEActType` Specifies the activation type for the MoE layer.
- **quantization_to_type** – *DataType* Specifies the quantization type for the MoE layer.
- **quantization_block_shape** – *Dims* Specifies the quantization block shape for the MoE layer.
- **dyn_q_output_scale_type** – *DataType* Specifies the dynamic quantization output scale type for the MoE layer.
- **swiglu_param_limit** – `float` Specifies the swiglu parameter limit for the MoE layer.
- **swiglu_param_alpha** – `float` Specifies the swiglu parameter alpha for the MoE layer.

- **swiglu_param_beta** – `float` Specifies the swiglu parameter beta for the MoE layer.

set_gated_biases(*self:* tensorrt.tensorrt.IMoELayer, *fc_gate_biases:* tensorrt.tensorrt.ITensor, *fc_up_biases:* tensorrt.tensorrt.ITensor, *fc_down_biases:* tensorrt.tensorrt.ITensor) → None

Set the gated biases for the MoE layer. :arg fc_gate_biases: The biases for the gate-projection layer of all experts in MoE. :arg fc_up_biases: The biases for the up-projection layer of all experts in MoE. :arg fc_down_biases: The biases for the down-projection layer of all experts in MoE.

set_gated_weights(*self:* tensorrt.tensorrt.IMoELayer, *fc_gate_weights:* tensorrt.tensorrt.ITensor, *fc_up_weights:* tensorrt.tensorrt.ITensor, *fc_down_weights:* tensorrt.tensorrt.ITensor, *activation_type: tensorrt.tensorrt.MoEActType*) → None

Set the gated weights for the MoE layer. :arg fc_gate_weights: The weights for the gate-projection layer of all experts in MoE. :arg fc_up_weights: The weights for the up-projection layer of all experts in MoE. :arg fc_down_weights: The weights for the down-projection layer of all experts in MoE. :arg activation_type: The activation type for the MoE layer.

set_input(*self:* tensorrt.tensorrt.IMoELayer, *index: SupportsInt*, *tensor:* tensorrt.tensorrt.ITensor) → None

Set the input tensor specified by the given index.

Parameters

- **index** – The index of the input tensor.
- **tensor** – The input tensor.

The indices are as follows:

Input 0: hidden_states: the hidden states tensor. Input 1: selected_experts_for_tokens: the selected experts for tokens tensor. Input 2: scores_for_selected_experts: the scores for selected experts tensor.

set_quantization_dynamic_dbl_q(*self:* tensorrt.tensorrt.IMoELayer, *fc_down_activation_dbl_q_scale:* tensorrt.tensorrt.ITensor, *data_type: tensorrt.tensorrt.DataType*, *block_shape:* tensorrt.tensorrt.Dims, *dyn_q_output_scale_type: tensorrt.tensorrt.DataType*) → None

Set the quantization dynamic double quantization for the MoE layer. :arg fc_down_activation_dbl_q_scale: The down activation double quantization scale tensor. :arg data_type: The data type for the quantization. :arg block_shape: The block shape for the quantization. :arg dyn_q_output_scale_type: The dynamic quantization output scale type.

set_quantization_static(*self:* tensorrt.tensorrt.IMoELayer, *fc_down_activation_scale:* tensorrt.tensorrt.ITensor, *data_type: tensorrt.tensorrt.DataType*) → None

Set the quantization static for the MoE layer. :arg fc_down_activation_scale: The down activation scale tensor. :arg data_type: The data type for the quantization.

set_swiglu_params(*self:* tensorrt.tensorrt.IMoELayer, *limit: SupportsFloat*, *alpha: SupportsFloat*, *beta: SupportsFloat*) → None

Set the swiglu parameters for the MoE layer. :arg limit: The limit for the swiglu parameters. :arg alpha: The alpha for the swiglu parameters. :arg beta: The beta for the swiglu parameters.

CHAPTER

SEVEN

PLUGIN

7.1 IPluginCreator

tensorrt.**PluginFieldType**

The possible field types for custom layer.

Members:

FLOAT16

FLOAT32

FLOAT64

INT8

INT16

INT32

CHAR

DIMS

UNKNOWN

BF16

INT64

FP8

INT4

FP4

class tensorrt.**PluginField**(**args*, ***kwargs*)

Contains plugin attribute field names and associated data. This information can be parsed to decode necessary plugin metadata

Variables

- **name** – str Plugin field attribute name.
- **data** – buffer Plugin field attribute data.
- **type** – *PluginFieldType* Plugin field attribute type.
- **size** – int Number of data entries in the Plugin attribute.

Overloaded function.

1. __init__(self: tensorrt.tensorrt.PluginField, name: tensorrt.tensorrt.FallbackString = '') -> None
2. __init__(self: tensorrt.tensorrt.PluginField, name: tensorrt.tensorrt.FallbackString, data: collections.abc.Buffer, type: tensorrt.tensorrt.PluginFieldType = <PluginFieldType.UNKNOWN: 8>) -> None

class tensorrt.**PluginFieldCollection**(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: tensorrt.tensorrt.PluginFieldCollection) -> None
2. __init__(self: tensorrt.tensorrt.PluginFieldCollection, arg0: tensorrt.tensorrt.PluginFieldCollection) -> None

Copy constructor

3. __init__(self: tensorrt.tensorrt.PluginFieldCollection, arg0: collections.abc.Iterable) -> None

append(*self:* tensorrt.tensorrt.PluginFieldCollection, *x:* tensorrt.PluginField) → None

Add an item to the end of the list

clear(*self:* tensorrt.tensorrt.PluginFieldCollection) → None

Clear the contents

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: tensorrt.tensorrt.PluginFieldCollection, L: tensorrt.tensorrt.PluginFieldCollection) -> None

Extend the list by appending all the items in the given list

2. extend(self: tensorrt.tensorrt.PluginFieldCollection, L: collections.abc.Iterable) -> None

Extend the list by appending all the items in the given list

insert(*self:* tensorrt.tensorrt.PluginFieldCollection, *i: SupportsInt*, *x:* tensorrt.PluginField) → None

Insert an item at a given position.

pop(**args*, ***kwargs*)

Overloaded function.

1. pop(self: tensorrt.tensorrt.PluginFieldCollection) -> tensorrt.PluginField

Remove and return the last item

2. pop(self: tensorrt.tensorrt.PluginFieldCollection, i: typing.SupportsInt) -> tensorrt.PluginField

Remove and return the item at index `i`

class tensorrt.**IPluginCreatorInterface**

Base class for for plugin sub-interfaces.

class tensorrt.**IPluginCreatorV3One**(*self:* tensorrt.tensorrt.IPluginCreatorV3One)

Plugin creator class for user implemented layers

Variables

- **name** – `str` Plugin name.
- **plugin_version** – `str` Plugin version.
- **field_names** – `list` List of fields that needs to be passed to *create_plugin()* .
- **plugin_namespace** – `str` The namespace of the plugin creator based on the plugin library it belongs to. This can be set while registering the plugin creator.

create_plugin(*self:* tensorrt.tensorrt.IPluginCreatorV3One, *name: str*, *field_collection: tensorrt.tensorrt.PluginFieldCollection_*, *phase:* tensorrt.TensorRTPhase) → *tensorrt.tensorrt.IPluginV3*

Creates a new plugin.

Parameters

- **name** – The name of the plugin.
- **field_collection** – The *PluginFieldCollection* for this plugin.

Returns
`IPluginV2` or `None` on failure.

class `tensorrt.IPluginCreator`(*self:* tensorrt.tensorrt.IPluginCreator)

Plugin creator class for user implemented layers

Variables

- **name** – `str` Plugin name.
- **plugin_version** – `str` Plugin version.
- **field_names** – `list` List of fields that needs to be passed to *create_plugin()* .
- **plugin_namespace** – `str` The namespace of the plugin creator based on the plugin library it belongs to. This can be set while registering the plugin creator.

create_plugin(*self:* tensorrt.tensorrt.IPluginCreator, *name: str*, *field_collection: tensorrt.tensorrt.PluginFieldCollection_*) → tensorrt.tensorrt.IPluginV2

Creates a new plugin.

Parameters

- **name** – The name of the plugin.
- **field_collection** – The *PluginFieldCollection* for this plugin.

Returns
`IPluginV2` or `None` on failure.

deserialize_plugin(**args*, ***kwargs*)

Overloaded function.

1. deserialize_plugin(self: tensorrt.tensorrt.IPluginCreator, name: str, serialized_plugin: collections.abc.Buffer) -> tensorrt.tensorrt.IPluginV2

   Creates a plugin object from a serialized plugin.

   > **Warning:** This API only applies when called on a C++ plugin from a Python program.

   serialized_plugin will contain a Python bytes object containing the serialized representation of the plugin.

   arg name
   Name of the plugin.

   arg serialized_plugin
   A buffer containing a serialized plugin.

   returns
   A new `IPluginV2`

2. deserialize_plugin(self: tensorrt.tensorrt.IPluginV2DynamicExt, name: str, serialized_plugin: bytes) -> tensorrt.tensorrt.IPluginV2DynamicExt

   Creates a plugin object from a serialized plugin.

   Warning: This API only applies when implementing a Python-based plugin.

   serialized_plugin contains a serialized representation of the plugin.

   arg name
   Name of the plugin.

   arg serialized_plugin
   A string containing a serialized plugin.

   returns
   A new `IPluginV2`

7.2 IPluginRegistry

class tensorrt.**IPluginRegistry**

Registers plugin creators.

Variables

- **plugin_creator_list** – [DEPRECATED] Deprecated in TensorRT 10.0. List of IPluginV2-descendent plugin creators in current registry.
- **all_creators** – List of all registered plugin creators of current registry.
- **all_creators_recursive** – List of all registered plugin creators of current registry and its parents (if `parent_search_enabled` is True).
- **error_recorder** – *IErrorRecorder* Application-implemented error reporting interface for TensorRT objects.
- **parent_search_enabled** – bool variable indicating whether parent search is enabled. Default is True.

acquire_plugin_resource(*self:* tensorrt.tensorrt.IPluginRegistry, *key: str*, *resource:* tensorrt.tensorrt.IPluginResource) → *tensorrt.tensorrt.IPluginResource*

Get a handle to a plugin resource registered against the provided key.

Arg
key: Key for identifying the resource.

Arg
resource: A plugin resource object. The object will only need to be valid until this method returns, as only a clone of this object will be registered by TRT. Cannot be null.

deregister_creator(**args*, ***kwargs*)

Overloaded function.

1. deregister_creator(self: tensorrt.tensorrt.IPluginRegistry, creator: tensorrt.tensorrt.IPluginCreatorInterface) -> bool

Deregister a previously registered plugin creator inheriting from IPluginCreator.

Since there may be a desire to limit the number of plugins, this function provides a mechanism for removing plugin creators registered in TensorRT. The plugin creator that is specified by `creator` is removed from TensorRT and no longer tracked.

arg creator
The IPluginCreator instance.

returns
`True` if the plugin creator was deregistered, `False` if it was not found in the registry or otherwise could not be deregistered.

2. deregister_creator(self: tensorrt.tensorrt.IPluginRegistry, creator: tensorrt.tensorrt.IPluginCreatorInterface) -> bool

Deregister a previously registered plugin creator.

Since there may be a desire to limit the number of plugins, this function provides a mechanism for removing plugin creators registered in TensorRT. The plugin creator that is specified by `creator` is removed from TensorRT and no longer tracked.

arg creator
The plugin creator instance.

returns
`True` if the plugin creator was deregistered, `False` if it was not found in the registry or otherwise could not be deregistered.

deregister_library(*self:* tensorrt.tensorrt.IPluginRegistry, *handle: typing_extensions.CapsuleType*) → None

Deregister plugins associated with a library. Any resources acquired when the library was loaded will be released.

Arg
handle: the plugin library handle to deregister.

get_creator(*self:* tensorrt.tensorrt.IPluginRegistry, *type: str*, *version: str*, *plugin_namespace: str = ''*) → object

Return plugin creator based on type, version and namespace

Parameters

- **type** – The type of the plugin.
- **version** – The version of the plugin.
- **plugin_namespace** – The namespace of the plugin.

Returns
An *IPluginCreator* .

load_library(*self:* tensorrt.tensorrt.IPluginRegistry, *plugin_path: str*) → typing_extensions.CapsuleType

Load and register a shared library of plugins.

Arg
plugin_path: the plugin library path.

Returns
The loaded plugin library handle. The call will fail and return None if any of the plugins are already registered.

register_creator(**args*, ***kwargs*)

Overloaded function.

1. register_creator(self: tensorrt.tensorrt.IPluginRegistry, creator: tensorrt.tensorrt.IPluginCreatorInterface, plugin_namespace: str = '') -> bool

   Register a plugin creator implementing IPluginCreator.

   arg creator
   The IPluginCreator instance.

   arg plugin_namespace
   The namespace of the plugin creator.

   returns
   False if any plugin creator with the same name, version and namespace is already registered.

2. register_creator(self: tensorrt.tensorrt.IPluginRegistry, creator: tensorrt.tensorrt.IPluginCreatorInterface, plugin_namespace: str = '') -> bool

   Register a plugin creator.

   arg creator
   The plugin creator instance.

   arg plugin_namespace
   The namespace of the plugin creator.

   returns
   False if any plugin creator with the same name, version and namespace is already registered..

release_plugin_resource(*self:* tensorrt.tensorrt.IPluginRegistry, *key: str*) → int

Decrement reference count for the resource with this key. If reference count goes to zero after decrement, release() will be invoked on the resource, and the key will be deregistered.

Arg
key: Key that was used to register the resource.

class tensorrt.**IPluginResource**(*self:* tensorrt.tensorrt.IPluginResource)

Interface for plugins to define custom resources that could be shared through the plugin registry

clone(*self:* tensorrt.tensorrt.IPluginResource) → *tensorrt.tensorrt.IPluginResource*

Resource initialization (if any) may be skipped for non-cloned objects since only clones will be registered by TensorRT.

release(*self:* tensorrt.tensorrt.IPluginResource) → None

This will only be called for IPluginResource objects that were produced from IPluginResource::clone().

The IPluginResource object on which release() is called must still be in a clone-able state after release() returns.

tensorrt.**get_plugin_registry**() → detail::accessor<detail::accessor_policies::str_attr>

Return the plugin registry for standard runtime

tensorrt.**init_libnvinfer_plugins**(*logger: typing_extensions.CapsuleType*, *namespace: str*) → bool

Initialize and register all the existing TensorRT plugins to the *IPluginRegistry* with an optional namespace. The plugin library author should ensure that this function name is unique to the library. This function should be called once before accessing the Plugin Registry.

Parameters

- **logger** – Logger to print plugin registration information.
- **namespace** – Namespace used to register all the plugins in this library.

tensorrt.**get_builder_plugin_registry**(*arg0: tensorrt.EngineCapability*) → *tensorrt.tensorrt.IPluginRegistry*

Return the plugin registry used for building engines for the specified runtime

7.3 IPluginV2DynamicExt

class tensorrt.**IPluginV2DynamicExt**(**args*, ***kwargs*)

Plugin class for user-implemented layers.

Plugins are a mechanism for applications to implement custom layers.

Similar to *IPluginV2Ext* (including capability to support different output data types), but with support for dynamic shapes.

This class is made available for the purpose of implementing *IPluginV2DynamicExt* plugins with Python. Inherited Python->C++ bindings from *IPluginV2* and *IPluginV2Ext* will continue to work on C++-based *IPluginV2DynamicExt* plugins.

Note: Every attribute except *tensorrt_version* must be explicitly initialized on Python-based plugins. Except *plugin_namespace*, these attributes will be read-only when accessed through a C++-based plugin.

