



NVIDIA TensorRT 8.5.10 API Reference for DRIVE OS

NVIDIA DRIVE OS 6.0

TensorRT API Reference

8.6.0

December 2022

Table 1 Revision history

Date	Summary of change
Aug, 2022	Initial draft
Dec, 2022	Review
Dec, 2022	Approval review

Contents

1	Standard and Safe	1
2	TensorRT	3
3	Deprecated List	5
4	Namespace Index	13
4.1	Namespace List	13
5	Hierarchical Index	15
5.1	Class Hierarchy	15
6	Class Index	19
6.1	Class List	19
7	File Index	25
7.1	File List	25
8	Namespace Documentation	27
8.1	nvcaffeparser1 Namespace Reference	27
8.1.1	Detailed Description	27
8.1.2	Function Documentation	27
8.1.2.1	createCaffeParser()	28
8.1.2.2	shutdownProtobufLibrary()	28
8.2	nvinfer1 Namespace Reference	28
8.2.1	Detailed Description	38
8.2.2	Typedef Documentation	38
8.2.2.1	AllocatorFlags	38
8.2.2.2	AsciiChar	38
8.2.2.3	BuilderFlags	38
8.2.2.4	char_t	38
8.2.2.5	Dims	38
8.2.2.6	NetworkDefinitionCreationFlags	39
8.2.2.7	PluginFormat	39
8.2.2.8	QuantizationFlags	39
8.2.2.9	ResizeMode	40
8.2.2.10	SliceMode	40
8.2.2.11	TacticSources	40
8.2.2.12	TempfileControlFlags	40
8.2.2.13	TensorFormats	41
8.2.3	Enumeration Type Documentation	41
8.2.3.1	ActivationType	41
8.2.3.2	AllocatorFlag	41

8.2.3.3	BoundingBoxFormat	42
8.2.3.4	BuilderFlag	42
8.2.3.5	CalibrationAlgoType	44
8.2.3.6	DataType	44
8.2.3.7	DeviceType	45
8.2.3.8	DimensionOperation	45
8.2.3.9	ElementWiseOperation	46
8.2.3.10	EngineCapability	47
8.2.3.11	ErrorCode	47
8.2.3.12	FillOperation	49
8.2.3.13	GatherMode	49
8.2.3.14	HardwareCompatibilityLevel	49
8.2.3.15	InterpolationMode	50
8.2.3.16	LayerInformationFormat	50
8.2.3.17	LayerType	51
8.2.3.18	LoopOutput	52
8.2.3.19	MatrixOperation	52
8.2.3.20	MemoryPoolType	53
8.2.3.21	NetworkDefinitionCreationFlag	54
8.2.3.22	OptProfileSelector	54
8.2.3.23	PaddingMode	55
8.2.3.24	PluginFieldType	58
8.2.3.25	PluginVersion	58
8.2.3.26	PoolingType	58
8.2.3.27	PreviewFeature	59
8.2.3.28	ProfilingVerbosity	60
8.2.3.29	QuantizationFlag	60
8.2.3.30	ReduceOperation	60
8.2.3.31	ResizeCoordinateTransformation	61
8.2.3.32	ResizeRoundMode	62
8.2.3.33	ResizeSelector	62
8.2.3.34	RNNDirection	62
8.2.3.35	RNNGateType	63
8.2.3.36	RNNInputMode	63
8.2.3.37	RNNOperation	64
8.2.3.38	SampleMode	66
8.2.3.39	ScaleMode	66
8.2.3.40	ScatterMode	67
8.2.3.41	TacticSource	67
8.2.3.42	TempfileControlFlag	68
8.2.3.43	TensorFormat	68
8.2.3.44	TensorIOMode	71
8.2.3.45	TensorLocation	71
8.2.3.46	TopKOperation	72
8.2.3.47	TripLimit	72
8.2.3.48	UnaryOperation	72
8.2.3.49	WeightsRole	73
8.2.4	Function Documentation	74
8.2.4.1	EnumMax()	74
8.2.4.2	EnumMax< BoundingBoxFormat >()	74
8.2.4.3	EnumMax< BuilderFlag >()	74
8.2.4.4	EnumMax< CalibrationAlgoType >()	74
8.2.4.5	EnumMax< DeviceType >()	75
8.2.4.6	EnumMax< DimensionOperation >()	75

8.2.4.7	EnumMax< FillOperation >()	75
8.2.4.8	EnumMax< GatherMode >()	75
8.2.4.9	EnumMax< LayerInformationFormat >()	76
8.2.4.10	EnumMax< LayerType >()	76
8.2.4.11	EnumMax< LoopOutput >()	76
8.2.4.12	EnumMax< MatrixOperation >()	76
8.2.4.13	EnumMax< MemoryPoolType >()	77
8.2.4.14	EnumMax< NetworkDefinitionCreationFlag >()	77
8.2.4.15	EnumMax< OptProfileSelector >()	77
8.2.4.16	EnumMax< ProfilingVerbosity >()	77
8.2.4.17	EnumMax< QuantizationFlag >()	78
8.2.4.18	EnumMax< ReduceOperation >()	78
8.2.4.19	EnumMax< RNNDirection >()	78
8.2.4.20	EnumMax< RNNGateType >()	78
8.2.4.21	EnumMax< RNNInputMode >()	79
8.2.4.22	EnumMax< RNNOperation >()	79
8.2.4.23	EnumMax< SampleMode >()	79
8.2.4.24	EnumMax< ScaleMode >()	79
8.2.4.25	EnumMax< ScatterMode >()	80
8.2.4.26	EnumMax< TacticSource >()	80
8.2.4.27	EnumMax< TempfileControlFlag >()	80
8.2.4.28	EnumMax< TopKOperation >()	80
8.2.4.29	EnumMax< TripLimit >()	81
8.2.4.30	EnumMax< UnaryOperation >()	81
8.2.4.31	EnumMax< WeightsRole >()	81
8.2.4.32	getBuilderPluginRegistry()	81
8.3	nvinfer1::consistency Namespace Reference	82
8.4	nvinfer1::impl Namespace Reference	82
8.5	nvinfer1::plugin Namespace Reference	83
8.5.1	Enumeration Type Documentation	83
8.5.1.1	CodeTypeSSD	83
8.6	nvinfer1::safe Namespace Reference	84
8.6.1	Detailed Description	84
8.6.2	Function Documentation	84
8.6.2.1	createInferRuntime()	85
8.6.2.2	getSafePluginRegistry()	85
8.7	nvinfer1::utils Namespace Reference	85
8.7.1	Function Documentation	86
8.7.1.1	reorderSubBuffers()	86
8.7.1.2	reshapeWeights()	87
8.7.1.3	transposeSubBuffers()	88
8.8	nvonnxparser Namespace Reference	88
8.8.1	Detailed Description	89
8.8.2	Enumeration Type Documentation	89
8.8.2.1	ErrorCode	89
8.8.3	Function Documentation	90
8.8.3.1	createONNXConfig()	90
8.8.3.2	EnumMax()	90
8.8.3.3	EnumMax< ErrorCode >()	90
8.9	nvuffparser Namespace Reference	90
8.9.1	Detailed Description	91
8.9.2	Enumeration Type Documentation	91
8.9.2.1	FieldType	91
8.9.2.2	UffInputOrder	91

8.9.3	Function Documentation	92
8.9.3.1	createUffParser()	92
8.9.3.2	shutdownProtobufLibrary()	92
9	Class Documentation	93
9.1	nvinfer1::plugin::DetectionOutputParameters Struct Reference	93
9.1.1	Detailed Description	93
9.1.2	Member Data Documentation	94
9.1.2.1	backgroundLabelId	94
9.1.2.2	codeType	94
9.1.2.3	confidenceThreshold	94
9.1.2.4	confSigmoid	95
9.1.2.5	inputOrder	95
9.1.2.6	isBatchAgnostic	95
9.1.2.7	isNormalized	95
9.1.2.8	keepTopK	95
9.1.2.9	nmsThreshold	95
9.1.2.10	numClasses	95
9.1.2.11	shareLocation	96
9.1.2.12	topK	96
9.1.2.13	varianceEncodedInTarget	96
9.2	Dims Class Reference	96
9.2.1	Detailed Description	96
9.3	nvinfer1::Dims2 Class Reference	97
9.3.1	Detailed Description	97
9.3.2	Constructor & Destructor Documentation	97
9.3.2.1	Dims2() [1/2]	97
9.3.2.2	Dims2() [2/2]	97
9.4	nvinfer1::Dims3 Class Reference	98
9.4.1	Detailed Description	98
9.4.2	Constructor & Destructor Documentation	98
9.4.2.1	Dims3() [1/2]	99
9.4.2.2	Dims3() [2/2]	99
9.5	nvinfer1::Dims32 Class Reference	99
9.5.1	Member Data Documentation	100
9.5.1.1	d	100
9.5.1.2	MAX_DIMS	100
9.5.1.3	nbDims	100
9.6	nvinfer1::Dims4 Class Reference	101
9.6.1	Detailed Description	101
9.6.2	Constructor & Destructor Documentation	101
9.6.2.1	Dims4() [1/2]	101
9.6.2.2	Dims4() [2/2]	101
9.7	nvinfer1::DimsExprs Class Reference	102
9.7.1	Detailed Description	102
9.7.2	Member Data Documentation	102
9.7.2.1	d	102
9.7.2.2	nbDims	103
9.8	nvinfer1::DimsHW Class Reference	103
9.8.1	Detailed Description	103
9.8.2	Constructor & Destructor Documentation	104
9.8.2.1	DimsHW() [1/2]	104
9.8.2.2	DimsHW() [2/2]	104
9.8.3	Member Function Documentation	104

9.8.3.1	h() [1/2]	104
9.8.3.2	h() [2/2]	105
9.8.3.3	w() [1/2]	105
9.8.3.4	w() [2/2]	105
9.9	nvinfer1::DynamicPluginTensorDesc Class Reference	105
9.9.1	Detailed Description	106
9.9.2	Member Data Documentation	106
9.9.2.1	desc	106
9.9.2.2	max	106
9.9.2.3	min	106
9.10	nvinfer1::impl::EnumMaxImpl< T > Struct Template Reference	107
9.10.1	Detailed Description	107
9.11	nvinfer1::impl::EnumMaxImpl< ActivationType > Struct Reference	107
9.11.1	Detailed Description	107
9.11.2	Member Data Documentation	107
9.11.2.1	kVALUE	107
9.12	nvinfer1::impl::EnumMaxImpl< AllocatorFlag > Struct Reference	108
9.12.1	Detailed Description	108
9.12.2	Member Data Documentation	108
9.12.2.1	kVALUE	108
9.13	nvinfer1::impl::EnumMaxImpl< DataType > Struct Reference	108
9.13.1	Detailed Description	109
9.13.2	Member Data Documentation	109
9.13.2.1	kVALUE	109
9.14	nvinfer1::impl::EnumMaxImpl< ElementWiseOperation > Struct Reference	109
9.14.1	Detailed Description	109
9.14.2	Member Data Documentation	109
9.14.2.1	kVALUE	110
9.15	nvinfer1::impl::EnumMaxImpl< EngineCapability > Struct Reference	110
9.15.1	Detailed Description	110
9.15.2	Member Data Documentation	110
9.15.2.1	kVALUE	110
9.16	nvinfer1::impl::EnumMaxImpl< ErrorCode > Struct Reference	111
9.16.1	Detailed Description	111
9.16.2	Member Data Documentation	111
9.16.2.1	kVALUE	111
9.17	nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel > Struct Reference	111
9.17.1	Detailed Description	112
9.17.2	Member Data Documentation	112
9.17.2.1	kVALUE	112
9.18	nvinfer1::impl::EnumMaxImpl< ILogger::Severity > Struct Reference	112
9.18.1	Detailed Description	112
9.18.2	Member Data Documentation	113
9.18.2.1	kVALUE	113
9.19	nvinfer1::impl::EnumMaxImpl< InterpolationMode > Struct Reference	113
9.19.1	Detailed Description	113
9.19.2	Member Data Documentation	113
9.19.2.1	kVALUE	113
9.20	nvinfer1::impl::EnumMaxImpl< PaddingMode > Struct Reference	114
9.20.1	Detailed Description	114
9.20.2	Member Data Documentation	114
9.20.2.1	kVALUE	114
9.21	nvinfer1::impl::EnumMaxImpl< PoolingType > Struct Reference	114
9.21.1	Detailed Description	114

9.21.2	Member Data Documentation	115
9.21.2.1	kVALUE	115
9.22	nvinfer1::impl::EnumMaxImpl< PreviewFeature > Struct Reference	115
9.22.1	Detailed Description	115
9.22.2	Member Data Documentation	115
9.22.2.1	kVALUE	115
9.23	nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation > Struct Reference	116
9.23.1	Detailed Description	116
9.23.2	Member Data Documentation	116
9.23.2.1	kVALUE	116
9.24	nvinfer1::impl::EnumMaxImpl< ResizeRoundMode > Struct Reference	116
9.24.1	Detailed Description	117
9.24.2	Member Data Documentation	117
9.24.2.1	kVALUE	117
9.25	nvinfer1::impl::EnumMaxImpl< ResizeSelector > Struct Reference	117
9.25.1	Detailed Description	117
9.25.2	Member Data Documentation	117
9.25.2.1	kVALUE	118
9.26	nvinfer1::impl::EnumMaxImpl< TensorFormat > Struct Reference	118
9.26.1	Detailed Description	118
9.26.2	Member Data Documentation	118
9.26.2.1	kVALUE	118
9.27	nvinfer1::impl::EnumMaxImpl< TensorIOMode > Struct Reference	119
9.27.1	Detailed Description	119
9.27.2	Member Data Documentation	119
9.27.2.1	kVALUE	119
9.28	nvinfer1::impl::EnumMaxImpl< TensorLocation > Struct Reference	119
9.28.1	Detailed Description	120
9.28.2	Member Data Documentation	120
9.28.2.1	kVALUE	120
9.29	nvuffparser::FieldCollection Struct Reference	120
9.29.1	Member Data Documentation	120
9.29.1.1	fields	120
9.29.1.2	nbFields	121
9.30	nvuffparser::FieldMap Class Reference	121
9.30.1	Detailed Description	121
9.30.2	Constructor & Destructor Documentation	121
9.30.2.1	FieldMap()	121
9.30.3	Member Data Documentation	122
9.30.3.1	data	122
9.30.3.2	length	122
9.30.3.3	name	122
9.30.3.4	type	122
9.31	nvinfer1::safe::FloatingPointErrorInformation Struct Reference	122
9.31.1	Detailed Description	123
9.31.2	Member Data Documentation	123
9.31.2.1	nbInfErrors	123
9.31.2.2	nbNanErrors	123
9.32	nvinfer1::plugin::GridAnchorParameters Struct Reference	123
9.32.1	Detailed Description	123
9.32.2	Member Data Documentation	124
9.32.2.1	aspectRatios	124
9.32.2.2	H	124
9.32.2.3	maxSize	124

9.32.2.4	minSize	124
9.32.2.5	numAspectRatios	124
9.32.2.6	variance	125
9.32.2.7	W	125
9.33	nvinfer1::IActivationLayer Class Reference	125
9.33.1	Detailed Description	126
9.33.2	Constructor & Destructor Documentation	126
9.33.2.1	~IActivationLayer()	126
9.33.3	Member Function Documentation	126
9.33.3.1	getActivationType()	126
9.33.3.2	getAlpha()	127
9.33.3.3	getBeta()	127
9.33.3.4	setActivationType()	127
9.33.3.5	setAlpha()	128
9.33.3.6	setBeta()	128
9.33.4	Member Data Documentation	128
9.33.4.1	mImpl	128
9.34	nvinfer1::IAlgorithm Class Reference	129
9.34.1	Detailed Description	129
9.34.2	Constructor & Destructor Documentation	130
9.34.2.1	~IAlgorithm()	130
9.34.3	Member Function Documentation	130
9.34.3.1	getAlgorithmIOInfo()	130
9.34.3.2	getAlgorithmIOInfoByIndex()	130
9.34.3.3	getAlgorithmVariant()	131
9.34.3.4	getTimingMSec()	131
9.34.3.5	getWorkspaceSize()	131
9.34.4	Member Data Documentation	131
9.34.4.1	mImpl	131
9.35	nvinfer1::IAlgorithmContext Class Reference	132
9.35.1	Detailed Description	132
9.35.2	Constructor & Destructor Documentation	133
9.35.2.1	~IAlgorithmContext()	133
9.35.3	Member Function Documentation	133
9.35.3.1	getDimensions()	133
9.35.3.2	getName()	133
9.35.3.3	getNbInputs()	133
9.35.3.4	getNbOutputs()	134
9.35.4	Member Data Documentation	134
9.35.4.1	mImpl	134
9.36	nvinfer1::IAlgorithmIOInfo Class Reference	134
9.36.1	Detailed Description	135
9.36.2	Constructor & Destructor Documentation	135
9.36.2.1	~IAlgorithmIOInfo()	135
9.36.3	Member Function Documentation	135
9.36.3.1	getDataType()	135
9.36.3.2	getStrides()	136
9.36.3.3	getTensorFormat()	136
9.36.4	Member Data Documentation	136
9.36.4.1	mImpl	136
9.37	nvinfer1::IAlgorithmSelector Class Reference	136
9.37.1	Detailed Description	137
9.37.2	Constructor & Destructor Documentation	137
9.37.2.1	~IAlgorithmSelector()	137

9.37.3	Member Function Documentation	137
9.37.3.1	reportAlgorithms()	137
9.37.3.2	selectAlgorithms()	138
9.38	nvinfer1::IAlgorithmVariant Class Reference	138
9.38.1	Detailed Description	139
9.38.2	Constructor & Destructor Documentation	139
9.38.2.1	~IAlgorithmVariant()	139
9.38.3	Member Function Documentation	139
9.38.3.1	getImplementation()	140
9.38.3.2	getTactic()	140
9.38.4	Member Data Documentation	140
9.38.4.1	mImpl	140
9.39	nvinfer1::IAssertionLayer Class Reference	140
9.39.1	Detailed Description	141
9.39.2	Constructor & Destructor Documentation	141
9.39.2.1	~IAssertionLayer()	141
9.39.3	Member Function Documentation	141
9.39.3.1	getMessage()	142
9.39.3.2	setMessage()	142
9.39.4	Member Data Documentation	142
9.39.4.1	mImpl	142
9.40	nvcaffeparser1::IBinaryProtoBlob Class Reference	142
9.40.1	Detailed Description	143
9.40.2	Constructor & Destructor Documentation	143
9.40.2.1	~IBinaryProtoBlob()	143
9.40.3	Member Function Documentation	143
9.40.3.1	destroy()	143
9.40.3.2	getData()	144
9.40.3.3	getDataType()	144
9.40.3.4	getDimensions()	144
9.41	nvcaffeparser1::IBlobNameToTensor Class Reference	144
9.41.1	Detailed Description	144
9.41.2	Constructor & Destructor Documentation	145
9.41.2.1	~IBlobNameToTensor()	145
9.41.3	Member Function Documentation	145
9.41.3.1	find()	145
9.42	nvinfer1::IBuilder Class Reference	145
9.42.1	Detailed Description	147
9.42.2	Constructor & Destructor Documentation	147
9.42.2.1	~IBuilder()	147
9.42.3	Member Function Documentation	147
9.42.3.1	buildEngineWithConfig()	148
9.42.3.2	buildSerializedNetwork()	148
9.42.3.3	createBuilderConfig()	149
9.42.3.4	createNetworkV2()	149
9.42.3.5	createOptimizationProfile()	149
9.42.3.6	destroy()	150
9.42.3.7	getErrorRecorder()	150
9.42.3.8	getLogger()	150
9.42.3.9	getMaxBatchSize()	151
9.42.3.10	getMaxDLABatchSize()	151
9.42.3.11	getMaxThreads()	151
9.42.3.12	getNbDLACores()	152
9.42.3.13	getPluginRegistry()	152

9.42.3.14	isNetworkSupported()	152
9.42.3.15	platformHasFastFp16()	153
9.42.3.16	platformHasFastInt8()	153
9.42.3.17	platformHasTf32()	153
9.42.3.18	reset()	153
9.42.3.19	setErrorRecorder()	153
9.42.3.20	setGpuAllocator()	154
9.42.3.21	setMaxBatchSize()	154
9.42.3.22	setMaxThreads()	155
9.42.4	Member Data Documentation	155
9.42.4.1	mImpl	155
9.43	nvinfer1::IBuilderConfig Class Reference	155
9.43.1	Detailed Description	158
9.43.2	Constructor & Destructor Documentation	158
9.43.2.1	~IBuilderConfig()	159
9.43.3	Member Function Documentation	159
9.43.3.1	addOptimizationProfile()	159
9.43.3.2	canRunOnDLA()	159
9.43.3.3	clearFlag()	160
9.43.3.4	clearQuantizationFlag()	160
9.43.3.5	createTimingCache()	160
9.43.3.6	destroy()	161
9.43.3.7	getAlgorithmSelector()	161
9.43.3.8	getAvgTimingIterations()	161
9.43.3.9	getBuilderOptimizationLevel()	162
9.43.3.10	getCalibrationProfile()	162
9.43.3.11	getDefaultDeviceType()	162
9.43.3.12	getDeviceType()	162
9.43.3.13	getDLACore()	163
9.43.3.14	getEngineCapability()	163
9.43.3.15	getFlag()	163
9.43.3.16	getFlags()	164
9.43.3.17	getHardwareCompatibilityLevel()	164
9.43.3.18	getInt8Calibrator()	164
9.43.3.19	getMaxWorkspaceSize()	165
9.43.3.20	getMemoryPoolLimit()	165
9.43.3.21	getMinTimingIterations()	166
9.43.3.22	getNbOptimizationProfiles()	166
9.43.3.23	getNbPluginsToSerialize()	166
9.43.3.24	getPluginToSerialize()	167
9.43.3.25	getPreviewFeature()	167
9.43.3.26	getProfileStream()	167
9.43.3.27	getProfilingVerbosity()	168
9.43.3.28	getQuantizationFlag()	168
9.43.3.29	getQuantizationFlags()	168
9.43.3.30	getTacticSources()	169
9.43.3.31	getTimingCache()	169
9.43.3.32	isDeviceTypeSet()	169
9.43.3.33	reset()	170
9.43.3.34	resetDeviceType()	170
9.43.3.35	setAlgorithmSelector()	170
9.43.3.36	setAvgTimingIterations()	170
9.43.3.37	setBuilderOptimizationLevel()	171
9.43.3.38	setCalibrationProfile()	171

9.43.3.39	setDefaultDeviceType()	172
9.43.3.40	setDeviceType()	172
9.43.3.41	setDLACore()	172
9.43.3.42	setEngineCapability()	173
9.43.3.43	setFlag()	173
9.43.3.44	setFlags()	173
9.43.3.45	setHardwareCompatibilityLevel()	174
9.43.3.46	setInt8Calibrator()	174
9.43.3.47	setMaxWorkspaceSize()	174
9.43.3.48	setMemoryPoolLimit()	175
9.43.3.49	setMinTimingIterations()	176
9.43.3.50	setPluginsToSerialize()	176
9.43.3.51	setPreviewFeature()	176
9.43.3.52	setProfileStream()	177
9.43.3.53	setProfilingVerbosity()	177
9.43.3.54	setQuantizationFlag()	178
9.43.3.55	setQuantizationFlags()	178
9.43.3.56	setTacticSources()	178
9.43.3.57	setTimingCache()	179
9.43.4	Member Data Documentation	179
9.43.4.1	mImpl	179
9.44	nvcaffeparser1::ICaffeParser Class Reference	180
9.44.1	Detailed Description	180
9.44.2	Constructor & Destructor Documentation	180
9.44.2.1	~ICaffeParser()	181
9.44.3	Member Function Documentation	181
9.44.3.1	destroy()	181
9.44.3.2	getErrorRecorder()	181
9.44.3.3	parse()	181
9.44.3.4	parseBinaryProto()	182
9.44.3.5	parseBuffers()	183
9.44.3.6	setErrorRecorder()	183
9.44.3.7	setPluginFactoryV2()	184
9.44.3.8	setPluginNamespace()	184
9.44.3.9	setProtobufBufferSize()	184
9.45	nvinfer1::ICastLayer Class Reference	185
9.45.1	Detailed Description	185
9.45.2	Constructor & Destructor Documentation	186
9.45.2.1	~ICastLayer()	186
9.45.3	Member Function Documentation	186
9.45.3.1	getToType()	186
9.45.3.2	setToType()	186
9.45.4	Member Data Documentation	186
9.45.4.1	mImpl	186
9.46	nvinfer1::IConcatenationLayer Class Reference	187
9.46.1	Detailed Description	187
9.46.2	Constructor & Destructor Documentation	187
9.46.2.1	~IConcatenationLayer()	188
9.46.3	Member Function Documentation	188
9.46.3.1	getAxis()	188
9.46.3.2	setAxis()	188
9.46.4	Member Data Documentation	188
9.46.4.1	mImpl	189
9.47	nvinfer1::IConditionLayer Class Reference	189

9.47.1	Detailed Description	189
9.47.2	Constructor & Destructor Documentation	189
9.47.2.1	~IConditionLayer()	190
9.47.3	Member Data Documentation	190
9.47.3.1	mImpl	190
9.48	nvinfer1::consistency::IConsistencyChecker Class Reference	190
9.48.1	Detailed Description	190
9.48.2	Constructor & Destructor Documentation	191
9.48.2.1	~IConsistencyChecker()	191
9.48.2.2	IConsistencyChecker() [1/3]	191
9.48.2.3	IConsistencyChecker() [2/3]	191
9.48.2.4	IConsistencyChecker() [3/3]	191
9.48.3	Member Function Documentation	191
9.48.3.1	operator=() [1/2]	191
9.48.3.2	operator=() [2/2]	192
9.48.3.3	validate()	192
9.48.4	Member Data Documentation	192
9.48.4.1	mImpl	192
9.49	nvinfer1::IConstantLayer Class Reference	192
9.49.1	Detailed Description	193
9.49.2	Constructor & Destructor Documentation	193
9.49.2.1	~IConstantLayer()	193
9.49.3	Member Function Documentation	193
9.49.3.1	getDimensions()	194
9.49.3.2	getWeights()	194
9.49.3.3	setDimensions()	194
9.49.3.4	setWeights()	195
9.49.4	Member Data Documentation	195
9.49.4.1	mImpl	195
9.50	nvinfer1::IConvolutionLayer Class Reference	195
9.50.1	Detailed Description	197
9.50.2	Constructor & Destructor Documentation	197
9.50.2.1	~IConvolutionLayer()	198
9.50.3	Member Function Documentation	198
9.50.3.1	getBiasWeights()	198
9.50.3.2	getDilation()	198
9.50.3.3	getDilationNd()	198
9.50.3.4	getKernelSize()	199
9.50.3.5	getKernelSizeNd()	199
9.50.3.6	getKernelWeights()	199
9.50.3.7	getNbGroups()	199
9.50.3.8	getNbOutputMaps()	200
9.50.3.9	getPadding()	200
9.50.3.10	getPaddingMode()	200
9.50.3.11	getPaddingNd()	201
9.50.3.12	getPostPadding()	201
9.50.3.13	getPrePadding()	201
9.50.3.14	getStride()	201
9.50.3.15	getStrideNd()	202
9.50.3.16	setBiasWeights()	202
9.50.3.17	setDilation()	202
9.50.3.18	setDilationNd()	203
9.50.3.19	setInput()	203
9.50.3.20	setKernelSize()	203

9.50.3.21	setKernelSizeNd()	204
9.50.3.22	setKernelWeights()	204
9.50.3.23	setNbGroups()	205
9.50.3.24	setNbOutputMaps()	205
9.50.3.25	setPadding()	206
9.50.3.26	setPaddingMode()	206
9.50.3.27	setPaddingNd()	206
9.50.3.28	setPostPadding()	207
9.50.3.29	setPrePadding()	207
9.50.3.30	setStride()	207
9.50.3.31	setStrideNd()	208
9.50.4	Member Data Documentation	208
9.50.4.1	mImpl	208
9.51	nvinfer1::ICudaEngine Class Reference	208
9.51.1	Detailed Description	211
9.51.2	Constructor & Destructor Documentation	211
9.51.2.1	~ICudaEngine()	211
9.51.3	Member Function Documentation	211
9.51.3.1	bindingIsInput()	211
9.51.3.2	createEngineInspector()	212
9.51.3.3	createExecutionContext()	212
9.51.3.4	createExecutionContextWithoutDeviceMemory()	213
9.51.3.5	destroy()	213
9.51.3.6	getBindingBytesPerComponent()	213
9.51.3.7	getBindingComponentsPerElement()	214
9.51.3.8	getBindingDataType()	214
9.51.3.9	getBindingDimensions()	215
9.51.3.10	getBindingFormat()	215
9.51.3.11	getBindingFormatDesc()	216
9.51.3.12	getBindingIndex()	216
9.51.3.13	getBindingName()	217
9.51.3.14	getBindingVectorizedDim()	218
9.51.3.15	getDeviceMemorySize()	218
9.51.3.16	getEngineCapability()	219
9.51.3.17	getErrorRecorder()	219
9.51.3.18	getHardwareCompatibilityLevel()	219
9.51.3.19	getIOTensorName()	219
9.51.3.20	getLocation()	220
9.51.3.21	getMaxBatchSize()	220
9.51.3.22	getName()	221
9.51.3.23	getNbBindings()	221
9.51.3.24	getNbIOTensors()	222
9.51.3.25	getNbLayers()	222
9.51.3.26	getNbOptimizationProfiles()	222
9.51.3.27	getProfileDimensions()	222
9.51.3.28	getProfileShape()	223
9.51.3.29	getProfileShapeValues()	224
9.51.3.30	getProfilingVerbosity()	225
9.51.3.31	getTacticSources()	225
9.51.3.32	getTensorBytesPerComponent()	225
9.51.3.33	getTensorComponentsPerElement()	226
9.51.3.34	getTensorDataType()	226
9.51.3.35	getTensorFormat()	227
9.51.3.36	getTensorFormatDesc()	227