Variables

- **num_outputs** – `int` The number of outputs from the plugin. This is used by the implementations of *INetworkDefinition* and *Builder*. In particular, it is called prior to any call to *initialize()*.
- **tensorrt_version** – `int` [READ ONLY] The API version with which this plugin was built.
- **plugin_type** – `str` The plugin type. Should match the plugin name returned by the corresponding plugin creator.
- **plugin_version** – `str` The plugin version. Should match the plugin version returned by the corresponding plugin creator.
- **plugin_namespace** – `str` The namespace that this plugin object belongs to. Ideally, all plugin objects from the same plugin library should have the same namespace.
- **serialization_size** – `int` [READ ONLY] The size of the serialization buffer required.

Overloaded function.

1. __init__(self: tensorrt.tensorrt.IPluginV2DynamicExt) -> None
2. __init__(self: tensorrt.tensorrt.IPluginV2DynamicExt, arg0: tensorrt.tensorrt.IPluginV2DynamicExt) -> None

clone(*self:* tensorrt.tensorrt.IPluginV2DynamicExt) → *tensorrt.tensorrt.IPluginV2DynamicExt*

Clone the plugin object. This copies over internal plugin parameters as well and returns a new plugin object with these parameters.

If the source plugin is pre-configured with *configure_plugin()*, the returned object should also be pre-configured. Cloned plugin objects can share the same per-engine immutable resource (e.g. weights) with the source object to avoid duplication.

configure_plugin(*self:* tensorrt.tensorrt.IPluginV2DynamicExt, *pos:* *collections.abc.Sequence[*tensorrt.tensorrt.DynamicPluginTensorDesc*]*, *in_out:* *collections.abc.Sequence[*tensorrt.tensorrt.DynamicPluginTensorDesc*]*) → None

Configure the plugin.

This function can be called multiple times in both the build and execution phases. The build phase happens before *initialize()* is called and only occurs during creation of an engine by *IBuilder*. The execution phase happens after *initialize()* is called and occurs during both creation of an engine by *IBuilder* and execution of an engine by *IExecutionContext*.

Build phase: *configure_plugin()* is called when a plugin is being prepared for profiling but not for any specific input size. This provides an opportunity for the plugin to make algorithmic choices on the basis of input and output formats, along with the bound of possible dimensions. The min and max value of the *DynamicPluginTensorDesc* correspond to the *kMIN* and *kMAX* value of the current optimization profile that the plugin is being profiled for, with the *desc.dims* field corresponding to the dimensions of plugin specified at network creation. Wildcard dimensions will exist during this phase in the *desc.dims* field.

Execution phase: *configure_plugin()* is called when a plugin is being prepared for executing the plugin for specific dimensions. This provides an opportunity for the plugin to change algorithmic choices based on the explicit input dimensions stored in *desc.dims* field.

Warning: This *configure_plugin()* method is not available to be called from Python on C++-based plugins

Parameters

- **in** – The input tensors attributes that are used for configuration.
- **out** – The output tensors attributes that are used for configuration.

destroy(*self:* tensorrt.tensorrt.IPluginV2DynamicExt) → None

Destroy the plugin object. This will be called when the *INetworkDefinition*, *Builder* or *ICudaEngine* is destroyed.

Note: When implementing a Python-based plugin, implementing this method is optional. The default behavior is a *pass*.

enqueue(*self:* tensorrt.tensorrt.IPluginV2DynamicExt, *input_desc:* *collections.abc.Sequence[tensorrt.tensorrt.PluginTensorDesc]*, *output_desc:* *collections.abc.Sequence[tensorrt.tensorrt.PluginTensorDesc]*, *inputs:* *collections.abc.Sequence[SupportsInt]*, *outputs: collections.abc.Sequence[SupportsInt]*, *workspace:* *SupportsInt*, *stream: SupportsInt*) → None

Execute the layer.

inputs and *outputs* contains pointers to the corresponding input and output device buffers as their *intptr_t* casts. *stream* also represents an *intptr_t* cast of the CUDA stream in which enqueue should be executed.

Warning: Since input, output, and workspace buffers are created and owned by TRT, care must be taken when writing to them from the Python side.

Warning: In contrast to the C++ API for *enqueue()*, this method must not return an error code. The expected behavior is to throw an appropriate exception. if an error occurs.

Warning: This *enqueue()* method is not available to be called from Python on C++-based plugins.

Parameters

- **input_desc** – how to interpret the memory for the input tensors.
- **output_desc** – how to interpret the memory for the output tensors.
- **inputs** – The memory for the input tensors.
- **outputs** – The memory for the output tensors.
- **workspace** – Workspace for execution.
- **stream** – The stream in which to execute the kernels.

get_output_datatype(*self:* tensorrt.tensorrt.IPluginV2DynamicExt, *index: SupportsInt*, *input_types: collections.abc.Sequence[tensorrt.tensorrt.DataType]*) → tensorrt.tensorrt.DataType

Return the *DataType* of the plugin output at the requested index. The default behavior should be to return the type of the first input, or *DataType::kFLOAT* if the layer has no inputs. The returned data type must have a format that is supported by the plugin.

Parameters

- **index** – Index of the output for which the data type is requested.
- **input_types** – Data types of the inputs.

Returns

DataType of the plugin output at the requested *index*.

get_output_dimensions(*self:* tensorrt.tensorrt.IPluginV2DynamicExt, *output_index: SupportsInt*, *inputs: collections.abc.Sequence[*tensorrt.tensorrt.DimsExprs*]*, *expr_builder:* tensorrt.tensorrt.IExprBuilder) → *tensorrt.tensorrt.DimsExprs*

Get expressions for computing dimensions of an output tensor from dimensions of the input tensors.

This function is called by the implementations of *IBuilder* during analysis of the network.

Warning: This *get_output_dimensions()* method is not available to be called from Python on C++-based plugins

Parameters

- **output_index** – The index of the output tensor
- **inputs** – Expressions for dimensions of the input tensors
- **expr_builder** – Object for generating new expressions

Returns

Expression for the output dimensions at the given *output_index*.

get_serialization_size(*self:* tensorrt.tensorrt.IPluginV2DynamicExt) → int

Return the serialization size (in bytes) required by the plugin.

Note: When implementing a Python-based plugin, implementing this method is optional. The default behavior is equivalent to *return len(serialize())*.

---

get_workspace_size(*self: tensorrt.tensorrt.IPluginV2DynamicExt, in: collections.abc.Sequence[tensorrt.tensorrt.PluginTensorDesc], out: collections.abc.Sequence[tensorrt.tensorrt.PluginTensorDesc]*) → int

Return the workspace size (in bytes) required by the plugin.

This function is called after the plugin is configured, and possibly during execution. The result should be a sufficient workspace size to deal with inputs and outputs of the given size or any smaller problem.

Note: When implementing a Python-based plugin, implementing this method is optional. The default behavior is equivalent to *return 0*.

Warning: This *get_workspace_size()* method is not available to be called from Python on C++-based plugins

Parameters

- **input_desc** – How to interpret the memory for the input tensors.
- **output_desc** – How to interpret the memory for the output tensors.

Returns

The workspace size (in bytes).

initialize(*self:* tensorrt.tensorrt.IPluginV2DynamicExt) → int

Initialize the plugin for execution. This is called when the engine is created.

Note: When implementing a Python-based plugin, implementing this method is optional. The default behavior is equivalent to *pass*.

Warning: In contrast to the C++ API for *initialize()*, this method must not return an error code. The expected behavior is to throw an appropriate exception if an error occurs.

Warning: This *initialize()* method is not available to be called from Python on C++-based plugins.

serialize(*self:* tensorrt.tensorrt.IPluginV2DynamicExt) → bytes

Serialize the plugin.

Warning: This API only applies when implementing a Python-based plugin.

Returns
A bytes object containing the serialized representation of the plugin.

supports_format_combination(*self:* tensorrt.tensorrt.IPluginV2DynamicExt, *pos: SupportsInt*, *in_out: collections.abc.Sequence[tensorrt.tensorrt.PluginTensorDesc]*, *num_inputs: SupportsInt*) → bool

Return true if plugin supports the format and datatype for the input/output indexed by pos.

For this method, inputs are indexed from *[0, num_inputs-1]* and outputs are indexed from *[num_inputs, (num_inputs + num_outputs - 1)]*. *pos* is an index into *in_ou`t, where `0 <= pos < (num_inputs + num_outputs - 1)*.

TensorRT invokes this method to query if the input/output tensor indexed by *pos* supports the format and datatype specified by *in_out[pos].format* and *in_out[pos].type*. The override shall return true if that format and datatype at *in_out[pos]* are supported by the plugin. It is undefined behavior to examine the format or datatype or any tensor that is indexed by a number greater than *pos*.

> **Warning:** This *supports_format_combination()* method is not available to be called from Python on C++-based plugins

Parameters

- **pos** – The input or output tensor index being queried.
- **in_out** – The combined input and output tensor descriptions.
- **num_inputs** – The number of inputs.

Returns
boolean indicating whether the format combination is supported or not.

terminate(*self:* tensorrt.tensorrt.IPluginV2DynamicExt) → None

Release resources acquired during plugin layer initialization. This is called when the engine is destroyed.

Note: When implementing a Python-based plugin, implementing this method is optional. The default behavior is equivalent to *pass*.

7.4 IPluginV3

class tensorrt.**IPluginV3**(**args*, ***kwargs*)

Plugin class for the V3 generation of user-implemented layers.

IPluginV3 acts as a wrapper around the plugin capability interfaces that define the actual behavior of the plugin.

This class is made available for the purpose of implementing *IPluginV3* plugins with Python.

Note: Every attribute must be explicitly initialized on Python-based plugins. These attributes will be read-only when accessed through a C++-based plugin.

Overloaded function.

1. __init__(self: tensorrt.tensorrt.IPluginV3) -> None

2. __init__(self: tensorrt.tensorrt.IPluginV3, arg0: tensorrt.tensorrt.IPluginV3) -> None

clone(*self:* tensorrt.tensorrt.IPluginV3) → *tensorrt.tensorrt.IPluginV3*

Clone the plugin object. This copies over internal plugin parameters as well and returns a new plugin object with these parameters.

If the source plugin is pre-configured with *configure_plugin()*, the returned object should also be pre-configured. Cloned plugin objects can share the same per-engine immutable resource (e.g. weights) with the source object to avoid duplication.

destroy(*self:* tensorrt.tensorrt.IPluginV3) → None

Perform any cleanup or resource release(s) needed before plugin object is destroyed. This will be called when the *INetworkDefinition* , *Builder* or *ICudaEngine* is destroyed.

Note: There is no direct equivalent to this method in the C++ API.

Note: Implementing this method is optional. The default behavior is a *pass*.

---

get_capability_interface(*self:* tensorrt.tensorrt.IPluginV3, *type:* tensorrt.PluginCapabilityType) → object

Return a plugin object implementing the specified PluginCapabilityType.

Note: IPluginV3 objects added for the build phase (through add_plugin_v3()) must return valid objects for PluginCapabilityType.CORE, PluginCapabilityType.BUILD and PluginCapabilityType.RUNTIME.

Note: IPluginV3 objects added for the runtime phase must return valid objects for PluginCapabilityType.CORE and PluginCapabilityType.RUNTIME.

---

class tensorrt.**IPluginCapability**

Base class for plugin capability interfaces

IPluginCapability represents a split in TensorRT V3 plugins to sub-objects that expose different types of capabilites a plugin may have, as opposed to a single interface which defines all capabilities and behaviors of a plugin.

class tensorrt.**IPluginV3OneCore**(**args*, ***kwargs*)

A plugin capability interface that enables the core capability (PluginCapabilityType.CORE).

Note: Every attribute must be explicitly initialized on Python-based plugins. These attributes will be read-only when accessed through a C++-based plugin.

Variables

- **plugin_name** – str The plugin name. Should match the plugin name returned by the corresponding plugin creator.
- **plugin_version** – str The plugin version. Should match the plugin version returned by the corresponding plugin creator.

- **plugin_namespace** – str The namespace that this plugin object belongs to. Ideally, all plugin objects from the same plugin library should have the same namespace.

Overloaded function.

1. __init__(self: tensorrt.tensorrt.IPluginV3OneCore) -> None
2. __init__(self: tensorrt.tensorrt.IPluginV3OneCore, arg0: tensorrt.tensorrt.IPluginV3OneCore) -> None

class tensorrt.**IPluginV3OneBuild**(**args*, ***kwargs*)

A plugin capability interface that enables the build capability (PluginCapabilityType.BUILD).

Exposes methods that allow the expression of the build time properties and behavior of a plugin.

Note: Every attribute must be explicitly initialized on Python-based plugins. These attributes will be read-only when accessed through a C++-based plugin.

Variables

- **num_outputs** – int The number of outputs from the plugin. This is used by the implementations of *INetworkDefinition* and *Builder*.
- **format_combination_limit** – int The maximum number of format combinations that the plugin supports.
- **metadata_string** – str The metadata string for the plugin.
- **timing_cache_id** – str The timing cache ID for the plugin.

Overloaded function.

1. __init__(self: tensorrt.tensorrt.IPluginV3OneBuild) -> None
2. __init__(self: tensorrt.tensorrt.IPluginV3OneBuild, arg0: tensorrt.tensorrt.IPluginV3OneBuild) -> None

configure_plugin(*self: tensorrt.tensorrt.IPluginV3, in: collections.abc.Sequence[tensorrt.tensorrt.DynamicPluginTensorDesc], out: collections.abc.Sequence[tensorrt.tensorrt.DynamicPluginTensorDesc]*) → None

Configure the plugin.

This function can be called multiple times in the build phase during creation of an engine by IBuilder.

Build phase: *configure_plugin()* is called when a plugin is being prepared for profiling but not for any specific input size. This provides an opportunity for the plugin to make algorithmic choices on the basis of input and output formats, along with the bound of possible dimensions. The min, opt and max value of the *DynamicPluginTensorDesc* correspond to the *MIN*, *OPT* and *MAX* value of the current profile that the plugin is being profiled for, with the desc.dims field corresponding to the dimensions of plugin specified at network creation. Wildcard dimensions may exist during this phase in the desc.dims field.

Warning: In contrast to the C++ API for *configurePlugin()*, this method must not return an error code. The expected behavior is to throw an appropriate exception if an error occurs.

Warning: This *configure_plugin()* method is not available to be called from Python on C++-based plugins

Parameters

- **in** – The input tensors attributes that are used for configuration.
- **out** – The output tensors attributes that are used for configuration.

get_output_data_types(*self:* tensorrt.tensorrt.IPluginV3, *input_types: collections.abc.Sequence[tensorrt.tensorrt.DataType]*) → list[tensorrt.tensorrt.DataType]

Return *DataType* s of the plugin outputs.

Provide *DataType.FLOAT* s if the layer has no inputs. The data type for any size tensor outputs must be *DataType.INT32*. The returned data types must each have a format that is supported by the plugin.

Parameters
input_types – Data types of the inputs.

Returns
DataType of the plugin output at the requested *index*.

get_output_shapes(*self:* tensorrt.tensorrt.IPluginV3, *inputs: collections.abc.Sequence[*tensorrt.tensorrt.DimsExprs*]*, *shape_inputs: collections.abc.Sequence[*tensorrt.tensorrt.DimsExprs*]*, *expr_builder:* tensorrt.tensorrt.IExprBuilder) → list[*tensorrt.tensorrt.DimsExprs*]

Get expressions for computing shapes of an output tensor from shapes of the input tensors.

This function is called by the implementations of *IBuilder* during analysis of the network.

Warning: This get_output_shapes() method is not available to be called from Python on C++-based plugins

Parameters

- **inputs** – Expressions for shapes of the input tensors
- **shape_inputs** – Expressions for shapes of the shape inputs
- **expr_builder** – Object for generating new expressions

Returns
Expressions for the output shapes.

get_valid_tactics(*self:* tensorrt.tensorrt.IPluginV3) → list[int]

Return any custom tactics that the plugin intends to use.

Note: The provided tactic values must be unique and positive

Warning: This *get_valid_tactics()* method is not available to be called from Python on C++-based plugins.

get_workspace_size(*self: tensorrt.tensorrt.IPluginV3, in: collections.abc.Sequence[tensorrt.tensorrt.DynamicPluginTensorDesc], out: collections.abc.Sequence[tensorrt.tensorrt.DynamicPluginTensorDesc]*) → int

Return the workspace size (in bytes) required by the plugin.

This function is called after the plugin is configured, and possibly during execution. The result should be a sufficient workspace size to deal with inputs and outputs of the given size or any smaller problem.

Note: When implementing a Python-based plugin, implementing this method is optional. The default behavior is equivalent to *return 0*.

Warning: This *get_workspace_size()* method is not available to be called from Python on C++-based plugins

Parameters

- **input_desc** – How to interpret the memory for the input tensors.
- **output_desc** – How to interpret the memory for the output tensors.

Returns
The workspace size (in bytes).

supports_format_combination(*self:* tensorrt.tensorrt.IPluginV3, *pos: SupportsInt*, *in_out: collections.abc.Sequence[*tensorrt.tensorrt.DynamicPluginTensorDesc*]*, *num_inputs: SupportsInt*) → bool

Return true if plugin supports the format and datatype for the input/output indexed by pos.

For this method, inputs are indexed from *[0, num_inputs-1]* and outputs are indexed from *[num_inputs, (num_inputs + num_outputs - 1)]*. *pos* is an index into *in_ou`t, where `0 <= pos < (num_inputs + num_outputs - 1)*.

TensorRT invokes this method to query if the input/output tensor indexed by *pos* supports the format and datatype specified by *in_out[pos].format* and *in_out[pos].type*. The override shall return true if that format and datatype at *in_out[pos]* are supported by the plugin. It is undefined behavior to examine the format or datatype or any tensor that is indexed by a number greater than *pos*.

Warning: This *supports_format_combination()* method is not available to be called from Python on C++-based plugins

Parameters

- **pos** – The input or output tensor index being queried.
- **in_out** – The combined input and output tensor descriptions.
- **num_inputs** – The number of inputs.

Returns
boolean indicating whether the format combination is supported or not.

class tensorrt.**IPluginV3OneRuntime**(**args*, ***kwargs*)

A plugin capability interface that enables the runtime capability (PluginCapabilityType.RUNTIME).

Exposes methods that allow the expression of the runtime properties and behavior of a plugin.

Overloaded function.

1. __init__(self: tensorrt.tensorrt.IPluginV3OneRuntime) -> None
2. __init__(self: tensorrt.tensorrt.IPluginV3OneRuntime, arg0: tensorrt.tensorrt.IPluginV3OneRuntime) -> None

attach_to_context(*self:* tensorrt.tensorrt.IPluginV3, *resource_context:* tensorrt.IPluginResourceContext) → *tensorrt.tensorrt.IPluginV3*

Clone the plugin, attach the cloned plugin object to a execution context and grant the cloned plugin access to some context resources.

This function is called automatically for each plugin when a new execution context is created.

The plugin may use resources provided by the resource_context until the plugin is deleted by TensorRT.

Parameters

resource_context – A resource context that exposes methods to get access to execution context specific resources. A different resource context is guaranteed for each different execution context to which the plugin is attached.

Note: This method should clone the entire IPluginV3 object, not just the runtime interface

---

enqueue(*self:* tensorrt.tensorrt.IPluginV3, *input_desc:* *collections.abc.Sequence[tensorrt.tensorrt.PluginTensorDesc]*, *output_desc:* *collections.abc.Sequence[tensorrt.tensorrt.PluginTensorDesc]*, *inputs:* *collections.abc.Sequence[SupportsInt]*, *outputs: collections.abc.Sequence[SupportsInt]*, *workspace:* *SupportsInt*, *stream: SupportsInt*) → None

Execute the layer.

inputs and *outputs* contains pointers to the corresponding input and output device buffers as their *intptr_t* casts. *stream* also represents an *intptr_t* cast of the CUDA stream in which enqueue should be executed.

Warning: Since input, output, and workspace buffers are created and owned by TRT, care must be taken when writing to them from the Python side.

Warning: In contrast to the C++ API for *enqueue()*, this method must not return an error code. The expected behavior is to throw an appropriate exception. if an error occurs.

Warning: This *enqueue()* method is not available to be called from Python on C++-based plugins.

Parameters

- **input_desc** – how to interpret the memory for the input tensors.
- **output_desc** – how to interpret the memory for the output tensors.
- **inputs** – The memory for the input tensors.
- **outputs** – The memory for the output tensors.
- **workspace** – Workspace for execution.
- **stream** – The stream in which to execute the kernels.

get_fields_to_serialize(*self:* tensorrt.tensorrt.IPluginV3) → tensorrt.tensorrt.PluginFieldCollection_

Return the plugin fields which should be serialized.

Note: The set of plugin fields returned does not necessarily need to match that advertised through get_field_names() of the corresponding plugin creator.

Warning: This *get_fields_to_serialize()* method is not available to be called from Python on C++-based plugins.

on_shape_change(*self: tensorrt.tensorrt.IPluginV3, in: collections.abc.Sequence[tensorrt.tensorrt.PluginTensorDesc], out: collections.abc.Sequence[tensorrt.tensorrt.PluginTensorDesc]*) → None

Called when a plugin is being prepared for execution for specific dimensions. This could happen multiple times in the execution phase, both during creation of an engine by IBuilder and execution of an engine by IExecutionContext.

- IBuilder will call this function once per profile, with *in* resolved to the values specified by the kOPT field of the current profile.
- IExecutionContext will call this during the next subsequent instance of enqueue_v2() or execute_v3() if: (1) The optimization profile is changed (2). An input binding is changed.

Warning: In contrast to the C++ API for *onShapeChange()*, this method must not return an error code. The expected behavior is to throw an appropriate exception if an error occurs.

Warning: This *on_shape_change()* method is not available to be called from Python on C++-based plugins

Parameters

- **in** – The input tensors attributes that are used for configuration.
- **out** – The output tensors attributes that are used for configuration.

set_tactic(*self:* tensorrt.tensorrt.IPluginV3, *tactic: SupportsInt*) → None

Set the tactic to be used in the subsequent call to enqueue().

If no custom tactics were advertised, this will have a value of 0, which is designated as the default tactic.

Warning: In contrast to the C++ API for *setTactic()*, this method must not return an error code. The expected behavior is to throw an appropriate exception if an error occurs.

Warning: This *set_tactic()* method is not available to be called from Python on C++-based plugins.

class tensorrt.**IPluginResourceContext**

Interface for plugins to access per context resources provided by TensorRT

There is no public way to construct an IPluginResourceContext. It appears as an argument to trt.IPluginV3OneRuntime.attach_to_context().

tensorrt.**PluginTensorDesc**

Fields that a plugin might see for an input or output.

scale is only valid when the *type* is *DataType.INT8*. TensorRT will set the value to -1.0 if it is invalid.

Variables

- **dims** – *Dims* Dimensions.
- **format** – *TensorFormat* Tensor format.
- **type** – *DataType* Type.
- **scale** – float Scale for INT8 data type.

class tensorrt.**DynamicPluginTensorDesc**(*self:* tensorrt.tensorrt.DynamicPluginTensorDesc)

Summarizes tensors that a plugin might see for an input or output.

Variables

- **desc** – *PluginTensorDesc* Information required to interpret a pointer to tensor data, except that desc.dims has -1 in place of any runtime dimension..
- **min** – *Dims* Lower bounds on tensor's dimensions.
- **max** – *Dims* Upper bounds on tensor's dimensions.

class tensorrt.**IDimensionExpr**

An *IDimensionExpr* represents an integer expression constructed from constants, input dimensions, and binary operations.

These expressions are can be used in overrides of *IPluginV2DynamicExt::get_output_dimensions()* to define output dimensions in terms of input dimensions.

get_constant_value(*self:* tensorrt.tensorrt.IDimensionExpr) → int

Get the value of the constant.

If is_constant(), returns value of the constant. Else, return int64 minimum.

is_constant(*self:* tensorrt.tensorrt.IDimensionExpr) → bool

Return true if expression is a build-time constant.

is_size_tensor(*self:* tensorrt.tensorrt.IDimensionExpr) → bool

Return true if this denotes the value of a size tensor.

class tensorrt.**DimsExprs**(**args*, ***kwargs*)

Analog of class *Dims* with expressions (*IDimensionExpr*) instead of constants for the dimensions.

Behaves like a Python iterable and lists or tuples of *IDimensionExpr* can be used to construct it.

Overloaded function.

1. __init__(self: tensorrt.tensorrt.DimsExprs) -> None
2. __init__(self: tensorrt.tensorrt.DimsExprs, arg0: collections.abc.Sequence[tensorrt.tensorrt.IDimensionExpr]) -> None
3. __init__(self: tensorrt.tensorrt.DimsExprs, arg0: typing.SupportsInt) -> None

class tensorrt.**IExprBuilder**(*self:* tensorrt.tensorrt.IExprBuilder)

Object for constructing *IDimensionExpr*.

There is no public way to construct an *IExprBuilder*. It appears as an argument to method *IPluginV2DynamicExt::get_output_dimensions()*. Overrides of that method can use that *IExprBuilder* argument to construct expressions that define output dimensions in terms of input dimensions.

Clients should assume that any values constructed by the *IExprBuilder* are destroyed after *IPluginV2DynamicExt::get_output_dimensions()* returns.

constant(*self:* tensorrt.tensorrt.IExprBuilder, *arg0: SupportsInt*) → *tensorrt.tensorrt.IDimensionExpr*

Return a IDimensionExpr for the given value.

declare_size_tensor(*self:* tensorrt.tensorrt.IExprBuilder, *arg0: SupportsInt*, *arg1:* tensorrt.tensorrt.IDimensionExpr, *arg2:* tensorrt.tensorrt.IDimensionExpr) → *tensorrt.tensorrt.IDimensionExpr*

Declare a size tensor at the given output index, with the specified auto-tuning formula and upper bound.

A size tensor allows a plugin to have output dimensions that cannot be computed solely from input dimensions. For example, suppose a plugin implements the equivalent of INonZeroLayer for 2D input. The plugin can have one output for the indices of non-zero elements, and a second output containing the number of non-zero elements. Suppose the input has size [M,N] and has K non-zero elements. The plugin can write K to the second output. When telling TensorRT that the first output has shape [2,K], plugin uses IExprBuilder.constant() and IExprBuilder.declare_size_tensor(1,...) to create the IDimensionExpr that respectively denote 2 and K.

TensorRT also needs to know the value of K to use for auto-tuning and an upper bound on K so that it can allocate memory for the output tensor. In the example, suppose typically half of the plugin's input elements are non-zero, and all the elements might be nonzero. then using M*N/2 might be a good expression for the opt parameter, and M*N for the upper bound. IDimensionsExpr for these expressions can be constructed from IDimensionsExpr for the input dimensions.

operation(*self:* tensorrt.tensorrt.IExprBuilder, *arg0:* tensorrt.tensorrt.DimensionOperation, *arg1:* tensorrt.tensorrt.IDimensionExpr, *arg2:* tensorrt.tensorrt.IDimensionExpr) → *tensorrt.tensorrt.IDimensionExpr*

Return a IDimensionExpr that represents the given operation applied to first and second. Returns None if op is not a valid DimensionOperation.

class tensorrt.**DimensionOperation**(*self:* tensorrt.tensorrt.DimensionOperation, *value: SupportsInt*)

An operation on two IDimensionExprs, which represent integer expressions used in dimension computations.

For example, given two IDimensionExprs x and y and an IExprBuilder eb, eb.operation(DimensionOperation.SUM, x, y) creates a representation of x + y.

Members:

SUM

PROD

MAX

MIN

SUB

EQUAL

LESS

FLOOR_DIV

CEIL_DIV

property name

class tensorrt.**PluginCapabilityType**(*self:* tensorrt.tensorrt.PluginCapabilityType, *value: SupportsInt*)

Enumerates the different capability types a IPluginV3 object may have.

Members:

CORE

BUILD

RUNTIME

property name

class tensorrt.**TensorRTPhase**(*self:* tensorrt.tensorrt.TensorRTPhase, *value: SupportsInt*)

Indicates a phase of operation of TensorRT

Members:

BUILD

RUNTIME

property name

CHAPTER

EIGHT

TENSORRT.PLUGIN

8.1 tensorrt.plugin.register

tensorrt.plugin.**register**(*plugin_id: str*, *lazy_register: bool = False*) → Callable

Wraps a function to register and describe a TensorRT plugin's IO characteristics. In addition, a complete plugin at least needs an *trt.plugin.impl* function to be registered.

This API is only intended to be used as a decorator. The decorated function must have type hints for all inputs as well as return value.

```
(inp0: TensorDesc, inp1: TensorDesc, ..., attr0: SupportedAttrType, attr1:␣
↪SupportedAttrType, ...) -> Union[TensorDesc, Tuple[TensorDesc]]
```

- Input tensors are declared first, each described by a tensor descriptor TensorDesc.
- **Plugin attributes are declared next. "SupportedAttrType" must be one of:**
  - Supported built-in types: int, float, str, bool, bytes (Note: Lists/tuples of these types are not supported)
  - 1-D Numpy arrays of the following types: int8, int16, int32, int64, float16, float32, float64, bool. These must be annotated with 'numpy.typing.NDArray[dtype]', where 'dtype' is the expected numpy dtype.
- If the plugin has only one output, the return annotation could be TensorDesc. Tuple[TensorDesc] could be used for any number of outputs.

By default, the plugin will be immediately registered in the TRT plugin registry. Use the lazy_register argument to change this.

Parameters

- **plugin_id** – An ID for the plugin in the form "{namespace}::{name}", e.g. "my_project::add_plugin". The namespace is used to avoid collisions so using your product/project name is recommended.
- **lazy_register** – During plugin development/when building engine, lazy registration may be used to delay plugin registration until the plugin is explicitly instantiated using *trt.plugin.op.ns.plugin_name(...)*

Listing 1: Registration of an elementwise plugin (output has same characteristics as the input)

```
import tensorrt.plugin as trtp

@trtp.register("my::add_plugin")
def add_plugin_desc(inp0: trtp.TensorDesc, block_size: int) -> Tuple[trtp.
↪TensorDesc]:
    return inp0.like()
```

8.2 tensorrt.plugin.impl

tensorrt.plugin.**impl**(*plugin_id: str*) → Callable

Wraps a function to define an implementation for a plugin already registered through *trt.plugin.register*.

This API is only intended to be used as a decorator. The decorated function is not required to have type hints for input arguments or return value; however, any type hints specified will be validated against the *trt.plugin.register* signature for consistency.