9.51.3.37	getTensorIOMode()	228
9.51.3.38	getTensorLocation()	228
9.51.3.39	getTensorShape()	229
9.51.3.40	getTensorVectorizedDim()	229
9.51.3.41	hasImplicitBatchDimension()	230
9.51.3.42	isExecutionBinding()	230
9.51.3.43	isRefittable()	231
9.51.3.44	isShapeBinding()	231
9.51.3.45	isShapeInferenceIO()	232
9.51.3.46	serialize()	232
9.51.3.47	setErrorRecorder()	232
9.51.4	Member Data Documentation	233
9.51.4.1	mImpl	233
9.52	nvinfer1::safe::ICudaEngine Class Reference	233
9.52.1	Detailed Description	235
9.52.2	Constructor & Destructor Documentation	235
9.52.2.1	ICudaEngine() [1/3]	235
9.52.2.2	~ICudaEngine()	235
9.52.2.3	ICudaEngine() [2/3]	235
9.52.2.4	ICudaEngine() [3/3]	235
9.52.3	Member Function Documentation	235
9.52.3.1	bindingIsInput()	235
9.52.3.2	createExecutionContext()	236
9.52.3.3	createExecutionContextWithoutDeviceMemory()	237
9.52.3.4	getBindingBytesPerComponent()	237
9.52.3.5	getBindingComponentsPerElement()	238
9.52.3.6	getBindingDataType()	238
9.52.3.7	getBindingDimensions()	239
9.52.3.8	getBindingFormat()	239
9.52.3.9	getBindingIndex()	240
9.52.3.10	getBindingName()	241
9.52.3.11	getBindingVectorizedDim()	241
9.52.3.12	getDeviceMemorySize()	242
9.52.3.13	getErrorRecorder()	243
9.52.3.14	getIOTensorName()	243
9.52.3.15	getName()	244
9.52.3.16	getNbBindings()	245
9.52.3.17	getNbIOTensors()	245
9.52.3.18	getTensorBytesPerComponent()	245
9.52.3.19	getTensorComponentsPerElement()	246
9.52.3.20	getTensorDataType()	247
9.52.3.21	getTensorFormat()	248
9.52.3.22	getTensorIOMode()	248
9.52.3.23	getTensorShape()	249
9.52.3.24	getTensorVectorizedDim()	249
9.52.3.25	operator=() [1/2]	250
9.52.3.26	operator=() [2/2]	250
9.52.3.27	setErrorRecorder()	250
9.53	nvinfer1::IDeconvolutionLayer Class Reference	251
9.53.1	Detailed Description	253
9.53.2	Constructor & Destructor Documentation	253
9.53.2.1	~IDeconvolutionLayer()	253
9.53.3	Member Function Documentation	253
9.53.3.1	getBiasWeights()	253

9.53.3.2	getDilationNd()	254
9.53.3.3	getKernelSize()	254
9.53.3.4	getKernelSizeNd()	254
9.53.3.5	getKernelWeights()	254
9.53.3.6	getNbGroups()	255
9.53.3.7	getNbOutputMaps()	255
9.53.3.8	getPadding()	255
9.53.3.9	getPaddingMode()	256
9.53.3.10	getPaddingNd()	256
9.53.3.11	getPostPadding()	256
9.53.3.12	getPrePadding()	256
9.53.3.13	getStride()	257
9.53.3.14	getStrideNd()	257
9.53.3.15	setBiasWeights()	257
9.53.3.16	setDilationNd()	258
9.53.3.17	setInput()	258
9.53.3.18	setKernelSize()	258
9.53.3.19	setKernelSizeNd()	259
9.53.3.20	setKernelWeights()	259
9.53.3.21	setNbGroups()	260
9.53.3.22	setNbOutputMaps()	260
9.53.3.23	setPadding()	261
9.53.3.24	setPaddingMode()	261
9.53.3.25	setPaddingNd()	261
9.53.3.26	setPostPadding()	262
9.53.3.27	setPrePadding()	262
9.53.3.28	setStride()	262
9.53.3.29	setStrideNd()	263
9.53.4	Member Data Documentation	263
9.53.4.1	mImpl	263
9.54	nvinfer1::IDequantizeLayer Class Reference	263
9.54.1	Detailed Description	264
9.54.2	Constructor & Destructor Documentation	265
9.54.2.1	~IDequantizeLayer()	265
9.54.3	Member Function Documentation	265
9.54.3.1	getAxis()	265
9.54.3.2	setAxis()	265
9.54.4	Member Data Documentation	265
9.54.4.1	mImpl	266
9.55	nvinfer1::IDimensionExpr Class Reference	266
9.55.1	Detailed Description	266
9.55.2	Constructor & Destructor Documentation	267
9.55.2.1	~IDimensionExpr()	267
9.55.3	Member Function Documentation	267
9.55.3.1	getConstantValue()	267
9.55.3.2	isConstant()	267
9.55.4	Member Data Documentation	267
9.55.4.1	mImpl	267
9.56	nvinfer1::IEinsumLayer Class Reference	268
9.56.1	Detailed Description	268
9.56.2	Constructor & Destructor Documentation	269
9.56.2.1	~IEinsumLayer()	269
9.56.3	Member Function Documentation	269
9.56.3.1	getEquation()	269

9.56.3.2	setEquation()	269
9.56.4	Member Data Documentation	270
9.56.4.1	mImpl	270
9.57	nvinfer1::IElementWiseLayer Class Reference	270
9.57.1	Detailed Description	271
9.57.2	Constructor & Destructor Documentation	271
9.57.2.1	~IElementWiseLayer()	271
9.57.3	Member Function Documentation	271
9.57.3.1	getOperation()	271
9.57.3.2	setOperation()	272
9.57.4	Member Data Documentation	272
9.57.4.1	mImpl	272
9.58	nvinfer1::IEngineInspector Class Reference	272
9.58.1	Detailed Description	273
9.58.2	Constructor & Destructor Documentation	273
9.58.2.1	~IEngineInspector()	274
9.58.3	Member Function Documentation	274
9.58.3.1	getEngineInformation()	274
9.58.3.2	getErrorRecorder()	274
9.58.3.3	getExecutionContext()	275
9.58.3.4	getLayerInformation()	275
9.58.3.5	setErrorRecorder()	276
9.58.3.6	setExecutionContext()	276
9.58.4	Member Data Documentation	277
9.58.4.1	mImpl	277
9.59	nvinfer1::IErrorRecorder Class Reference	277
9.59.1	Detailed Description	278
9.59.2	Member Typedef Documentation	278
9.59.2.1	ErrorDesc	278
9.59.2.2	RefCount	278
9.59.3	Constructor & Destructor Documentation	278
9.59.3.1	IErrorRecorder()	279
9.59.3.2	~IErrorRecorder()	279
9.59.4	Member Function Documentation	279
9.59.4.1	clear()	279
9.59.4.2	decRefCount()	280
9.59.4.3	getErrorCode()	280
9.59.4.4	getErrorDesc()	281
9.59.4.5	getNbErrors()	282
9.59.4.6	hasOverflowed()	282
9.59.4.7	incRefCount()	283
9.59.4.8	reportError()	283
9.59.5	Member Data Documentation	284
9.59.5.1	kMAX_DESC_LENGTH	284
9.60	nvinfer1::IExecutionContext Class Reference	284
9.60.1	Detailed Description	287
9.60.2	Constructor & Destructor Documentation	287
9.60.2.1	~IExecutionContext()	287
9.60.3	Member Function Documentation	287
9.60.3.1	allInputDimensionsSpecified()	288
9.60.3.2	allInputShapesSpecified()	288
9.60.3.3	destroy()	288
9.60.3.4	enqueue()	289
9.60.3.5	enqueueV2()	290

9.60.3.6	enqueueV3()	290
9.60.3.7	execute()	291
9.60.3.8	executeV2()	292
9.60.3.9	getBindingDimensions()	292
9.60.3.10	getDebugSync()	293
9.60.3.11	getEngine()	293
9.60.3.12	getEnqueueEmitsProfile()	294
9.60.3.13	getErrorRecorder()	294
9.60.3.14	getInputConsumedEvent()	294
9.60.3.15	getMaxOutputSize()	294
9.60.3.16	getName()	295
9.60.3.17	getNvtxVerbosity()	295
9.60.3.18	getOptimizationProfile()	296
9.60.3.19	getOutputAllocator()	296
9.60.3.20	getOutputTensorAddress()	296
9.60.3.21	getPersistentCacheLimit()	297
9.60.3.22	getProfiler()	297
9.60.3.23	getShapeBinding()	297
9.60.3.24	getStrides()	298
9.60.3.25	getTemporaryStorageAllocator()	299
9.60.3.26	getTensorAddress()	299
9.60.3.27	getTensorShape()	299
9.60.3.28	getTensorStrides()	300
9.60.3.29	inferShapes()	301
9.60.3.30	reportToProfiler()	301
9.60.3.31	setBindingDimensions()	302
9.60.3.32	setDebugSync()	303
9.60.3.33	setDeviceMemory()	304
9.60.3.34	setEnqueueEmitsProfile()	304
9.60.3.35	setErrorRecorder()	304
9.60.3.36	setInputConsumedEvent()	305
9.60.3.37	setInputShape()	305
9.60.3.38	setInputShapeBinding()	306
9.60.3.39	setInputTensorAddress()	307
9.60.3.40	setName()	307
9.60.3.41	setNvtxVerbosity()	308
9.60.3.42	setOptimizationProfile()	308
9.60.3.43	setOptimizationProfileAsync()	309
9.60.3.44	setOutputAllocator()	310
9.60.3.45	setPersistentCacheLimit()	311
9.60.3.46	setProfiler()	311
9.60.3.47	setTemporaryStorageAllocator()	311
9.60.3.48	setTensorAddress()	312
9.60.4	Member Data Documentation	313
9.60.4.1	mImpl	313
9.61	nvinfer1::safe::IExecutionContext Class Reference	313
9.61.1	Detailed Description	314
9.61.2	Constructor & Destructor Documentation	314
9.61.2.1	IExecutionContext() [1/3]	315
9.61.2.2	~IExecutionContext()	315
9.61.2.3	IExecutionContext() [2/3]	315
9.61.2.4	IExecutionContext() [3/3]	315
9.61.3	Member Function Documentation	315
9.61.3.1	enqueueV2()	315

9.61.3.2	enqueueV3()	316
9.61.3.3	getEngine()	317
9.61.3.4	getErrorBuffer()	317
9.61.3.5	getErrorRecorder()	318
9.61.3.6	getInputConsumedEvent()	318
9.61.3.7	getInputTensorAddress()	318
9.61.3.8	getName()	319
9.61.3.9	getOutputTensorAddress()	319
9.61.3.10	getStrides()	320
9.61.3.11	getTensorStrides()	321
9.61.3.12	operator=() [1/2]	321
9.61.3.13	operator=() [2/2]	321
9.61.3.14	setDeviceMemory()	322
9.61.3.15	setErrorBuffer()	322
9.61.3.16	setErrorRecorder()	323
9.61.3.17	setInputConsumedEvent()	323
9.61.3.18	setInputTensorAddress()	324
9.61.3.19	setName()	325
9.61.3.20	setOutputTensorAddress()	325
9.62	nvinfer1::IExprBuilder Class Reference	326
9.62.1	Detailed Description	327
9.62.2	Constructor & Destructor Documentation	327
9.62.2.1	~IExprBuilder()	327
9.62.3	Member Function Documentation	327
9.62.3.1	constant()	327
9.62.3.2	operation()	328
9.62.4	Member Data Documentation	328
9.62.4.1	mImpl	328
9.63	nvinfer1::IFillLayer Class Reference	328
9.63.1	Detailed Description	329
9.63.2	Constructor & Destructor Documentation	330
9.63.2.1	~IFillLayer()	330
9.63.3	Member Function Documentation	330
9.63.3.1	getAlpha()	330
9.63.3.2	getBeta()	331
9.63.3.3	getDimensions()	331
9.63.3.4	getOperation()	331
9.63.3.5	setAlpha()	331
9.63.3.6	setBeta()	332
9.63.3.7	setDimensions()	332
9.63.3.8	setInput()	333
9.63.3.9	setOperation()	334
9.63.4	Member Data Documentation	334
9.63.4.1	mImpl	334
9.64	nvinfer1::IFullyConnectedLayer Class Reference	334
9.64.1	Detailed Description	335
9.64.2	Constructor & Destructor Documentation	336
9.64.2.1	~IFullyConnectedLayer()	336
9.64.3	Member Function Documentation	336
9.64.3.1	getBiasWeights()	336
9.64.3.2	getKernelWeights()	336
9.64.3.3	getNbOutputChannels()	337
9.64.3.4	setBiasWeights()	337
9.64.3.5	setInput()	337

9.64.3.6	setKernelWeights()	338
9.64.3.7	setNbOutputChannels()	338
9.64.4	Member Data Documentation	338
9.64.4.1	mImpl	339
9.65	nvinfer1::IGatherLayer Class Reference	339
9.65.1	Detailed Description	340
9.65.2	Constructor & Destructor Documentation	341
9.65.2.1	~IGatherLayer()	341
9.65.3	Member Function Documentation	342
9.65.3.1	getGatherAxis()	342
9.65.3.2	getMode()	342
9.65.3.3	getNbElementWiseDims()	342
9.65.3.4	setGatherAxis()	342
9.65.3.5	setMode()	343
9.65.3.6	setNbElementWiseDims()	343
9.65.4	Member Data Documentation	344
9.65.4.1	mImpl	344
9.66	nvinfer1::IGpuAllocator Class Reference	344
9.66.1	Detailed Description	344
9.66.2	Constructor & Destructor Documentation	344
9.66.2.1	~IGpuAllocator()	344
9.66.2.2	IGpuAllocator()	345
9.66.3	Member Function Documentation	345
9.66.3.1	allocate()	345
9.66.3.2	deallocate()	345
9.66.3.3	free()	346
9.66.3.4	realloc()	347
9.67	nvinfer1::IGridSampleLayer Class Reference	348
9.67.1	Detailed Description	349
9.67.2	Constructor & Destructor Documentation	349
9.67.2.1	~IGridSampleLayer()	349
9.67.3	Member Function Documentation	349
9.67.3.1	getAlignCorners()	349
9.67.3.2	getInterpolationMode()	350
9.67.3.3	getSampleMode()	350
9.67.3.4	setAlignCorners()	350
9.67.3.5	setInterpolationMode()	351
9.67.3.6	setSampleMode()	351
9.67.4	Member Data Documentation	351
9.67.4.1	mImpl	351
9.68	nvinfer1::IHostMemory Class Reference	352
9.68.1	Detailed Description	352
9.68.2	Constructor & Destructor Documentation	352
9.68.2.1	~IHostMemory()	353
9.68.3	Member Function Documentation	353
9.68.3.1	data()	353
9.68.3.2	destroy()	353
9.68.3.3	size()	353
9.68.3.4	type()	353
9.68.4	Member Data Documentation	354
9.68.4.1	mImpl	354
9.69	nvinfer1::IIdentityLayer Class Reference	354
9.69.1	Detailed Description	355
9.69.2	Constructor & Destructor Documentation	355

9.69.2.1	~IIdentityLayer()	355
9.69.3	Member Data Documentation	355
9.69.3.1	mImpl	355
9.70	nvinfer1::IIfConditional Class Reference	356
9.70.1	Detailed Description	356
9.70.2	Constructor & Destructor Documentation	357
9.70.2.1	~IIfConditional()	357
9.70.3	Member Function Documentation	357
9.70.3.1	addInput()	357
9.70.3.2	addOutput()	357
9.70.3.3	getName()	358
9.70.3.4	setCondition()	358
9.70.3.5	setName()	358
9.70.4	Member Data Documentation	359
9.70.4.1	mImpl	359
9.71	nvinfer1::IIfConditionalBoundaryLayer Class Reference	359
9.71.1	Detailed Description	360
9.71.2	Constructor & Destructor Documentation	360
9.71.2.1	~IIfConditionalBoundaryLayer()	360
9.71.3	Member Function Documentation	360
9.71.3.1	getConditional()	360
9.71.4	Member Data Documentation	360
9.71.4.1	mBoundary	360
9.72	nvinfer1::IIfConditionalInputLayer Class Reference	361
9.72.1	Detailed Description	361
9.72.2	Constructor & Destructor Documentation	361
9.72.2.1	~IIfConditionalInputLayer()	361
9.72.3	Member Data Documentation	361
9.72.3.1	mImpl	362
9.73	nvinfer1::IIfConditionalOutputLayer Class Reference	362
9.73.1	Detailed Description	362
9.73.2	Constructor & Destructor Documentation	362
9.73.2.1	~IIfConditionalOutputLayer()	363
9.73.3	Member Data Documentation	363
9.73.3.1	mImpl	363
9.74	nvinfer1::IInt8Calibrator Class Reference	363
9.74.1	Detailed Description	364
9.74.2	Constructor & Destructor Documentation	364
9.74.2.1	~IInt8Calibrator()	364
9.74.3	Member Function Documentation	364
9.74.3.1	getAlgorithm()	364
9.74.3.2	getBatch()	364
9.74.3.3	getBatchSize()	365
9.74.3.4	readCalibrationCache()	365
9.74.3.5	writeCalibrationCache()	366
9.75	nvinfer1::IInt8EntropyCalibrator Class Reference	366
9.75.1	Detailed Description	367
9.75.2	Constructor & Destructor Documentation	367
9.75.2.1	~IInt8EntropyCalibrator()	367
9.75.3	Member Function Documentation	367
9.75.3.1	getAlgorithm()	367
9.76	nvinfer1::IInt8EntropyCalibrator2 Class Reference	367
9.76.1	Detailed Description	368
9.76.2	Constructor & Destructor Documentation	368

9.76.2.1	~IInt8EntropyCalibrator2()	368
9.76.3	Member Function Documentation	368
9.76.3.1	getAlgorithm()	368
9.77	nvinfer1::IInt8LegacyCalibrator Class Reference	368
9.77.1	Detailed Description	369
9.77.2	Constructor & Destructor Documentation	369
9.77.2.1	~IInt8LegacyCalibrator()	369
9.77.3	Member Function Documentation	369
9.77.3.1	getAlgorithm()	369
9.77.3.2	getQuantile()	370
9.77.3.3	getRegressionCutoff()	370
9.77.3.4	readHistogramCache()	370
9.77.3.5	writeHistogramCache()	370
9.78	nvinfer1::IInt8MinMaxCalibrator Class Reference	371
9.78.1	Detailed Description	371
9.78.2	Constructor & Destructor Documentation	371
9.78.2.1	~IInt8MinMaxCalibrator()	371
9.78.3	Member Function Documentation	372
9.78.3.1	getAlgorithm()	372
9.79	nvinfer1::IIteratorLayer Class Reference	372
9.79.1	Constructor & Destructor Documentation	373
9.79.1.1	~IIteratorLayer()	373
9.79.2	Member Function Documentation	373
9.79.2.1	getAxis()	373
9.79.2.2	getReverse()	373
9.79.2.3	setAxis()	373
9.79.2.4	setReverse()	374
9.79.3	Member Data Documentation	374
9.79.3.1	mImpl	374
9.80	nvinfer1::ILayer Class Reference	374
9.80.1	Detailed Description	376
9.80.2	Constructor & Destructor Documentation	376
9.80.2.1	~ILayer()	376
9.80.3	Member Function Documentation	376
9.80.3.1	getInput()	376
9.80.3.2	getName()	377
9.80.3.3	getNbInputs()	377
9.80.3.4	getNbOutputs()	377
9.80.3.5	getOutput()	377
9.80.3.6	getOutputType()	377
9.80.3.7	getPrecision()	378
9.80.3.8	getType()	378
9.80.3.9	outputTypesIsSet()	378
9.80.3.10	precisionIsSet()	379
9.80.3.11	resetOutputType()	379
9.80.3.12	resetPrecision()	380
9.80.3.13	setInput()	380
9.80.3.14	setName()	380
9.80.3.15	setOutputType()	381
9.80.3.16	setPrecision()	382
9.80.4	Member Data Documentation	382
9.80.4.1	mLayer	382
9.81	nvinfer1::ILogger Class Reference	383
9.81.1	Detailed Description	383

9.81.2	Member Enumeration Documentation	383
9.81.2.1	Severity	383
9.81.3	Constructor & Destructor Documentation	384
9.81.3.1	ILogger()	384
9.81.3.2	~ILogger()	384
9.81.4	Member Function Documentation	384
9.81.4.1	log()	384
9.82	nvinfer1::ILoop Class Reference	385
9.82.1	Detailed Description	385
9.82.2	Constructor & Destructor Documentation	385
9.82.2.1	~ILoop()	386
9.82.3	Member Function Documentation	386
9.82.3.1	addIterator()	386
9.82.3.2	addLoopOutput()	386
9.82.3.3	addRecurrence()	386
9.82.3.4	addTripLimit()	387
9.82.3.5	getName()	387
9.82.3.6	setName()	387
9.82.4	Member Data Documentation	388
9.82.4.1	mImpl	388
9.83	nvinfer1::ILoopBoundaryLayer Class Reference	388
9.83.1	Constructor & Destructor Documentation	388
9.83.1.1	~ILoopBoundaryLayer()	389
9.83.2	Member Function Documentation	389
9.83.2.1	getLoop()	389
9.83.3	Member Data Documentation	389
9.83.3.1	mBoundary	389
9.84	nvinfer1::ILoopOutputLayer Class Reference	389
9.84.1	Detailed Description	390
9.84.2	Constructor & Destructor Documentation	390
9.84.2.1	~ILoopOutputLayer()	390
9.84.3	Member Function Documentation	391
9.84.3.1	getAxis()	391
9.84.3.2	getLoopOutput()	391
9.84.3.3	setAxis()	391
9.84.3.4	setInput()	391
9.84.4	Member Data Documentation	392
9.84.4.1	mImpl	392
9.85	nvinfer1::ILRNLayer Class Reference	392
9.85.1	Detailed Description	393
9.85.2	Constructor & Destructor Documentation	393
9.85.2.1	~ILRNLayer()	393
9.85.3	Member Function Documentation	394
9.85.3.1	getAlpha()	394
9.85.3.2	getBeta()	394
9.85.3.3	getK()	394
9.85.3.4	getWindowSize()	395
9.85.3.5	setAlpha()	395
9.85.3.6	setBeta()	395
9.85.3.7	setK()	396
9.85.3.8	setWindowSize()	396
9.85.4	Member Data Documentation	396
9.85.4.1	mImpl	396
9.86	nvinfer1::IMatrixMultiplyLayer Class Reference	397

9.86.1	Detailed Description	397
9.86.2	Constructor & Destructor Documentation	398
9.86.2.1	~IMatrixMultiplyLayer()	398
9.86.3	Member Function Documentation	398
9.86.3.1	getOperation()	398
9.86.3.2	setOperation()	398
9.86.4	Member Data Documentation	399
9.86.4.1	mImpl	399
9.87	nvinfer1::INetworkDefinition Class Reference	399
9.87.1	Detailed Description	403
9.87.2	Constructor & Destructor Documentation	403
9.87.2.1	~INetworkDefinition()	403
9.87.3	Member Function Documentation	404
9.87.3.1	addActivation()	404
9.87.3.2	addAssertion()	404
9.87.3.3	addCast()	405
9.87.3.4	addConcatenation()	405
9.87.3.5	addConstant()	406
9.87.3.6	addConvolution()	407
9.87.3.7	addConvolutionNd()	407
9.87.3.8	addDeconvolution()	408
9.87.3.9	addDeconvolutionNd()	409
9.87.3.10	addDequantize()	410
9.87.3.11	addEinsum()	410
9.87.3.12	addElementWise()	411
9.87.3.13	addFill()	411
9.87.3.14	addFullyConnected()	412
9.87.3.15	addGather()	413
9.87.3.16	addGatherV2()	413
9.87.3.17	addGridSample()	414
9.87.3.18	addIdentity()	414
9.87.3.19	addIfConditional()	415
9.87.3.20	addInput()	415
9.87.3.21	addLoop()	416
9.87.3.22	addLRN()	416
9.87.3.23	addMatrixMultiply()	417
9.87.3.24	addNMS()	418
9.87.3.25	addNonZero()	418
9.87.3.26	addNormalization()	419
9.87.3.27	addOneHot()	420
9.87.3.28	addPadding()	420
9.87.3.29	addPaddingNd()	421
9.87.3.30	addParametricReLU()	421
9.87.3.31	addPluginV2()	422
9.87.3.32	addPooling()	423
9.87.3.33	addPoolingNd()	423
9.87.3.34	addQuantize()	424
9.87.3.35	addRaggedSoftMax()	424
9.87.3.36	addReduce()	425
9.87.3.37	addResize()	426
9.87.3.38	addReverseSequence()	426
9.87.3.39	addRNNv2()	427
9.87.3.40	addScale()	428
9.87.3.41	addScaleNd()	429

9.87.3.42	addScatter()	430
9.87.3.43	addSelect()	431
9.87.3.44	addShape()	432
9.87.3.45	addShuffle()	433
9.87.3.46	addSlice()	433
9.87.3.47	addSoftMax()	434
9.87.3.48	addTopK()	435
9.87.3.49	addUnary()	435
9.87.3.50	destroy()	436
9.87.3.51	getErrorRecorder()	437
9.87.3.52	getInput()	437
9.87.3.53	getLayer()	437
9.87.3.54	getName()	438
9.87.3.55	getNbInputs()	438
9.87.3.56	getNbLayers()	439
9.87.3.57	getNbOutputs()	439
9.87.3.58	getOutput()	439
9.87.3.59	hasExplicitPrecision()	440
9.87.3.60	hasImplicitBatchDimension()	440
9.87.3.61	markOutput()	440
9.87.3.62	markOutputForShapes()	441
9.87.3.63	removeTensor()	441
9.87.3.64	setErrorRecorder()	442
9.87.3.65	setName()	442
9.87.3.66	setWeightsName()	443
9.87.3.67	unmarkOutput()	443
9.87.3.68	unmarkOutputForShapes()	444
9.87.4	Member Data Documentation	444
9.87.4.1	mImpl	444
9.88	nvinfer1::INMSLayer Class Reference	444
9.88.1	Detailed Description	445
9.88.2	Constructor & Destructor Documentation	446
9.88.2.1	~INMSLayer()	446
9.88.3	Member Function Documentation	446
9.88.3.1	getBoundingBoxFormat()	446
9.88.3.2	getTopKBoxLimit()	447
9.88.3.3	setBoundingBoxFormat()	447
9.88.3.4	setInput()	447
9.88.3.5	setTopKBoxLimit()	448
9.88.4	Member Data Documentation	448
9.88.4.1	mImpl	448
9.89	nvinfer1::INoCopy Class Reference	449
9.89.1	Detailed Description	449
9.89.2	Constructor & Destructor Documentation	450
9.89.2.1	INoCopy() [1/3]	450
9.89.2.2	~INoCopy()	450
9.89.2.3	INoCopy() [2/3]	450
9.89.2.4	INoCopy() [3/3]	450
9.89.3	Member Function Documentation	450
9.89.3.1	operator=() [1/2]	450
9.89.3.2	operator=() [2/2]	451
9.90	INonZero Class Reference	451
9.90.1	Detailed Description	451
9.91	nvinfer1::INonZeroLayer Class Reference	451

9.91.1	Constructor & Destructor Documentation	452
9.91.1.1	~INonZeroLayer()	452
9.91.2	Member Data Documentation	452
9.91.2.1	mImpl	452
9.92	nvinfer1::INormalizationLayer Class Reference	452
9.92.1	Detailed Description	453
9.92.2	Constructor & Destructor Documentation	453
9.92.2.1	~INormalizationLayer()	454
9.92.3	Member Function Documentation	454
9.92.3.1	getAxes()	454
9.92.3.2	getComputePrecision()	454
9.92.3.3	getEpsilon()	454
9.92.3.4	getNbGroups()	455
9.92.3.5	setAxes()	455
9.92.3.6	setComputePrecision()	455
9.92.3.7	setEpsilon()	456
9.92.3.8	setNbGroups()	456
9.92.4	Member Data Documentation	456
9.92.4.1	mImpl	456
9.93	nvinfer1::IOneHotLayer Class Reference	457
9.93.1	Detailed Description	457
9.93.2	Member Function Documentation	458
9.93.2.1	getAxis()	458
9.93.2.2	setAxis()	458
9.93.3	Member Data Documentation	458
9.93.3.1	mImpl	459
9.94	nvonnxparser::IONnxConfig Class Reference	459
9.94.1	Detailed Description	460
9.94.2	Member Typedef Documentation	460
9.94.2.1	Verbosity	460
9.94.3	Constructor & Destructor Documentation	460
9.94.3.1	~IONnxConfig()	460
9.94.4	Member Function Documentation	460
9.94.4.1	addVerbosity()	460
9.94.4.2	destroy()	460
9.94.4.3	getFullTextFileName()	461
9.94.4.4	getModelDtype()	461
9.94.4.5	getModelFileName()	462
9.94.4.6	getPrintLayerInfo()	462
9.94.4.7	getTextFileName()	462
9.94.4.8	getVerbosityLevel()	463
9.94.4.9	reduceVerbosity()	463
9.94.4.10	setFullTextFileName()	463
9.94.4.11	setModelDtype()	464
9.94.4.12	setModelFileName()	464
9.94.4.13	setPrintLayerInfo()	464
9.94.4.14	setTextFileName()	465
9.94.4.15	setVerbosityLevel()	465
9.95	nvinfer1::IOptimizationProfile Class Reference	466
9.95.1	Detailed Description	467
9.95.2	Constructor & Destructor Documentation	467
9.95.2.1	~IOptimizationProfile()	467
9.95.3	Member Function Documentation	467
9.95.3.1	getDimensions()	467