The schema for the function is as follows:

```
(inp0: Tensor, inp1: Tensor, ..., attr0: SupportedAttrType, attr1:␣
↪SupportedAttrType, outputs: Tuple[Tensor], stream: int, tactic: Optional[int]) ->␣
↪None
```

- Input tensors are passed first, each described by a *Tensor*.
- **Plugin attributes are declared next.**
  - Not all attributes included in *trt.plugin.register* must be specified here – they could be a subset.
  - Included attributes will be serialized to the TRT engine. Therefore, only attributes the plugin actually needs to perform inference (within the body of *trt.plugin.impl*) should be included.
- *tactic* is an optional argument. If the plugin is using custom tactics, it must be specified to receive the tactic value to use for the current execution of the plugin.

Parameters

plugin_id – The ID for the plugin in the form "{namespace}::{name}", which must match that used during *trt.plugin.register*

Listing 2: Implementation of an elementwise plugin with an OpenAI Triton kernel

```
import tensorrt.plugin as trtp
import triton
import triton.language as tl

@triton.jit
def add_kernel(x_ptr, y_ptr, n_elements, BLOCK_SIZE: tl.constexpr):
    pid = tl.program_id(0)
    offsets = pid * BLOCK_SIZE + tl.arange(0, BLOCK_SIZE)
    mask = offsets < n_elements
```

(continues on next page)

(continued from previous page)

```
    x = tl.load(x_ptr + offsets, mask=mask)
    tl.store(y_ptr + offsets, x + 1, mask=mask)

@trtp.register("my::add_plugin")
def add_plugin_desc(inp0: trtp.TensorDesc, block_size: int) -> Tuple[trtp.
↪TensorDesc]:
    return inp0.like()

@trtp.impl("my::add_plugin")
def add_plugin_impl(inp0: trtp.Tensor, block_size: int, outputs: Tuple[trtp.Tensor],
↪ stream: int) -> None:

    n = inp0.numel()
    inp0_t = torch.as_tensor(inp0, device="cuda")
    out_t = torch.as_tensor(outputs[0], device="cuda")

    add_kernel[(triton.cdiv(n, block_size),)](inp0_t, out_t, n, BLOCK_SIZE = block_
↪size)
```

8.3 tensorrt.plugin.aot_impl

tensorrt.plugin.**aot_impl**(*plugin_id: str*) → Callable

Wraps a function to define an Ahead-of-Time (AOT) implementation for a plugin already registered through *trt.plugin.register*.

This API is only intended to be used as a decorator. The decorated function is not required to have type hints for input arguments or return value; however, any type hints specified will be validated against the *trt.plugin.register* signature for consistency.

The schema for the function is as follows: .. code-block:: text

(inp0: TensorDesc, inp1: TensorDesc, ..., attr0: SupportedAttrType, attr1: SupportedAttrType, outputs: Tuple[TensorDesc], tactic: Optional[int]) -> Tuple[str, str, KernelLaunchParams, SymExprs]

- Input tensors are passed first, each described by a *TensorDesc*.
- **Plugin attributes are declared next.**
  - Not all attributes included in *trt.plugin.register* must be specified here – they could be a subset.
  - NOTE: Plugin attributes are not serialized into the engine when using an AOT implementation.
- *tactic* is an optional argument. If the plugin is using custom tactics, it must be specified to receive the tactic value to use for the current execution of the plugin.

Parameters

plugin_id – The ID for the plugin in the form "{namespace}::{name}", which must match that used during *trt.plugin.register*

Returns

- kernel_name: The name of the kernel.
- compiled_kernel: Compiled form of the kernel. Presently, only PTX is supported.
- launch_params: The launch parameters for the kernel

- extra_args: Symbolic expressions for scalar inputs to the kernel, located after the tensor inputs and before the tensor outputs

Listing 3: Implementation of an elementwise plugin with an OpenAI Triton kernel

```
import tensorrt.plugin as trtp
import triton
import triton.language as tl

@triton.jit
def add_kernel(x_ptr, n_elements, y_ptr, BLOCK_SIZE: tl.constexpr):
    pid = tl.program_id(0)
    offsets = pid * BLOCK_SIZE + tl.arange(0, BLOCK_SIZE)
    mask = offsets < n_elements
    x = tl.load(x_ptr + offsets, mask=mask)
    tl.store(y_ptr + offsets, x + 1, mask=mask)

@trtp.register("my::add_plugin")
def add_plugin_desc(inp0: trtp.TensorDesc, block_size: int) -> Tuple[trtp.TensorDesc]:
    return inp0.like()

@trtp.aot_impl("my::elemwise_add_plugin")
def add_plugin_aot_impl(
    inp0: trtp.TensorDesc, block_size: int, single_tactic: bool, outputs: Tuple[trtp.TensorDesc], tactic: int
) -> Tuple[Union[str, bytes], Union[str, bytes], trtp.KernelLaunchParams, trtp.SymExprs]:

    type_str = "fp32" if inp0.dtype == trt.float32 else "fp16"

    src = triton.compiler.ASTSource(
        fn=add_kernel,
        signature={
            "x_ptr": f"*{type_str}",
            "n_elements": "i32",
            "y_ptr": f"*{type_str}",
        },
        constexprs={
            "BLOCK_SIZE": block_size,
        },
    )

    compiled_kernel = triton.compile(src)

    N = inp0.shape_expr.numel()
    launch_params = trtp.KernelLaunchParams()

    # grid dims
    launch_params.grid_x = trtp.cdiv(N, block_size)
    # block dims
    launch_params.block_x = compiled_kernel.metadata.num_warps * 32
```

(continues on next page)

(continued from previous page)

```
    # shared memory
    launch_params.shared_mem = compiled_kernel.metadata.shared

    extra_args = trtp.SymIntExprs(1)
    extra_args[0] = trtp.SymInt32(N)

    return compiled_kernel.metadata.name, compiled_kernel.asm["ptx"], launch_params,
↪ extra_args
```

See also:

8.3.1 KernelLaunchParams

class tensorrt.plugin.**KernelLaunchParams**(*grid_x: int* | IDimensionExpr | *SymInt32 = 1*, *grid_y: int* | IDimensionExpr | *SymInt32 = 1*, *grid_z: int* | IDimensionExpr | *SymInt32 = 1*, *block_x: int* | IDimensionExpr | *SymInt32 = 1*, *block_y: int* | IDimensionExpr | *SymInt32 = 1*, *block_z: int* | IDimensionExpr | *SymInt32 = 1*, *shared_mem: int* | IDimensionExpr | *SymInt32 = 0*)

Parameters

- **grid_x** (*Union[int, trt.IDimensionExpr, SymInt32], optional*) – The grid x dimension. Defaults to 1.
- **grid_y** (*Union[int, trt.IDimensionExpr, SymInt32], optional*) – The grid y dimension. Defaults to 1.
- **grid_z** (*Union[int, trt.IDimensionExpr, SymInt32], optional*) – The grid z dimension. Defaults to 1.
- **block_x** (*Union[int, trt.IDimensionExpr, SymInt32], optional*) – The x dimension of each thread block. Defaults to 1.
- **block_y** (*Union[int, trt.IDimensionExpr, SymInt32], optional*) – The y dimension of each thread block. Defaults to 1.
- **block_z** (*Union[int, trt.IDimensionExpr, SymInt32], optional*) – The z dimension of each thread block. Defaults to 1.
- **shared_mem** (*Union[int, trt.IDimensionExpr, SymInt32], optional*) – Shared-memory per thread block in bytes. Defaults to 0.

8.4 tensorrt.plugin.autotune

tensorrt.plugin.**autotune**(*plugin_id: str*) → Callable

Wraps a function to define autotune logic for a plugin already registered through *trt.plugin.register*.

Autotuning is the process by which TensorRT executes the plugin over IO type/format combinations, and any custom tactics advertised as being supported by the plugin. The (type, format, tactic) combination with the lowest latency is used to execute the plugin once the engine is built.

Note: An autotune function is optional. If not specified, TensorRT will assume the plugin only supports input types specified at network creation, output types specifeid through *trt.plugin.register*, and linear formats for all I/O.

This API is only intended to be used as a decorator. The decorated function is not required to have type hints for input arguments or return value; however, any type hints specified will be validated against the *trt.plugin.register* signature for consistency.

The schema for the function is as follows:

```
(inp0: TensorDesc, inp1: TensorDesc, ..., attr0: SupportedAttrType, attr1:␣
↪SupportedAttrType, outputs: Tuple[TensorDesc]) -> List[AutoTuneCombination]
```

- Input tensors are passed first, each described by a *TensorDesc*.
- Plugin attributes are declared next. Not all attributes included in *trt.plugin.register* must be specified here – they could be a subset.
- The function should return a list of *AutoTuneCombination*s.

Parameters

plugin_id – The ID for the plugin in the form "{namespace}::{name}", which must match that used during *trt.plugin.register*

Listing 4: An elementwise add plugin which supports both FP32 and FP16 linear I/O and wants to be tuned over 2 custom tactics.

```
import tensorrt.plugin as trtp

@trtp.register("my::add_plugin")
def add_plugin_desc(inp0: trtp.TensorDesc, block_size: int) -> Tuple[trtp.
↪TensorDesc]:
    return inp0.like()

@trtp.autotune("my::add_plugin")
def add_plugin_autotune(inp0: trtp.TensorDesc, block_size: int, outputs: Tuple[trtp.
↪TensorDesc]) -> List[trtp.AutoTuneCombination]:

    return [trtp.AutoTuneCombination("FP32|FP16, FP32|FP16", "LINEAR", [1, 2])]
```

Listing 5: Same as above example but using index-by-index construction of an *AutoTuneCombination*

```
import tensorrt.plugin as trtp

@trtp.register("my::add_plugin")
def add_plugin_desc(inp0: trtp.TensorDesc, block_size: int) -> Tuple[trtp.
↪TensorDesc]:
    return inp0.like()

@trtp.autotune("my::add_plugin")
def add_plugin_autotune(inp0: trtp.TensorDesc, block_size: int, outputs: Tuple[trtp.
↪TensorDesc]) -> List[trtp.AutoTuneCombination]:
```

(continues on next page)

(continued from previous page)

```
    c = trtp.AutoTuneCombination()
    c.pos(0, "FP32|FP16", "LINEAR")
    c.pos(1, "FP32|FP16") # index 1 is the output. Omitting format is the same as␣
↪declaring it to be LINEAR.
    c.tactics([1, 2])
    return [c]
```

See also:

8.4.1 AutoTuneCombination

class tensorrt.plugin.**AutoTuneCombination**(*io_types: str = None*, *layouts: str = None*, *tactics: Iterable[int] = None*)

Construct a set of supported type/format combinations of a plugin's I/O.

Any custom *tactic* s per each such type/format combination can also be advertised. A tactic is simply another way to calculate the output of a plugin for the same type/format combination of the I/O (e.g. if there are multiple kernels available).

Parameters

- **io_types** (*str, optional*) – A string representation of a type combination.

  Valid format is "type0,type1,...,type#io" where 'type' is of the form "TYPE0[sep]TYPE1[sep]...".

  TYPE is a valid string representation of a *trt.DataType*. These include "FP32" for trt.float32, "FP16" for trt.float16. The string representation of other data types is the same as their name in the trt.DataType enum.

  [sep] is a valid separator, which is either '|' or '*'. Only one of these separators can appear in a given *io_types*.

  (1). '|' indicates a dependent combination: the dependence of the type of one I/O to another I/O. e.g. "FP32|FP16,FP32|FP16" indicates the IO can only be both FP32 or both FP16.

  (2). '*' indicates an independent combination. e.g. "FP32*FP16,FP32|FP16,FP32|FP16" indicates that the first input is independently either FP32 or FP16 regardless of the rest of the IO.

- **layouts** (*str, optional*) – A string representation of a format combination.

  Valid format is "format0,format1,...,format#io" where 'format' is of the form "FORMAT0[sep]FORMAT1[sep]...".

  FORMAT is a valid string representation of a *trt.TensorFormat*. These are string versions for the enum values of *trt.TensorFormat*. e.g. "LINEAR" for *trt.TensorFormat.LINEAR*.

  [sep] is a valid separator, which is either '|' or '*'. The rules are the same as for *io_types*.

- **tactics** (*Iterable[int], optional*) – Custom tactics for this type/format combination. Each custom tactic must be a positive integer. Defaults to default tactic (0).

Listing 6: For a plugin with 3 I/Os, I/O indices 0 and 1 are dependently either FP32/FP16 and index 2 is independently FP32/FP16.

```
@trtp.autotune("my::plugin")
def autotune(inp0: trtp.TensorDesc, inp1: trtp.TensorDesc, outputs: Tuple[trtp.TensorDesc]) -> List[trtp.AutoTuneCombination]:
    # The following would result in the following type combinations:
    # [FP32, FP32, FP32], [FP16, FP16, FP32], [FP32, FP32, FP16], [FP16, FP16, FP16]
    return [trtp.AutoTuneCombination("FP32|FP16, FP32|FP16, FP32|FP16", "LINEAR", [1, 2])]
```

Listing 7: For a plugin with 2 I/Os, the input/output supports either LINEAR or HWC format for FP32 and LINEAR format for FP16.

```
@trtp.autotune("my::plugin")
def autotune(inp0: trtp.TensorDesc, outputs: Tuple[trtp.TensorDesc]) -> List[trtp.AutoTuneCombination]:
    # Even though (FP16, HWC) is not a valid combination (see next example), TRT should intelligently reject those
    # and pass the following combinations to the impl function:
    # [{FP32, FP32}, {LINEAR, LINEAR}], [{FP32, FP32}, {HWC, LINEAR}], [{FP16, FP32}, {LINEAR, LINEAR}]
    return [trtp.AutoTuneCombination("FP32*FP16, FP32", "LINEAR*HWC, LINEAR", [1, 2])]
```

Listing 8: For a plugin with 2 I/Os, the input/output supports either LINEAR or HWC format for FP32 and LINEAR format for FP16 (second method).

```
@trtp.autotune("my::plugin")
def autotune(inp0: trtp.TensorDesc, outputs: Tuple[trtp.TensorDesc]) -> List[trtp.AutoTuneCombination]:
    # We can use two AutoTuneCombination objects to avoid communicating illegal combinations
    return [trtp.AutoTuneCombination("FP32*FP16, FP32", "LINEAR, LINEAR", [1, 2]), trtp.AutoTuneCombination("FP32, FP32", "HWC, LINEAR", [1, 2])]
```

pos(*pos: Iterable[int]*, *io_types: str*, *layouts: str = 'LINEAR'*) → None

Specify I/O types and formats for a specified set of I/O indices.

Parameters

- **pos** (*Iterable[int]*) – I/O indices. Input indices are [0, 1, …, #inputs - 1] and output indices are [#inputs, #inputs + 1, …, #inputs + #outputs - 1].
- **io_types** (*str*) – Data types for these I/O indices.
- **layouts** (*str, optional*) – Tensor format(s) for these I/O indices. Defaults to "LINEAR".

Raises

ValueError – If types or layouts for any of these I/O indices is already specified.

Listing 9: For a plugin with 3 I/Os, I/O indices 0 and 1 are dependently either FP32/FP16 and index 2 is independently FP32/FP16.

```
@trtp.autotune("my::plugin")
def autotune(inp0: trtp.TensorDesc, inp1: trtp.TensorDesc, outputs: Tuple[trtp.
↪TensorDesc]) -> List[trtp.AutoTuneCombination]:
    c = trtp.AutoTuneCombination()
    c.pos([0, 1], "FP32|FP16", "LINEAR")
    c.pos(2, "FP32*FP16") # Omitting format is the same as declaring it to be␣
↪LINEAR.
    c.tactics([1, 2])
    return [c]
```

tactics(*tactics: Iterable[int]*) → None

Specify custom tactics for this type/format combination

Parameters

tactics (*Iterable[int]*) – Custom tactics. These must be positive integers.

8.5 Shape

8.5.1 Shape

class tensorrt.plugin.**Shape**(*tensor_desc: Tuple[int] | DynamicPluginTensorDesc | PluginTensorDesc = None*)

Numerical representation of a tensor shape

property is_dynamic: bool

True if this tensor has at least one dynamic dimension, *False* otherwise.

property max: Tuple[int]

Upper bounds on tensor's dimensions.

property min: Tuple[int]

Lower bounds on tensor's dimensions.

numel() → int

Number of elements contained

Raises

ValueError – When *is_dynamic* is *True*

property opt: Tuple[int]

Optimum value of dimensions specified for auto-tuning.

8.5.2 ShapeExpr

class tensorrt.plugin.**ShapeExpr**(*value: int | IDimensionExpr | ShapeExpr | SymIntExpr = None*)

Symbolic expression for single dimension of a tensor

Parameters

value (*Union[int, trt.IDimensionExpr, ShapeExpr, SymIntExpr], optional*) – Constant or another symbolic expression. Defaults to creating a fake shape expression.

constant_value() → int

Return value of the constant shape expression.

Raises

RuntimeError – For non-constant shape expressions. Check *is_constant* to determine accessibility.

property is_constant: bool

True if this shape expression is a build-time constant, *False* otherwise.

Raises

RuntimeError – For fake *ShapeExpr*s. Check *is_fake* to determine accessibility.

property is_fake: bool

A ShapeExpr may be "fake" when it is accessed in a non-shape calculation context. Fake *ShapeExpr`s are externally indistinguishable unless `is_constant* or *constant_value* is required.

property is_size_tensor: bool

True if this represents a size tensor, *False* otherwise.

class tensorrt.plugin.**ShapeExprs**(*length*, *_is_dummy=False*)

Iterable holding *ShapeExpr*s, representing a tensor shape

Parameters

length (*int*) – Number of dimensions of the tensor

classmethod from_tuple(*shape_exprs: Tuple[*ShapeExpr | *int]*) → *ShapeExpr*

Parameters

shape_exprs (*Tuple[Union[*ShapeExpr*, int]]*) – Tuple to construct *ShapeExprs* from

numel() → *ShapeExpr*

Returns a symbolic expression for the number of elements

class tensorrt.plugin.**SizeTensorShapeExpr**(*size_tensor_desc:* SizeTensorDesc)

Extends *ShapeExpr*

A shape expression that represent a size tensor

Note: It is recommended to use *SizeTensorDesc.expr* to get a *SizeTensorShapeExpr* representing a size tensor

---

8.6 Tensor

8.6.1 TensorDesc

class tensorrt.plugin.**TensorDesc**(*shape_expr:* ShapeExprs = *None*, *dtype: DataType* = *None*, *format: TensorFormat* = *None*, *scale: float* = *None*)

Descriptor for a tensor A *TensorDesc* never contains nor refers to any tensor data.

Parameters

- **shape_expr** (ShapeExprs) – The data with which to initialize the tensor.
- **dtype** (*trt.DataType*) – The data type of the tensor.
- **format** (*trt.TensorFormat*) – Format (layout) of the tensor.
- **scale** (*float*) – Scale for INT8 data type.

Listing 10: Creates a TensorDesc with constant shape expressions

```
tensor = trt.TensorDesc((10, 2, 32, 32), dtype=trt.float32)
```

Listing 11: Creates a TensorDesc from shape expression of another TensorDesc

```
tensor = trt.from_shape_expr(other.shape_expr, dtype=trt.float32)
```

aliased() → *TensorDesc*

Returns

Returns a TensorDesc which has identical properties and is aliased to this tensor (would result in a *Tensor* during enqueue sharing the same data buffer). Returned TensorDesc is immutable.

Listing 12: Communicate that output tensor has identical properties to the input tensor

```
@tensorrt.plugin.register("my::plugin")
def _(inp: tensorrt.plugin.TensorDesc) -> tensorrt.plugin.TensorDesc:
    return inp.aliased()
```

property dtype: DataType

Data type of the tensor.

property format: TensorFormat

The format of the tensor.

> **Warning:** Only accessible when TensorDesc.has_shape is true.

get_aliased() → *TensorDesc*

Returns

Returns a TensorDesc for the tensor which this tensor is aliased to. Returns None is this tensor is not aliased to any other tensor.

property has_shape: bool

True if this tensor has concrete shape information, False otherwise.

property has_shape_expr: bool

True if this tensor has symbolic shape expressions, False otherwise.

property is_aliased: bool

True if this tensor is aliased to another tensor, False otherwise.

property is_dynamic: bool

True if this tensor has at least one dynamic dimension, *False* otherwise.

like() → *TensorDesc*

Returns

Returns a TensorDesc which has identical properties to this tensor, and is mutable.

Listing 13: Communicate that output tensor has identical properties to the input tensor

```
@tensorrt.plugin.register("my::plugin")
def _(inp: tensorrt.plugin.TensorDesc) -> tensorrt.plugin.TensorDesc:
    return inp.like()
```

property ndim: int

Number of dimensions

numel() → int

Returns

Returns an int with the number of elements of the tensor.

> **Warning:** Should only be called when TensorDesc.has_shape is true. If a symbolic expression for the number of elements is required, query TensorDesc.shape_expr.numel().

property scale: float

Scale for INT8 data type.

> **Warning:** Only accessible when TensorDesc.has_shape is true.

property shape: *Shape*

The (concrete) shape of the tensor.

> **Warning:** Only accessible when TensorDesc.has_shape is true.

property shape_expr: *ShapeExprs*

Symbolic expressions for the tensor shape.

class tensorrt.plugin.**SizeTensorDesc**(*opt:* ShapeExpr, *upper_bound:* ShapeExpr)

Extends *TensorDesc*

Descriptor for a size tensor: a scalar of either INT32 or INT64 data type used to express the extent of a data-dependent dimension.

Parameters

- **opt** (ShapeExpr) – Symbolic expression for the extent of this size tensor to use in the autotune process of the engine build
- **upper_bound** (ShapeExpr) – Symbolic expression for the upper-bound of this size tensor

---

Note: It is recommended to construct a size tensor using *size_tensor()* instead of using this constructor directly

---

expr() → *SizeTensorShapeExpr*

Symbolic expression for this size tensor

property index: int

Output index at which this size tensor resides

property opt: *ShapeExpr*

Symbolic expression for the extent of this size tensor to use in the autotune process of the engine build

property upper_bound: *ShapeExpr*

Symbolic expression for the upper-bound of this size tensor

8.6.2 Tensor

class tensorrt.plugin.**Tensor**

Representation of a tensor that carries data

Tensor objects are strictly *descriptors* of a tensor with an underlying data buffer. *tensorrt.plugin* does not provide any APIs that perform standard data-altering operations on *Tensor*s.

Supports *__cuda_array_interface__* for interoperability with other frameworks.

aliased(*shape:* Shape | *Tuple[int]* | *PluginTensorDesc* = *None*) → *Tensor*

Return a *Tensor* which has the same *data_ptr* as this but has the provided *shape*.

Parameters

shape (*Union[*Shape*, Tuple[int], trt.PluginTensorDesc], optional*) – Required shape of the new tensor (must have the same volume). Defaults to same shape.

Raises

ValueError – If *shape* is not a supported type or if it does not have the same volume

property data_ptr: int

Pointer to the data buffer of this tensor

property dtype: DataType

Data type of the tensor.

property format: TensorFormat

The format of the tensor.

get_aliased() → *Tensor*

Returns

Returns *Tensor* of the tensor which this tensor is aliased to. Returns None is this tensor is not aliased to any other tensor.

property is_aliased

True if this tensor is aliased to another tensor, False otherwise.

property is_data_dependent

True if this tensor contains at least one data-dependent dimension, False otherwise.

property ndim: int

Number of dimensions

numel() → int

Returns the number of elements of the tensor

Raises
ValueError – If the tensor has a data-dependent dimension. Examine *is_data_dependent* to determine whether the tensor is data-dependent.

Returns
Number of elements of the tensor

Return type
int

property scale: float

Scale for INT8 data type.

property shape: *Shape*

The (concrete) shape of the tensor.

property strides: Tuple[int]

Strides of this tensor.

8.7 Functions

tensorrt.plugin.**cdiv**(*first: int* | ShapeExpr, *second: int* | ShapeExpr) → *ShapeExpr*

Computes symbolic ceiling division of *first* by *second*

Parameters

- **first** (*Union[int, ShapeExpr]*) – Dividend
- **second** (*Union[int, ShapeExpr]*) – Divisor

Raises
ValueError – If both arguments are *int*s or if *second* evaluates to 0

Returns
Symbolic expression for the ceiling division of *first* by *second*

Return type
ShapeExpr

tensorrt.plugin.**max**(*first: int* | ShapeExpr, *second: int* | ShapeExpr) → *ShapeExpr*

Computes the maximum of *first* and *second*

Parameters

- **first** (*Union[int, ShapeExpr]*) – First operand
- **second** (*Union[int, ShapeExpr]*) – Second operand

Raises
ValueError – If both arguments are *int*s

Returns
Symbolic expression for the maximum of *first* and *second*

Return type
ShapeExpr

tensorrt.plugin.**min**(*first: int* | ShapeExpr, *second: int* | ShapeExpr) → *ShapeExpr*

Computes the minimum of *first* and *second*

Parameters

- **first** (*Union[int, ShapeExpr]*) – First operand
- **second** (*Union[int, ShapeExpr]*) – Second operand

Raises
ValueError – If both arguments are *int*s

Returns
Symbolic expression for the minimum of *first* and *second*

Return type
ShapeExpr

tensorrt.plugin.**size_tensor**(*opt:* ShapeExpr, *upper_bound:* ShapeExpr) → *SizeTensorDesc*

Constructs a size tensor with the specified autotune shape expression *opt* and *upper_bound*

Parameters

- **opt** (ShapeExpr) – Symbolic expression for the extent of this size tensor to use in the autotune process of the engine build
- **upper_bound** (ShapeExpr) – Symbolic expression for the upper-bound of this size tensor

Returns
A tensor descriptor for a size tensor with the specified autotune extent and upper-bound

Return type
SizeTensorDesc

tensorrt.plugin.**from_shape_expr**(*shape_expr: Tuple[*ShapeExpr | *int]* | ShapeExprs, *dtype: DataType*) → *TensorDesc*

Constructs a tensor descriptor with the specified shape expression and data type

Parameters

- **shape_expr** (*Union[Tuple[Union[ShapeExpr, int]], ShapeExprs]*) – Expressions or constants denoting the shape of the tensor
- **dtype** (*trt.DataType*) – Data type of the tensor

Returns
Tensor descriptor with the specified shape expression and data type

Return type
TensorDesc

CHAPTER

NINE

ONNX PARSER

class tensorrt.**OnnxParser**(*self:* tensorrt.tensorrt.OnnxParser, *network:* tensorrt.tensorrt.INetworkDefinition, *logger:* tensorrt.tensorrt.ILogger)

This class is used for parsing ONNX models into a TensorRT network definition

Variables
num_errors – int The number of errors that occurred during prior calls to *parse()*

Parameters

- **network** – The network definition to which the parser will write.
- **logger** – The logger to use.

__del__(*self:* tensorrt.tensorrt.OnnxParser) → None

__exit__(*exc_type*, *exc_value*, *traceback*)

Context managers are deprecated and have no effect. Objects are automatically freed when the reference count reaches 0.

__init__(*self:* tensorrt.tensorrt.OnnxParser, *network:* tensorrt.tensorrt.INetworkDefinition, *logger:* tensorrt.tensorrt.ILogger) → None

Parameters

- **network** – The network definition to which the parser will write.
- **logger** – The logger to use.

clear_errors(*self:* tensorrt.tensorrt.OnnxParser) → None

Clear errors from prior calls to *parse()*

clear_flag(*self: tensorrt.tensorrt.OnnxParser*, *flag: nvonnxparser::OnnxParserFlag*) → None

Clears the parser flag from the enabled flags.

Parameters
flag – The flag to clear.

get_error(*self:* tensorrt.tensorrt.OnnxParser, *index: SupportsInt*) → nvonnxparser::IParserError

Get an error that occurred during prior calls to *parse()*

Parameters
index – Index of the error

get_flag(*self: tensorrt.tensorrt.OnnxParser*, *flag: nvonnxparser::OnnxParserFlag*) → bool

Check if a build mode flag is set.

Parameters
flag – The flag to check.

Returns
A *bool* indicating whether the flag is set.

get_layer_output_tensor(*self:* tensorrt.tensorrt.OnnxParser, *name: str*, *i: SupportsInt*) → *tensorrt.tensorrt.ITensor*

Get the i-th output ITensor object for the ONNX layer "name".

In the case of multiple nodes sharing the same name this function will return the output tensors of the first instance of the node in the ONNX graph.

arg name
The name of the ONNX layer.

arg i
The index of the output.

returns
The output tensor or None if the layer was not found or an invalid index was provided.

get_subgraph_nodes(*self:* tensorrt.tensorrt.OnnxParser, *index: SupportsInt*) → list

Get the nodes of the specified subgraph. Calling before p supportsModelV2 is an undefined behavior. Will return an empty list by default.

Parameters
index – Index of the subgraph.

Returns
List[int] A list of node indices in the subgraph.

get_used_vc_plugin_libraries(*self:* tensorrt.tensorrt.OnnxParser) → list[str]

Query the plugin libraries needed to implement operations used by the parser in a version-compatible engine.

This provides a list of plugin libraries on the filesystem needed to implement operations in the parsed network. If you are building a version-compatible engine using this network, provide this list to IBuilderConfig.set_plugins_to_serialize() to serialize these plugins along with the version-compatible engine, or, if you want to ship these plugin libraries externally to the engine, ensure that IPluginRegistry.load_library() is used to load these libraries in the appropriate runtime before deserializing the corresponding engine.

Returns
List[str] List of plugin libraries found by the parser.

Raises
`RuntimeError` if an internal error occurred when trying to fetch the list of plugin libraries.

is_subgraph_supported(*self:* tensorrt.tensorrt.OnnxParser, *index: SupportsInt*) → bool

Returns whether the subgraph is supported. Calling before p supportsModelV2 is an undefined behavior. Will return false by default.

Parameters
index – Index of the subgraph to be checked.

Returns
true if subgraph is supported

load_initializer(*self:* tensorrt.tensorrt.OnnxParser, *name: str*, *data: SupportsInt*, *size: SupportsInt*) → bool

Prompt the ONNX parser to load an initializer with user-provided binary data. The lifetime of the data must exceed the lifetime of the parser.

All user-provided initializers must be provided prior to calling parse_model_proto().

This function can be called multiple times to specify the names of multiple initializers.

Calling this function with an initializer previously specified will overwrite the previous instance.

This function will return false if initializer validation fails. Possible validation errors are:

- This function was called prior to load_model_proto().
- The requested initializer was not found in the model.
- The size of the data provided is different from the corresponding initializer in the model.

Parameters

- **name** – name of the initializer.
- **data** – binary data of the initializer.
- **size** – the size of the binary data.

Returns
true if the initializer was successfully loaded

load_model_proto(*self:* tensorrt.tensorrt.OnnxParser, *model: collections.abc.Buffer*, *path: str = None*) → bool

Load a serialized ONNX model into the parser. Unlike the parse() or parse_from_file() functions, this function does not immediately convert the model into a TensorRT INetworkDefinition. Using this function allows users to provide their own initializers for the ONNX model through the load_initializer() function.

Only one model can be loaded at a time. Subsequent calls to load_model_proto() will result in an error.