9.95.3.2	getExtraMemoryTarget()	468
9.95.3.3	getNbShapeValues()	468
9.95.3.4	getShapeValues()	468
9.95.3.5	isValid()	469
9.95.3.6	setDimensions()	469
9.95.3.7	setExtraMemoryTarget()	470
9.95.3.8	setShapeValues()	470
9.95.4	Member Data Documentation	471
9.95.4.1	mImpl	471
9.96	nvinfer1::IOutputAllocator Class Reference	472
9.96.1	Detailed Description	472
9.96.2	Constructor & Destructor Documentation	472
9.96.2.1	~IOutputAllocator()	472
9.96.3	Member Function Documentation	472
9.96.3.1	getInterfaceVersion()	473
9.96.3.2	notifyShape()	473
9.96.3.3	reallocateOutput()	473
9.97	nvinfer1::IPaddingLayer Class Reference	474
9.97.1	Detailed Description	475
9.97.2	Constructor & Destructor Documentation	475
9.97.2.1	~IPaddingLayer()	475
9.97.3	Member Function Documentation	475
9.97.3.1	getPostPadding()	475
9.97.3.2	getPostPaddingNd()	475
9.97.3.3	getPrePadding()	476
9.97.3.4	getPrePaddingNd()	476
9.97.3.5	setPostPadding()	477
9.97.3.6	setPostPaddingNd()	477
9.97.3.7	setPrePadding()	477
9.97.3.8	setPrePaddingNd()	478
9.97.4	Member Data Documentation	478
9.97.4.1	mImpl	478
9.98	nvinfer1::IParametricReLULayer Class Reference	478
9.98.1	Detailed Description	479
9.98.2	Constructor & Destructor Documentation	479
9.98.2.1	~IParametricReLULayer()	479
9.98.3	Member Data Documentation	479
9.98.3.1	mImpl	479
9.99	nvonnxparser::IParser Class Reference	480
9.99.1	Detailed Description	480
9.99.2	Constructor & Destructor Documentation	480
9.99.2.1	~IParser()	480
9.99.3	Member Function Documentation	481
9.99.3.1	clearErrors()	481
9.99.3.2	destroy()	481
9.99.3.3	getError()	481
9.99.3.4	getNbErrors()	482
9.99.3.5	parse()	482
9.99.3.6	parseFromFile()	482
9.99.3.7	parseWithWeightDescriptors()	483
9.99.3.8	supportsModel()	483
9.99.3.9	supportsOperator()	484
9.100	nvonnxparser::IParserError Class Reference	484
9.100.1	Detailed Description	485

9.100.2 Constructor & Destructor Documentation	485
9.100.2.1 ~IParserError()	485
9.100.3 Member Function Documentation	485
9.100.3.1 code()	485
9.100.3.2 desc()	485
9.100.3.3 file()	486
9.100.3.4 func()	486
9.100.3.5 line()	486
9.100.3.6 node()	486
9.101 nvinfer1::consistency::IPluginChecker Class Reference	486
9.101.1 Detailed Description	487
9.101.2 Constructor & Destructor Documentation	487
9.101.2.1 IPluginChecker() [1/3]	487
9.101.2.2 ~IPluginChecker()	487
9.101.2.3 IPluginChecker() [2/3]	488
9.101.2.4 IPluginChecker() [3/3]	488
9.101.3 Member Function Documentation	488
9.101.3.1 operator=() [1/2]	488
9.101.3.2 operator=() [2/2]	488
9.101.3.3 validate()	488
9.102 nvinfer1::IPluginCreator Class Reference	489
9.102.1 Detailed Description	490
9.102.2 Constructor & Destructor Documentation	490
9.102.2.1 IPluginCreator()	490
9.102.2.2 ~IPluginCreator()	490
9.102.3 Member Function Documentation	490
9.102.3.1 createPlugin()	490
9.102.3.2 deserializePlugin()	491
9.102.3.3 getFieldNames()	491
9.102.3.4 getPluginName()	491
9.102.3.5 getPluginNamespace()	492
9.102.3.6 getPluginVersion()	492
9.102.3.7 getTensorRTVersion()	493
9.102.3.8 setPluginNamespace()	493
9.103 nvcaffeparser1::IPluginFactoryV2 Class Reference	494
9.103.1 Detailed Description	494
9.103.2 Constructor & Destructor Documentation	494
9.103.2.1 ~IPluginFactoryV2()	494
9.103.3 Member Function Documentation	494
9.103.3.1 createPlugin()	494
9.103.3.2 isPluginV2()	495
9.104 nvinfer1::IPluginRegistry Class Reference	495
9.104.1 Detailed Description	496
9.104.2 Member Typedef Documentation	497
9.104.2.1 PluginLibraryHandle	497
9.104.3 Constructor & Destructor Documentation	497
9.104.3.1 ~IPluginRegistry()	497
9.104.4 Member Function Documentation	497
9.104.4.1 deregisterCreator()	497
9.104.4.2 deregisterLibrary()	497
9.104.4.3 deserializeLibrary()	498
9.104.4.4 getErrorRecorder()	498
9.104.4.5 getPluginCreator()	499
9.104.4.6 getPluginCreatorList()	499

9.104.4.7 isParentSearchEnabled()	499
9.104.4.8 loadLibrary()	500
9.104.4.9 registerCreator()	501
9.104.4.10 setErrorRecorder()	501
9.104.4.11 setParentSearchEnabled()	502
9.105 nvinfer1::IPluginV2 Class Reference	502
9.105.1 Detailed Description	503
9.105.2 Member Function Documentation	504
9.105.2.1 clone()	504
9.105.2.2 configureWithFormat()	504
9.105.2.3 destroy()	505
9.105.2.4 enqueue()	506
9.105.2.5 getNbOutputs()	506
9.105.2.6 getOutputDimensions()	507
9.105.2.7 getPluginNamespace()	507
9.105.2.8 getPluginType()	508
9.105.2.9 getPluginVersion()	508
9.105.2.10 getSerializationSize()	509
9.105.2.11 getTensorRTVersion()	509
9.105.2.12 getWorkspaceSize()	510
9.105.2.13 initialize()	510
9.105.2.14 serialize()	510
9.105.2.15 setPluginNamespace()	511
9.105.2.16 supportsFormat()	512
9.105.2.17 terminate()	512
9.106 nvinfer1::IPluginV2DynamicExt Class Reference	513
9.106.1 Detailed Description	514
9.106.2 Constructor & Destructor Documentation	514
9.106.2.1 ~IPluginV2DynamicExt()	514
9.106.3 Member Function Documentation	515
9.106.3.1 clone()	515
9.106.3.2 configurePlugin()	515
9.106.3.3 enqueue()	516
9.106.3.4 getOutputDimensions()	517
9.106.3.5 getTensorRTVersion()	517
9.106.3.6 getWorkspaceSize()	518
9.106.3.7 supportsFormatCombination()	518
9.106.4 Member Data Documentation	519
9.106.4.1 kFORMAT_COMBINATION_LIMIT	519
9.107 nvinfer1::IPluginV2Ext Class Reference	519
9.107.1 Detailed Description	520
9.107.2 Constructor & Destructor Documentation	520
9.107.2.1 IPluginV2Ext()	520
9.107.2.2 ~IPluginV2Ext()	521
9.107.3 Member Function Documentation	521
9.107.3.1 attachToContext()	521
9.107.3.2 canBroadcastInputAcrossBatch()	521
9.107.3.3 clone()	522
9.107.3.4 configurePlugin()	523
9.107.3.5 configureWithFormat()	524
9.107.3.6 detachFromContext()	524
9.107.3.7 getOutputDataType()	525
9.107.3.8 getTensorRTVersion()	525
9.107.3.9 isOutputBroadcastAcrossBatch()	525

9.108	nvinfer1::IPluginV2IOExt Class Reference	526
9.108.1	Detailed Description	527
9.108.2	Member Function Documentation	527
9.108.2.1	configurePlugin()	527
9.108.2.2	getTensorRTVersion()	528
9.108.2.3	supportsFormatCombination()	528
9.109	nvinfer1::IPluginV2Layer Class Reference	529
9.109.1	Detailed Description	530
9.109.2	Constructor & Destructor Documentation	530
9.109.2.1	~IPluginV2Layer()	530
9.109.3	Member Function Documentation	530
9.109.3.1	getPlugin()	530
9.109.4	Member Data Documentation	531
9.109.4.1	mImpl	531
9.110	nvinfer1::IPoolingLayer Class Reference	531
9.110.1	Detailed Description	533
9.110.2	Constructor & Destructor Documentation	533
9.110.2.1	~IPoolingLayer()	533
9.110.3	Member Function Documentation	533
9.110.3.1	getAverageCountExcludesPadding()	533
9.110.3.2	getBlendFactor()	533
9.110.3.3	getPadding()	534
9.110.3.4	getPaddingMode()	534
9.110.3.5	getPaddingNd()	534
9.110.3.6	getPoolingType()	535
9.110.3.7	getPostPadding()	535
9.110.3.8	getPrePadding()	535
9.110.3.9	getStride()	535
9.110.3.10	getStrideNd()	536
9.110.3.11	getWindowSize()	536
9.110.3.12	getWindowSizeNd()	536
9.110.3.13	setAverageCountExcludesPadding()	537
9.110.3.14	setBlendFactor()	537
9.110.3.15	setPadding()	537
9.110.3.16	setPaddingMode()	538
9.110.3.17	setPaddingNd()	538
9.110.3.18	setPoolingType()	538
9.110.3.19	setPostPadding()	539
9.110.3.20	setPrePadding()	539
9.110.3.21	setStride()	539
9.110.3.22	setStrideNd()	540
9.110.3.23	setWindowSize()	540
9.110.3.24	setWindowSizeNd()	540
9.110.4	Member Data Documentation	541
9.110.4.1	mImpl	541
9.111	nvinfer1::IProfiler Class Reference	541
9.111.1	Detailed Description	541
9.111.2	Constructor & Destructor Documentation	541
9.111.2.1	~IProfiler()	541
9.111.3	Member Function Documentation	542
9.111.3.1	reportLayerTime()	542
9.112	nvinfer1::IQuantizeLayer Class Reference	543
9.112.1	Detailed Description	544
9.112.2	Constructor & Destructor Documentation	544

9.112.2.1 ~IQuantizeLayer()	545
9.112.3 Member Function Documentation	545
9.112.3.1 getAxis()	545
9.112.3.2 setAxis()	545
9.112.4 Member Data Documentation	545
9.112.4.1 mImpl	545
9.113 nvinfer1::IRaggedSoftMaxLayer Class Reference	546
9.113.1 Detailed Description	546
9.113.2 Constructor & Destructor Documentation	546
9.113.2.1 ~IRaggedSoftMaxLayer()	547
9.113.3 Member Data Documentation	547
9.113.3.1 mImpl	547
9.114 nvinfer1::IRecurrenceLayer Class Reference	547
9.114.1 Constructor & Destructor Documentation	548
9.114.1.1 ~IRecurrenceLayer()	548
9.114.2 Member Function Documentation	548
9.114.2.1 setInput()	548
9.114.3 Member Data Documentation	548
9.114.3.1 mImpl	549
9.115 nvinfer1::IReduceLayer Class Reference	549
9.115.1 Detailed Description	549
9.115.2 Constructor & Destructor Documentation	550
9.115.2.1 ~IReduceLayer()	550
9.115.3 Member Function Documentation	550
9.115.3.1 getKeepDimensions()	550
9.115.3.2 getOperation()	550
9.115.3.3 getReduceAxes()	551
9.115.3.4 setKeepDimensions()	551
9.115.3.5 setOperation()	551
9.115.3.6 setReduceAxes()	551
9.115.4 Member Data Documentation	552
9.115.4.1 mImpl	552
9.116 nvinfer1::IRefitter Class Reference	552
9.116.1 Detailed Description	553
9.116.2 Constructor & Destructor Documentation	553
9.116.2.1 ~IRefitter()	553
9.116.3 Member Function Documentation	553
9.116.3.1 destroy()	554
9.116.3.2 getAll()	554
9.116.3.3 getAllWeights()	554
9.116.3.4 getDynamicRangeMax()	555
9.116.3.5 getDynamicRangeMin()	555
9.116.3.6 getErrorRecorder()	556
9.116.3.7 getLogger()	556
9.116.3.8 getMaxThreads()	556
9.116.3.9 getMissing()	556
9.116.3.10 getMissingWeights()	557
9.116.3.11 getTensorsWithDynamicRange()	557
9.116.3.12 refitCudaEngine()	558
9.116.3.13 setDynamicRange()	558
9.116.3.14 setErrorRecorder()	559
9.116.3.15 setMaxThreads()	559
9.116.3.16 setNamedWeights()	560
9.116.3.17 setWeights()	560

9.116.4 Member Data Documentation	561
9.116.4.1 mImpl	561
9.117 nvinfer1::IResizeLayer Class Reference	561
9.117.1 Detailed Description	563
9.117.2 Constructor & Destructor Documentation	563
9.117.2.1 ~IResizeLayer()	564
9.117.3 Member Function Documentation	564
9.117.3.1 getAlignCorners()	564
9.117.3.2 getCoordinateTransformation()	564
9.117.3.3 getCubicCoeff()	564
9.117.3.4 getExcludeOutside()	565
9.117.3.5 getNearestRounding()	565
9.117.3.6 getOutputDimensions()	565
9.117.3.7 getResizeMode()	565
9.117.3.8 getScales()	565
9.117.3.9 getSelectorForSinglePixel()	566
9.117.3.10 setAlignCorners()	566
9.117.3.11 setCoordinateTransformation()	567
9.117.3.12 setCubicCoeff()	567
9.117.3.13 setExcludeOutside()	567
9.117.3.14 setInput()	567
9.117.3.15 setNearestRounding()	568
9.117.3.16 setOutputDimensions()	568
9.117.3.17 setResizeMode()	569
9.117.3.18 setScales()	569
9.117.3.19 setSelectorForSinglePixel()	570
9.117.4 Member Data Documentation	570
9.117.4.1 mImpl	570
9.118 nvinfer1::IReverseSequenceLayer Class Reference	571
9.118.1 Detailed Description	571
9.118.2 Constructor & Destructor Documentation	572
9.118.2.1 ~IReverseSequenceLayer()	572
9.118.3 Member Function Documentation	572
9.118.3.1 getBatchAxis()	572
9.118.3.2 getSequenceAxis()	572
9.118.3.3 setBatchAxis()	573
9.118.3.4 setSequenceAxis()	573
9.118.4 Member Data Documentation	573
9.118.4.1 mImpl	573
9.119 nvinfer1::IRNNv2Layer Class Reference	574
9.119.1 Detailed Description	575
9.119.2 Constructor & Destructor Documentation	575
9.119.2.1 ~IRNNv2Layer()	575
9.119.3 Member Function Documentation	575
9.119.3.1 getBiasForGate()	576
9.119.3.2 getCellState()	576
9.119.3.3 getDataLength()	576
9.119.3.4 getDirection()	576
9.119.3.5 getHiddenSize()	577
9.119.3.6 getHiddenState()	577
9.119.3.7 getInputMode()	577
9.119.3.8 getLayerCount()	577
9.119.3.9 getMaxSeqLength()	577
9.119.3.10 getOperation()	578

9.119.3.1	lgetSequenceLengths()	578
9.119.3.12	getWeightsForGate()	578
9.119.3.13	setBiasForGate()	578
9.119.3.14	setCellState()	579
9.119.3.15	setDirection()	579
9.119.3.16	setHiddenState()	580
9.119.3.17	setInputMode()	580
9.119.3.18	setOperation()	580
9.119.3.19	setSequenceLengths()	581
9.119.3.20	setWeightsForGate()	581
9.119.4	Member Data Documentation	582
9.119.4.1	mImpl	582
9.120	nvinfer1::IRuntime Class Reference	582
9.120.1	Detailed Description	584
9.120.2	Constructor & Destructor Documentation	584
9.120.2.1	~IRuntime()	584
9.120.3	Member Function Documentation	584
9.120.3.1	deserializeCudaEngine() [1/2]	584
9.120.3.2	deserializeCudaEngine() [2/2]	585
9.120.3.3	destroy()	585
9.120.3.4	getDLACore()	586
9.120.3.5	getEngineHostCodeAllowed()	586
9.120.3.6	getErrorRecorder()	586
9.120.3.7	getLogger()	587
9.120.3.8	getMaxThreads()	587
9.120.3.9	getNbDLACores()	587
9.120.3.10	getPluginRegistry()	587
9.120.3.11	lgetTempfileControlFlags()	588
9.120.3.12	getTemporaryDirectory()	588
9.120.3.13	loadRuntime()	588
9.120.3.14	setDLACore()	589
9.120.3.15	setEngineHostCodeAllowed()	589
9.120.3.16	setErrorRecorder()	590
9.120.3.17	setGpuAllocator()	590
9.120.3.18	setMaxThreads()	590
9.120.3.19	setTempfileControlFlags()	591
9.120.3.20	setTemporaryDirectory()	591
9.120.4	Member Data Documentation	592
9.120.4.1	mImpl	592
9.121	nvinfer1::safe::IRuntime Class Reference	592
9.121.1	Detailed Description	593
9.121.2	Constructor & Destructor Documentation	593
9.121.2.1	IRuntime() [1/3]	593
9.121.2.2	~IRuntime()	594
9.121.2.3	IRuntime() [2/3]	594
9.121.2.4	IRuntime() [3/3]	594
9.121.3	Member Function Documentation	594
9.121.3.1	deserializeCudaEngine()	594
9.121.3.2	getErrorRecorder()	595
9.121.3.3	operator=() [1/2]	595
9.121.3.4	operator=() [2/2]	596
9.121.3.5	setErrorRecorder()	596
9.121.3.6	setGpuAllocator()	596
9.122	nvinfer1::IScaleLayer Class Reference	597

9.122.1 Detailed Description	598
9.122.2 Constructor & Destructor Documentation	598
9.122.2.1 ~IScaleLayer()	599
9.122.3 Member Function Documentation	599
9.122.3.1 getChannelAxis()	599
9.122.3.2 getMode()	599
9.122.3.3 getPower()	599
9.122.3.4 getScale()	600
9.122.3.5 getShift()	600
9.122.3.6 setChannelAxis()	600
9.122.3.7 setMode()	601
9.122.3.8 setPower()	601
9.122.3.9 setScale()	601
9.122.3.10setShift()	601
9.122.4 Member Data Documentation	602
9.122.4.1 mImpl	602
9.123nvinfer1::IScatterLayer Class Reference	602
9.123.1 Detailed Description	603
9.123.2 Constructor & Destructor Documentation	604
9.123.2.1 ~IScatterLayer()	604
9.123.3 Member Function Documentation	604
9.123.3.1 getAxis()	604
9.123.3.2 getMode()	604
9.123.3.3 setAxis()	604
9.123.3.4 setMode()	605
9.123.4 Member Data Documentation	605
9.123.4.1 mImpl	605
9.124nvinfer1::ISelectLayer Class Reference	605
9.124.1 Detailed Description	605
9.124.2 Constructor & Destructor Documentation	606
9.124.2.1 ~ISelectLayer()	606
9.124.3 Member Data Documentation	606
9.124.3.1 mImpl	606
9.125nvinfer1::IShapeLayer Class Reference	606
9.125.1 Detailed Description	607
9.125.2 Constructor & Destructor Documentation	607
9.125.2.1 ~IShapeLayer()	607
9.125.3 Member Data Documentation	607
9.125.3.1 mImpl	607
9.126nvinfer1::IShuffleLayer Class Reference	608
9.126.1 Detailed Description	609
9.126.2 Constructor & Destructor Documentation	609
9.126.2.1 ~IShuffleLayer()	609
9.126.3 Member Function Documentation	609
9.126.3.1 getFirstTranspose()	609
9.126.3.2 getReshapeDimensions()	610
9.126.3.3 getSecondTranspose()	610
9.126.3.4 getZeroIsPlaceholder()	610
9.126.3.5 setFirstTranspose()	610
9.126.3.6 setInput()	611
9.126.3.7 setReshapeDimensions()	612
9.126.3.8 setSecondTranspose()	612
9.126.3.9 setZeroIsPlaceholder()	613
9.126.4 Member Data Documentation	613

9.126.4.1 mImpl	613
9.127nvinfer1::ISliceLayer Class Reference	613
9.127.1 Detailed Description	614
9.127.2 Constructor & Destructor Documentation	615
9.127.2.1 ~ISliceLayer()	615
9.127.3 Member Function Documentation	615
9.127.3.1 getMode()	615
9.127.3.2 getSize()	616
9.127.3.3 getStart()	616
9.127.3.4 getStride()	616
9.127.3.5 setInput()	616
9.127.3.6 setMode()	617
9.127.3.7 setSize()	617
9.127.3.8 setStart()	618
9.127.3.9 setStride()	618
9.127.4 Member Data Documentation	619
9.127.4.1 mImpl	619
9.128nvinfer1::ISoftMaxLayer Class Reference	619
9.128.1 Detailed Description	620
9.128.2 Constructor & Destructor Documentation	620
9.128.2.1 ~ISoftMaxLayer()	620
9.128.3 Member Function Documentation	620
9.128.3.1 getAxes()	620
9.128.3.2 setAxes()	621
9.128.4 Member Data Documentation	621
9.128.4.1 mImpl	621
9.129nvinfer1::ITensor Class Reference	621
9.129.1 Detailed Description	623
9.129.2 Constructor & Destructor Documentation	623
9.129.2.1 ~ITensor()	623
9.129.3 Member Function Documentation	624
9.129.3.1 dynamicRangeIsSet()	624
9.129.3.2 getAllowedFormats()	624
9.129.3.3 getBroadcastAcrossBatch()	624
9.129.3.4 getDimensionName()	624
9.129.3.5 getDimensions()	625
9.129.3.6 getDynamicRangeMax()	625
9.129.3.7 getDynamicRangeMin()	626
9.129.3.8 getLocation()	626
9.129.3.9 getName()	626
9.129.3.10getType()	627
9.129.3.11isExecutionTensor()	627
9.129.3.12isNetworkInput()	627
9.129.3.13isNetworkOutput()	628
9.129.3.14isShapeTensor()	628
9.129.3.15resetDynamicRange()	628
9.129.3.16setAllowedFormats()	629
9.129.3.17setBroadcastAcrossBatch()	629
9.129.3.18setDimensionName()	630
9.129.3.19setDimensions()	630
9.129.3.20setDynamicRange()	631
9.129.3.21setLocation()	631
9.129.3.22setName()	632
9.129.3.23setType()	632

9.129.4 Member Data Documentation	632
9.129.4.1 mImpl	633
9.130nvinfer1::ITimingCache Class Reference	633
9.130.1 Detailed Description	633
9.130.2 Constructor & Destructor Documentation	634
9.130.2.1 ~ITimingCache()	634
9.130.3 Member Function Documentation	634
9.130.3.1 combine()	634
9.130.3.2 reset()	635
9.130.3.3 serialize()	635
9.130.4 Member Data Documentation	635
9.130.4.1 mImpl	635
9.131nvinfer1::ITopKLayer Class Reference	636
9.131.1 Detailed Description	636
9.131.2 Constructor & Destructor Documentation	637
9.131.2.1 ~ITopKLayer()	637
9.131.3 Member Function Documentation	637
9.131.3.1 getK()	637
9.131.3.2 getOperation()	637
9.131.3.3 getReduceAxes()	638
9.131.3.4 setInput()	638
9.131.3.5 setK()	638
9.131.3.6 setOperation()	639
9.131.3.7 setReduceAxes()	639
9.131.4 Member Data Documentation	639
9.131.4.1 mImpl	639
9.132nvinfer1::ITripLimitLayer Class Reference	639
9.132.1 Constructor & Destructor Documentation	640
9.132.1.1 ~ITripLimitLayer()	640
9.132.2 Member Function Documentation	640
9.132.2.1 getTripLimit()	640
9.132.3 Member Data Documentation	640
9.132.3.1 mImpl	640
9.133nvuffparser::IUffParser Class Reference	641
9.133.1 Detailed Description	641
9.133.2 Constructor & Destructor Documentation	641
9.133.2.1 ~IUffParser()	642
9.133.3 Member Function Documentation	642
9.133.3.1 destroy()	642
9.133.3.2 getErrorRecorder()	642
9.133.3.3 getUffRequiredVersionMajor()	642
9.133.3.4 getUffRequiredVersionMinor()	643
9.133.3.5 getUffRequiredVersionPatch()	643
9.133.3.6 parse()	643
9.133.3.7 parseBuffer()	643
9.133.3.8 registerInput()	644
9.133.3.9 registerOutput()	644
9.133.3.10 setErrorRecorder()	644
9.133.3.11 setPluginNamespace()	645
9.134nvinfer1::IUnaryLayer Class Reference	645
9.134.1 Detailed Description	646
9.134.2 Constructor & Destructor Documentation	646
9.134.2.1 ~IUnaryLayer()	646
9.134.3 Member Function Documentation	646

9.134.3.1	getOperation()	646
9.134.3.2	setOperation()	647
9.134.4	Member Data Documentation	647
9.134.4.1	mImpl	647
9.135	nvinfer1::plugin::NMSParameters Struct Reference	647
9.135.1	Detailed Description	647
9.135.2	Member Data Documentation	648
9.135.2.1	backgroundLabelId	648
9.135.2.2	iouThreshold	648
9.135.2.3	isNormalized	648
9.135.2.4	keepTopK	648
9.135.2.5	numClasses	649
9.135.2.6	scoreThreshold	649
9.135.2.7	shareLocation	649
9.135.2.8	topK	649
9.136	nvinfer1::Permutation Struct Reference	649
9.136.1	Member Data Documentation	649
9.136.1.1	order	650
9.137	nvinfer1::PluginField Class Reference	650
9.137.1	Detailed Description	650
9.137.2	Constructor & Destructor Documentation	650
9.137.2.1	PluginField()	651
9.137.3	Member Data Documentation	651
9.137.3.1	data	651
9.137.3.2	length	651
9.137.3.3	name	651
9.137.3.4	type	651
9.138	nvinfer1::PluginFieldCollection Struct Reference	652
9.138.1	Detailed Description	652
9.138.2	Member Data Documentation	652
9.138.2.1	fields	652
9.138.2.2	nbFields	652
9.139	nvinfer1::PluginRegistrar< T > Class Template Reference	653
9.139.1	Detailed Description	653
9.139.2	Constructor & Destructor Documentation	653
9.139.2.1	PluginRegistrar()	653
9.140	nvinfer1::safe::PluginRegistrar< T > Class Template Reference	653
9.140.1	Detailed Description	654
9.140.2	Constructor & Destructor Documentation	654
9.140.2.1	PluginRegistrar()	654
9.141	nvinfer1::PluginTensorDesc Struct Reference	654
9.141.1	Detailed Description	655
9.141.2	Member Data Documentation	655
9.141.2.1	dims	655
9.141.2.2	format	655
9.141.2.3	scale	655
9.141.2.4	type	655
9.142	PluginVersion Struct Reference	656
9.142.1	Detailed Description	656
9.143	nvinfer1::plugin::PriorBoxParameters Struct Reference	656
9.143.1	Detailed Description	656
9.143.2	Member Data Documentation	657
9.143.2.1	aspectRatios	657
9.143.2.2	clip	657

9.143.2.3 flip	657
9.143.2.4 imgH	658
9.143.2.5 imgW	658
9.143.2.6 maxSize	658
9.143.2.7 minSize	658
9.143.2.8 numAspectRatios	658
9.143.2.9 numMaxSize	658
9.143.2.10 numMinSize	658
9.143.2.11 offset	659
9.143.2.12 stepH	659
9.143.2.13 stepW	659
9.143.2.14 variance	659
9.144 nvinfer1::plugin::Quadruple Struct Reference	659
9.144.1 Detailed Description	659
9.144.2 Member Data Documentation	660
9.144.2.1 data	660
9.145 nvinfer1::plugin::RegionParameters Struct Reference	660
9.145.1 Detailed Description	660
9.145.2 Member Data Documentation	661
9.145.2.1 classes	661
9.145.2.2 coords	661
9.145.2.3 num	661
9.145.2.4 smTree	661
9.146 nvinfer1::plugin::RPROIParams Struct Reference	661
9.146.1 Detailed Description	661
9.146.2 Member Data Documentation	662
9.146.2.1 anchorsRatioCount	662
9.146.2.2 anchorsScaleCount	662
9.146.2.3 featureStride	662
9.146.2.4 iouThreshold	662
9.146.2.5 minBoxSize	663
9.146.2.6 nmsMaxOut	663
9.146.2.7 poolingH	663
9.146.2.8 poolingW	663
9.146.2.9 preNmsTop	663
9.146.2.10 spatialScale	663
9.147 nvinfer1::plugin::softmaxTree Struct Reference	663
9.147.1 Detailed Description	664
9.147.2 Member Data Documentation	664
9.147.2.1 child	664
9.147.2.2 group	664
9.147.2.3 groupOffset	664
9.147.2.4 groups	664
9.147.2.5 groupSize	665
9.147.2.6 leaf	665
9.147.2.7 n	665
9.147.2.8 name	665
9.147.2.9 parent	665
9.148 nvinfer1::Weights Class Reference	665
9.148.1 Detailed Description	666
9.148.2 Member Data Documentation	666
9.148.2.1 count	666
9.148.2.2 type	666
9.148.2.3 values	666