To begin the conversion of the model into a TensorRT INetworkDefinition, use parse_model_proto().

Parameters

- **model** – The serialized ONNX model.
- **path** – The path to the model file. Only required if the model has externally stored weights.

Returns
true if the model was loaded successfully

parse(*self:* tensorrt.tensorrt.OnnxParser, *model: collections.abc.Buffer*, *path: str = None*) → bool

Parse a serialized ONNX model into the TensorRT network.

Parameters

- **model** – The serialized ONNX model.
- **path** – The path to the model file. Only required if the model has externally stored weights.

Returns
true if the model was parsed successfully

parse_from_file(*self:* tensorrt.tensorrt.OnnxParser, *model: str*) → bool

Parse an ONNX model from file into a TensorRT network.

Parameters
model – The path to an ONNX model.

Returns
true if the model was parsed successfully

parse_model_proto(*self:* tensorrt.tensorrt.OnnxParser) → bool

Begin the parsing and conversion process of the loaded ONNX model into a TensorRT INetworkDefinition.

Returns
true if the model was parsed successfully.

set_builder_config(*self:* tensorrt.tensorrt.OnnxParser, *builder_config:* tensorrt.tensorrt.IBuilderConfig) → bool

Set the BuilderConfig for the parser.

Parameters
builder_config – The BuilderConfig to set.

Returns
true if the BuilderConfig was set successfully, false otherwise.

set_flag(*self: tensorrt.tensorrt.OnnxParser*, *flag: nvonnxparser::OnnxParserFlag*) → None

Add the input parser flag to the already enabled flags.

Parameters
flag – The flag to set.

supports_model_v2(*self:* tensorrt.tensorrt.OnnxParser, *model: collections.abc.Buffer*, *path: str = None*) → bool

Check whether TensorRT supports a particular ONNX model. Query each subgraph with num_subgraphs, is_subgraph_supported, get_subgraph_nodes.

Parameters

- **model** – The serialized ONNX model.
- **path** – The path to the model file. Only required if the model has externally stored weights.

Returns
true if the model is supported

supports_operator(*self:* tensorrt.tensorrt.OnnxParser, *op_name: str*) → bool

Returns whether the specified operator may be supported by the parser. Note that a result of true does not guarantee that the operator will be supported in all cases (i.e., this function may return false-positives).

Parameters
op_name – The name of the ONNX operator to check for support

class tensorrt.**OnnxParserRefitter**(*self:* tensorrt.tensorrt.OnnxParserRefitter, *refitter:* tensorrt.tensorrt.Refitter, *logger:* tensorrt.tensorrt.ILogger)

This is an interface designed to refit weights from an ONNX model.

Parameters

- **refitter** – The Refitter object used to refit the model.
- **logger** – The logger to use.

__init__(*self:* tensorrt.tensorrt.OnnxParserRefitter, *refitter:* tensorrt.tensorrt.Refitter, *logger:* tensorrt.tensorrt.ILogger) → None

Parameters

- **refitter** – The Refitter object used to refit the model.
- **logger** – The logger to use.

clear_errors(*self:* tensorrt.tensorrt.OnnxParserRefitter) → None

Clear errors from prior calls to `refitFromBytes()` or `refitFromFile()`.

get_error(*self:* tensorrt.tensorrt.OnnxParserRefitter, *index: SupportsInt*) → *tensorrt.tensorrt.ParserError*

Get an error that occurred during prior calls to `refitFromBytes()` or `refitFromFile()`.

Parameters
index – Index of the error

load_initializer(*self:* tensorrt.tensorrt.OnnxParserRefitter, *name: str*, *data: SupportsInt*, *size: SupportsInt*) → bool

Prompt the ONNX refitter to load an initializer with user-provided binary data. The lifetime of the data must exceed the lifetime of the refitter.

All user-provided initializers must be provided prior to calling refit_model_proto().

This function can be called multiple times to specify the names of multiple initializers.

Calling this function with an initializer previously specified will overwrite the previous instance.

This function will return false if initializer validation fails. Possible validation errors are:

- This function was called prior to load_model_proto().
- The requested initializer was not found in the model.
- The size of the data provided is different from the corresponding initializer in the model.

Parameters

- **name** – name of the initializer.
- **data** – binary data of the initializer.
- **size** – the size of the binary data.

Returns
true if the initializer was successfully loaded.

load_model_proto(*self:* tensorrt.tensorrt.OnnxParserRefitter, *model: collections.abc.Buffer*, *path: str = None*) → bool

Load a serialized ONNX model into the refitter. Unlike the refit() or refit_from_file() functions, this function does not immediately begin the refit process. Using this function allows users to provide their own initializers for the ONNX model through the load_initializer() function.

Only one model can be loaded at a time. Subsequent calls to load_model_proto() will result in an error.

To begin the refit process, use refit_model_proto().

Parameters

- **model** – The serialized ONNX model.
- **path** – The path to the model file. Only required if the model has externally stored weights.

Returns
true if the model was loaded successfully.

refit_from_bytes(*self:* tensorrt.tensorrt.OnnxParserRefitter, *model: collections.abc.Buffer*, *path: str = None*) → bool

Load a serialized ONNX model from memory and perform weight refit.

Parameters

- **model** – The serialized ONNX model.
- **path** – The path to the model file. Only required if the model has externally stored weights.

Returns
true if all the weights in the engine were refit successfully.

refit_from_file(*self:* tensorrt.tensorrt.OnnxParserRefitter, *model: str*) → bool

Load and parse a ONNX model from disk and perform weight refit.

Parameters
model – The path to an ONNX model.

Returns
true if the model was loaded successfully, and if all the weights in the engine were refit successfully.

refit_model_proto(*self:* tensorrt.tensorrt.OnnxParserRefitter) → bool

Begin the refit process from the loaded ONNX model.

Returns
true if the model was refit successfully.

tensorrt.**ErrorCode**

The type of parser error

Members:

SUCCESS

INTERNAL_ERROR

MEM_ALLOC_FAILED

MODEL_DESERIALIZE_FAILED

INVALID_VALUE

INVALID_GRAPH

INVALID_NODE

UNSUPPORTED_GRAPH

UNSUPPORTED_NODE

UNSUPPORTED_NODE_ATTR

UNSUPPORTED_NODE_INPUT

UNSUPPORTED_NODE_DATATYPE

UNSUPPORTED_NODE_DYNAMIC

UNSUPPORTED_NODE_SHAPE

REFIT_FAILED

class tensorrt.**ParserError**

code(*self:* tensorrt.tensorrt.ParserError) → tensorrt.tensorrt.ErrorCode

Returns
The error code

desc(*self:* tensorrt.tensorrt.ParserError) → str

Returns
Description of the error

file(*self:* tensorrt.tensorrt.ParserError) → str

Returns
Source file in which the error occurred

func(*self:* tensorrt.tensorrt.ParserError) → str

Returns
Source function in which the error occurred

line(*self:* tensorrt.tensorrt.ParserError) → int

Returns
Source line at which the error occurred

local_function_stack(*self:* tensorrt.tensorrt.ParserError) → list[str]

Returns
Current stack trace of local functions in which the error occurred

local_function_stack_size(*self:* tensorrt.tensorrt.ParserError) → int

Returns
Size of the current stack trace of local functions in which the error occurred

node(*self:* tensorrt.tensorrt.ParserError) → int

Returns
Index of the Onnx model node in which the error occurred

node_name(*self:* tensorrt.tensorrt.ParserError) → str

Returns
Name of the node in the model in which the error occurred

node_operator(*self:* tensorrt.tensorrt.ParserError) → str

Returns
Name of the node operation in the model in which the error occurred

tensorrt.**OnnxParserFlag**

Flags that control how an ONNX model gets parsed.

Members:

NATIVE_INSTANCENORM :
Parse the ONNX model into the INetworkDefinition with the intention of using TensorRT's native layer implementation over the plugin implementation for InstanceNormalization nodes. This flag is required when building version-compatible or hardware-compatible engines. This flag is ON by default.

ENABLE_UINT8_AND_ASYMMETRIC_QUANTIZATION_DLA :
Enable UINT8 as a quantization data type and asymmetric quantization with non-zero zero-point values in Quantize and Dequantize nodes. The resulting engine must be built targeting DLA version >= 3.16. This flag is OFF by default.

REPORT_CAPABILITY_DLA :
Parse the ONNX model with per-node validation for DLA. If this flag is set, is_subgraph_supported() will also return capability in the context of DLA support. When this flag is set, a valid BuilderConfig must be provided to the parser via set_builder_config(). This flag is OFF by default.

ENABLE_PLUGIN_OVERRIDE :
Allow a loaded plugin with the same name as an ONNX operator type to override the default ONNX implementation, even if the plugin namespace attribute is not set. This flag is useful for custom plugins that are intended to replace standard ONNX operators, for example to provide alternative implementations or improved performance. This flag is OFF by default.

ADJUST_FOR_DLA :
Parse the ONNX model with adjustments to make layers more amenable to running on DLA. This flag is OFF by default.

tensorrt.**get_nv_onnx_parser_version**() → int

Returns
The Onnx Parser version

INDEX

Symbols

__buffer__() (*tensorrt.IHostMemory method*), 21
__del__() (*tensorrt.Builder method*), 36
__del__() (*tensorrt.IBuilderConfig method*), 32
__del__() (*tensorrt.ICudaEngine method*), 41
__del__() (*tensorrt.IExecutionContext method*), 48
__del__() (*tensorrt.IHostMemory method*), 21
__del__() (*tensorrt.INetworkDefinition method*), 67
__del__() (*tensorrt.OnnxParser method*), 163
__del__() (*tensorrt.Refitter method*), 55
__del__() (*tensorrt.Runtime method*), 53
__exit__() (*tensorrt.Builder method*), 36
__exit__() (*tensorrt.IBuilderConfig method*), 32
__exit__() (*tensorrt.ICudaEngine method*), 41
__exit__() (*tensorrt.IExecutionContext method*), 48
__exit__() (*tensorrt.IHostMemory method*), 21
__exit__() (*tensorrt.INetworkDefinition method*), 67
__exit__() (*tensorrt.OnnxParser method*), 163
__exit__() (*tensorrt.Runtime method*), 53
__getitem__() (*tensorrt.ICudaEngine method*), 42
__getitem__() (*tensorrt.INetworkDefinition method*), 67
__init__() (*tensorrt.Builder method*), 36
__init__() (*tensorrt.EngineInspector method*), 65
__init__() (*tensorrt.IBuilderConfig method*), 32
__init__() (*tensorrt.ICudaEngine method*), 42
__init__() (*tensorrt.IDebugListener method*), 40
__init__() (*tensorrt.IExecutionContext method*), 48
__init__() (*tensorrt.IGpuAllocator method*), 63
__init__() (*tensorrt.IHostMemory method*), 21
__init__() (*tensorrt.INetworkDefinition method*), 67
__init__() (*tensorrt.IOutputAllocator method*), 46
__init__() (*tensorrt.IRuntimeConfig method*), 54
__init__() (*tensorrt.ISerializationConfig method*), 66
__init__() (*tensorrt.OnnxParser method*), 163
__init__() (*tensorrt.OnnxParserRefitter method*), 166
__init__() (*tensorrt.Refitter method*), 55
__init__() (*tensorrt.Runtime method*), 53
__len__() (*tensorrt.INetworkDefinition method*), 67
__release_buffer__() (*tensorrt.IHostMemory method*), 21

A

acquire_plugin_resource() (*tensorrt.IPluginRegistry method*), 130
ActivationType (*in module tensorrt*), 96
add_activation() (*tensorrt.INetworkDefinition method*), 67
add_assertion() (*tensorrt.INetworkDefinition method*), 68
add_attention() (*tensorrt.INetworkDefinition method*), 68
add_attention_v2() (*tensorrt.INetworkDefinition method*), 68
add_cast() (*tensorrt.INetworkDefinition method*), 68
add_concatenation() (*tensorrt.INetworkDefinition method*), 69
add_constant() (*tensorrt.INetworkDefinition method*), 69
add_convolution_nd() (*tensorrt.INetworkDefinition method*), 69
add_cumulative() (*tensorrt.INetworkDefinition method*), 69
add_deconvolution_nd() (*tensorrt.INetworkDefinition method*), 70
add_dequantize() (*tensorrt.INetworkDefinition method*), 70
add_dist_collective() (*tensorrt.INetworkDefinition method*), 70
add_dynamic_quantize() (*tensorrt.INetworkDefinition method*), 71
add_dynamic_quantize_v2() (*tensorrt.INetworkDefinition method*), 71
add_einsum() (*tensorrt.INetworkDefinition method*), 71
add_elementwise() (*tensorrt.INetworkDefinition method*), 72
add_fill() (*tensorrt.INetworkDefinition method*), 72
add_gather() (*tensorrt.INetworkDefinition method*), 72
add_gather_v2() (*tensorrt.INetworkDefinition method*), 72
add_grid_sample() (*tensorrt.INetworkDefinition method*), 73
add_identity() (*tensorrt.INetworkDefinition method*), 73

add_if_conditional() (*tensorrt.INetworkDefinition method*), 73
add_input() (*tensorrt.IIfConditional method*), 116
add_input() (*tensorrt.INetworkDefinition method*), 73
add_iterator() (*tensorrt.ILoop method*), 110
add_kv_cache_update() (*tensorrt.INetworkDefinition method*), 73
add_loop() (*tensorrt.INetworkDefinition method*), 74
add_loop_output() (*tensorrt.ILoop method*), 110
add_lrn() (*tensorrt.INetworkDefinition method*), 74
add_matrix_multiply() (*tensorrt.INetworkDefinition method*), 74
add_moe() (*tensorrt.INetworkDefinition method*), 74
add_nms() (*tensorrt.INetworkDefinition method*), 75
add_non_zero() (*tensorrt.INetworkDefinition method*), 76
add_normalization() (*tensorrt.INetworkDefinition method*), 76
add_normalization_v2() (*tensorrt.INetworkDefinition method*), 77
add_one_hot() (*tensorrt.INetworkDefinition method*), 77
add_optimization_profile() (*tensorrt.IBuilderConfig method*), 32
add_output() (*tensorrt.IIfConditional method*), 116
add_padding_nd() (*tensorrt.INetworkDefinition method*), 78
add_parametric_relu() (*tensorrt.INetworkDefinition method*), 78
add_plugin() (*tensorrt.INetworkDefinition method*), 78
add_plugin_v2() (*tensorrt.INetworkDefinition method*), 78
add_plugin_v3() (*tensorrt.INetworkDefinition method*), 79
add_pooling_nd() (*tensorrt.INetworkDefinition method*), 79
add_quantize() (*tensorrt.INetworkDefinition method*), 79
add_ragged_softmax() (*tensorrt.INetworkDefinition method*), 79
add_recurrence() (*tensorrt.ILoop method*), 110
add_reduce() (*tensorrt.INetworkDefinition method*), 80
add_resize() (*tensorrt.INetworkDefinition method*), 80
add_reverse_sequence() (*tensorrt.INetworkDefinition method*), 80
add_rotary_embedding() (*tensorrt.INetworkDefinition method*), 80
add_scale() (*tensorrt.INetworkDefinition method*), 81
add_scale_nd() (*tensorrt.INetworkDefinition method*), 81
add_scatter() (*tensorrt.INetworkDefinition method*), 82
add_select() (*tensorrt.INetworkDefinition method*), 82
add_shape() (*tensorrt.INetworkDefinition method*), 82
add_shuffle() (*tensorrt.INetworkDefinition method*), 82
add_slice() (*tensorrt.INetworkDefinition method*), 82
add_softmax() (*tensorrt.INetworkDefinition method*), 83
add_squeeze() (*tensorrt.INetworkDefinition method*), 83
add_topk() (*tensorrt.INetworkDefinition method*), 83
add_trip_limit() (*tensorrt.ILoop method*), 110
add_unary() (*tensorrt.INetworkDefinition method*), 84
add_unsqueeze() (*tensorrt.INetworkDefinition method*), 84
aliased() (*tensorrt.plugin.Tensor method*), 159
aliased() (*tensorrt.plugin.TensorDesc method*), 157
allocate() (*tensorrt.IGpuAllocator method*), 63
allocate_async() (*tensorrt.IGpuAllocator method*), 63
AllocatorFlag (*in module tensorrt*), 62
aot_impl() (*in module tensorrt.plugin*), 149
append() (*tensorrt.PluginFieldCollection method*), 128
are_weights_marked_refittable() (*tensorrt.INetworkDefinition method*), 85
attach_to_context() (*tensorrt.IPluginV3OneRuntime method*), 142
autotune() (*in module tensorrt.plugin*), 151
AutoTuneCombination (*class in tensorrt.plugin*), 153