10 File Documentation	667
10.1 NvCaffeParser.h File Reference	667
10.1.1 Detailed Description	668
10.2 NvCaffeParser.h	668
10.3 NvInfer.h File Reference	669
10.3.1 Detailed Description	676
10.4 NvInfer.h	676
10.5 NvInferConsistency.h File Reference	721
10.5.1 Function Documentation	722
10.5.1.1 createConsistencyChecker_INTERNAL()	722
10.6 NvInferConsistency.h	722
10.7 NvInferLegacyDims.h File Reference	723
10.7.1 Detailed Description	724
10.8 NvInferLegacyDims.h	724
10.9 NvInferPlugin.h File Reference	725
10.9.1 Detailed Description	727
10.9.2 Function Documentation	727
10.9.2.1 createAnchorGeneratorPlugin()	727
10.9.2.2 createBatchedNMSPlugin()	727
10.9.2.3 createInstanceNormalizationPlugin()	728
10.9.2.4 createNMSPlugin()	728
10.9.2.5 createNormalizePlugin()	728
10.9.2.6 createPriorBoxPlugin()	729
10.9.2.7 createRegionPlugin()	729
10.9.2.8 createReorgPlugin()	729
10.9.2.9 createRPNROIPlugin()	730
10.9.2.10 createSplitPlugin()	731
10.9.2.11 initLibNvInferPlugins()	731
10.10 NvInferPlugin.h	731
10.11 NvInferPluginUtils.h File Reference	732
10.11.1 Detailed Description	733
10.12 NvInferPluginUtils.h	733
10.13 NvInferRuntime.h File Reference	735
10.13.1 Detailed Description	738
10.13.2 Macro Definition Documentation	738
10.13.2.1 REGISTER_TENSORRT_PLUGIN	738
10.13.3 Function Documentation	738
10.13.3.1 getLogger()	738
10.13.3.2 getPluginRegistry()	738
10.14 NvInferRuntime.h	739
10.15 NvInferRuntimeCommon.h File Reference	754
10.15.1 Detailed Description	756
10.15.2 Macro Definition Documentation	756
10.15.2.1 NV_TENSORRT_VERSION	757
10.15.2.2 TENSORRTAPI	757
10.15.2.3 TRT_DEPRECATED	757
10.15.2.4 TRT_DEPRECATED_API	757
10.15.2.5 TRT_DEPRECATED_ENUM	757
10.15.2.6 TRTNOEXCEPT	757
10.15.3 Function Documentation	757
10.15.3.1 getInferLibVersion()	758
10.16 NvInferRuntimeCommon.h	758
10.17 NvInferSafeRuntime.h File Reference	766
10.17.1 Detailed Description	767

10.17.2 Macro Definition Documentation	767
10.17.2.1 REGISTER_SAFE_TENSORRT_PLUGIN	767
10.18NvInferSafeRuntime.h	767
10.19NvInferVersion.h File Reference	770
10.19.1 Detailed Description	770
10.19.2 Macro Definition Documentation	770
10.19.2.1 NV_TENSORRT_BUILD	770
10.19.2.2 NV_TENSORRT_LWS_MAJOR	771
10.19.2.3 NV_TENSORRT_LWS_MINOR	771
10.19.2.4 NV_TENSORRT_LWS_PATCH	771
10.19.2.5 NV_TENSORRT_MAJOR	771
10.19.2.6 NV_TENSORRT_MINOR	771
10.19.2.7 NV_TENSORRT_PATCH	771
10.19.2.8 NV_TENSORRT_SONAME_MAJOR	772
10.19.2.9 NV_TENSORRT_SONAME_MINOR	772
10.19.2.10 NV_TENSORRT_SONAME_PATCH	772
10.20NvInferVersion.h	772
10.21NvOnnxConfig.h File Reference	773
10.21.1 Detailed Description	773
10.22NvOnnxConfig.h	773
10.23NvUffParser.h File Reference	774
10.23.1 Detailed Description	775
10.23.2 Macro Definition Documentation	775
10.23.2.1 UFF_REQUIRED_VERSION_MAJOR	775
10.23.2.2 UFF_REQUIRED_VERSION_MINOR	775
10.23.2.3 UFF_REQUIRED_VERSION_PATCH	775
10.24NvUffParser.h	776
10.25NvUtils.h File Reference	777
10.25.1 Detailed Description	777
10.26NvUtils.h	778
10.27NvOnnxParser.h File Reference	778
10.27.1 Detailed Description	779
10.27.2 Macro Definition Documentation	779
10.27.2.1 NV_ONNX_PARSER_MAJOR	779
10.27.2.2 NV_ONNX_PARSER_MINOR	780
10.27.2.3 NV_ONNX_PARSER_PATCH	780
10.27.3 Typedef Documentation	780
10.27.3.1 SubGraph_t	780
10.27.3.2 SubGraphCollection_t	780
10.27.4 Function Documentation	780
10.27.4.1 createNvOnnxParser_INTERNAL()	780
10.27.4.2 getNvOnnxParserVersion()	781
10.28NvOnnxParser.h	781
Index	783

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:

NvInferRuntimeCommon.h	754
NvInferSafeRuntime.h	766
NvInferVersion.h	770

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 DRIVE OS 6.0 Safety Manual*. The safety runtime makes use of CUDA® and is designed to work with applications using CUDA.

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.

Chapter 3

Deprecated List

Member `createAnchorGeneratorPlugin` (`nvinfer1::plugin::GridAnchorParameters` *param, `int32_t` num←
Layers)

Deprecated in TensorRT 8.5. Use `GridAnchorPluginCreator::createPlugin()` to create an instance of "GridAnchor←
_TRT" version 1 plugin.

Member `createBatchedNMSPlugin` (`nvinfer1::plugin::NMSPParameters` param)

Deprecated in TensorRT 8.5. Use `BatchedNMSPluginCreator::createPlugin()` to create an instance of "Batched←
NMS-TRT" version 1 plugin.

Member `createInstanceNormalizationPlugin` (`float` epsilon, `nvinfer1::Weights` scale_weights, `nvinfer1::Weights`
bias_weights)

Deprecated in TensorRT 8.5. Use `InstanceNormalizationPluginCreator::createPlugin()` to create an instance of "←
InstanceNormalization-TRT" version 1 plugin.

Member `createNMSPlugin` (`nvinfer1::plugin::DetectionOutputParameters` param)

Deprecated in TensorRT 8.5. Use `NMSPluginCreator::createPlugin()` to create an instance of "NMS-TRT" version
1 plugin.

Member `createNormalizePlugin` (`nvinfer1::Weights` const *scales, `bool` acrossSpatial, `bool` channelShared, `float`
eps)

Deprecated in TensorRT 8.5. Use `NormalizePluginCreator::createPlugin()` to create an instance of "Normalize-←
TRT" version 1 plugin.

Member `createPriorBoxPlugin` (`nvinfer1::plugin::PriorBoxParameters` param)

Deprecated in TensorRT 8.5. Use `PriorBoxPluginCreator::createPlugin()` to create an instance of "PriorBox-TRT"
version 1 plugin.

Member `createRegionPlugin` (`nvinfer1::plugin::RegionParameters` params)

Deprecated in TensorRT 8.5. Use `RegionPluginCreator::createPlugin()` to create an instance of "Region-TRT"
version 1 plugin.

Member `createReorgPlugin` (`int32_t` stride)

Deprecated in TensorRT 8.5. Use `ReorgPluginCreator::createPlugin()` to create an instance of "Reorg-TRT" version
1 plugin.

Member `createRPNROIPlugin` (`int32_t` featureStride, `int32_t` preNmsTop, `int32_t` nmsMaxOut, `float` iou←
Threshold, `float` minBoxSize, `float` spatialScale, `nvinfer1::DimsHW` pooling, `nvinfer1::Weights` anchor←
Ratios, `nvinfer1::Weights` anchorScales)

Deprecated in TensorRT 8.5. Use `RPROIPluginCreator::createPlugin()` to create an instance of "RPROI-TRT"
version 1 plugin.

Member [createSplitPlugin](#) (int32_t axis, int32_t *output_lengths, int32_t noutput)

Deprecated in TensorRT 8.5 along with the "Split" plugin. Use [INetworkDefinition::addSlice\(\)](#) to add slice layer(s) as necessary to accomplish the required effect.

Member [nvcaffeparser1::createCaffeParser](#) () noexcept

[ICaffeParser](#) will be removed in TensorRT 9.0. Plan to migrate your workflow to use [nvonnxparser::IParser](#) for deployment.

Member [nvcaffeparser1::IBinaryProtoBlob::destroy](#) () noexcept=0

Deprecated in TensorRT 8.0. Superseded by `delete`.

Member [nvcaffeparser1::ICaffeParser::destroy](#) () noexcept=0

Deprecated in TensorRT 8.0. Superseded by `delete`.

Member [nvinfer1::IAlgorithm::getAlgorithmIOInfo](#) (int32_t index) const noexcept

Deprecated in TensorRT 8.0. Superseded by [IAlgorithm::getAlgorithmIOInfoByIndex\(\)](#).

Member [nvinfer1::IBuilder::buildEngineWithConfig](#) (INetworkDefinition &network, IBuilderConfig &config) noexcept

Deprecated in TensorRT 8.0. Superseded by [IBuilder::buildSerializedNetwork\(\)](#).

Member [nvinfer1::IBuilder::destroy](#) () noexcept

Deprecated in TensorRT 8.0. Superseded by `delete`.

Member [nvinfer1::IBuilder::getMaxBatchSize](#) () const noexcept

Deprecated in TensorRT 8.4.

Member [nvinfer1::IBuilder::setMaxBatchSize](#) (int32_t batchSize) noexcept

Deprecated in TensorRT 8.4.

Member [nvinfer1::IBuilderConfig::destroy](#) () noexcept

Deprecated in TensorRT 8.0. Superseded by `delete`.

Member [nvinfer1::IBuilderConfig::getMaxWorkspaceSize](#) () const noexcept

Deprecated in TensorRT 8.3. Superseded by [IBuilderConfig::getMemoryPoolLimit\(\)](#) with [MemoryPoolType::kWORKSPACE](#).

Member [nvinfer1::IBuilderConfig::getMinTimingIterations](#) () const noexcept

Deprecated in TensorRT 8.4. Superseded by [getAvgTimingIterations\(\)](#).

Member [nvinfer1::IBuilderConfig::setMaxWorkspaceSize](#) (std::size_t workspaceSize) noexcept

Deprecated in TensorRT 8.3. Superseded by [IBuilderConfig::setMemoryPoolLimit\(\)](#) with [MemoryPoolType::kWORKSPACE](#).

Member [nvinfer1::IBuilderConfig::setMinTimingIterations](#) (int32_t minTiming) noexcept

Deprecated in TensorRT 8.4. Superseded by [setAvgTimingIterations\(\)](#).

Member [nvinfer1::IConvolutionLayer::getDilation](#) () const noexcept

Superseded by [getDilationNd](#). Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member [nvinfer1::IConvolutionLayer::getKernelSize](#) () const noexcept

Superseded by [getKernelSizeNd](#). Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member [nvinfer1::IConvolutionLayer::getPadding](#) () const noexcept

Superseded by [getPaddingNd](#). Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member [nvinfer1::IConvolutionLayer::getStride](#) () const noexcept

Superseded by [getStrideNd](#). Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member [nvinfer1::IConvolutionLayer::setDilation](#) (DimsHW dilation) noexcept

Superseded by [setDilationNd](#). Deprecated prior to TensorRT 8.0 and will be removed in 9.0

-
- Member [nvinfer1::IConvolutionLayer::setKernelSize](#) (DimsHW kernelSize) noexcept**
Superseded by [setKernelSizeNd](#). Deprecated prior to TensorRT 8.0 and will be removed in 9.0
- Member [nvinfer1::IConvolutionLayer::setPadding](#) (DimsHW padding) noexcept**
Superseded by [setPaddingNd](#). Deprecated prior to TensorRT 8.0 and will be removed in 9.0
- Member [nvinfer1::IConvolutionLayer::setStride](#) (DimsHW stride) noexcept**
Superseded by [setStrideNd](#). Deprecated prior to TensorRT 8.0 and will be removed in 9.0
- Member [nvinfer1::ICudaEngine::bindingIsInput](#) (int32_t bindingIndex) const noexcept**
Deprecated in TensorRT 8.5. Superseded by [getTensorIOMode\(\)](#).
- Member [nvinfer1::ICudaEngine::destroy](#) () noexcept**
Deprecated in TRT 8.0. Superseded by `delete`.
- Member [nvinfer1::ICudaEngine::getBindingBytesPerComponent](#) (int32_t bindingIndex) const noexcept**
Deprecated in TensorRT 8.5. Superseded by [getTensorBytesPerComponent\(\)](#).
- Member [nvinfer1::ICudaEngine::getBindingComponentsPerElement](#) (int32_t bindingIndex) const noexcept**
Deprecated in TensorRT 8.5. Superseded by [getTensorComponentsPerElement\(\)](#).
- Member [nvinfer1::ICudaEngine::getBindingDataType](#) (int32_t bindingIndex) const noexcept**
Deprecated in TensorRT 8.5. Superseded by [getTensorDataType\(\)](#).
- Member [nvinfer1::ICudaEngine::getBindingDimensions](#) (int32_t bindingIndex) const noexcept**
Deprecated in TensorRT 8.5. Superseded by [getTensorShape\(\)](#).
- Member [nvinfer1::ICudaEngine::getBindingFormat](#) (int32_t bindingIndex) const noexcept**
Deprecated in TensorRT 8.5. Superseded by [getTensorFormat\(\)](#).
- Member [nvinfer1::ICudaEngine::getBindingFormatDesc](#) (int32_t bindingIndex) const noexcept**
Deprecated in TensorRT 8.5. Superseded by [getTensorFormatDesc\(\)](#).
- Member [nvinfer1::ICudaEngine::getBindingIndex](#) (char const *name) const noexcept**
Deprecated in TensorRT 8.5. Superseded by name-based methods. Use them instead of binding-index based methods.
- Member [nvinfer1::ICudaEngine::getBindingName](#) (int32_t bindingIndex) const noexcept**
Deprecated in TensorRT 8.5. Superseded by name-based methods. Use them instead of binding-index based methods.
- Member [nvinfer1::ICudaEngine::getBindingVectorizedDim](#) (int32_t bindingIndex) const noexcept**
Deprecated in TensorRT 8.5. Superseded by [getTensorVectorizedDim\(\)](#).
- Member [nvinfer1::ICudaEngine::getLocation](#) (int32_t bindingIndex) const noexcept**
Deprecated in TensorRT 8.5. Superseded by [getTensorLocation\(\)](#).
- Member [nvinfer1::ICudaEngine::getMaxBatchSize](#) () const noexcept**
Deprecated in TensorRT 8.4.
- Member [nvinfer1::ICudaEngine::getNbBindings](#) () const noexcept**
Deprecated in TensorRT 8.5. Superseded by [getNbIOTensors](#).
- Member [nvinfer1::ICudaEngine::getProfileDimensions](#) (int32_t bindingIndex, int32_t profileIndex, Opt↵ ProfileSelector select) const noexcept**
Deprecated in TensorRT 8.5. Superseded by [getProfileShape\(\)](#).
- Member [nvinfer1::ICudaEngine::getProfileShapeValues](#) (int32_t profileIndex, int32_t inputIndex, OptProfile↵ Selector select) const noexcept**
Deprecated in TensorRT 8.5. Superseded by [getShapeValues\(\)](#). Difference between Execution and shape tensor is superficial since TensorRT 8.5.
-

Member [nvinfer1::ICudaEngine::isExecutionBinding](#) (int32_t bindingIndex) const noexcept

No name-based equivalent replacement. Use [getTensorLocation\(\)](#) instead to know the location of tensor data. Distinction between execution binding and shape binding is superficial since TensorRT 8.5.

Member [nvinfer1::ICudaEngine::isShapeBinding](#) (int32_t bindingIndex) const noexcept

Use name-based [isShapeInferenceIO\(\)](#) instead to know whether a tensor is a shape tensor.

Member [nvinfer1::IDeconvolutionLayer::getKernelSize](#) () const noexcept

Superseded by [getKernelSizeNd](#). Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member [nvinfer1::IDeconvolutionLayer::getPadding](#) () const noexcept

Superseded by [getPaddingNd](#). Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member [nvinfer1::IDeconvolutionLayer::getStride](#) () const noexcept

Superseded by [getStrideNd](#). Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member [nvinfer1::IDeconvolutionLayer::setKernelSize](#) (DimsHW kernelSize) noexcept

Superseded by [setKernelSizeNd](#). Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member [nvinfer1::IDeconvolutionLayer::setPadding](#) (DimsHW padding) noexcept

Superseded by [setPaddingNd](#). Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member [nvinfer1::IDeconvolutionLayer::setStride](#) (DimsHW stride) noexcept

Superseded by [setStrideNd](#). Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member [nvinfer1::IExecutionContext::destroy](#) () noexcept

Deprecated in TRT 8.0. Superseded by `delete`.

Member [nvinfer1::IExecutionContext::enqueue](#) (int32_t batchSize, void *const *bindings, cudaStream_t stream, cudaEvent_t *inputConsumed) noexcept

Deprecated in TensorRT 8.4. Superseded by [enqueueV2\(\)](#) if the network is created with [NetworkDefinitionCreationFlag::kEXPLICIT](#) flag.

Member [nvinfer1::IExecutionContext::enqueueV2](#) (void *const *bindings, cudaStream_t stream, cudaEvent_t *inputConsumed) noexcept

Superseded by [enqueueV3\(\)](#). Deprecated in TensorRT 8.5

Member [nvinfer1::IExecutionContext::execute](#) (int32_t batchSize, void *const *bindings) noexcept

Deprecated in TensorRT 8.4. Superseded by [executeV2\(\)](#) if the network is created with [NetworkDefinitionCreationFlag::kEXPLICIT](#) flag.

Member [nvinfer1::IExecutionContext::getBindingDimensions](#) (int32_t bindingIndex) const noexcept

Deprecated in TensorRT 8.5. Superseded by [getTensorShape\(\)](#).

Member [nvinfer1::IExecutionContext::getShapeBinding](#) (int32_t bindingIndex, int32_t *data) const noexcept

Deprecated in TensorRT 8.5. Superseded by [getTensorAddress\(\)](#) or [getOutputTensorAddress\(\)](#).

Member [nvinfer1::IExecutionContext::getStrides](#) (int32_t bindingIndex) const noexcept

Deprecated in TensorRT 8.5. Superseded by [getTensorStrides\(\)](#).

Member [nvinfer1::IExecutionContext::setBindingDimensions](#) (int32_t bindingIndex, Dims dimensions) noexcept

Deprecated in TensorRT 8.5. Superseded by [setInputShape\(\)](#).

Member [nvinfer1::IExecutionContext::setInputShapeBinding](#) (int32_t bindingIndex, int32_t const *data) noexcept

Deprecated in TensorRT 8.5. Superseded by [setInputTensorAddress\(\)](#) or [setTensorAddress\(\)](#).

Member [nvinfer1::IExecutionContext::setOptimizationProfile](#) (int32_t profileIndex) noexcept

Superseded by [setOptimizationProfileAsync](#). Deprecated prior to TensorRT 8.0 and will be removed in 9.0.

Class `nvinfer1::IFullyConnectedLayer`

Deprecated in TensorRT 8.4. Superseded by [IMatrixMultiplyLayer](#).

Member `nvinfer1::IGpuAllocator::free (void *const memory) noexcept=0`

Deprecated in TensorRT 8.0. Superseded by `deallocate`.

Member `nvinfer1::IHostMemory::destroy () noexcept`

Deprecated in TRT 8.0. Superseded by `delete`.

Member `nvinfer1::INetworkDefinition::addConvolution (ITensor &input, int32_t nbOutputMaps, DimsHW kernelSize, Weights kernelWeights, Weights biasWeights) noexcept`

Superseded by `addConvolutionNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member `nvinfer1::INetworkDefinition::addDeconvolution (ITensor &input, int32_t nbOutputMaps, DimsHW kernelSize, Weights kernelWeights, Weights biasWeights) noexcept`

Superseded by `addDeconvolutionNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member `nvinfer1::INetworkDefinition::addFullyConnected (ITensor &input, int32_t nbOutputs, Weights kernelWeights, Weights biasWeights) noexcept`

Deprecated in TensorRT 8.4. Superseded by `addMatrixMultiply()`.

Member `nvinfer1::INetworkDefinition::addPadding (ITensor &input, DimsHW prePadding, DimsHW postPadding) noexcept`

Superseded by `addPaddingNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member `nvinfer1::INetworkDefinition::addPaddingNd (ITensor &input, Dims prePadding, Dims postPadding) noexcept`

Deprecated in TensorRT 8.0. Superseded by `addSlice()`.

Member `nvinfer1::INetworkDefinition::addPooling (ITensor &input, PoolingType type, DimsHW windowSize) noexcept`

Superseded by `addPoolingNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member `nvinfer1::INetworkDefinition::addRNNv2 (ITensor &input, int32_t layerCount, int32_t hiddenSize, int32_t maxSeqLen, RNNOperation op) noexcept`

Deprecated prior to TensorRT 8.0 and will be removed in 9.0. Superseded by `INetworkDefinition::addLoop()`.

Member `nvinfer1::INetworkDefinition::destroy () noexcept`

Deprecated in TensorRT 8.0. Superseded by `delete`.

Member `nvinfer1::INetworkDefinition::hasExplicitPrecision () const noexcept`

Deprecated in TensorRT 8.0.

Member `nvinfer1::IPaddingLayer::getPostPadding () const noexcept`

Superseded by `getPostPaddingNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member `nvinfer1::IPaddingLayer::getPrePadding () const noexcept`

Superseded by `getPrePaddingNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member `nvinfer1::IPaddingLayer::setPostPadding (DimsHW padding) noexcept`

Superseded by `setPostPaddingNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member `nvinfer1::IPaddingLayer::setPrePadding (DimsHW padding) noexcept`

Superseded by `setPrePaddingNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Class `nvinfer1::IPluginV2`

Deprecated in TensorRT 8.5. Implement [IPluginV2DynamicExt](#) or [IPluginV2IOExt](#) depending on your requirement.

Class [nvinfer1::IPluginV2Ext](#)

Deprecated in TensorRT 8.5. Implement [IPluginV2DynamicExt](#) or [IPluginV2IOExt](#) depending on your requirement.

Member [nvinfer1::IPoolingLayer::getPadding \(\)](#) const noexcept

Superseded by `getPaddingNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member [nvinfer1::IPoolingLayer::getStride \(\)](#) const noexcept

Superseded by `getStrideNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member [nvinfer1::IPoolingLayer::getWindowSize \(\)](#) const noexcept

Superseded by `getWindowSizeNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member [nvinfer1::IPoolingLayer::setPadding \(DimsHW padding\)](#) noexcept

Superseded by `setPaddingNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member [nvinfer1::IPoolingLayer::setStride \(DimsHW stride\)](#) noexcept

Superseded by `setStrideNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member [nvinfer1::IPoolingLayer::setWindowSize \(DimsHW windowSize\)](#) noexcept

Superseded by `setWindowSizeNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member [nvinfer1::IRefitter::destroy \(\)](#) noexcept

Deprecated in TRT 8.0. Superseded by `delete`.

Member [nvinfer1::IResizeLayer::getAlignCorners \(\)](#) const noexcept

Deprecated in TensorRT 8.0. Superseded by [IResizeLayer::getCoordinateTransformation\(\)](#).

Member [nvinfer1::IResizeLayer::setAlignCorners \(bool alignCorners\)](#) noexcept

Deprecated in TensorRT 8.0. Superseded by [IResizeLayer::setCoordinateTransformation\(\)](#).

Class [nvinfer1::IRNNv2Layer](#)

Deprecated prior to TensorRT 8.0 and will be removed in 9.0. Superseded by [INetworkDefinition::addLoop\(\)](#).

Member [nvinfer1::IRuntime::deserializeCudaEngine \(void const *blob, std::size_t size, IPluginFactory *pluginFactory\)](#) noexcept

Deprecated in TensorRT 8.0.

Member [nvinfer1::IRuntime::destroy \(\)](#) noexcept

Deprecated in TRT 8.0. Superseded by `delete`.

Member [nvinfer1::kDEFAULT](#)

Deprecated in TensorRT 8.0. Superseded by `kLAYER_NAMES_ONLY`.

Member [nvinfer1::kDEFAULT](#)

Deprecated in TensorRT 8.0. Superseded by `kSTANDARD`.

Member [nvinfer1::kSAFE_DLA](#)

Deprecated in TensorRT 8.0. Superseded by `kDLA_STANDALONE`.

Member [nvinfer1::kSAFE_GPU](#)

Deprecated in TensorRT 8.0. Superseded by `kSAFETY`.

Member [nvinfer1::kVERBOSE](#)

Deprecated in TensorRT 8.0. Superseded by `kDETAILED`.

Member [nvinfer1::kWRAP](#)

Use `kSTRICT_BOUNDS`.

Member [nvinfer1::ResizeMode](#)

Deprecated in TensorRT 8.5. Superseded by `InterpolationMode`.

Member **[nvinfer1::safe::ICudaEngine::bindingIsInput](#)** (std::int32_t const bindingIndex) const noexcept=0

Deprecated in TensorRT 8.5. Superseded by [tensorIOMode\(\)](#).

Member **[nvinfer1::safe::ICudaEngine::getBindingBytesPerComponent](#)** (std::int32_t const bindingIndex) const noexcept=0

Deprecated in TensorRT 8.5. Superseded by [getTensorBytesPerComponent\(\)](#).

Member **[nvinfer1::safe::ICudaEngine::getBindingComponentsPerElement](#)** (std::int32_t const bindingIndex) const noexcept=0

Deprecated in TensorRT 8.5. Superseded by [getTensorComponentsPerElement\(\)](#).

Member **[nvinfer1::safe::ICudaEngine::getBindingDataType](#)** (std::int32_t const bindingIndex) const noexcept=0

Deprecated in TensorRT 8.5. Superseded by [getTensorDataType\(\)](#).

Member **[nvinfer1::safe::ICudaEngine::getBindingDimensions](#)** (std::int32_t const bindingIndex) const noexcept=0

Deprecated in TensorRT 8.5. Superseded by [getTensorShape\(\)](#).

Member **[nvinfer1::safe::ICudaEngine::getBindingFormat](#)** (std::int32_t const bindingIndex) const noexcept=0

Deprecated in TensorRT 8.5. Superseded by [getTensorFormat\(\)](#).

Member **[nvinfer1::safe::ICudaEngine::getBindingIndex](#)** (AsciiChar const *const name) const noexcept=0

Deprecated in TensorRT 8.5. Superseded by name-based methods. Use them instead of binding-index based methods.

Member **[nvinfer1::safe::ICudaEngine::getBindingName](#)** (std::int32_t const bindingIndex) const noexcept=0

Deprecated in TensorRT 8.5. Superseded by name-based methods. Use them instead of binding-index based methods.

Member **[nvinfer1::safe::ICudaEngine::getBindingVectorizedDim](#)** (std::int32_t const bindingIndex) const noexcept=0

Deprecated in TensorRT 8.5. Superseded by [getTensorVectorizedDim\(\)](#).

Member **[nvinfer1::safe::ICudaEngine::getNbBindings](#)** () const noexcept=0

Deprecated in TensorRT 8.5. Superseded by [getNbIOTensors\(\)](#).

Member **[nvinfer1::safe::IExecutionContext::enqueueV2](#)** (void *const *const bindings, cudaStream_t const stream, cudaEvent_t const *const inputConsumed) noexcept=0

Deprecated in TensorRT 8.5. Superseded by [enqueueV3\(\)](#).

Member **[nvinfer1::safe::IExecutionContext::getStrides](#)** (std::int32_t const bindingIndex) const noexcept=0

Deprecated in TensorRT 8.5. Superseded by [getTensorStrides\(\)](#).

Member **[nvinfer1::SliceMode](#)**

Deprecated in TensorRT 8.5. Superseded by [SampleMode](#).

Member **[nvinfer1::utils::reorderSubBuffers](#)** (void *input, int32_t const *order, int32_t num, int32_t size) noexcept

Deprecated in TensorRT 8.0.

Member **[nvinfer1::utils::reshapeWeights](#)** (Weights const &input, int32_t const *shape, int32_t const *shape←Order, void *data, int32_t nbDims) noexcept

Deprecated in TensorRT 8.0.

Member **[nvinfer1::utils::transposeSubBuffers](#)** (void *input, DataType type, int32_t num, int32_t height, int32_t width) noexcept

Deprecated in TensorRT 8.0.

Member **[nvonnxparser::IOnnxConfig::destroy](#)** () noexcept=0

Use `delete` instead. Deprecated in TRT 8.0.

Member `nvuffparser::createUffParser () noexcept`

`IUffParser` will be removed in TensorRT 9.0. Plan to migrate your workflow to use `nvonnxparser::IParser` for deployment.

Member `nvuffparser::IUffParser::destroy () noexcept=0`

Use `delete` instead. Deprecated in TRT 8.0.

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

nvcaffeparser1	
The TensorRT Caffe parser API namespace	27
nvinfer1	
The TensorRT API version 1 namespace	28
nvinfer1::consistency	82
nvinfer1::impl	82
nvinfer1::plugin	83
nvinfer1::safe	
The safety subset of TensorRT's API version 1 namespace	84
nvinfer1::utils	85
nvonnxparser	
The TensorRT ONNX parser API namespace	88
nvuffparser	
The TensorRT UFF parser API namespace	90

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

<code>nvinfer1::plugin::DetectionOutputParameters</code>	93
<code>Dims</code>	96
<code>nvinfer1::Dims32</code>	99
<code>nvinfer1::Dims2</code>	97
<code>nvinfer1::Dims3</code>	98
<code>nvinfer1::Dims4</code>	101
<code>nvinfer1::DimsHW</code>	103
<code>nvinfer1::DimsExprs</code>	102
<code>nvinfer1::DynamicPluginTensorDesc</code>	105
<code>nvinfer1::impl::EnumMaxImpl< T ></code>	107
<code>nvinfer1::impl::EnumMaxImpl< ActivationType ></code>	107
<code>nvinfer1::impl::EnumMaxImpl< AllocatorFlag ></code>	108
<code>nvinfer1::impl::EnumMaxImpl< DataType ></code>	108
<code>nvinfer1::impl::EnumMaxImpl< ElementWiseOperation ></code>	109
<code>nvinfer1::impl::EnumMaxImpl< EngineCapability ></code>	110
<code>nvinfer1::impl::EnumMaxImpl< ErrorCode ></code>	111
<code>nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel ></code>	111
<code>nvinfer1::impl::EnumMaxImpl< ILogger::Severity ></code>	112
<code>nvinfer1::impl::EnumMaxImpl< InterpolationMode ></code>	113
<code>nvinfer1::impl::EnumMaxImpl< PaddingMode ></code>	114
<code>nvinfer1::impl::EnumMaxImpl< PoolingType ></code>	114
<code>nvinfer1::impl::EnumMaxImpl< PreviewFeature ></code>	115
<code>nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation ></code>	116
<code>nvinfer1::impl::EnumMaxImpl< ResizeRoundMode ></code>	116
<code>nvinfer1::impl::EnumMaxImpl< ResizeSelector ></code>	117
<code>nvinfer1::impl::EnumMaxImpl< TensorFormat ></code>	118
<code>nvinfer1::impl::EnumMaxImpl< TensorIOMode ></code>	119
<code>nvinfer1::impl::EnumMaxImpl< TensorLocation ></code>	119
<code>nvuffparser::FieldCollection</code>	120
<code>nvuffparser::FieldMap</code>	121
<code>nvinfer1::safe::FloatingPointErrorInformation</code>	122

nvinfer1::plugin::GridAnchorParameters	123
nvinfer1::IAlgorithmSelector	136
nvcaffeparser1::IBinaryProtoBlob	142
nvcaffeparser1::IBlobNameToTensor	144
nvcaffeparser1::ICaffeParser	180
nvinfer1::consistency::IConsistencyChecker	190
nvinfer1::safe::ICudaEngine	233
nvinfer1::IErrorRecorder	277
nvinfer1::safe::IExecutionContext	313
nvinfer1::IGpuAllocator	344
nvinfer1::IInt8Calibrator	363
nvinfer1::IInt8EntropyCalibrator	366
nvinfer1::IInt8EntropyCalibrator2	367
nvinfer1::IInt8LegacyCalibrator	368
nvinfer1::IInt8MinMaxCalibrator	371
nvinfer1::ILogger	383
nvinfer1::INoCopy	449
nvinfer1::IAlgorithm	129
nvinfer1::IAlgorithmContext	132
nvinfer1::IAlgorithmIOInfo	134
nvinfer1::IAlgorithmVariant	138
nvinfer1::IBuilder	145
nvinfer1::IBuilderConfig	155
nvinfer1::ICudaEngine	208
nvinfer1::IDimensionExpr	266
nvinfer1::IEngineInspector	272
nvinfer1::IExecutionContext	284
nvinfer1::IExprBuilder	326
nvinfer1::IHostMemory	352
nvinfer1::IIfConditional	356
nvinfer1::ILayer	374
nvinfer1::IActivationLayer	125
nvinfer1::IAssertionLayer	140
nvinfer1::ICastLayer	185
nvinfer1::IConcatenationLayer	187
nvinfer1::IConstantLayer	192
nvinfer1::IConvolutionLayer	195
nvinfer1::IDeconvolutionLayer	251
nvinfer1::IDequantizeLayer	263
nvinfer1::IEinsumLayer	268
nvinfer1::IElementWiseLayer	270
nvinfer1::IFillLayer	328
nvinfer1::IFullyConnectedLayer	334
nvinfer1::IGatherLayer	339
nvinfer1::IGridSampleLayer	348
nvinfer1::IIdentityLayer	354
nvinfer1::IIfConditionalBoundaryLayer	359
nvinfer1::IConditionLayer	189
nvinfer1::IIfConditionalInputLayer	361
nvinfer1::IIfConditionalOutputLayer	362
nvinfer1::ILRNLayer	392
nvinfer1::ILoopBoundaryLayer	388
nvinfer1::IIteratorLayer	372
nvinfer1::ILoopOutputLayer	389

nvinfer1::IRecurrenceLayer	547
nvinfer1::ITripLimitLayer	639
nvinfer1::IMatrixMultiplyLayer	397
nvinfer1::INMSLayer	444
nvinfer1::INonZeroLayer	451
nvinfer1::INormalizationLayer	452
nvinfer1::IOneHotLayer	457
nvinfer1::IPaddingLayer	474
nvinfer1::IParametricReLULayer	478
nvinfer1::IPluginV2Layer	529
nvinfer1::IPoolingLayer	531
nvinfer1::IQuantizeLayer	543
nvinfer1::IRNNv2Layer	574
nvinfer1::IRaggedSoftMaxLayer	546
nvinfer1::IReduceLayer	549
nvinfer1::IResizeLayer	561
nvinfer1::IReverseSequenceLayer	571
nvinfer1::IScaleLayer	597
nvinfer1::IScatterLayer	602
nvinfer1::ISelectLayer	605
nvinfer1::IShapeLayer	606
nvinfer1::IShuffleLayer	608
nvinfer1::ISliceLayer	613
nvinfer1::ISoftMaxLayer	619
nvinfer1::ITopKLayer	636
nvinfer1::UnaryLayer	645
nvinfer1::ILoop	385
nvinfer1::INetworkDefinition	399
nvinfer1::IOptimizationProfile	466
nvinfer1::IRefitter	552
nvinfer1::IRuntime	582
nvinfer1::ITensor	621
nvinfer1::ITimingCache	633
INonZero	451
nvonnxparser::IONnxConfig	459
nvinfer1::IOutputAllocator	472
nvonnxparser::IParser	480
nvonnxparser::IParserError	484
nvinfer1::IPluginCreator	489
nvinfer1::consistency::IPluginChecker	486
nvcaffeparser1::IPluginFactoryV2	494
nvinfer1::IPluginRegistry	495
nvinfer1::IPluginV2	502
nvinfer1::IPluginV2Ext	519
nvinfer1::IPluginV2DynamicExt	513
nvinfer1::IPluginV2IOExt	526
nvinfer1::IProfiler	541
nvinfer1::safe::IRuntime	592
nvuffparser::IUffParser	641
nvinfer1::plugin::NMSPParameters	647
nvinfer1::Permutation	649
nvinfer1::PluginField	650
nvinfer1::PluginFieldCollection	652

<code>nvinfer1::PluginRegistrar< T ></code>	653
<code>nvinfer1::safe::PluginRegistrar< T ></code>	653
<code>nvinfer1::PluginTensorDesc</code>	654
<code>PluginVersion</code>	656
<code>nvinfer1::plugin::PriorBoxParameters</code>	656
<code>nvinfer1::plugin::Quadruple</code>	659
<code>nvinfer1::plugin::RegionParameters</code>	660
<code>nvinfer1::plugin::RPROIParams</code>	661
<code>nvinfer1::plugin::softmaxTree</code>	663
<code>nvinfer1::Weights</code>	665

Chapter 6

Class Index

6.1 Class List

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

nvinfer1::plugin::DetectionOutputParameters	
The DetectionOutput plugin layer generates the detection output based on location and confidence predictions by doing non maximum suppression. This plugin first decodes the bounding boxes based on the anchors generated. It then performs non_max_suppression on the decoded bounding boxes. DetectionOutputParameters defines a set of parameters for creating the DetectionOutput plugin layer. It contains:	93
Dims	
Structure to define the dimensions of a tensor	96
nvinfer1::Dims2	
Descriptor for two-dimensional data	97
nvinfer1::Dims3	
Descriptor for three-dimensional data	98
nvinfer1::Dims32	99
nvinfer1::Dims4	
Descriptor for four-dimensional data	101
nvinfer1::DimsExprs	102
nvinfer1::DimsHW	
Descriptor for two-dimensional spatial data	103
nvinfer1::DynamicPluginTensorDesc	105
nvinfer1::impl::EnumMaxImpl< T >	
Declaration of EnumMaxImpl struct to store maximum number of elements in an enumeration type	107
nvinfer1::impl::EnumMaxImpl< ActivationType >	107
nvinfer1::impl::EnumMaxImpl< AllocatorFlag >	
Maximum number of elements in AllocatorFlag enum	108
nvinfer1::impl::EnumMaxImpl< DataType >	
Maximum number of elements in DataType enum	108
nvinfer1::impl::EnumMaxImpl< ElementWiseOperation >	109
nvinfer1::impl::EnumMaxImpl< EngineCapability >	
Maximum number of elements in EngineCapability enum	110
nvinfer1::impl::EnumMaxImpl< ErrorCode >	
Maximum number of elements in ErrorCode enum	111

nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel >	111
nvinfer1::impl::EnumMaxImpl< ILogger::Severity >	
Maximum number of elements in ILogger::Severity enum	112
nvinfer1::impl::EnumMaxImpl< InterpolationMode >	113
nvinfer1::impl::EnumMaxImpl< PaddingMode >	114
nvinfer1::impl::EnumMaxImpl< PoolingType >	114
nvinfer1::impl::EnumMaxImpl< PreviewFeature >	115
nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation >	116
nvinfer1::impl::EnumMaxImpl< ResizeRoundMode >	116
nvinfer1::impl::EnumMaxImpl< ResizeSelector >	117
nvinfer1::impl::EnumMaxImpl< TensorFormat >	
Maximum number of elements in TensorFormat enum	118
nvinfer1::impl::EnumMaxImpl< TensorIOMode >	
Maximum number of elements in TensorIOMode enum	119
nvinfer1::impl::EnumMaxImpl< TensorLocation >	
Maximum number of elements in TensorLocation enum	119
nvuffparser::FieldCollection	120
nvuffparser::FieldMap	
An array of field params used as a layer parameter for plugin layers	121
nvinfer1::safe::FloatingPointErrorInformation	
Space to record information about floating point runtime errors	122
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. It contains:	123
nvinfer1::IActivationLayer	
An Activation layer in a network definition	125
nvinfer1::IAlgorithm	
Describes a variation of execution of a layer. An algorithm is represented by IAlgorithmVariant and the IAlgorithmIOInfo for each of its inputs and outputs. An algorithm can be selected or reproduced using AlgorithmSelector::selectAlgorithms() .”	129
nvinfer1::IAlgorithmContext	
Describes the context and requirements, that could be fulfilled by one or more instances of IAlgorithm	132
nvinfer1::IAlgorithmIOInfo	
Carries information about input or output of the algorithm. IAlgorithmIOInfo for all the input and output along with IAlgorithmVariant denotes the variation of algorithm and can be used to select or reproduce an algorithm using IAlgorithmSelector::selectAlgorithms()	134
nvinfer1::IAlgorithmSelector	
Interface implemented by application for selecting and reporting algorithms of a layer provided by the builder	136
nvinfer1::IAlgorithmVariant	
Unique 128-bit identifier, which along with the input and output information denotes the variation of algorithm and can be used to select or reproduce an algorithm, using IAlgorithmSelector::selectAlgorithms()	138
nvinfer1::IAssertionLayer	
An assertion layer in a network	140
nvcaffeparser1::IBinaryProtoBlob	
Object used to store and query data extracted from a binaryproto file using the ICaffeParser	142
nvcaffeparser1::IBlobNameToTensor	
Object used to store and query Tensors after they have been extracted from a Caffe model using the ICaffeParser	144
nvinfer1::IBuilder	
Builds an engine from a network definition	145

nvinfer1::IBuilderConfig	155
Holds properties for configuring a builder to produce an engine	
nvcaffeparser1::ICaffeParser	180
Class used for parsing Caffe models	
nvinfer1::ICastLayer	185
A cast layer in a network	
nvinfer1::IConcatenationLayer	187
A concatenation layer in a network definition	
nvinfer1::IConditionLayer	189
nvinfer1::consistency::IConsistencyChecker	190
Validates a serialized engine blob	
nvinfer1::IConstantLayer	192
Layer that represents a constant value	
nvinfer1::IConvolutionLayer	195
A convolution layer in a network definition	
nvinfer1::ICudaEngine	208
An engine for executing inference on a built network, with functionally unsafe features	
nvinfer1::safe::ICudaEngine	233
A functionally safe engine for executing inference on a built network	
nvinfer1::IDeconvolutionLayer	251
A deconvolution layer in a network definition	
nvinfer1::IDequantizeLayer	263
A Dequantize layer in a network definition	
nvinfer1::IDimensionExpr	266
nvinfer1::IEinsumLayer	268
An Einsum layer in a network	
nvinfer1::IElementWiseLayer	270
A elementwise layer in a network definition	
nvinfer1::IEngineInspector	272
An engine inspector which prints out the layer information of an engine or an execution context	
nvinfer1::IErrorRecorder	277
Reference counted application-implemented error reporting interface for TensorRT objects	
nvinfer1::IExecutionContext	284
Context for executing inference using an engine, with functionally unsafe features	
nvinfer1::safe::IExecutionContext	313
Functionally safe context for executing inference using an engine	
nvinfer1::IExprBuilder	326
nvinfer1::IFillLayer	328
Generate an output tensor with specified mode	
nvinfer1::IFullyConnectedLayer	334
A fully connected layer in a network definition. This layer expects an input tensor of three or more non-batch dimensions. The input is automatically reshaped into an $M \times V$ tensor X , where V is a product of the last three dimensions and M is a product of the remaining dimensions (where the product over 0 dimensions is defined as 1). For example:	
nvinfer1::IGatherLayer	339
A Gather layer in a network definition. Supports several kinds of gathering	
nvinfer1::IGpuAllocator	344
Application-implemented class for controlling allocation on the GPU	
nvinfer1::IGridSampleLayer	348
A GridSample layer in a network definition	
nvinfer1::IHostMemory	352
Class to handle library allocated memory that is accessible to the user	
nvinfer1::IIdentityLayer	354
A layer that represents the identity function	

nvinfer1::IfConditional	356
nvinfer1::IfConditionalBoundaryLayer	359
nvinfer1::IfConditionalInputLayer	361
nvinfer1::IfConditionalOutputLayer	362
nvinfer1::Int8Calibrator	
Application-implemented interface for calibration	363
nvinfer1::Int8EntropyCalibrator	366
nvinfer1::Int8EntropyCalibrator2	367
nvinfer1::Int8LegacyCalibrator	368
nvinfer1::Int8MinMaxCalibrator	371
nvinfer1::IteratorLayer	372
nvinfer1::ILayer	
Base class for all layer classes in a network definition	374
nvinfer1::ILogger	
Application-implemented logging interface for the builder, refitter and runtime	383
nvinfer1::ILoop	385
nvinfer1::ILoopBoundaryLayer	388
nvinfer1::ILoopOutputLayer	389
nvinfer1::ILRNLayer	
A LRN layer in a network definition	392
nvinfer1::IMatrixMultiplyLayer	
Layer that represents a Matrix Multiplication	397
nvinfer1::INetworkDefinition	
A network definition for input to the builder	399
nvinfer1::INMSLayer	
A non-maximum suppression layer in a network definition	444
nvinfer1::INoCopy	
Forward declaration of IEngineInspector for use by other interfaces	449
INonZero	
A NonZero layer in a network	451
nvinfer1::INonZeroLayer	451
nvinfer1::INormalizationLayer	
A normalization layer in a network definition	452
nvinfer1::IOneHotLayer	
A OneHot layer in a network definition	457
nvonnxparser::IOnnxConfig	
Configuration Manager Class	459
nvinfer1::IOptimizationProfile	
Optimization profile for dynamic input dimensions and shape tensors	466
nvinfer1::IOutputAllocator	
Callback from ExecutionContext::enqueueV3()	472
nvinfer1::IPaddingLayer	
Layer that represents a padding operation	474
nvinfer1::IParametricReLULayer	
Layer that represents a parametric ReLU operation	478
nvonnxparser::IParser	
Object for parsing ONNX models into a TensorRT network definition	480
nvonnxparser::IParserError	
Object containing information about an error	484
nvinfer1::consistency::IPluginChecker	
Consistency Checker plugin class for user implemented Plugins	486
nvinfer1::IPluginCreator	
Plugin creator class for user implemented layers	489

nvcaffeparser1::IPluginFactoryV2	
Plugin factory used to configure plugins	494
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	495
nvinfer1::IPluginV2	
Plugin class for user-implemented layers	502
nvinfer1::IPluginV2DynamicExt	513
nvinfer1::IPluginV2Ext	
Plugin class for user-implemented layers	519
nvinfer1::IPluginV2IOExt	
Plugin class for user-implemented layers	526
nvinfer1::IPluginV2Layer	
Layer type for pluginV2	529
nvinfer1::IPoolingLayer	
A Pooling layer in a network definition	531
nvinfer1::IProfiler	
Application-implemented interface for profiling	541
nvinfer1::IQuantizeLayer	
A Quantize layer in a network definition	543
nvinfer1::IRaggedSoftMaxLayer	
A RaggedSoftmax layer in a network definition	546
nvinfer1::IRecurrenceLayer	547
nvinfer1::IReduceLayer	
Layer that represents a reduction across a non-bool tensor	549
nvinfer1::IRefitter	
Updates weights in an engine	552
nvinfer1::IResizeLayer	
A resize layer in a network definition	561
nvinfer1::IReverseSequenceLayer	
A ReverseSequence layer in a network definition	571
nvinfer1::IRNNv2Layer	
An RNN layer in a network definition, version 2	574
nvinfer1::IRuntime	
Allows a serialized functionally unsafe engine to be deserialized	582
nvinfer1::safe::IRuntime	
Allows a serialized functionally safe engine to be deserialized	592
nvinfer1::IScaleLayer	
A Scale layer in a network definition	597
nvinfer1::IScatterLayer	
A scatter layer in a network definition. Supports several kinds of scattering	602
nvinfer1::ISelectLayer	605
nvinfer1::IShapeLayer	
Layer type for getting shape of a tensor	606
nvinfer1::IShuffleLayer	
Layer type for shuffling data	608
nvinfer1::ISliceLayer	
Slices an input tensor into an output tensor based on the offset and strides	613
nvinfer1::ISoftMaxLayer	
A Softmax layer in a network definition	619

nvinfer1::ITensor	621
A tensor in a network definition	
nvinfer1::ITimingCache	633
Class to handle tactic timing info collected from builder	
nvinfer1::ITopKLayer	636
Layer that represents a TopK reduction	
nvinfer1::ITripLimitLayer	639
nvuffparser::IUFFParser	641
Class used for parsing models described using the UFF format	
nvinfer1::IUnaryLayer	645
Layer that represents an unary operation	
nvinfer1::plugin::NMSPParameters	647
The NMSPParameters are used by the BatchedNMSPPlugin for performing the non_max_suppression operation over boxes for object detection networks	
nvinfer1::Permutation	649
nvinfer1::PluginField	650
Structure containing plugin attribute field names and associated data This information can be parsed to decode necessary plugin metadata	
nvinfer1::PluginFieldCollection	652
Plugin field collection struct	
nvinfer1::PluginRegistrar< T >	653
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	
nvinfer1::safe::PluginRegistrar< T >	653
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	
nvinfer1::PluginTensorDesc	654
Fields that a plugin might see for an input or output	
PluginVersion	656
Definition of plugin versions	
nvinfer1::plugin::PriorBoxParameters	656
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. It contains:	
nvinfer1::plugin::Quadruple	659
The Permute plugin layer permutes the input tensor by changing the memory order of the data. Quadruple defines a structure that contains an array of 4 integers. They can represent the permute orders or the strides in each dimension	
nvinfer1::plugin::RegionParameters	660
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	
nvinfer1::plugin::RPROIParams	661
RPROIParams is used to create the RPROIPlugin instance. It contains:	
nvinfer1::plugin::softmaxTree	663
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	
nvinfer1::Weights	665
An array of weights used as a layer parameter	

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

NvCaffeParser.h	667
NvInfer.h	669
NvInferConsistency.h	721
NvInferLegacyDims.h	723
NvInferPlugin.h	725
NvInferPluginUtils.h	732
NvInferRuntime.h	735
NvInferRuntimeCommon.h	754
NvInferSafeRuntime.h	766
NvInferVersion.h	770
NvOnnxConfig.h	773
NvUffParser.h	774
NvUtils.h	777
NvOnnxParser.h	778

Chapter 8

Namespace Documentation

8.1 nvcaffeparser1 Namespace Reference

The TensorRT Caffe parser API namespace.

Classes

- class [IBinaryProtoBlob](#)
Object used to store and query data extracted from a binaryproto file using the [ICaffeParser](#).
- class [IBlobNameToTensor](#)
Object used to store and query Tensors after they have been extracted from a Caffe model using the [ICaffeParser](#).
- class [ICaffeParser](#)
Class used for parsing Caffe models.
- class [IPluginFactoryV2](#)
Plugin factory used to configure plugins.

Functions

- [ICaffeParser](#) * [createCaffeParser](#) () noexcept
Creates a [ICaffeParser](#) object.
- void [shutdownProtobufLibrary](#) () noexcept
Shuts down protocol buffers library.

8.1.1 Detailed Description

The TensorRT Caffe parser API namespace.

8.1.2 Function Documentation

8.1.2.1 createCaffeParser()

```
ICaffeParser * nvcaffeparser1::createCaffeParser ( ) [noexcept]
```

Creates a [ICaffeParser](#) object.

Returns

A pointer to the [ICaffeParser](#) object is returned.

See also

[nvcaffeparser1::ICaffeParser](#)

Deprecated [ICaffeParser](#) will be removed in TensorRT 9.0. Plan to migrate your workflow to use [nvonnxparser::IParser](#) for deployment.

8.1.2.2 shutdownProtobufLibrary()

```
void nvcaffeparser1::shutdownProtobufLibrary ( ) [noexcept]
```

Shuts down protocol buffers library.

Note

No part of the protocol buffers library can be used after this function is called.

8.2 nvinfer1 Namespace Reference

The TensorRT API version 1 namespace.

Namespaces

- namespace [consistency](#)
- namespace [impl](#)
- namespace [plugin](#)
- namespace [safe](#)

The safety subset of TensorRT's API version 1 namespace.

- namespace [utils](#)

Classes

- class [Dims2](#)
Descriptor for two-dimensional data.
- class [Dims3](#)
Descriptor for three-dimensional data.
- class [Dims32](#)
- class [Dims4](#)
Descriptor for four-dimensional data.
- class [DimsExprs](#)
- class [DimsHW](#)
Descriptor for two-dimensional spatial data.
- class [DynamicPluginTensorDesc](#)
- class [IActivationLayer](#)
An Activation layer in a network definition.
- class [IAlgorithm](#)
Describes a variation of execution of a layer. An algorithm is represented by [IAlgorithmVariant](#) and the [IAlgorithmIOInfo](#) for each of its inputs and outputs. An algorithm can be selected or reproduced using `AlgorithmSelector::selectAlgorithms()`.
- class [IAlgorithmContext](#)
Describes the context and requirements, that could be fulfilled by one or more instances of [IAlgorithm](#).
- class [IAlgorithmIOInfo](#)
Carries information about input or output of the algorithm. [IAlgorithmIOInfo](#) for all the input and output along with [IAlgorithmVariant](#) denotes the variation of algorithm and can be used to select or reproduce an algorithm using `IAlgorithmSelector::selectAlgorithms()`.
- class [IAlgorithmSelector](#)
Interface implemented by application for selecting and reporting algorithms of a layer provided by the builder.
- class [IAlgorithmVariant](#)
provides a unique 128-bit identifier, which along with the input and output information denotes the variation of algorithm and can be used to select or reproduce an algorithm, using `IAlgorithmSelector::selectAlgorithms()`
- class [IAssertionLayer](#)
An assertion layer in a network.
- 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](#)
- 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 [IDeconvolutionLayer](#)
A deconvolution layer in a network definition.

- class [IDequantizeLayer](#)
A Dequantize layer in a network definition.
- class [IDimensionExpr](#)
- 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 [IErrorRecorder](#)
Reference counted application-implemented error reporting interface for TensorRT objects.
- class [IExecutionContext](#)
Context for executing inference using an engine, with functionally unsafe features.
- class [IExprBuilder](#)
- class [IFillLayer](#)
Generate an output tensor with specified mode.
- class [IFullyConnectedLayer](#)
A fully connected layer in a network definition. This layer expects an input tensor of three or more non-batch dimensions. The input is automatically reshaped into an $M \times V$ tensor X , where V is a product of the last three dimensions and M is a product of the remaining dimensions (where the product over 0 dimensions is defined as 1). For example:
- class [IGatherLayer](#)
A Gather layer in a network definition. Supports several kinds of gathering.
- class [IGpuAllocator](#)
Application-implemented class for controlling allocation on the GPU.
- 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 [IIifConditional](#)
- class [IIifConditionalBoundaryLayer](#)
- class [IIifConditionalInputLayer](#)
- class [IIifConditionalOutputLayer](#)
- class [IInt8Calibrator](#)
Application-implemented interface for calibration.
- class [IInt8EntropyCalibrator](#)
- class [IInt8EntropyCalibrator2](#)
- class [IInt8LegacyCalibrator](#)
- class [IInt8MinMaxCalibrator](#)
- class [IIteratorLayer](#)
- class [ILayer](#)
Base class for all layer classes in a network definition.
- class [ILogger](#)
Application-implemented logging interface for the builder, refitter and runtime.
- class [ILoop](#)
- class [ILoopBoundaryLayer](#)
- class [ILoopOutputLayer](#)

- class [ILRNLayer](#)
A LRN layer in a network definition.
- class [IMatrixMultiplyLayer](#)
Layer that represents a Matrix Multiplication.
- 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 [IOneHotLayer](#)
A OneHot layer in a network definition.
- class [IOptimizationProfile](#)
Optimization profile for dynamic input dimensions and shape tensors.
- class [IOutputAllocator](#)
Callback from `ExecutionContext::enqueueV3()`
- class [IPaddingLayer](#)
Layer that represents a padding operation.
- class [IParametricReLULayer](#)
Layer that represents a parametric ReLU operation.
- class [IPluginCreator](#)
Plugin creator class for user implemented layers.
- 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 [IPluginV2](#)
Plugin class for user-implemented layers.
- class [IPluginV2DynamicExt](#)
- class [IPluginV2Ext](#)
Plugin class for user-implemented layers.
- class [IPluginV2IOExt](#)
Plugin class for user-implemented layers.
- class [IPluginV2Layer](#)
Layer type for pluginV2.
- class [IPoolingLayer](#)
A Pooling layer in a network definition.
- class [IProfiler](#)
Application-implemented interface for profiling.
- class [IQuantizeLayer](#)
A Quantize layer in a network definition.
- class [IRaggedSoftMaxLayer](#)
A RaggedSoftmax layer in a network definition.
- class [IRecurrenceLayer](#)

- 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 [IRReverseSequenceLayer](#)
A ReverseSequence layer in a network definition.
- class [IRNNv2Layer](#)
An RNN layer in a network definition, version 2.
- class [IRuntime](#)
Allows a serialized functionally unsafe engine to be deserialized.
- 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](#)
- 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 [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](#)
- class [IUnaryLayer](#)
Layer that represents an unary operation.
- struct [Permutation](#)
- 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 [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 [SliceMode](#) = [SampleMode](#)
- using [ResizeMode](#) = [InterpolationMode](#)
- using [QuantizationFlags](#) = uint32_t
Represents one or more QuantizationFlag values using binary OR operations.
- using [BuilderFlags](#) = uint32_t
Represents one or more QuantizationFlag values using binary OR operations, e.g., 1U << [BuilderFlag::kFP16](#) | 1U << [BuilderFlag::kDEBUG](#).
- using [NetworkDefinitionCreationFlags](#) = uint32_t
Represents one or more NetworkDefinitionCreationFlag flags using binary OR operations. e.g., 1U << [NetworkDefinitionCreationFlag::kEXPLICIT_BATCH](#).
- using [TempfileControlFlags](#) = uint32_t
Represents a collection of one or more TempfileControlFlag values combined using bitwise-OR operations.
- using [TacticSources](#) = uint32_t
Represents a collection of one or more TacticSource values combine using bitwise-OR operations.
- using [char_t](#) = char
char_t is the type used by TensorRT to represent all valid characters.
- using [AsciiChar](#) = [char_t](#)
- using [Dims](#) = [Dims32](#)
- using [PluginFormat](#) = [TensorFormat](#)
PluginFormat is reserved for backward compatibility.
- using [AllocatorFlags](#) = uint32_t

Enumerations

- enum class [LayerType](#) : int32_t {
[kCONVOLUTION](#) = 0 , [kFULLY_CONNECTED](#) = 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 ,
[kRNN_V2](#) = 20 , [kIDENTITY](#) = 21 , [kPLUGIN_V2](#) = 22 , [kSLICE](#) = 23 ,
[kSHAPE](#) = 24 , [kPARAMETRIC_RELU](#) = 25 , [kRESIZE](#) = 26 , [kTRIP_LIMIT](#) = 27 ,
[kRECURRENCE](#) = 28 , [kITERATOR](#) = 29 , [kLOOP_OUTPUT](#) = 30 , [kSELECT](#) = 31 ,
[kFILL](#) = 32 , [kQUANTIZE](#) = 33 , [kDEQUANTIZE](#) = 34 , [kCONDITION](#) = 35 ,
[kCONDITIONAL_INPUT](#) = 36 , [kCONDITIONAL_OUTPUT](#) = 37 , [kSCATTER](#) = 38 , [keINSUM](#) = 39 ,
[kASSERTION](#) = 40 , [kONE_HOT](#) = 41 , [kNON_ZERO](#) = 42 , [kGRID_SAMPLE](#) = 43 ,
[kNMS](#) = 44 , [kREVERSE_SEQUENCE](#) = 45 , [kNORMALIZATION](#) = 46 , [kCAST](#) = 47 }
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 }
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 ,
kCAFFE_ROUND_DOWN = 4 , **kCAFFE_ROUND_UP** = 5 }
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 **RNNOperation** : int32_t { **kRELU** = 0 , **kTANH** = 1 , **kLSTM** = 2 , **kGRU** = 3 }
Enumerates the RNN operations that may be performed by an RNN layer.
- enum class **RNNDirection** : int32_t { **kUNIDIRECTION** = 0 , **kBIDIRECTION** = 1 }
Enumerates the RNN direction that may be performed by an RNN layer.
- enum class **RNNInputMode** : int32_t { **kLINEAR** = 0 , **kSKIP** = 1 }
Enumerates the RNN input modes that may occur with an RNN layer.
- enum class **RNNGateType** : int32_t {
kINPUT = 0 , **kOUTPUT** = 1 , **kFORGET** = 2 , **kUPDATE** = 3 ,
kRESET = 4 , **kCELL** = 5 , **kHIDDEN** = 6 }
Identifies an individual gate within an RNN cell.
- 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 , **kSINF** = 23 }
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 }
Enumerates the reduce operations that may be performed by a Reduce layer.
- enum class **SampleMode** : int32_t {
kSTRICT_BOUNDS = 0 , **kDEFAULT** = kSTRICT_BOUNDS , **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** , **kTRANPOSE** , **kVECTOR** }
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 { [kLINSAPCE](#) = 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 [CalibrationAlgoType](#) : int32_t { [kLEGACY_CALIBRATION](#) = 0 , [kENTROPY_CALIBRATION](#) = 1 , [kENTROPY_CALIBRATION_2](#) = 2 , [kMINMAX_CALIBRATION](#) = 3 }

Version of calibration algorithm to use.