B

build_engine_with_config() (*tensorrt.Builder method*), 36
build_serialized_network() (*tensorrt.Builder method*), 36
build_serialized_network_to_stream() (*tensorrt.Builder method*), 37
Builder (*class in tensorrt*), 36
builder (*tensorrt.INetworkDefinition property*), 85
BuilderFlag (*in module tensorrt*), 28

C

can_run_on_DLA() (*tensorrt.IBuilderConfig method*), 32
cdiv() (*in module tensorrt.plugin*), 160
clear() (*tensorrt.IErrorRecorder method*), 60
clear() (*tensorrt.PluginFieldCollection method*), 128
clear_errors() (*tensorrt.OnnxParser method*), 163
clear_errors() (*tensorrt.OnnxParserRefitter method*), 166
clear_flag() (*tensorrt.IBuilderConfig method*), 32
clear_flag() (*tensorrt.ISerializationConfig method*), 66
clear_flag() (*tensorrt.OnnxParser method*), 163
clone() (*tensorrt.IPluginResource method*), 132
clone() (*tensorrt.IPluginV2DynamicExt method*), 133
clone() (*tensorrt.IPluginV3 method*), 138

code() (*tensorrt.ParserError method*), 168
CollectiveOperation (*in module tensorrt*), 122
combine() (*tensorrt.ITimingCache method*), 61
configure_plugin() (*tensorrt.IPluginV2DynamicExt method*), 134
configure_plugin() (*tensorrt.IPluginV3OneBuild method*), 139
constant() (*tensorrt.IExprBuilder method*), 145
constant_value() (*tensorrt.plugin.ShapeExpr method*), 156
create_builder_config() (*tensorrt.Builder method*), 37
create_engine_inspector() (*tensorrt.ICudaEngine method*), 42
create_execution_context() (*tensorrt.ICudaEngine method*), 42
create_network() (*tensorrt.Builder method*), 37
create_optimization_profile() (*tensorrt.Builder method*), 38
create_plugin() (*tensorrt.IPluginCreator method*), 129
create_plugin() (*tensorrt.IPluginCreatorV3One method*), 128
create_runtime_config() (*tensorrt.ICudaEngine method*), 42
create_serialization_config() (*tensorrt.ICudaEngine method*), 42
create_timing_cache() (*tensorrt.IBuilderConfig method*), 32

D

data_ptr (*tensorrt.plugin.Tensor property*), 159
DataType (*in module tensorrt*), 17
deallocate() (*tensorrt.IGpuAllocator method*), 64
deallocate_async() (*tensorrt.IGpuAllocator method*), 64
declare_size_tensor() (*tensorrt.IExprBuilder method*), 145
deregister_creator() (*tensorrt.IPluginRegistry method*), 130
deregister_library() (*tensorrt.IPluginRegistry method*), 131
desc() (*tensorrt.ParserError method*), 168
deserialize_cuda_engine() (*tensorrt.Runtime method*), 53
deserialize_plugin() (*tensorrt.IPluginCreator method*), 129
destroy() (*tensorrt.IPluginV2DynamicExt method*), 134
destroy() (*tensorrt.IPluginV3 method*), 138
DeviceType (*in module tensorrt*), 27
DimensionOperation (*class in tensorrt*), 145
Dims (*class in tensorrt*), 20
Dims2 (*class in tensorrt*), 20
Dims3 (*class in tensorrt*), 20
Dims4 (*class in tensorrt*), 21
DimsExprs (*class in tensorrt*), 144
DimsHW (*class in tensorrt*), 20
dtype (*tensorrt.plugin.Tensor property*), 159
dtype (*tensorrt.plugin.TensorDesc property*), 157
DynamicPluginTensorDesc (*class in tensorrt*), 144

E

ElementWiseOperation (*in module tensorrt*), 100
EngineCapability (*in module tensorrt*), 27
EngineInspector (*class in tensorrt*), 65
enqueue() (*tensorrt.IPluginV2DynamicExt method*), 134
enqueue() (*tensorrt.IPluginV3OneRuntime method*), 142
ErrorCode (*in module tensorrt*), 168
ErrorCodeTRT (*in module tensorrt*), 59
execute_async_v3() (*tensorrt.IExecutionContext method*), 48
execute_v2() (*tensorrt.IExecutionContext method*), 48
ExecutionContextAllocationStrategy (*in module tensorrt*), 54
expr() (*tensorrt.plugin.SizeTensorDesc method*), 159
extend() (*tensorrt.PluginFieldCollection method*), 128

F

file() (*tensorrt.ParserError method*), 169
FillOperation (*in module tensorrt*), 113
format (*tensorrt.plugin.Tensor property*), 159
format (*tensorrt.plugin.TensorDesc property*), 157
from_shape_expr() (*in module tensorrt.plugin*), 161
from_tuple() (*tensorrt.plugin.ShapeExprs class method*), 156
func() (*tensorrt.ParserError method*), 169

G

get_aliased() (*tensorrt.plugin.Tensor method*), 159
get_aliased() (*tensorrt.plugin.TensorDesc method*), 157
get_aliased_input_tensor() (*tensorrt.ICudaEngine method*), 42
get_all() (*tensorrt.Refitter method*), 55
get_all_weights() (*tensorrt.Refitter method*), 55
get_builder_plugin_registry() (*in module tensorrt*), 133
get_capability_interface() (*tensorrt.IPluginV3 method*), 138
get_constant_value() (*tensorrt.IDimensionExpr method*), 144
get_creator() (*tensorrt.IPluginRegistry method*), 131
get_debug_listener() (*tensorrt.IExecutionContext method*), 48

get_debug_state() (*tensorrt.IExecutionContext method*), 48
get_device_memory_size_for_profile_v2() (*tensorrt.ICudaEngine method*), 42
get_device_type() (*tensorrt.IBuilderConfig method*), 33
get_dimension_name() (*tensorrt.ITensor method*), 90
get_engine_information() (*tensorrt.EngineInspector method*), 65
get_engine_stat() (*tensorrt.ICudaEngine method*), 43
get_error() (*tensorrt.OnnxParser method*), 163
get_error() (*tensorrt.OnnxParserRefitter method*), 167
get_error_code() (*tensorrt.IErrorRecorder method*), 60
get_error_desc() (*tensorrt.IErrorRecorder method*), 60
get_execution_context_allocation_strategy() (*tensorrt.IRuntimeConfig method*), 54
get_fields_to_serialize() (*tensorrt.IPluginV3OneRuntime method*), 142
get_flag() (*tensorrt.IBuilderConfig method*), 33
get_flag() (*tensorrt.INetworkDefinition method*), 85
get_flag() (*tensorrt.ISerializationConfig method*), 66
get_flag() (*tensorrt.OnnxParser method*), 163
get_input() (*tensorrt.IAttention method*), 123
get_input() (*tensorrt.ILayer method*), 92
get_input() (*tensorrt.INetworkDefinition method*), 85
get_input_consumed_event() (*tensorrt.IExecutionContext method*), 48
get_layer() (*tensorrt.INetworkDefinition method*), 85
get_layer_information() (*tensorrt.EngineInspector method*), 65
get_layer_output_tensor() (*tensorrt.OnnxParser method*), 164
get_max_output_size() (*tensorrt.IExecutionContext method*), 48
get_memory_pool_limit() (*tensorrt.IBuilderConfig method*), 33
get_missing() (*tensorrt.Refitter method*), 56
get_missing_weights() (*tensorrt.Refitter method*), 56
get_named_weights() (*tensorrt.Refitter method*), 56
get_nv_onnx_parser_version() (*in module tensorrt*), 170
get_output() (*tensorrt.IAttention method*), 123
get_output() (*tensorrt.ILayer method*), 93
get_output() (*tensorrt.INetworkDefinition method*), 85
get_output_allocator() (*tensorrt.IExecutionContext method*), 48
get_output_data_types() (*tensorrt.IPluginV3OneBuild method*), 140
get_output_datatype() (*tensorrt.IPluginV2DynamicExt method*), 135
get_output_dimensions() (*tensorrt.IPluginV2DynamicExt method*), 135
get_output_shapes() (*tensorrt.IPluginV3OneBuild method*), 140
get_output_type() (*tensorrt.ILayer method*), 93
get_plugin_registry() (*in module tensorrt*), 132
get_plugin_registry() (*tensorrt.Builder method*), 38
get_plugin_registry() (*tensorrt.Runtime method*), 54
get_preview_feature() (*tensorrt.IBuilderConfig method*), 33
get_runtime_config() (*tensorrt.IExecutionContext method*), 49
get_serialization_size() (*tensorrt.IPluginV2DynamicExt method*), 135
get_shape() (*tensorrt.IOptimizationProfile method*), 26
get_shape_input() (*tensorrt.IOptimizationProfile method*), 26
get_subgraph_nodes() (*tensorrt.OnnxParser method*), 164
get_tactic_sources() (*tensorrt.IBuilderConfig method*), 33
get_tensor_address() (*tensorrt.IExecutionContext method*), 49
get_tensor_bytes_per_component() (*tensorrt.ICudaEngine method*), 43
get_tensor_components_per_element() (*tensorrt.ICudaEngine method*), 43
get_tensor_dtype() (*tensorrt.ICudaEngine method*), 43
get_tensor_format() (*tensorrt.ICudaEngine method*), 44
get_tensor_format_desc() (*tensorrt.ICudaEngine method*), 44
get_tensor_location() (*tensorrt.ICudaEngine method*), 45
get_tensor_mode() (*tensorrt.ICudaEngine method*), 45
get_tensor_name() (*tensorrt.ICudaEngine method*), 45
get_tensor_profile_shape() (*tensorrt.ICudaEngine method*), 45
get_tensor_profile_values() (*tensorrt.ICudaEngine method*), 45
get_tensor_shape() (*tensorrt.ICudaEngine method*), 45
get_tensor_shape() (*tensorrt.IExecutionContext method*), 49
get_tensor_strides() (*tensorrt.IExecutionContext method*), 49
get_tensor_vectorized_dim() (*tensorrt.ICudaEngine method*), 45
get_timing_cache() (*tensorrt.IBuilderConfig method*), 33
get_used_vc_plugin_libraries() (*ten-*

sorrt.OnnxParser method), 164
get_valid_tactics() (*tensorrt.IPluginV3OneBuild method*), 140
get_weight_streaming_automatic_budget() (*tensorrt.ICudaEngine method*), 46
get_weights_location() (*tensorrt.Refitter method*), 56
get_weights_prototype() (*tensorrt.Refitter method*), 56
get_workspace_size() (*tensorrt.IPluginV2DynamicExt method*), 136
get_workspace_size() (*tensorrt.IPluginV3OneBuild method*), 140

H

HardwareCompatibilityLevel (*in module tensorrt*), 30
has_overflowed() (*tensorrt.IErrorRecorder method*), 60
has_shape (*tensorrt.plugin.TensorDesc property*), 157
has_shape_expr (*tensorrt.plugin.TensorDesc property*), 158

I

IActivationLayer (*class in tensorrt*), 96
IAssertionLayer (*class in tensorrt*), 118
IAttention (*class in tensorrt*), 123
IBuilderConfig (*class in tensorrt*), 30
ICastLayer (*class in tensorrt*), 94
IConcatenationLayer (*class in tensorrt*), 99
IConditionLayer (*class in tensorrt*), 117
IConstantLayer (*class in tensorrt*), 108
IConvolutionLayer (*class in tensorrt*), 94
ICudaEngine (*class in tensorrt*), 40
ICumulativeLayer (*class in tensorrt*), 122
IDebugListener (*class in tensorrt*), 40
IDeconvolutionLayer (*class in tensorrt*), 99
IDequantizeLayer (*class in tensorrt*), 114
IDimensionExpr (*class in tensorrt*), 144
IDistCollectiveLayer (*class in tensorrt*), 122
IDynamicQuantizeLayer (*class in tensorrt*), 115
IEinsumLayer (*class in tensorrt*), 117
IElementWiseLayer (*class in tensorrt*), 100
IErrorRecorder (*class in tensorrt*), 60
IExecutionContext (*class in tensorrt*), 47
IExprBuilder (*class in tensorrt*), 144
IFillLayer (*class in tensorrt*), 113
IGatherLayer (*class in tensorrt*), 101
IGpuAllocator (*class in tensorrt*), 63
IGridSampleLayer (*class in tensorrt*), 95
IHostMemory (*class in tensorrt*), 21
IIdentityLayer (*class in tensorrt*), 108
IIfConditional (*class in tensorrt*), 116
IIfConditionalInputLayer (*class in tensorrt*), 117
IIfConditionalOutputLayer (*class in tensorrt*), 117
IIteratorLayer (*class in tensorrt*), 112
IKVCacheUpdateLayer (*class in tensorrt*), 125
ILayer (*class in tensorrt*), 92
ILogger (*class in tensorrt*), 23
ILogger.Severity (*class in tensorrt*), 23
ILoop (*class in tensorrt*), 110
ILoopBoundaryLayer (*class in tensorrt*), 111
ILoopOutputLayer (*class in tensorrt*), 112
ILRNLayer (*class in tensorrt*), 97
IMatrixMultiplyLayer (*class in tensorrt*), 107
IMoELayer (*class in tensorrt*), 125
impl() (*in module tensorrt.plugin*), 148
index (*tensorrt.plugin.SizeTensorDesc property*), 159
INetworkDefinition (*class in tensorrt*), 67
infer_shapes() (*tensorrt.IExecutionContext method*), 49
init_libnvinfer_plugins() (*in module tensorrt*), 132
initialize() (*tensorrt.IPluginV2DynamicExt method*), 136
INMSLayer (*class in tensorrt*), 119
INonZeroLayer (*class in tensorrt*), 118
INormalizationLayer (*class in tensorrt*), 120
insert() (*tensorrt.PluginFieldCollection method*), 128
InterpolationMode (*in module tensorrt*), 95
IOneHotLayer (*class in tensorrt*), 118
IOptimizationProfile (*class in tensorrt*), 25
IOutputAllocator (*class in tensorrt*), 46
IPaddingLayer (*class in tensorrt*), 103
IParametricReLULayer (*class in tensorrt*), 103
IPluginCapability (*class in tensorrt*), 138
IPluginCreator (*class in tensorrt*), 129
IPluginCreatorInterface (*class in tensorrt*), 128
IPluginCreatorV3One (*class in tensorrt*), 128
IPluginRegistry (*class in tensorrt*), 130
IPluginResource (*class in tensorrt*), 132
IPluginResourceContext (*class in tensorrt*), 143
IPluginV2DynamicExt (*class in tensorrt*), 133
IPluginV2Layer (*class in tensorrt*), 101
IPluginV3 (*class in tensorrt*), 137
IPluginV3Layer (*class in tensorrt*), 101
IPluginV3OneBuild (*class in tensorrt*), 139
IPluginV3OneCore (*class in tensorrt*), 138
IPluginV3OneRuntime (*class in tensorrt*), 141
IPoolingLayer (*class in tensorrt*), 97
IProfiler (*class in tensorrt*), 24
IProgressMonitor (*class in tensorrt*), 39
IQuantizeLayer (*class in tensorrt*), 114
IRaggedSoftMaxLayer (*class in tensorrt*), 108
IRecurrenceLayer (*class in tensorrt*), 111
IReduceLayer (*class in tensorrt*), 102
IResizeLayer (*class in tensorrt*), 108
IReverseSequenceLayer (*class in tensorrt*), 120