- enum class [QuantizationFlag](#) : int32_t { [kCALIBRATE_BEFORE_FUSION](#) = 0 }

List of valid flags for quantizing the network to int8.

- enum class [BuilderFlag](#) : int32_t { [kFP16](#) = 0 , [kINT8](#) = 1 , [kDEBUG](#) = 2 , [kGPU_FALLBACK](#) = 3 , [kSTRICT_TYPES](#) = 4 , [kREFIT](#) = 5 , [kDISABLE_TIMING_CACHE](#) = 6 , [kTF32](#) = 7 , [kSPARSE_WEIGHTS](#) = 8 , [kSAFETY_SCOPE](#) = 9 , [kOBEY_PRECISION_CONSTRAINTS](#) = 10 , [kPREFER_PRECISION_CONSTRAINTS](#) = 11 , [kDIRECT_IO](#) = 12 , [kREJECT_EMPTY_ALGORITHMS](#) = 13 , [kENABLE_TACTIC_HEURISTIC](#) = 14 , [kVERSION_COMPATIBLE](#) = 15 , [kEXCLUDE_LEAN_RUNTIME](#) = 16 , [kFP8](#) = 17 }

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 }

The type for memory pools used by TensorRT.

- enum class [PreviewFeature](#) : int32_t { [kFASTER_DYNAMIC_SHAPES_0805](#) = 0 , [kDISABLE_EXTERNAL_TACTIC_SOURCES](#) = 1 , [kPROFILE_SHARING_0806](#) = 2 , [kENABLE_MULTI_STREAM_ENGINE_0806](#) = 3 }

Define preview features.

- enum class [HardwareCompatibilityLevel](#) : int32_t { [kNONE](#) = 0 , [kAMPERE_PLUS](#) = 1 }

- enum class [NetworkDefinitionCreationFlag](#) : int32_t { [kEXPLICIT_BATCH](#) = 0 , [kEXPLICIT_PRECISION](#) = 1 }

List of immutable network properties expressed at network creation time. NetworkDefinitionCreationFlag is used with createNetworkV2() to specify immutable properties of the network. Creating a network without NetworkDefinitionCreationFlag::kEXPLICIT_BATCH flag has been deprecated.

- enum class [EngineCapability](#) : int32_t { [kSTANDARD](#) = 0 , [kDEFAULT](#) = kSTANDARD , [kSAFETY](#) = 1 , [kSAFE_GPU](#) = kSAFETY , [kDLA_STANDALONE](#) = 2 , [kSAFE_DLA](#) = kDLA_STANDALONE }

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 `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` , `kDLA` }
- 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 `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` {
`kCUBLAS` = 0 , `kCUBLAS_LT` = 1 , `kCUDNN` = 2 , `kEDGE_MASK_CONVOLUTIONS` = 3 ,
`kJIT_CONVOLUTIONS` = 4 }
- List of tactic sources for TensorRT.*
- enum class `ProfilingVerbosity` : `int32_t` {
`kLAYER_NAMES_ONLY` = 0 , `kNONE` = 1 , `kDETAILED` = 2 , `kDEFAULT` = `kLAYER_NAMES_ONLY` ,
`kVERBOSE` = `kDETAILED` }
- List of verbosity levels of layer information exposed in NVTX annotations and in IEngineInspector.*
- enum class `LayerInformationFormat` : `int32_t` { `kONELINE` = 0 , `kJSON` = 1 }
- The format in which the IEngineInspector prints the layer information.*
- enum class `DataType` : `int32_t` {
`kFLOAT` = 0 , `kHALF` = 1 , `kINT8` = 2 , `kINT32` = 3 ,
`kBOOL` = 4 , `kUINT8` = 5 , `kFP8` = 6 }
- The type of weights and tensors.*
- enum class `TensorFormat` : `int32_t` {
`kLINEAR` = 0 , `kCHW2` = 1 , `kHWC8` = 2 , `kCHW4` = 3 ,
`kCHW16` = 4 , `kCHW32` = 5 , `kDHW8` = 6 , `kCDHW32` = 7 ,
`kHWC` = 8 , `kDLA_LINEAR` = 9 , `kDLA_HWC4` = 10 , `kHWC16` = 11 ,
`kDHW8` = 12 }
- Format of the input/output tensors.*
- enum class `PluginVersion` : `uint8_t` { `kV2` = 0 , `kV2_EXT` = 1 , `kV2_IOEXT` = 2 , `kV2_DYNAMICEXT` = 3 }
- enum class `PluginFieldType` : `int32_t` {
`kFLOAT16` = 0 , `kFLOAT32` = 1 , `kFLOAT64` = 2 , `kINT8` = 3 ,
`kINT16` = 4 , `kINT32` = 5 , `kCHAR` = 6 , `kDIMS` = 7 ,
`kUNKNOWN` = 8 }
- enum class `AllocatorFlag` : `int32_t` { `kRESIZABLE` = 0 }
- 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.

Functions

- template<> constexpr int32_t [EnumMax< LayerType >](#) () noexcept
- template<> constexpr int32_t [EnumMax< ScaleMode >](#) () noexcept
- template<> constexpr int32_t [EnumMax< GatherMode >](#) () noexcept
- template<> constexpr int32_t [EnumMax< RNNOperation >](#) () noexcept
- template<> constexpr int32_t [EnumMax< RNNDirection >](#) () noexcept
- template<> constexpr int32_t [EnumMax< RNNInputMode >](#) () noexcept
- template<> constexpr int32_t [EnumMax< RNNGateType >](#) () noexcept
- template<> constexpr int32_t [EnumMax< UnaryOperation >](#) () noexcept
- template<> constexpr int32_t [EnumMax< ReduceOperation >](#) () noexcept
- template<> constexpr int32_t [EnumMax< SampleMode >](#) () noexcept
- template<> constexpr int32_t [EnumMax< TopKOperation >](#) () noexcept
- template<> constexpr int32_t [EnumMax< MatrixOperation >](#) () noexcept
- template<> constexpr int32_t [EnumMax< LoopOutput >](#) () noexcept
- template<> constexpr int32_t [EnumMax< TripLimit >](#) () noexcept
- template<> constexpr int32_t [EnumMax< FillOperation >](#) () noexcept
- template<> constexpr int32_t [EnumMax< ScatterMode >](#) () noexcept
- template<> constexpr int32_t [EnumMax< BoundingBoxFormat >](#) () noexcept
- template<> constexpr int32_t [EnumMax< CalibrationAlgoType >](#) () noexcept
- template<> constexpr int32_t [EnumMax< QuantizationFlag >](#) () noexcept
- template<> constexpr int32_t [EnumMax< BuilderFlag >](#) () noexcept
- template<> constexpr int32_t [EnumMax< MemoryPoolType >](#) () noexcept
- template<> constexpr int32_t [EnumMax< NetworkDefinitionCreationFlag >](#) () noexcept
- [nvinfer1::IPluginRegistry * getBuilderPluginRegistry \(nvinfer1::EngineCapability capability\)](#) noexcept

Return the plugin registry for the given capability or nullptr if no registry exists.

- template<> constexpr int32_t [EnumMax< DimensionOperation >](#) () noexcept

Maximum number of elements in DimensionOperation enum.

- template<> constexpr int32_t [EnumMax< WeightsRole >](#) () noexcept

Maximum number of elements in WeightsRole enum.

- template<> constexpr int32_t [EnumMax< DeviceType >](#) () noexcept

Maximum number of elements in DeviceType enum.

- template<> constexpr int32_t [EnumMax< TempfileControlFlag >](#) () noexcept

Maximum number of elements in TempfileControlFlag enum.

- template<> constexpr int32_t [EnumMax< OptProfileSelector >](#) () noexcept

Number of different values of OptProfileSelector enum.

- template<> constexpr int32_t [EnumMax< TacticSource >](#) () noexcept

Maximum number of tactic sources in TacticSource enum.

- template<> constexpr int32_t [EnumMax< ProfilingVerbosity >](#) () noexcept

Maximum number of profile verbosity levels in ProfilingVerbosity enum.

- template<> constexpr int32_t [EnumMax< LayerInformationFormat >](#) () noexcept

- template<typename T>

constexpr int32_t [EnumMax](#) () noexcept

Maximum number of elements in an enumeration type.

8.2.1 Detailed Description

The TensorRT API version 1 namespace.

8.2.2 Typedef Documentation

8.2.2.1 AllocatorFlags

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

8.2.2.2 AsciiChar

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

AsciiChar is the type used by TensorRT to represent valid ASCII characters. This type is used by [IPluginV2](#), [PluginField](#), [IPluginCreator](#), [IPluginRegistry](#), and [ILogger](#) due to their use in automotive safety context.

8.2.2.3 BuilderFlags

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

Represents one or more QuantizationFlag values using binary OR operations, e.g., `1U << BuilderFlag::kFP16 | 1U << BuilderFlag::kDEBUG`.

See also

[IBuilderConfig::getFlags\(\)](#), [ITensor::setFlags\(\)](#),

8.2.2.4 char_t

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

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

8.2.2.5 Dims

```
using nvinfer1::Dims = typedef Dims32
```

Alias for [Dims32](#).

Warning

: This alias might change in the future.

8.2.2.6 NetworkDefinitionCreationFlags

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

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

See also

[IBuilder::createNetworkV2](#)

8.2.2.7 PluginFormat

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

PluginFormat is reserved for backward compatibility.

See also

[IPluginV2::supportsFormat\(\)](#)

8.2.2.8 QuantizationFlags

```
using nvinfer1::QuantizationFlags = typedef uint32_t
```

Represents one or more `QuantizationFlag` values using binary OR operations.

See also

[IBuilderConfig::getQuantizationFlags\(\)](#), [IBuilderConfig::setQuantizationFlags\(\)](#)

8.2.2.9 ResizeMode

```
using nvinfer1::ResizeMode = typedef InterpolationMode
```

Deprecated Deprecated in TensorRT 8.5. Superseded by InterpolationMode.

8.2.2.10 SliceMode

```
using nvinfer1::SliceMode = typedef SampleMode
```

Deprecated Deprecated in TensorRT 8.5. Superseded by SampleMode.

8.2.2.11 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\(\)](#)

8.2.2.12 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\(\)](#)

8.2.2.13 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\(\)](#),

8.2.3 Enumeration Type Documentation

8.2.3.1 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: $x \geq 0 ? x : \alpha * x$.
kELU	Elu activation: $x \geq 0 ? x : \alpha * (\exp(x) - 1)$.
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: $x > \alpha ? x : 0$.

8.2.3.2 AllocatorFlag

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

Enumerator

kRESIZABLE	TensorRT may call realloc() on this allocation.
------------	---

8.2.3.3 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

8.2.3.4 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::setFlag\(\)](#), [IBuilderConfig::getFlag\(\)](#)

Enumerator

kFP16	Enable FP16 layer selection, with FP32 fallback.
kINT8	Enable Int8 layer selection, with FP32 fallback with FP16 fallback if kFP16 also specified.
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.

Enumerator

kSTRICT_TYPES	<p>Legacy flag with effect similar to setting all of these three flags:</p> <ul style="list-style-type: none"> * kPREFER_PRECISION_CONSTRAINTS * kDIRECT_IO * kREJECT_EMPTY_ALGORITHMS <p>except that if the direct I/O requirement cannot be met and kDIRECT_IO was not set, instead of the build failing, the build falls back as if kDIRECT_IO was not set.</p> <p>\deprecated Deprecatd in TensorRT 8.2.</p>
kREFIT	Enable building a refittable engine.
kDISABLE_TIMING_CACHE	Disable reuse of timing information across identical layers.
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.
kOBEY_PRECISION_CONSTRAINTS	Require that layers execute in specified precisions. Build fails otherwise.
kPREFER_PRECISION_CONSTRAINTS	Prefer that layers execute in specified precisions. Fall back (with warning) to another precision if build would otherwise fail.
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.
kREJECT_EMPTY_ALGORITHMS	Fail if IAlgorithmSelector::selectAlgorithms returns an empty set of algorithms.
kENABLE_TACTIC_HEURISTIC	<p>Enable heuristic-based tactic selection for shorter engine generation time. The engine may not be as performant as when built with a profiling-based builder.</p> <p>This flag is only supported by NVIDIA Ampere and later GPUs.</p>
kVERSION_COMPATIBLE	<p>Restrict to lean runtime operators to provide version forward compatibility for the plan.</p> <p>Using this flag with ICudaEngine::serialize() and BuilderFlag::kREFIT would result in error. This flag is only supported by NVIDIA Volta and later GPUs. This flag is not supported in NVIDIA Drive(R) products. This flag is not supported with implicit batch mode. Network must be created with NetworkDefinitionCreationFlag::kEXPLICIT_BATCH.</p>

Enumerator

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. This flag is not supported with implicit batch mode. Network must be created with NetworkDefinitionCreationFlag::kEXPLICIT_BATCH .
kFP8	Enable FP8 layer selection, with FP32 fallback.

8.2.3.5 CalibrationAlgoType

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

Version of calibration algorithm to use.

```
enum CalibrationAlgoType
```

Enumerator

kLEGACY_CALIBRATION	
kENTROPY_CALIBRATION	
kENTROPY_↔ CALIBRATION_2	
kMINMAX_CALIBRATION	

8.2.3.6 DataType

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

The type of weights and tensors.

Enumerator

kFLOAT	32-bit floating point format.
kHALF	IEEE 16-bit floating-point format.
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.

Enumerator

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.

8.2.3.7 DeviceType

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

The device that this layer/network will execute on.

Enumerator

kGPU	GPU Device.
kDLA	DLA Core.

8.2.3.8 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.

Enumerator

kMIN	Minimum of the two operands.
kSUB	Subtract 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.

8.2.3.9 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.

Operation kPOW must have inputs of [DataType](#) kFLOAT, kHALF, or kINT8.

All other operations must have inputs of [DataType](#) kFLOAT, kHALF, kINT8, or kINT32.

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.

8.2.3.10 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. When [BuilderFlag::kSAFETY_SCOPE](#) is not set (by default), [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 [nvinfer1::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 .
kDEFAULT	Deprecated Deprecated in TensorRT 8.0. Superseded by kSTANDARD.
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.
kSAFE_GPU	Deprecated Deprecated in TensorRT 8.0. Superseded by kSAFETY.
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 .
kSAFE_DLA	Deprecated Deprecated in TensorRT 8.0. Superseded by kDLA_STANDALONE.

8.2.3.11 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.

Enumerator

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.
kFAILED_EXECUTION	An error occurred during execution that caused TensorRT to end prematurely, either an asynchronous error 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	<p>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.</p> <p>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.</p>
kUNSUPPORTED_STATE	<p>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:</p> <ul style="list-style-type: none"> • 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.

8.2.3.12 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	Generate evenly spaced numbers over a specified interval.
kRandomUniform	Generate a tensor with random values drawn from a uniform distribution.
kRandomNormal	Generate a tensor with random values drawn from a normal distribution.

8.2.3.13 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.

8.2.3.14 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. Levels except kNone are only supported for engines built on NVIDIA Ampere and later GPUs. 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 max shared memory usage 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.

8.2.3.15 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.

8.2.3.16 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.

8.2.3.17 LayerType

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

The type values of layer classes.

See also

[ILayer::getType\(\)](#)

Enumerator

kCONVOLUTION	Convolution layer.
kFULLY_CONNECTED	Fully connected 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.
krnn_v2	RNNv2 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.

Enumerator

kQUANTIZE	Quantize layer.
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.
kCAST	Cast layer.

8.2.3.18 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.

8.2.3.19 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.
kTRANPOSE	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::kTRANPOSE . 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::kTRANPOSE .

8.2.3.20 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 is equivalent to the deprecated IBuilderConfig::setMaxWorkspaceSize and overrides that value. This defaults to max device memory. Set to a smaller value to restrict tactics that use over the threshold en masse. For more targeted removal of tactics use the IAlgorithmSelector interface.
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, and Xavier shares 4 MiB across all of its accelerator cores.
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.
kTACTIC_DRAM	kTACTIC_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
TensorRT 8.6.0 API Reference	cudaGetDeviceProperties when cudaGetDeviceProperties.embedded is true, and 100% otherwise.

8.2.3.21 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. Creating a network without NetworkDefinitionCreationFlag::kEXPLICIT_BATCH flag has been deprecated.

See also

[IBuilder::createNetworkV2](#)

Enumerator

kEXPLICIT_BATCH	Mark the network to be an explicit batch network. Dynamic shape support requires that the kEXPLICIT_BATCH flag is set. With dynamic shapes, any of the input dimensions can vary at run-time, and there are no implicit dimensions in the network specification. Varying dimensions are specified by using the wildcard dimension value -1.
kEXPLICIT_PRECISION	Deprecated. This flag has no effect now, but is only kept for backward compatability.

8.2.3.22 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::setShapeValues\(\)](#)

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.

8.2.3.23 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 three padding styles, EXPLICIT, SAME, and CAFFE, with each style having two variants. The EXPLICIT and CAFFE styles 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. For deconvolution, prePadding is set before output.
A = postPadding, after the image data. For deconvolution, postPadding is set after output.
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:**

$$O = \text{floor}((M - DK) / S) + 1$$
- **CAFFE_ROUND_DOWN:**

$$O = \text{floor}((I + B * 2 - DK) / S) + 1$$
- **EXPLICIT_ROUND_UP:**

$$O = \text{ceil}((M - DK) / S) + 1$$
- **CAFFE_ROUND_UP:**

$$O = \text{ceil}((I + B * 2 - DK) / S) + 1$$
- **SAME_UPPER:**

$$O = \text{ceil}(I / S)$$

$$P = \text{floor}((I - 1) / S) * S + DK - I;$$

$$B = \text{floor}(P / 2)$$

$$A = P - B$$
- **SAME_LOWER:**

$$O = \text{ceil}(I / S)$$

$$P = \text{floor}((I - 1) / S) * S + DK - I;$$

$$A = \text{floor}(P / 2)$$

$$B = P - A$$

Formulas for Deconvolution:

- **EXPLICIT_ROUND_DOWN:**
- **CAFFE_ROUND_DOWN:**
- **EXPLICIT_ROUND_UP:**
- **CAFFE_ROUND_UP:**

$$O = (I - 1) * S + DK - (B + A)$$
- **SAME_UPPER:**

$$O = \min(I * S, (I - 1) * S + DK)$$

$$P = \max(DK - S, 0)$$

$$B = \text{floor}(P / 2)$$

$$A = P - B$$

- **SAME_LOWER:**

$$O = \min(I * S, (I - 1) * S + DK)$$

$$P = \max(DK - S, 0)$$

$$A = \text{floor}(P / 2)$$

$$B = P - A$$

Formulas for Pooling:

- **EXPLICIT_ROUND_DOWN:**

$$O = \text{floor}((M - F) / S) + 1$$
- **EXPLICIT_ROUND_UP:**

$$O = \text{ceil}((M - F) / S) + 1$$
- **SAME_UPPER:**

$$O = \text{ceil}(I / S)$$

$$P = \text{floor}((I - 1) / S) * S + F - I;$$

$$B = \text{floor}(P / 2)$$

$$A = P - B$$
- **SAME_LOWER:**

$$O = \text{ceil}(I / S)$$

$$P = \text{floor}((I - 1) / S) * S + F - I;$$

$$A = \text{floor}(P / 2)$$

$$B = P - A$$
- **CAFFE_ROUND_DOWN:**

$$\text{EXPLICIT_ROUND_DOWN} - ((\text{EXPLICIT_ROUND_DOWN} - 1) * S \geq I + B)$$
- **CAFFE_ROUND_UP:**

$$\text{EXPLICIT_ROUND_UP} - ((\text{EXPLICIT_ROUND_UP} - 1) * S \geq I + B)$$

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:

$$M = \{6, 6\} + \{3, 3\} + \{2, 2\} \Rightarrow \{11, 11\}$$

$$O \Rightarrow \text{floor}((M - F) / S) + 1$$

$$\Rightarrow \text{floor}((\{11, 11\} - \{3, 3\}) / \{2, 2\}) + \{1, 1\}$$

$$\Rightarrow \text{floor}(\{8, 8\} / \{2, 2\}) + \{1, 1\}$$

$$\Rightarrow \{5, 5\}$$

- **EXPLICIT_ROUND_UP:**
 Computation:

$$M = \{6, 6\} + \{3, 3\} + \{2, 2\} \Rightarrow \{11, 11\}$$

$$O \Rightarrow \text{ceil}((M - F) / S) + 1$$

$$\Rightarrow \text{ceil}((\{11, 11\} - \{3, 3\}) / \{2, 2\}) + \{1, 1\}$$

$$\Rightarrow \text{ceil}(\{8, 8\} / \{2, 2\}) + \{1, 1\}$$

$$\Rightarrow \{5, 5\}$$

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

- **SAME_UPPER:**
 Computation:

$$I = \{6, 6\}$$

$$S = \{2, 2\}$$

$$O = \text{ceil}(I / S) = \{3, 3\}$$

$$P = \text{floor}((I - 1) / S) * S + F - I$$

$$\Rightarrow \text{floor}((\{6, 6\} - \{1, 1\}) / \{2, 2\}) * \{2, 2\} + \{3, 3\} - \{6, 6\}$$

$$\Rightarrow \{4, 4\} + \{3, 3\} - \{6, 6\}$$

$$\Rightarrow \{1, 1\}$$

$$B = \text{floor}(\{1, 1\} / \{2, 2\})$$

$$\Rightarrow \{0, 0\}$$

$$A = \{1, 1\} - \{0, 0\}$$

$$\Rightarrow \{1, 1\}$$

- **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?

- **CAFFE_ROUND_DOWN:**

Computation:

```

M = {6, 6} + {3, 3} + {3, 3} ==> {12, 12}
EXPLICIT_ROUND_DOWN ==> floor((M - F) / S) + 1
  ==> floor(({12, 12} - {3, 3}) / {2, 2}) + {1, 1}
  ==> {5, 5}
DIFF = (((EXPLICIT_ROUND_DOWN - 1) * S >= I + B) ? {1, 1} : {0, 0})
  ==> ({5, 5} - {1, 1}) * {2, 2} >= {6, 6} + {3, 3} ? {1, 1} : {0, 0}
  ==> {0, 0}
O ==> EXPLICIT_ROUND_DOWN - DIFF
  ==> {5, 5} - {0, 0}
  ==> {5, 5}

```

- **CAFFE_ROUND_UP:**

Computation:

```

M = {6, 6} + {3, 3} + {3, 3} ==> {12, 12}
EXPLICIT_ROUND_UP ==> ceil((M - F) / S) + 1
  ==> ceil(({12, 12} - {3, 3}) / {2, 2}) + {1, 1}
  ==> {6, 6}
DIFF = (((EXPLICIT_ROUND_UP - 1) * S >= I + B) ? {1, 1} : {0, 0})
  ==> ({6, 6} - {1, 1}) * {2, 2} >= {6, 6} + {3, 3} ? {1, 1} : {0, 0}
  ==> {1, 1}
O ==> EXPLICIT_ROUND_UP - DIFF
  ==> {6, 6} - {1, 1}
  ==> {5, 5}

```

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

CAFFE_ROUND_DOWN and CAFFE_ROUND_UP have two restrictions each on usage with pooling operations. This will cause getDimensions to return an empty dimension and also to reject the network at validation time.

For more information on original reference code, see https://github.com/BVLC/caffe/blob/master/src/caffe/layer_factory.cpp

- **Restriction 1:**

CAFFE_ROUND_DOWN: $B \geq F$ is an error if $(B - S) < F$
 CAFFE_ROUND_UP: $(B + S) \geq (F + 1)$ is an error if $B < (F + 1)$

- **Restriction 2:**

CAFFE_ROUND_DOWN: $(B - S) \geq F$ is an error if $B \geq F$
 CAFFE_ROUND_UP: $B \geq (F + 1)$ is an error if $(B + S) \geq (F + 1)$

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.
kCAFFE_ROUND_DOWN	Use CAFFE padding, rounding output size down, uses prePadding value.
kCAFFE_ROUND_UP	Use CAFFE padding, rounding output size up, uses prePadding value.

8.2.3.24 PluginFieldType

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

Enumerator

kFLOAT16	FP16 field type.
kFLOAT32	FP32 field type.
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.

8.2.3.25 PluginVersion

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

Enumerator

kV2	IPluginV2 .
kV2_EXT	IPluginV2Ext .
kV2_IOEXT	IPluginV2IOExt .
kV2_DYNAMICEXT	IPluginV2DynamicExt .

8.2.3.26 PoolingType

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

The type of pooling to perform in a pooling layer.

Enumerator

kMAX	
kAVERAGE	
kMAX_AVERAGE_BLEND	

8.2.3.27 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

kFASTER_DYNAMIC_SHAPES_0805	Optimize runtime dimensions with TensorRT's DL Compiler. Potentially reduces run time and decreases device memory usage and engine size. Models most likely to benefit from enabling kFASTER_DYNAMIC_SHAPES_0805 are transformer-based models, and models containing dynamic control flows.
kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805	<p>Disable usage of cuDNN/cuBLAS/cuBLASLt tactics in the TensorRT core library.</p> <p>When the flag is enabled, TensorRT core will not use these tactics even if they are specified in IBuilderConfig::setTacticSources(), but cudnnContext and cublasContext handles will still be passed to plugins via IPluginV2::attachToContext() if the appropriate tactic sources are set.</p> <p>This allows users to experiment with disabling external library tactics without having to modify their application's plugins to support nullptr handles. The default value for this flag is on.</p> <p>See also</p> <p>TacticSource</p>
kPROFILE_SHARING_0806	Allows optimization profiles to be shared across execution contexts. This flag defaults to false and will become the default behavior in TensorRT 9.0. At that point this flag will do nothing.
kENABLE_MULTI_STREAM_ENGINE_0806	<p>Build an engine with multi-stream data.</p> <p>Using multi-stream increases layer dispatching overhead and the activation memory consumption. Users shall decide which configuration is better based on experimentation. Using CUDA graphs can help reduce dispatching overheads.</p>

8.2.3.28 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.
kDEFAULT	Deprecated Deprecated in TensorRT 8.0. Superseded by kLAYER_NAMES_ONLY.
kVERBOSE	Deprecated Deprecated in TensorRT 8.0. Superseded by kDETAILED.

8.2.3.29 QuantizationFlag

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

List of valid flags for quantizing the network to int8.

See also

[IBuilderConfig::setQuantizationFlag\(\)](#), [IBuilderConfig::getQuantizationFlag\(\)](#)

Enumerator

kCALIBRATE_BEFORE_FUSION	Run int8 calibration pass before layer fusion. Only valid for IInt8LegacyCalibrator and IInt8EntropyCalibrator . The builder always runs the int8 calibration pass before layer fusion for IInt8MinMaxCalibrator and IInt8EntropyCalibrator2 . Disabled by default.
--------------------------	---

8.2.3.30 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

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.

Enumerator

kSUM	
kPROD	
kMAX	
kMIN	
kAVG	

8.2.3.31 ResizeCoordinateTransformation

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

The resize coordinate transformation function.

See also

[IResizeLayer::setCoordinateTransformation\(\)](#)

Enumerator

kALIGN_CORNERS	<p>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 <code>x_origin</code> as the coordinate of axis x in the input tensor, <code>x_resized</code> as the coordinate of axis x in the output tensor, <code>length_origin</code> as length of the input tensor in axis x, and <code>length_resize</code> as length of the output tensor in axis x.</p> <pre> <-----length-----> 0 1 2 3 * * * * </pre> $x_origin = x_resized * (length_origin - 1) / (length_resize - 1)$
kASYMMETRIC	<pre> <-----length-----> 0 1 2 3 x_origin = x_resized * (length_origin / length_resize) </pre>
kHALF_PIXEL	<pre> <-----length-----> 0 1 2 3 x_origin = (x_resized + 0.5) * (length_origin / length_resize) - 0.5 </pre>

8.2.3.32 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.

8.2.3.33 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.

8.2.3.34 RNNDirection

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

Enumerates the RNN direction that may be performed by an RNN layer.

See also

[IRNNv2Layer](#)

Enumerator

kUNIDIRECTION	Network iterations from first input to last input.
kBIDIRECTION	Network iterates from first to last and vice versa and outputs concatenated.

8.2.3.35 RNNGateType

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

Identifies an individual gate within an RNN cell.

See also

[RNNOperation](#)

Enumerator

kINPUT	Input gate (i).
kOUTPUT	Output gate (o).
kFORGET	Forget gate (f).
kUPDATE	Update gate (z).
kRESET	Reset gate (r).
kCELL	Cell gate (c).
kHIDDEN	Hidden gate (h).

8.2.3.36 RNNInputMode

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

Enumerates the RNN input modes that may occur with an RNN layer.

If the RNN is configured with [RNNInputMode::kLINEAR](#), then for each gate g in the first layer of the RNN, the input vector $X[t]$ (length E) is left-multiplied by the gate's corresponding weight matrix $W[g]$ (dimensions $H \times E$) as usual, before being used to compute the gate output as described by [RNNOperation](#).

If the RNN is configured with [RNNInputMode::kSKIP](#), then this initial matrix multiplication is "skipped" and $W[g]$ is conceptually an identity matrix. In this case, the input vector $X[t]$ must have length H (the size of the hidden state).

See also

[IRNNv2Layer](#)

Enumerator

kLINEAR	Perform the normal matrix multiplication in the first recurrent layer.
kSKIP	No operation is performed on the first recurrent layer.

8.2.3.37 RNNOperation

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

Enumerates the RNN operations that may be performed by an RNN layer.

Equation definitions

The equations below have the following naming convention:

```
t := current time step
i := input gate
o := output gate
f := forget gate
z := update gate
r := reset gate
c := cell gate
h := hidden gate
g[t] denotes the output of gate g at timestep t, e.g.
f[t] is the output of the forget gate f.
X[t] := input tensor for timestep t
C[t] := cell state for timestep t
H[t] := hidden state for timestep t
W[g] := W (input) parameter weight matrix for gate g
R[g] := U (recurrent) parameter weight matrix for gate g
Wb[g] := W (input) parameter bias vector for gate g
Rb[g] := U (recurrent) parameter bias vector for gate g
Unless otherwise specified, all operations apply pointwise
to elements of each operand tensor.
ReLU(X) := max(X, 0)
tanh(X) := hyperbolic tangent of X
sigmoid(X) := 1 / (1 + exp(-X))
exp(X) := e^X
A.B denotes matrix multiplication of A and B.
A*B denotes pointwise multiplication of A and B.
```

Equations

Depending on the value of RNNOperation chosen, each sub-layer of the RNN layer will perform one of the following operations:

```
::kReLU
H[t] := ReLU(W[i].X[t] + R[i].H[t-1] + Wb[i] + Rb[i])
::kTANH
H[t] := tanh(W[i].X[t] + R[i].H[t-1] + Wb[i] + Rb[i])
::kLSTM
i[t] := sigmoid(W[i].X[t] + R[i].H[t-1] + Wb[i] + Rb[i])
f[t] := sigmoid(W[f].X[t] + R[f].H[t-1] + Wb[f] + Rb[f])
o[t] := sigmoid(W[o].X[t] + R[o].H[t-1] + Wb[o] + Rb[o])
c[t] := tanh(W[c].X[t] + R[c].H[t-1] + Wb[c] + Rb[c])
C[t] := f[t]*C[t-1] + i[t]*c[t]
H[t] := o[t]*tanh(C[t])
::kGRU
z[t] := sigmoid(W[z].X[t] + R[z].H[t-1] + Wb[z] + Rb[z])
r[t] := sigmoid(W[r].X[t] + R[r].H[t-1] + Wb[r] + Rb[r])
h[t] := tanh(W[h].X[t] + r[t]*(R[h].H[t-1] + Rb[h]) + Wb[h])
H[t] := (1 - z[t])*h[t] + z[t]*H[t-1]
```

See also

[IRNNv2Layer](#)

Enumerator

kRELU	Single gate RNN w/ ReLU activation function.
kTANH	Single gate RNN w/ TANH activation function.
kLSTM	Four-gate LSTM network w/o peephole connections.
kGRU	Three-gate network consisting of Gated Recurrent Units.

8.2.3.38 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.
kDEFAULT	
kWRAP	Coordinates wrap around periodically. Deprecated Use kSTRICT_BOUNDS.
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.

8.2.3.39 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.

8.2.3.40 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.

8.2.3.41 TacticSource

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

List of tactic sources for TensorRT.

See also

[TacticSources](#), [IBuilderConfig::setTacticSources\(\)](#), [IBuilderConfig::getTacticSources\(\)](#), [PreviewFeature::kDISABLE_EXTERNAL](#)

Enumerator

kCUBLAS	cuBLAS tactics. Enabled by default. Note Disabling kCUBLAS will cause the cublas handle passed to plugins in attachToContext to be null.
kCUBLAS_LT	cuBLAS LT tactics. Enabled for x86 platforms and only enabled for non-x86 platforms when CUDA >= 11.0 by default.

Enumerator

kCUDNN	cuDNN tactics. Enabled by default. Note Disabling kCUDNN will cause the cuDNN handle passed to plugins in attachToContext to be null.
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.

8.2.3.42 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 <code>IRuntime::setTemporaryDirectory()</code>

8.2.3.43 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\(\)](#)

For more information about data formats, see the topic "Data Format Description" located in the TensorRT Developer Guide.

Enumerator

kLINEAR	Row major linear format. For a tensor with dimensions $\{N, C, H, W\}$ or $\{\text{numbers, channels, columns, rows}\}$, the dimensional index corresponds to $\{3, 2, 1, 0\}$ and thus the order is W minor. For DLA usage, the tensor sizes are limited to C,H,W in the range [1,8192].
kCHW2	Two wide channel vectorized row major format. This format is bound to 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 $[N][(C+1)/2][H][W][2]$, with the tensor coordinates (n, c, h, w) mapping to array subscript $[n][c/2][h][w][c\%2]$.
kHWC8	Eight channel format where C is padded to a multiple of 8. This format is bound to FP16. 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 $[n][h][w][c]$.
kCHW4	Four wide channel vectorized row major format. This format is bound to INT8 or FP16. It is only available for dimensions ≥ 3 . For INT8, the C dimension must be a build-time constant. 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 $[n][c/4][h][w][c\%4]$. Deprecated usage: If running on the DLA, this format can be used for acceleration with the caveat that C must be equal or lesser than 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 should be a multiple of 32 on Xavier and 64 on Orin.
kCHW16	Sixteen wide channel vectorized row major format. This format is bound to 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 $[n][c/16][h][w][c\%16]$. For DLA usage, 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	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 $[n][c/32][h][w][c\%32]$. 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].
kDHW8	Eight channel format where C is padded to a multiple of 8. This format is bound to FP16, 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 $[n][d][h][w][c]$.
kCDHW32	Thirty-two wide channel vectorized row major format. This format is bound to FP16 and INT8 and 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, c, d, h, w) mapping to array subscript $[n][c/32][d][h][w][c\%32]$.
kHWC	Non-vectorized channel-last format. This format is bound to either FP32 or UINT8, and is only available for dimensions ≥ 3 .
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 $[N][C][H][\text{roundUp}(W, 64/\text{elementSize})]$ where elementSize is 2 for FP16 and 1 for Int8, with the tensor coordinates (n, c, h, w) mapping to array subscript $[n][c][h][w]$.

Enumerator

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 32 bytes on Xavier and 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 $[N][H][\text{roundUp}(W, 32/C'/\text{elementSize})][C']$ on Xavier and $[N][H][\text{roundUp}(W, 64/C'/\text{elementSize})][C']$ 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	Sixteen channel format where C is padded to a multiple of 16. This format is bound to FP16. 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 $[n][h][w][c]$.
kDHWC	Non-vectorized channel-last format. This format is bound to FP32. It is only available for dimensions ≥ 4 .

8.2.3.44 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.

8.2.3.45 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.

8.2.3.46 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.

8.2.3.47 TripLimit

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

Enum that describes kinds of trip limits.

Enumerator

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

8.2.3.48 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 must have inputs of [DataType](#) kFLOAT, kHALF, kINT8, or kINT32.

Operation kISINF must have inputs of [DataType](#) kFLOAT or kHALF.

All other operations must have inputs of [DataType](#) kFLOAT, kHALF, or kINT8.

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.

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

8.2.4 Function Documentation

8.2.4.1 EnumMax()

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

Maximum number of elements in an enumeration type.

8.2.4.2 EnumMax< BoundingBoxFormat >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< BoundingBoxFormat > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in BoundingBoxFormat enum.

See also

[BoundingBoxFormat](#)

8.2.4.3 EnumMax< BuilderFlag >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< BuilderFlag > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of builder flags in BuilderFlag enum.

See also

[BuilderFlag](#)

8.2.4.4 EnumMax< CalibrationAlgoType >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< CalibrationAlgoType > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in CalibrationAlgoType enum.

See also

[DataType](#)

8.2.4.5 EnumMax< DeviceType >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< DeviceType > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in DeviceType enum.

See also

[DeviceType](#)

8.2.4.6 EnumMax< DimensionOperation >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< DimensionOperation > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in DimensionOperation enum.

See also

[DimensionOperation](#)

8.2.4.7 EnumMax< FillOperation >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< FillOperation > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in FillOperation enum.

See also

[FillOperation](#)

8.2.4.8 EnumMax< GatherMode >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< GatherMode > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in GatherMode enum.

See also

[GatherMode](#)

8.2.4.9 EnumMax< LayerInformationFormat >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< LayerInformationFormat > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of layer information formats in LayerInformationFormat enum.

See also

[LayerInformationFormat](#)

8.2.4.10 EnumMax< LayerType >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< LayerType > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in LayerType enum.

See also

[LayerType](#)

8.2.4.11 EnumMax< LoopOutput >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< LoopOutput > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in LoopOutput enum.

See also

[DataType](#)

8.2.4.12 EnumMax< MatrixOperation >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< MatrixOperation > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in MatrixOperation enum.

See also

[DataType](#)

8.2.4.13 EnumMax< MemoryPoolType >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< MemoryPoolType > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of memory pool types in the MemoryPoolType enum.

See also

[MemoryPoolType](#)

8.2.4.14 EnumMax< NetworkDefinitionCreationFlag >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< NetworkDefinitionCreationFlag > ( ) [inline], [constexpr],
[noexcept]
```

Maximum number of elements in NetworkDefinitionCreationFlag enum.

See also

[NetworkDefinitionCreationFlag](#)

8.2.4.15 EnumMax< OptProfileSelector >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< OptProfileSelector > ( ) [inline], [constexpr], [noexcept]
```

Number of different values of OptProfileSelector enum.

See also

[OptProfileSelector](#)

8.2.4.16 EnumMax< ProfilingVerbosity >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< ProfilingVerbosity > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of profile verbosity levels in ProfilingVerbosity enum.

See also

[ProfilingVerbosity](#)

8.2.4.17 EnumMax< QuantizationFlag >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< QuantizationFlag > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of quantization flags in QuantizationFlag enum.

See also

[QuantizationFlag](#)

8.2.4.18 EnumMax< ReduceOperation >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< ReduceOperation > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in ReduceOperation enum.

See also

[ReduceOperation](#)

8.2.4.19 EnumMax< RNNDirection >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< RNNDirection > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in RNNDirection enum.

See also

[RNNDirection](#)

8.2.4.20 EnumMax< RNNGateType >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< RNNGateType > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in RNNGateType enum.

See also

[RNNGateType](#)

8.2.4.21 EnumMax< RNNInputMode >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< RNNInputMode > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in RNNInputMode enum.

See also

[RNNInputMode](#)

8.2.4.22 EnumMax< RNNOperation >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< RNNOperation > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in RNNOperation enum.

See also

[RNNOperation](#)

8.2.4.23 EnumMax< SampleMode >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< SampleMode > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in SampleMode enum.

See also

[SampleMode](#)

8.2.4.24 EnumMax< ScaleMode >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< ScaleMode > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in ScaleMode enum.

See also

[ScaleMode](#)

8.2.4.25 EnumMax< ScatterMode >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< ScatterMode > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in ScatterMode enum.

See also

[ScatterMode](#)

8.2.4.26 EnumMax< TacticSource >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< TacticSource > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of tactic sources in TacticSource enum.

See also

[TacticSource](#)

8.2.4.27 EnumMax< TempfileControlFlag >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< TempfileControlFlag > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in TempfileControlFlag enum.

See also

[TempfileControlFlag](#)

8.2.4.28 EnumMax< TopKOperation >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< TopKOperation > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in TopKOperation enum.

See also

[TopKOperation](#)

8.2.4.29 EnumMax< TripLimit >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< TripLimit > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in TripLimit enum.

See also

[DataType](#)

8.2.4.30 EnumMax< UnaryOperation >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< UnaryOperation > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in UnaryOperation enum.

See also

[UnaryOperation](#)

8.2.4.31 EnumMax< WeightsRole >()

```
template<>
constexpr int32_t nvinfer1::EnumMax< WeightsRole > ( ) [inline], [constexpr], [noexcept]
```

Maximum number of elements in WeightsRole enum.

See also

[WeightsRole](#)

8.2.4.32 getBuilderPluginRegistry()

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

Return the plugin registry for the given capability or nullptr if no registry exists.

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](#).

8.3 nvinfer1::consistency Namespace Reference

Classes

- class [IConsistencyChecker](#)
Validates a serialized engine blob.
- class [IPluginChecker](#)
Consistency Checker plugin class for user implemented Plugins.

8.4 nvinfer1::impl Namespace Reference

Classes

- struct [EnumMaxImpl](#)
Declaration of [EnumMaxImpl](#) struct to store maximum number of elements in an enumeration type.
- struct [EnumMaxImpl< ActivationType >](#)
- struct [EnumMaxImpl< AllocatorFlag >](#)
Maximum number of elements in AllocatorFlag enum.
- struct [EnumMaxImpl< DataType >](#)
Maximum number of elements in DataType enum.
- struct [EnumMaxImpl< ElementWiseOperation >](#)
- struct [EnumMaxImpl< EngineCapability >](#)
Maximum number of elements in EngineCapability enum.
- struct [EnumMaxImpl< ErrorCode >](#)
Maximum number of elements in ErrorCode enum.
- struct [EnumMaxImpl< HardwareCompatibilityLevel >](#)
- struct [EnumMaxImpl< ILogger::Severity >](#)
Maximum number of elements in [ILogger::Severity](#) enum.
- struct [EnumMaxImpl< InterpolationMode >](#)
- struct [EnumMaxImpl< PaddingMode >](#)
- struct [EnumMaxImpl< PoolingType >](#)
- struct [EnumMaxImpl< PreviewFeature >](#)
- struct [EnumMaxImpl< ResizeCoordinateTransformation >](#)
- struct [EnumMaxImpl< ResizeRoundMode >](#)
- struct [EnumMaxImpl< ResizeSelector >](#)
- struct [EnumMaxImpl< TensorFormat >](#)
Maximum number of elements in TensorFormat enum.
- struct [EnumMaxImpl< TensorIOMode >](#)
Maximum number of elements in TensorIOMode enum.
- struct [EnumMaxImpl< TensorLocation >](#)
Maximum number of elements in TensorLocation enum.

8.5 nvinfer1::plugin Namespace Reference

Classes

- struct [DetectionOutputParameters](#)
The DetectionOutput plugin layer generates the detection output based on location and confidence predictions by doing non maximum suppression. This plugin first decodes the bounding boxes based on the anchors generated. It then performs non_max_suppression on the decoded bounding boxes. [DetectionOutputParameters](#) defines a set of parameters for creating the DetectionOutput plugin layer. It contains:
- 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. It contains:
- struct [NMSPParameters](#)
The [NMSPParameters](#) are used by the BatchedNMSPPlugin for performing the non_max_suppression operation over boxes for object detection networks.
- struct [PriorBoxParameters](#)
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. It contains:
- struct [Quadruple](#)
The Permute plugin layer permutes the input tensor by changing the memory order of the data. [Quadruple](#) defines a structure that contains an array of 4 integers. They can represent the permute orders or the strides in each dimension.
- struct [RegionParameters](#)
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.
- struct [RPROIParams](#)
[RPROIParams](#) is used to create the RPROIPlugin instance. It contains:
- 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.

Enumerations

- enum class [CodeTypeSSD](#) : int32_t { [CORNER](#) = 0 , [CENTER_SIZE](#) = 1 , [CORNER_SIZE](#) = 2 , [TF_CENTER](#) = 3 }
The type of encoding used for decoding the bounding boxes and loc.data.

8.5.1 Enumeration Type Documentation

8.5.1.1 CodeTypeSSD

```
enum class nvinfer1::plugin::CodeTypeSSD : int32_t [strong]
```

The type of encoding used for decoding the bounding boxes and loc.data.

Enumerator

CORNER	Use box corners.
CENTER_SIZE	Use box centers and size.
CORNER_SIZE	Use box centers and size.
TF_CENTER	Use box centers and size but flip x and y coordinates.

8.6 nvinfer1::safe Namespace Reference

The safety subset of TensorRT's API version 1 namespace.

Classes

- struct [FloatingPointErrorInformation](#)
Space to record information about floating point runtime errors.
- class [ICudaEngine](#)
A functionally safe engine for executing inference on a built network.
- class [IExecutionContext](#)
Functionally safe context for executing inference using an engine.
- class [IRuntime](#)
Allows a serialized functionally safe engine to be deserialized.
- 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.

Functions

- [IRuntime](#) * [createInferRuntime](#) ([ILogger](#) &logger) noexcept
Create an instance of an [safe::IRuntime](#) class.
- [nvinfer1::IPluginRegistry](#) * [getSafePluginRegistry](#) () noexcept
Return the safe plugin registry.

8.6.1 Detailed Description

The safety subset of TensorRT's API version 1 namespace.

8.6.2 Function Documentation

8.6.2.1 createInferRuntime()

```
IRuntime * nvinfer1::safe::createInferRuntime (
    ILogger & logger ) [noexcept]
```

Create an instance of an [safe::IRuntime](#) class.

This class is the logging class for the runtime.

Usage considerations

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

8.6.2.2 getSafePluginRegistry()

```
nvinfer1::IPluginRegistry * nvinfer1::safe::getSafePluginRegistry ( ) [noexcept]
```

Return the safe plugin registry.

Usage considerations

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

8.7 nvinfer1::utils Namespace Reference

Functions

- **TRT_DEPRECATED** bool [reshapeWeights](#) ([Weights](#) const &input, int32_t const *shape, int32_t const *shape←
Order, void *data, int32_t nbDims) noexcept
Reformat the input weights of the given shape based on the new order of dimensions.
- **TRT_DEPRECATED** bool [reorderSubBuffers](#) (void *input, int32_t const *order, int32_t num, int32_t size) noexcept
Takes an input stream and re-orders num chunks of the data given the size and order.
- **TRT_DEPRECATED** bool [transposeSubBuffers](#) (void *input, [DataType](#) type, int32_t num, int32_t height, int32_t←
width) noexcept
*Transpose num sub-buffers of height * width.*

8.7.1 Function Documentation

8.7.1.1 reorderSubBuffers()

```
TRT_DEPRECATED bool nvinfer1::utils::reorderSubBuffers (
    void * input,
    int32_t const * order,
    int32_t num,
    int32_t size ) [noexcept]
```

Takes an input stream and re-orders num chunks of the data given the size and order.

Parameters

<i>input</i>	The input data to re-order.
<i>order</i>	The new order of the data sub-buffers.
<i>num</i>	The number of data sub-buffers to re-order.
<i>size</i>	The size of each data sub-buffer in bytes.

In some frameworks, the ordering of the sub-buffers within a dimension is different than the way that TensorRT expects them. TensorRT expects the gate/bias sub-buffers for LSTM's to be in fico order. TensorFlow however formats the sub-buffers in icfo order. This helper function solves this in a generic fashion.

Example usage output of reshapeWeights above: `int32_t indir[1]{1, 0} int32_t stride = W*H; for (int32_t x = 0, y = N*C; x < y; ++x) reorderSubBuffers(out + x * stride, indir, H, W);`

Input Matrix{2, 3, 2, 3}: { 0 2 4}, { 1 3 5} <- {0, 0, *, *} {12 14 16}, {13 15 17} <- {0, 1, *, *} {24 26 28}, {25 27 29} <- {0, 2, *, *} { 6 8 10}, { 7 9 11} <- {1, 0, *, *} {18 20 22}, {19 21 23} <- {1, 1, *, *} {30 32 34}, {31 33 35} <- {1, 2, *, *}

Output Matrix{2, 3, 2, 3}: { 1 3 5}, { 0 2 4} <- {0, 0, *, *} {13 15 17}, {12 14 16} <- {0, 1, *, *} {25 27 29}, {24 26 28} <- {0, 2, *, *} { 7 9 11}, { 6 8 10} <- {1, 0, *, *} {19 21 23}, {18 20 22} <- {1, 1, *, *} {31 33 35}, {30 32 34} <- {1, 2, *, *}

Returns

True on success, false on failure.

See also

[reshapeWeights\(\)](#)

Deprecated Deprecated in TensorRT 8.0.

Warning

This file will be removed in TensorRT 10.0.

8.7.1.2 reshapeWeights()

```
TRT_DEPRECATED bool nvinfer1::utils::reshapeWeights (
    Weights const & input,
    int32_t const * shape,
    int32_t const * shapeOrder,
    void * data,
    int32_t nbDims ) [noexcept]
```

Reformat the input weights of the given shape based on the new order of dimensions.

Parameters

<i>input</i>	The input weights to reshape.
<i>shape</i>	The shape of the weights.
<i>shapeOrder</i>	The order of the dimensions to process for the output.
<i>data</i>	The location where the output data is placed.
<i>nbDims</i>	The number of dimensions to process.

Take the weights specified by *input* with the dimensions specified by *shape* and re-order the weights based on the new dimensions specified by *shapeOrder*. The size of each dimension and the input data is not modified. The output volume pointed to by *data* must be the same as the *input* volume.

Example usage: `float *out = new float[N*C*H*W]; Weights input{DataType::kFLOAT, {0 ... N*C*H*W-1}, N*←C*H*W size}; int32_t order[4]{1, 0, 3, 2}; int32_t shape[4]{C, N, W, H}; reshapeWeights(input, shape, order, out, 4); Weights reshaped{input.type, out, input.count};`

Input Matrix{3, 2, 3, 2}: { 0 1}, { 2 3}, { 4 5} <- {0, 0, *, *} { 6 7}, { 8 9}, {10 11} <- {0, 1, *, *} {12 13}, {14 15}, {16 17} <- {1, 0, *, *} {18 19}, {20 21}, {22 23} <- {1, 1, *, *} {24 25}, {26 27}, {28 29} <- {2, 0, *, *} {30 31}, {32 33}, {34 35} <- {2, 1, *, *}

Output Matrix{2, 3, 2, 3}: { 0 2 4}, { 1 3 5} <- {0, 0, *, *} {12 14 16}, {13 15 17} <- {0, 1, *, *} {24 26 28}, {25 27 29} <- {0, 2, *, *} { 6 8 10}, { 7 9 11} <- {1, 0, *, *} {18 20 22}, {19 21 23} <- {1, 1, *, *} {30 32 34}, {31 33 35} <- {1, 2, *, *}

Returns

True on success, false on failure.

Deprecated Deprecated in TensorRT 8.0.

Warning

This file will be removed in TensorRT 10.0.

8.7.1.3 transposeSubBuffers()

```
TRT_DEPRECATED bool nvinfer1::utils::transposeSubBuffers (
    void * input,
    DataType type,
    int32_t num,
    int32_t height,
    int32_t width ) [noexcept]
```

Transpose num sub-buffers of height * width.

Parameters

<i>input</i>	The input data to transpose.
<i>type</i>	The type of the data to transpose.
<i>num</i>	The number of data sub-buffers to transpose.
<i>height</i>	The size of the height dimension to transpose.
<i>width</i>	The size of the width dimension to transpose.

Returns

True on success, false on failure.

Deprecated Deprecated in TensorRT 8.0.

Warning

This file will be removed in TensorRT 10.0.

8.8 nvonnxparser Namespace Reference

The TensorRT ONNX parser API namespace.

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

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 }
the type of parser error

Functions

- `IOnnxConfig * createONNXConfig ()`
- template<typename T >
`int32_t EnumMax ()`
- template<> `int32_t EnumMax< ErrorCode > ()`

8.8.1 Detailed Description

The TensorRT ONNX parser API namespace.

8.8.2 Enumeration Type Documentation

8.8.2.1 ErrorCode

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

the type of parser error

Enumerator

<code>kSUCCESS</code>	
<code>kINTERNAL_ERROR</code>	
<code>kMEM_ALLOC_FAILED</code>	
<code>kMODEL_DESERIALIZE_FAILED</code>	
<code>kINVALID_VALUE</code>	
<code>kINVALID_GRAPH</code>	
<code>kINVALID_NODE</code>	
<code>kUNSUPPORTED_GRAPH</code>	
<code>kUNSUPPORTED_NODE</code>	

8.8.3 Function Documentation

8.8.3.1 createONNXConfig()

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

8.8.3.2 EnumMax()

```
template<typename T >
int32_t nvonnxparser::EnumMax ( ) [inline]
```

8.8.3.3 EnumMax< ErrorCode >()

```
template<>
int32_t nvonnxparser::EnumMax< ErrorCode > ( ) [inline]
```

8.9 nvuffparser Namespace Reference

The TensorRT UFF parser API namespace.

Classes

- struct [FieldCollection](#)
- class [FieldMap](#)
An array of field params used as a layer parameter for plugin layers.
- class [IUFFParser](#)
Class used for parsing models described using the UFF format.

Enumerations

- enum class [UffInputOrder](#) : int32_t { [kNCHW](#) = 0 , [kNHWC](#) = 1 , [kNC](#) = 2 }
 - enum class [FieldType](#) : int32_t {
[kFLOAT](#) = 0 , [kINT32](#) = 1 , [kCHAR](#) = 2 , [kDIMS](#) = 4 ,
[kDATATYPE](#) = 5 , [kUNKNOWN](#) = 6 }
- The possible field types for custom layer.*

Functions

- [IUffParser](#) * [createUffParser](#) () noexcept
Creates a [IUffParser](#) object.
- void [shutdownProtobufLibrary](#) (void) noexcept
Shuts down protocol buffers library.

8.9.1 Detailed Description

The TensorRT UFF parser API namespace.

8.9.2 Enumeration Type Documentation

8.9.2.1 FieldType

```
enum class nvuffparser::FieldType : int32_t [strong]
```

The possible field types for custom layer.

Enumerator

kFLOAT	FP32 field type.
kINT32	INT32 field type.
kCHAR	char field type. String for length>1.
kDIMS	nvinfer1::Dims field type.
kDATATYPE	nvinfer1::DataType field type.
kUNKNOWN	

8.9.2.2 UffInputOrder

```
enum class nvuffparser::UffInputOrder : int32_t [strong]
```

The different possible supported input order.

Enumerator

kNCHW	NCHW order.
kNHWC	NHWC order.
kNC	NC order.

8.9.3 Function Documentation

8.9.3.1 createUffParser()

```
IUffParser * nvuffparser::createUffParser ( ) [noexcept]
```

Creates a [IUffParser](#) object.

Returns

A pointer to the [IUffParser](#) object is returned.

See also

[nvuffparser::IUffParser](#)

Deprecated [IUffParser](#) will be removed in TensorRT 9.0. Plan to migrate your workflow to use [nvonnxparser::IParser](#) for deployment.

8.9.3.2 shutdownProtobufLibrary()

```
void nvuffparser::shutdownProtobufLibrary (
    void ) [noexcept]
```

Shuts down protocol buffers library.

Note

No part of the protocol buffers library can be used after this function is called.