IRotaryEmbeddingLayer (*class in tensorrt*), 124
IRuntimeConfig (*class in tensorrt*), 54
is_aliased (*tensorrt.plugin.Tensor property*), 159
is_aliased (*tensorrt.plugin.TensorDesc property*), 158
is_alpha_beta_int64() (*tensorrt.IFillLayer method*), 113
is_constant (*tensorrt.plugin.ShapeExpr property*), 156
is_constant() (*tensorrt.IDimensionExpr method*), 144
is_data_dependent (*tensorrt.plugin.Tensor property*), 160
is_debug_tensor() (*tensorrt.ICudaEngine method*), 46
is_debug_tensor() (*tensorrt.INetworkDefinition method*), 85
is_device_type_set() (*tensorrt.IBuilderConfig method*), 33
is_dynamic (*tensorrt.plugin.Shape property*), 155
is_dynamic (*tensorrt.plugin.TensorDesc property*), 158
is_fake (*tensorrt.plugin.ShapeExpr property*), 156
is_network_supported() (*tensorrt.Builder method*), 38
is_shape_inference_io() (*tensorrt.ICudaEngine method*), 46
is_size_tensor (*tensorrt.plugin.ShapeExpr property*), 156
is_size_tensor() (*tensorrt.IDimensionExpr method*), 144
is_subgraph_supported() (*tensorrt.OnnxParser method*), 164
IScaleLayer (*class in tensorrt*), 98
IScatterLayer (*class in tensorrt*), 116
ISelectLayer (*class in tensorrt*), 103
ISerializationConfig (*class in tensorrt*), 66
IShapeLayer (*class in tensorrt*), 106
IShuffleLayer (*class in tensorrt*), 104
ISliceLayer (*class in tensorrt*), 104
ISoftMaxLayer (*class in tensorrt*), 98
ISqueezeLayer (*class in tensorrt*), 121
ITensor (*class in tensorrt*), 90
ITimingCache (*class in tensorrt*), 61
ITopKLayer (*class in tensorrt*), 106
ITripLimitLayer (*class in tensorrt*), 111
IUnaryLayer (*class in tensorrt*), 102
IUnsqueezeLayer (*class in tensorrt*), 121

K

KernelLaunchParams (*class in tensorrt.plugin*), 151

L

LayerType (*in module tensorrt*), 91
like() (*tensorrt.plugin.TensorDesc method*), 158
line() (*tensorrt.ParserError method*), 169
load_initializer() (*tensorrt.OnnxParser method*), 164
load_initializer() (*tensorrt.OnnxParserRefitter method*), 167
load_library() (*tensorrt.IPluginRegistry method*), 131
load_model_proto() (*tensorrt.OnnxParser method*), 165
load_model_proto() (*tensorrt.OnnxParserRefitter method*), 167
load_runtime() (*tensorrt.Runtime method*), 54
local_function_stack() (*tensorrt.ParserError method*), 169
local_function_stack_size() (*tensorrt.ParserError method*), 169
log() (*tensorrt.ILogger method*), 24
log() (*tensorrt.Logger method*), 24
Logger (*class in tensorrt*), 24
LoopOutput (*in module tensorrt*), 112

M

mark_debug() (*tensorrt.INetworkDefinition method*), 86
mark_output() (*tensorrt.INetworkDefinition method*), 86
mark_output_for_shapes() (*tensorrt.INetworkDefinition method*), 86
mark_unfused_tensors_as_debug_tensors() (*tensorrt.INetworkDefinition method*), 86
mark_weights_refittable() (*tensorrt.INetworkDefinition method*), 86
MatrixOperation (*in module tensorrt*), 107
max (*tensorrt.plugin.Shape property*), 155
max() (*in module tensorrt.plugin*), 160
MemoryPoolType (*in module tensorrt*), 29
min (*tensorrt.plugin.Shape property*), 155
min() (*in module tensorrt.plugin*), 161

N

name (*tensorrt.DimensionOperation property*), 146
name (*tensorrt.ILogger.Severity property*), 24
name (*tensorrt.PluginCapabilityType property*), 146
name (*tensorrt.TensorRTPhase property*), 146
ndim (*tensorrt.plugin.Tensor property*), 160
ndim (*tensorrt.plugin.TensorDesc property*), 158
NetworkDefinitionCreationFlag (*in module tensorrt*), 35
node() (*tensorrt.ParserError method*), 169
node_name() (*tensorrt.ParserError method*), 169
node_operator() (*tensorrt.ParserError method*), 169
nptype() (*in module tensorrt*), 18
num_errors() (*tensorrt.IErrorRecorder method*), 61
num_ranks (*tensorrt.IAttention property*), 124
num_ranks (*tensorrt.ILayer property*), 93
numel() (*tensorrt.plugin.Shape method*), 155
numel() (*tensorrt.plugin.ShapeExprs method*), 156
numel() (*tensorrt.plugin.Tensor method*), 160

numel() (*tensorrt.plugin.TensorDesc method*), 158
numpy() (*tensorrt.Weights method*), 19

O

on_shape_change() (*tensorrt.IPluginV3OneRuntime method*), 143
OnnxParser (*class in tensorrt*), 163
OnnxParserFlag (*in module tensorrt*), 169
OnnxParserRefitter (*class in tensorrt*), 166
operation() (*tensorrt.IExprBuilder method*), 145
opt (*tensorrt.plugin.Shape property*), 155
opt (*tensorrt.plugin.SizeTensorDesc property*), 159

P

PaddingMode (*in module tensorrt*), 94
parse() (*tensorrt.OnnxParser method*), 165
parse_from_file() (*tensorrt.OnnxParser method*), 165
parse_model_proto() (*tensorrt.OnnxParser method*), 165
ParserError (*class in tensorrt*), 168
Permutation (*class in tensorrt*), 103
phase_finish() (*tensorrt.IProgressMonitor method*), 39
phase_start() (*tensorrt.IProgressMonitor method*), 39
PluginCapabilityType (*class in tensorrt*), 146
PluginField (*class in tensorrt*), 127
PluginFieldCollection (*class in tensorrt*), 128
PluginFieldType (*in module tensorrt*), 127
PluginTensorDesc (*in module tensorrt*), 144
PoolingType (*in module tensorrt*), 97
pop() (*tensorrt.PluginFieldCollection method*), 128
pos() (*tensorrt.plugin.AutoTuneCombination method*), 154
PreviewFeature (*in module tensorrt*), 29
process_debug_tensor() (*tensorrt.IDebugListener method*), 40
Profiler (*class in tensorrt*), 25
ProfilingVerbosity (*in module tensorrt*), 27

Q

query() (*tensorrt.ITimingCache method*), 61
queryKeys() (*tensorrt.ITimingCache method*), 62

R

reallocate() (*tensorrt.IGpuAllocator method*), 64
ReduceOperation (*in module tensorrt*), 102
refit_cuda_engine() (*tensorrt.Refitter method*), 56
refit_cuda_engine_async() (*tensorrt.Refitter method*), 57
refit_from_bytes() (*tensorrt.OnnxParserRefitter method*), 167
refit_from_file() (*tensorrt.OnnxParserRefitter method*), 168
refit_model_proto() (*tensorrt.OnnxParserRefitter method*), 168
Refitter (*class in tensorrt*), 55
register() (*in module tensorrt.plugin*), 147
register_creator() (*tensorrt.IPluginRegistry method*), 131
release() (*tensorrt.IPluginResource method*), 132
release_plugin_resource() (*tensorrt.IPluginRegistry method*), 132
remove_tensor() (*tensorrt.INetworkDefinition method*), 86
report_error() (*tensorrt.IErrorRecorder method*), 61
report_layer_time() (*tensorrt.IProfiler method*), 24
report_layer_time() (*tensorrt.Profiler method*), 25
report_to_profiler() (*tensorrt.IExecutionContext method*), 49
reset() (*tensorrt.Builder method*), 38
reset() (*tensorrt.IBuilderConfig method*), 33
reset() (*tensorrt.ITimingCache method*), 62
reset_device_type() (*tensorrt.IBuilderConfig method*), 34
Runtime (*class in tensorrt*), 53

S

SampleMode (*in module tensorrt*), 95
scale (*tensorrt.plugin.Tensor property*), 160
scale (*tensorrt.plugin.TensorDesc property*), 158
ScaleMode (*in module tensorrt*), 98
SerializationFlag (*in module tensorrt*), 65
serialize() (*tensorrt.ICudaEngine method*), 46
serialize() (*tensorrt.IPluginV2DynamicExt method*), 136
serialize() (*tensorrt.ITimingCache method*), 62
serialize_with_config() (*tensorrt.ICudaEngine method*), 46
set_all_tensors_debug_state() (*tensorrt.IExecutionContext method*), 50
set_aux_streams() (*tensorrt.IExecutionContext method*), 50
set_builder_config() (*tensorrt.OnnxParser method*), 166
set_communicator() (*tensorrt.IExecutionContext method*), 50
set_condition() (*tensorrt.IIfConditional method*), 116
set_debug_listener() (*tensorrt.IExecutionContext method*), 50
set_device_memory() (*tensorrt.IExecutionContext method*), 50
set_device_type() (*tensorrt.IBuilderConfig method*), 34
set_dimension_name() (*tensorrt.ITensor method*), 90
set_execution_context_allocation_strategy() (*tensorrt.IRuntimeConfig method*), 55
set_flag() (*tensorrt.IBuilderConfig method*), 34

set_flag() (*tensorrt.ISerializationConfig method*), 66
set_flag() (*tensorrt.OnnxParser method*), 166
set_gated_biases() (*tensorrt.IMoELayer method*), 126
set_gated_weights() (*tensorrt.IMoELayer method*), 126
set_input() (*tensorrt.IAttention method*), 124
set_input() (*tensorrt.IFillLayer method*), 113
set_input() (*tensorrt.IKVCacheUpdateLayer method*), 125
set_input() (*tensorrt.ILayer method*), 93
set_input() (*tensorrt.ILoopOutputLayer method*), 112
set_input() (*tensorrt.IMoELayer method*), 126
set_input() (*tensorrt.INMSLayer method*), 119
set_input() (*tensorrt.IRecurrenceLayer method*), 111
set_input() (*tensorrt.IResizeLayer method*), 109
set_input() (*tensorrt.IRotaryEmbeddingLayer method*), 124
set_input() (*tensorrt.IShuffleLayer method*), 104
set_input() (*tensorrt.ISliceLayer method*), 105
set_input() (*tensorrt.ISqueezeLayer method*), 121
set_input() (*tensorrt.ITopKLayer method*), 106
set_input() (*tensorrt.IUnsqueezeLayer method*), 121
set_input_consumed_event() (*tensorrt.IExecutionContext method*), 51
set_input_shape() (*tensorrt.IExecutionContext method*), 51
set_memory_pool_limit() (*tensorrt.IBuilderConfig method*), 34
set_named_weights() (*tensorrt.Refitter method*), 57
set_optimization_profile_async() (*tensorrt.IExecutionContext method*), 51
set_output_allocator() (*tensorrt.IExecutionContext method*), 52
set_preview_feature() (*tensorrt.IBuilderConfig method*), 34
set_quantization_dynamic_dbl_q() (*tensorrt.IMoELayer method*), 126
set_quantization_static() (*tensorrt.IMoELayer method*), 126
set_shape() (*tensorrt.IOptimizationProfile method*), 26
set_shape_input() (*tensorrt.IOptimizationProfile method*), 26
set_swiglu_params() (*tensorrt.IMoELayer method*), 126
set_tactic() (*tensorrt.IPluginV3OneRuntime method*), 143
set_tactic_sources() (*tensorrt.IBuilderConfig method*), 35
set_tensor_address() (*tensorrt.IExecutionContext method*), 52
set_tensor_debug_state() (*tensorrt.IExecutionContext method*), 52
set_timing_cache() (*tensorrt.IBuilderConfig method*), 35
set_weights() (*tensorrt.Refitter method*), 58
set_weights_name() (*tensorrt.INetworkDefinition method*), 86
Shape (*class in tensorrt.plugin*), 155
shape (*tensorrt.plugin.Tensor property*), 160
shape (*tensorrt.plugin.TensorDesc property*), 158
shape_expr (*tensorrt.plugin.TensorDesc property*), 158
ShapeExpr (*class in tensorrt.plugin*), 156
ShapeExprs (*class in tensorrt.plugin*), 156
size_tensor() (*in module tensorrt.plugin*), 161
SizeTensorDesc (*class in tensorrt.plugin*), 158
SizeTensorShapeExpr (*class in tensorrt.plugin*), 156
step_complete() (*tensorrt.IProgressMonitor method*), 39
strides (*tensorrt.plugin.Tensor property*), 160
supports_format_combination() (*tensorrt.IPluginV2DynamicExt method*), 137
supports_format_combination() (*tensorrt.IPluginV3OneBuild method*), 141
supports_model_v2() (*tensorrt.OnnxParser method*), 166
supports_operator() (*tensorrt.OnnxParser method*), 166

T

tactics() (*tensorrt.plugin.AutoTuneCombination method*), 155
TacticSource (*in module tensorrt*), 27
TempfileControlFlag (*in module tensorrt*), 52
Tensor (*class in tensorrt.plugin*), 159
TensorDesc (*class in tensorrt.plugin*), 157
TensorFormat (*in module tensorrt*), 88
TensorIOMode (*in module tensorrt*), 40
TensorLocation (*in module tensorrt*), 88
TensorRTPhase (*class in tensorrt*), 146
terminate() (*tensorrt.IPluginV2DynamicExt method*), 137
TilingOptimizationLevel (*in module tensorrt*), 30
TopKOperation (*in module tensorrt*), 106
TripLimit (*in module tensorrt*), 111

U

UnaryOperation (*in module tensorrt*), 101
unmark_debug() (*tensorrt.INetworkDefinition method*), 87
unmark_output() (*tensorrt.INetworkDefinition method*), 87
unmark_output_for_shapes() (*tensorrt.INetworkDefinition method*), 87
unmark_unfused_tensors_as_debug_tensors() (*tensorrt.INetworkDefinition method*), 87
unmark_weights_refittable() (*tensorrt.INetworkDefinition method*), 87

unset_named_weights() (*tensorrt.Refitter method*), 58
update() (*tensorrt.ITimingCache method*), 62
update_device_memory_size_for_shapes() (*tensorrt.IExecutionContext method*), 52
upper_bound (*tensorrt.plugin.SizeTensorDesc property*), 159

V

volume() (*in module tensorrt*), 19

W

Weights (*class in tensorrt*), 18
WeightsRole (*in module tensorrt*), 18

Copyright

©2021-2026, NVIDIA Corporation