Chapter 9

Class Documentation

9.1 nvinfer1::plugin::DetectionOutputParameters Struct Reference

The DetectionOutput plugin layer generates the detection output based on location and confidence predictions by doing non maximum suppression. This plugin first decodes the bounding boxes based on the anchors generated. It then performs non_max_suppression on the decoded bounding boxes. [DetectionOutputParameters](#) defines a set of parameters for creating the DetectionOutput plugin layer. It contains:

```
#include <NvInferPluginUtils.h>
```

Public Attributes

- bool [shareLocation](#)
- bool [varianceEncodedInTarget](#)
- int32_t [backgroundLabelId](#)
- int32_t [numClasses](#)
- int32_t [topK](#)
- int32_t [keepTopK](#)
- float [confidenceThreshold](#)
- float [nmsThreshold](#)
- [CodeTypeSSD](#) [codeType](#)
- int32_t [inputOrder](#) [3]
- bool [confSigmoid](#)
- bool [isNormalized](#)
- bool [isBatchAgnostic](#) {true}

9.1.1 Detailed Description

The DetectionOutput plugin layer generates the detection output based on location and confidence predictions by doing non maximum suppression. This plugin first decodes the bounding boxes based on the anchors generated. It then performs non_max_suppression on the decoded bounding boxes. [DetectionOutputParameters](#) defines a set of parameters for creating the DetectionOutput plugin layer. It contains:

Parameters

<i>shareLocation</i>	If true, bounding box are shared among different classes.
<i>varianceEncodedInTarget</i>	If true, variance is encoded in target. Otherwise we need to adjust the predicted offset accordingly.
<i>backgroundLabelId</i>	Background label ID. If there is no background class, set it as -1.
<i>numClasses</i>	Number of classes to be predicted.
<i>topK</i>	Number of boxes per image with top confidence scores that are fed into the NMS algorithm.
<i>keepTopK</i>	Number of total bounding boxes to be kept per image after NMS step.
<i>confidenceThreshold</i>	Only consider detections whose confidences are larger than a threshold.
<i>nmsThreshold</i>	Threshold to be used in NMS.
<i>codeType</i>	Type of coding method for bbox.
<i>inputOrder</i>	Specifies the order of inputs {loc_data, conf_data, priorbox_data}.
<i>confSigmoid</i>	Set to true to calculate sigmoid of confidence scores.
<i>isNormalized</i>	Set to true if bounding box data is normalized by the network.
<i>isBatchAgnostic</i>	Defaults to true. Set to false if prior boxes are unique per batch

9.1.2 Member Data Documentation

9.1.2.1 backgroundLabelId

```
int32_t nvinfer1::plugin::DetectionOutputParameters::backgroundLabelId
```

9.1.2.2 codeType

```
CodeTypeSSD nvinfer1::plugin::DetectionOutputParameters::codeType
```

9.1.2.3 confidenceThreshold

```
float nvinfer1::plugin::DetectionOutputParameters::confidenceThreshold
```

9.1.2.4 confSigmoid

```
bool nvinfer1::plugin::DetectionOutputParameters::confSigmoid
```

9.1.2.5 inputOrder

```
int32_t nvinfer1::plugin::DetectionOutputParameters::inputOrder[3]
```

9.1.2.6 isBatchAgnostic

```
bool nvinfer1::plugin::DetectionOutputParameters::isBatchAgnostic {true}
```

9.1.2.7 isNormalized

```
bool nvinfer1::plugin::DetectionOutputParameters::isNormalized
```

9.1.2.8 keepTopK

```
int32_t nvinfer1::plugin::DetectionOutputParameters::keepTopK
```

9.1.2.9 nmsThreshold

```
float nvinfer1::plugin::DetectionOutputParameters::nmsThreshold
```

9.1.2.10 numClasses

```
int32_t nvinfer1::plugin::DetectionOutputParameters::numClasses
```


9.1.2.11 shareLocation

```
bool nvinfer1::plugin::DetectionOutputParameters::shareLocation
```

9.1.2.12 topK

```
int32_t nvinfer1::plugin::DetectionOutputParameters::topK
```

9.1.2.13 varianceEncodedInTarget

```
bool nvinfer1::plugin::DetectionOutputParameters::varianceEncodedInTarget
```

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

- [NvInferPluginUtils.h](#)

9.2 Dims Class Reference

Structure to define the dimensions of a tensor.

```
#include <NvInferRuntimeCommon.h>
```

9.2.1 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 d.

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

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

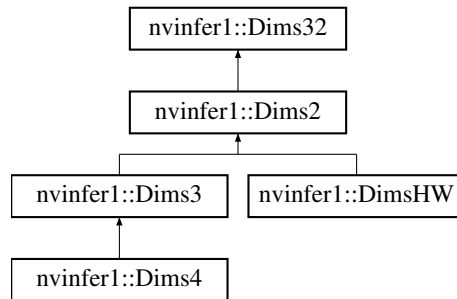
- [NvInferRuntimeCommon.h](#)

9.3 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](#) (int32_t d0, int32_t d1)
Construct a [Dims2](#) from 2 elements.

Additional Inherited Members

9.3.1 Detailed Description

Descriptor for two-dimensional data.

9.3.2 Constructor & Destructor Documentation

9.3.2.1 Dims2() [1/2]

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

Construct an empty [Dims2](#) object.

9.3.2.2 Dims2() [2/2]

```
nvinfer1::Dims2::Dims2 (
    int32_t d0,
    int32_t d1 ) [inline]
```

Construct a [Dims2](#) from 2 elements.

Parameters

<i>d0</i>	The first element.
<i>d1</i>	The second element.

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

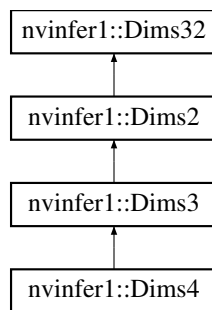
- [NvInferLegacyDims.h](#)

9.4 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](#) (int32_t d0, int32_t d1, int32_t d2)
Construct a [Dims3](#) from 3 elements.

Additional Inherited Members

9.4.1 Detailed Description

Descriptor for three-dimensional data.

9.4.2 Constructor & Destructor Documentation

9.4.2.1 Dims3() [1/2]

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

Construct an empty [Dims3](#) object.

9.4.2.2 Dims3() [2/2]

```
nvinfer1::Dims3::Dims3 (
    int32_t d0,
    int32_t d1,
    int32_t d2 ) [inline]
```

Construct a [Dims3](#) from 3 elements.

Parameters

<i>d0</i>	The first element.
<i>d1</i>	The second element.
<i>d2</i>	The third element.

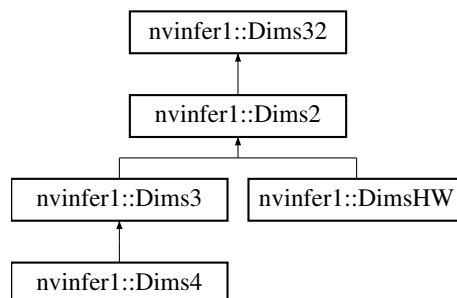
The documentation for this class was generated from the following file:

- [NvInferLegacyDims.h](#)

9.5 nvinfer1::Dims32 Class Reference

```
#include <NvInferRuntimeCommon.h>
```

Inheritance diagram for nvinfer1::Dims32:



Public Attributes

- `int32_t nbDims`
The rank (number of dimensions).
- `int32_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

```
int32_t nvinfer1::Dims32::d[MAX_DIMS]
```

The extent of each dimension.

9.5.1.2 MAX_DIMS

```
constexpr int32_t nvinfer1::Dims32::MAX_DIMS {8} [static], [constexpr]
```

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

9.5.1.3 nbDims

```
int32_t nvinfer1::Dims32::nbDims
```

The rank (number of dimensions).

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

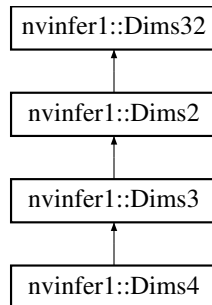
- [NvInferRuntimeCommon.h](#)

9.6 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\(int32_t d0, int32_t d1, int32_t d2, int32_t d3\)](#)
Construct a [Dims4](#) from 4 elements.

Additional Inherited Members

9.6.1 Detailed Description

Descriptor for four-dimensional data.

9.6.2 Constructor & Destructor Documentation

9.6.2.1 Dims4() [1/2]

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

Construct an empty [Dims4](#) object.

9.6.2.2 Dims4() [2/2]

```
nvinfer1::Dims4::Dims4 (
    int32_t d0,
    int32_t d1,
    int32_t d2,
    int32_t d3 ) [inline]
```

Construct a [Dims4](#) from 4 elements.

Parameters

<i>d0</i>	The first element.
<i>d1</i>	The second element.
<i>d2</i>	The third element.
<i>d3</i>	The fourth element.

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

- [NvInferLegacyDims.h](#)

9.7 nvinfer1::DimsExprs Class Reference

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

Public Attributes

- `int32_t` [nbDims](#)
The number of dimensions.
- [IDimensionExpr](#) `const * d` [[Dims::MAX_DIMS](#)]
The extent of each dimension.

9.7.1 Detailed Description

Analog of class [Dims](#) with expressions instead of constants for the dimensions.

9.7.2 Member Data Documentation

9.7.2.1 d

```
IDimensionExpr const* nvinfer1::DimsExprs::d[Dims::MAX\_DIMS]
```

The extent of each dimension.

9.7.2.2 nbDims

```
int32_t nvinfer1::DimsExprs::nbDims
```

The number of dimensions.

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

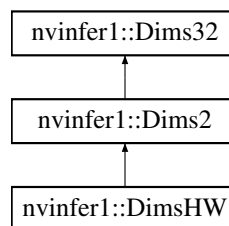
- [NvInferRuntime.h](#)

9.8 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](#) (int32_t height, int32_t width)
Construct a [DimsHW](#) given height and width.
- int32_t & [h](#) ()
Get the height.
- int32_t [h](#) () const
Get the height.
- int32_t & [w](#) ()
Get the width.
- int32_t [w](#) () const
Get the width.

Additional Inherited Members

9.8.1 Detailed Description

Descriptor for two-dimensional spatial data.

9.8.2 Constructor & Destructor Documentation

9.8.2.1 DimsHW() [1/2]

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

Construct an empty [DimsHW](#) object.

9.8.2.2 DimsHW() [2/2]

```
nvinfer1::DimsHW::DimsHW (
    int32_t height,
    int32_t width ) [inline]
```

Construct a [DimsHW](#) given height and width.

Parameters

<i>height</i>	the height of the data
<i>width</i>	the width of the data

9.8.3 Member Function Documentation

9.8.3.1 h() [1/2]

```
int32_t & nvinfer1::DimsHW::h ( ) [inline]
```

Get the height.

Returns

The height.

9.8.3.2 h() [2/2]

```
int32_t nvinfer1::DimsHW::h ( ) const [inline]
```

Get the height.

Returns

The height.

9.8.3.3 w() [1/2]

```
int32_t & nvinfer1::DimsHW::w ( ) [inline]
```

Get the width.

Returns

The width.

9.8.3.4 w() [2/2]

```
int32_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](#)

9.9 nvinfer1::DynamicPluginTensorDesc Class Reference

```
#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.

9.9.1 Detailed Description

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

9.9.2 Member Data Documentation

9.9.2.1 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.

9.9.2.2 max

`Dims nvinfer1::DynamicPluginTensorDesc::max`

Upper bounds on tensor's dimensions.

9.9.2.3 min

`Dims nvinfer1::DynamicPluginTensorDesc::min`

Lower bounds on tensor's dimensions.

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

- [NvInferRuntime.h](#)

9.10 nvinfer1::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 nvinfer1::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:

- [NvInferRuntimeCommon.h](#)

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

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

Static Public Attributes

- static constexpr int32_t [kVALUE](#) = 12

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 = 12 [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 <NvInferRuntimeCommon.h>
```

Static Public Attributes

- static constexpr int32_t `kVALUE` = 1
maximum number of elements in 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]
```

maximum number of elements in AllocatorFlag enum

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

- [NvInferRuntimeCommon.h](#)

9.13 `nvinfer1::impl::EnumMaxImpl< DataType >` Struct Reference

Maximum number of elements in DataType enum.

```
#include <NvInferRuntimeCommon.h>
```

Static Public Attributes

- static constexpr int32_t `kVALUE` = 7

9.13.1 Detailed Description

Maximum number of elements in `DataType` enum.

See also

[DataType](#)

9.13.2 Member Data Documentation

9.13.2.1 `kVALUE`

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< DataType >::kVALUE = 7 [static], [constexpr]
```

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

- [NvInferRuntimeCommon.h](#)

9.14 `nvinfer1::impl::EnumMaxImpl< ElementWiseOperation >` Struct Reference

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

Static Public Attributes

- static constexpr int32_t `kVALUE` = 14

9.14.1 Detailed Description

Maximum number of elements in `ElementWiseOperation` enum.

See also

[ElementWiseOperation](#)

9.14.2 Member Data Documentation

9.14.2.1 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](#)

9.15 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

9.15.1 Detailed Description

Maximum number of elements in EngineCapability enum.

See also

[EngineCapability](#)

9.15.2 Member Data Documentation

9.15.2.1 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](#)

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

Maximum number of elements in ErrorCode enum.

```
#include <NvInferRuntimeCommon.h>
```

Static Public Attributes

- static constexpr int32_t [kVALUE](#) = 11

Declaration of kVALUE.

9.16.1 Detailed Description

Maximum number of elements in ErrorCode enum.

See also

[ErrorCode](#)

9.16.2 Member Data Documentation

9.16.2.1 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:

- [NvInferRuntimeCommon.h](#)

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

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

Static Public Attributes

- static constexpr int32_t [kVALUE](#) = 2

9.17.1 Detailed Description

Maximum number of elements in `HardwareCompatibilityLevel` enum.

See also

[HardwareCompatibilityLevel](#)

9.17.2 Member Data Documentation

9.17.2.1 kVALUE

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

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

- [NvInfer.h](#)

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

Maximum number of elements in [ILogger::Severity](#) enum.

```
#include <NvInferRuntimeCommon.h>
```

Static Public Attributes

- static constexpr int32_t [kVALUE](#) = 5
Declaration of kVALUE that represents maximum number of elements in [ILogger::Severity](#) enum.

9.18.1 Detailed Description

Maximum number of elements in [ILogger::Severity](#) enum.

See also

[ILogger::Severity](#)

9.18.2 Member Data Documentation

9.18.2.1 kVALUE

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

Declaration of kVALUE that represents maximum number of elements in [ILogger::Severity](#) enum.

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

- [NvInferRuntimeCommon.h](#)

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

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

Static Public Attributes

- static constexpr int32_t [kVALUE](#) = 3

9.19.1 Detailed Description

Maximum number of elements in InterpolationMode enum.

See also

[InterpolationMode](#)

9.19.2 Member Data Documentation

9.19.2.1 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](#)

9.20 `nvinfer1::impl::EnumMaxImpl< PaddingMode >` Struct Reference

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

Static Public Attributes

- static constexpr int32_t `kVALUE` = 6

9.20.1 Detailed Description

Maximum number of elements in `PaddingMode` enum.

See also

[PaddingMode](#)

9.20.2 Member Data Documentation

9.20.2.1 `kVALUE`

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< PaddingMode >::kVALUE = 6 [static], [constexpr]
```

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

- [NvInfer.h](#)

9.21 `nvinfer1::impl::EnumMaxImpl< PoolingType >` Struct Reference

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

Static Public Attributes

- static constexpr int32_t `kVALUE` = 3

9.21.1 Detailed Description

Maximum number of elements in `PoolingType` enum.

See also

[PoolingType](#)

9.21.2 Member Data Documentation

9.21.2.1 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](#)

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

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

Static Public Attributes

- static constexpr int32_t [kVALUE](#) = 4

9.22.1 Detailed Description

Maximum number of elements in PreviewFeature enum.

See also

[PreviewFeature](#)

9.22.2 Member Data Documentation

9.22.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< PreviewFeature >::kVALUE = 4 [static], [constexpr]
```

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

- [NvInfer.h](#)

9.23 `nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation >` Struct Reference

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

Static Public Attributes

- static constexpr int32_t `kVALUE` = 3

9.23.1 Detailed Description

Maximum number of elements in `ResizeCoordinateTransformation` enum.

See also

[ResizeCoordinateTransformation](#)

9.23.2 Member Data Documentation

9.23.2.1 `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](#)

9.24 `nvinfer1::impl::EnumMaxImpl< ResizeRoundMode >` Struct Reference

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

Static Public Attributes

- static constexpr int32_t `kVALUE` = 4

9.24.1 Detailed Description

Maximum number of elements in ResizeRoundMode enum.

See also

[ResizeRoundMode](#)

9.24.2 Member Data Documentation

9.24.2.1 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](#)

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

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

Static Public Attributes

- static constexpr int32_t [kVALUE](#) = 2

9.25.1 Detailed Description

Maximum number of elements in ResizeSelector enum.

See also

[ResizeSelector](#)

9.25.2 Member Data Documentation

9.25.2.1 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](#)

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

Maximum number of elements in TensorFormat enum.

```
#include <NvInferRuntimeCommon.h>
```

Static Public Attributes

- static constexpr int32_t [kVALUE](#) = 13
Declaration of kVALUE that represents maximum number of elements in TensorFormat enum.

9.26.1 Detailed Description

Maximum number of elements in TensorFormat enum.

See also

[TensorFormat](#)

9.26.2 Member Data Documentation

9.26.2.1 kVALUE

```
constexpr int32_t nvinfer1::impl::EnumMaxImpl< TensorFormat >::kVALUE = 13 [static], [constexpr]
```

Declaration of kVALUE that represents maximum number of elements in TensorFormat enum.

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

- [NvInferRuntimeCommon.h](#)

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

Maximum number of elements in TensorIOMode enum.

```
#include <NvInferRuntimeCommon.h>
```

Static Public Attributes

- static constexpr int32_t [kVALUE](#) = 3

9.27.1 Detailed Description

Maximum number of elements in TensorIOMode enum.

See also

[TensorIOMode](#)

9.27.2 Member Data Documentation

9.27.2.1 kVALUE

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

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

- [NvInferRuntimeCommon.h](#)

9.28 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

9.28.1 Detailed Description

Maximum number of elements in TensorLocation enum.

See also

[TensorLocation](#)

9.28.2 Member Data Documentation

9.28.2.1 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](#)

9.29 nvuffparser::FieldCollection Struct Reference

```
#include <NvUffParser.h>
```

Public Attributes

- int32_t [nbFields](#)
- [FieldMap](#) const * [fields](#)

9.29.1 Member Data Documentation

9.29.1.1 fields

```
FieldMap const* nvuffparser::FieldCollection::fields
```

9.29.1.2 nbFields

```
int32_t nvuffparser::FieldCollection::nbFields
```

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

- [NvUffParser.h](#)

9.30 nvuffparser::FieldMap Class Reference

An array of field params used as a layer parameter for plugin layers.

```
#include <NvUffParser.h>
```

Public Member Functions

- [FieldMap](#) (char const *[name](#), void const *[data](#), const [FieldType](#) [type](#), int32_t [length](#)=1)

Public Attributes

- char const * [name](#)
- void const * [data](#)
- [FieldType](#) [type](#) = [FieldType::kUNKNOWN](#)
- int32_t [length](#) = 1

9.30.1 Detailed Description

An array of field params used as a layer parameter for plugin layers.

The node fields are passed by the parser to the API through the plugin constructor. The implementation of the plugin should parse the contents of the fieldMap as part of the plugin constructor

9.30.2 Constructor & Destructor Documentation

9.30.2.1 FieldMap()

```
nvuffparser::FieldMap::FieldMap (
    char const * name,
    void const * data,
    const FieldType type,
    int32_t length = 1 )
```

9.30.3 Member Data Documentation

9.30.3.1 data

```
void const* nvuffparser::FieldMap::data
```

9.30.3.2 length

```
int32_t nvuffparser::FieldMap::length = 1
```

9.30.3.3 name

```
char const* nvuffparser::FieldMap::name
```

9.30.3.4 type

```
FieldType nvuffparser::FieldMap::type = FieldType::kUNKNOWN
```

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

- [NvUffParser.h](#)

9.31 nvinfer1::safe::FloatingPointErrorInformation Struct Reference

Space to record information about floating point runtime errors.

```
#include <NvInferSafeRuntime.h>
```

Public Attributes

- int32_t [nbNanErrors](#)
Total count of errors relating to NAN values (0 if none)
- int32_t [nbInfErrors](#)
Total count of errors relating to INF values (0 if none)

9.31.1 Detailed Description

Space to record information about floating point runtime errors.

NAN errors occur when NAN values are stored in an INT8 quantized datatype. INF errors occur when +-INF values are stored in an INT8 quantized datatype.

9.31.2 Member Data Documentation

9.31.2.1 nbInfErrors

```
int32_t nvinfer1::safe::FloatingPointErrorInformation::nbInfErrors
```

Total count of errors relating to INF values (0 if none)

9.31.2.2 nbNanErrors

```
int32_t nvinfer1::safe::FloatingPointErrorInformation::nbNanErrors
```

Total count of errors relating to NAN values (0 if none)

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

- [NvInferSafeRuntime.h](#)

9.32 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. It contains:

```
#include <NvInferPluginUtils.h>
```

Public Attributes

- float [minSize](#)
- float [maxSize](#)
- float * [aspectRatios](#)
- int32_t [numAspectRatios](#)
- int32_t [H](#)
- int32_t [W](#)
- float [variance](#) [4]

9.32.1 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. It contains:

Parameters

<i>minScale</i>	Scale of anchors corresponding to finest resolution.
<i>maxScale</i>	Scale of anchors corresponding to coarsest resolution.
<i>aspectRatios</i>	List of aspect ratios to place on each grid point.
<i>numAspectRatios</i>	Number of elements in aspectRatios.
<i>H</i>	Height of feature map to generate anchors for.
<i>W</i>	Width of feature map to generate anchors for.
<i>variance</i>	Variance for adjusting the prior boxes.

9.32.2 Member Data Documentation

9.32.2.1 aspectRatios

```
float* nvinfer1::plugin::GridAnchorParameters::aspectRatios
```

9.32.2.2 H

```
int32_t nvinfer1::plugin::GridAnchorParameters::H
```

9.32.2.3 maxSize

```
float nvinfer1::plugin::GridAnchorParameters::maxSize
```

9.32.2.4 minSize

```
float nvinfer1::plugin::GridAnchorParameters::minSize
```

9.32.2.5 numAspectRatios

```
int32_t nvinfer1::plugin::GridAnchorParameters::numAspectRatios
```

9.32.2.6 variance

```
float nvinfer1::plugin::GridAnchorParameters::variance[4]
```

9.32.2.7 W

```
int32_t nvinfer1::plugin::GridAnchorParameters::W
```

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

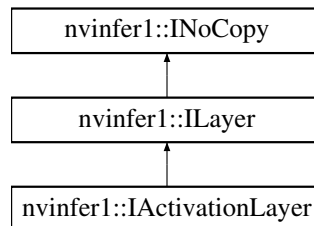
- [NvInferPluginUtils.h](#)

9.33 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=default

Protected Attributes

- apiv::VActivationLayer * [mImpl](#)

9.33.1 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.

9.33.2 Constructor & Destructor Documentation

9.33.2.1 ~IActivationLayer()

```
virtual nvinfer1::IActivationLayer::~~IActivationLayer ( ) [protected], [virtual], [default],
[noexcept]
```

9.33.3 Member Function Documentation

9.33.3.1 getActivationType()

```
ActivationType nvinfer1::IActivationLayer::getActivationType ( ) const [inline], [noexcept]
```

Get the type of activation to be performed.

See also

[setActivationType\(\)](#), [ActivationType](#)

9.33.3.2 getAlpha()

```
float nvinfer1::IActivationLayer::getAlpha ( ) const [inline], [noexcept]
```

Get the alpha parameter.

See also

[getBeta\(\)](#), [setAlpha\(\)](#)

9.33.3.3 getBeta()

```
float nvinfer1::IActivationLayer::getBeta ( ) const [inline], [noexcept]
```

Get the beta parameter.

See also

[getAlpha\(\)](#), [setBeta\(\)](#)

9.33.3.4 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](#)

9.33.3.5 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\(\)](#)

9.33.3.6 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\(\)](#)

9.33.4 Member Data Documentation

9.33.4.1 mImpl

```
apiv::VActivationLayer* nvinfer1::IActivationLayer::mImpl [protected]
```

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

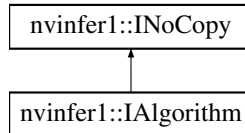
- [NvInfer.h](#)

9.34 nvinfer1::IAlgorithm Class Reference

Describes a variation of execution of a layer. An algorithm is represented by [IAlgorithmVariant](#) and the [IAlgorithmIOInfo](#) for each of its inputs and outputs. An algorithm can be selected or reproduced using `AlgorithmSelector::selectAlgorithms()`.”.

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

Inheritance diagram for nvinfer1::IAlgorithm:



Public Member Functions

- `TRT_DEPRECATED IAlgorithmIOInfo const & getAlgorithmIOInfo (int32_t index) const noexcept`
Returns the format of an Algorithm input or output. Algorithm inputs are incrementally numbered first, followed by algorithm outputs.
- `IAlgorithmVariant const & getAlgorithmVariant () const noexcept`
Returns the algorithm variant.
- `float getTimingMSec () const noexcept`
The time in milliseconds to execute the algorithm.
- `std::size_t getWorkspaceSize () const noexcept`
The size of the GPU temporary memory in bytes which the algorithm uses at execution time.
- `IAlgorithmIOInfo const * getAlgorithmIOInfoByIndex (int32_t index) const noexcept`
Returns the format of an Algorithm input or output. Algorithm inputs are incrementally numbered first, followed by algorithm outputs.

Protected Member Functions

- `virtual ~IAlgorithm () noexcept=default`

Protected Attributes

- `apiv::VAlgorithm * mImpl`

9.34.1 Detailed Description

Describes a variation of execution of a layer. An algorithm is represented by [IAlgorithmVariant](#) and the [IAlgorithmIOInfo](#) for each of its inputs and outputs. An algorithm can be selected or reproduced using `AlgorithmSelector::selectAlgorithms()`.”.

See also

[IAlgorithmIOInfo](#), [IAlgorithmVariant](#), [IAlgorithmSelector::selectAlgorithms\(\)](#)

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.34.2 Constructor & Destructor Documentation

9.34.2.1 ~IAlgorithm()

```
virtual nvinfer1::IAlgorithm::~IAlgorithm ( ) [protected], [virtual], [default], [noexcept]
```

9.34.3 Member Function Documentation

9.34.3.1 getAlgorithmIOInfo()

```
TRT_DEPRECATED IAlgorithmIOInfo const & nvinfer1::IAlgorithm::getAlgorithmIOInfo (
    int32_t index ) const [inline], [noexcept]
```

Returns the format of an Algorithm input or output. Algorithm inputs are incrementally numbered first, followed by algorithm outputs.

Parameters

<i>index</i>	Index of the input or output of the algorithm. Incremental numbers assigned to indices of inputs and the outputs.
--------------	---

Returns

a reference to [IAlgorithmIOInfo](#) specified by index or the first algorithm if index is out of range.

Deprecated Deprecated in TensorRT 8.0. Superseded by [IAlgorithm::getAlgorithmIOInfoByIndex\(\)](#).

9.34.3.2 getAlgorithmIOInfoByIndex()

```
IAlgorithmIOInfo const * nvinfer1::IAlgorithm::getAlgorithmIOInfoByIndex (
    int32_t index ) const [inline], [noexcept]
```

Returns the format of an Algorithm input or output. Algorithm inputs are incrementally numbered first, followed by algorithm outputs.

Parameters

<i>index</i>	Index of the input or output of the algorithm. Incremental numbers assigned to indices of inputs and the outputs.
--------------	---

Returns

a pointer to a [IAlgorithmIOInfo](#) interface or nullptr if index is out of range.

9.34.3.3 getAlgorithmVariant()

```
IAlgorithmVariant const & nvinfer1::IAlgorithm::getAlgorithmVariant ( ) const [inline], [noexcept]
```

Returns the algorithm variant.

9.34.3.4 getTimingMSec()

```
float nvinfer1::IAlgorithm::getTimingMSec ( ) const [inline], [noexcept]
```

The time in milliseconds to execute the algorithm.

9.34.3.5 getWorkspaceSize()

```
std::size_t nvinfer1::IAlgorithm::getWorkspaceSize ( ) const [inline], [noexcept]
```

The size of the GPU temporary memory in bytes which the algorithm uses at execution time.

9.34.4 Member Data Documentation**9.34.4.1 mImpl**

```
apiv::VAlgorithm* nvinfer1::IAlgorithm::mImpl [protected]
```

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

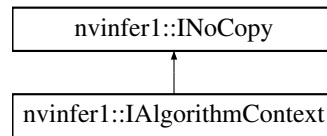
- [NvInfer.h](#)

9.35 nvinfer1::IAlgorithmContext Class Reference

Describes the context and requirements, that could be fulfilled by one or more instances of [IAlgorithm](#).

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

Inheritance diagram for nvinfer1::IAlgorithmContext:



Public Member Functions

- `char const * getName () const noexcept`
Return name of the algorithm node. This is a unique identifier for the [IAlgorithmContext](#).
- `Dims getDimensions (int32_t index, OptProfileSelector select) const noexcept`
Get the minimum / optimum / maximum dimensions for input or output tensor.
- `int32_t getNbInputs () const noexcept`
Return number of inputs of the algorithm.
- `int32_t getNbOutputs () const noexcept`
Return number of outputs of the algorithm.

Protected Member Functions

- `virtual ~IAlgorithmContext () noexcept=default`

Protected Attributes

- `apiv::VAlgorithmContext * mImpl`

9.35.1 Detailed Description

Describes the context and requirements, that could be fulfilled by one or more instances of [IAlgorithm](#).

See also

[IAlgorithm](#)

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.35.2 Constructor & Destructor Documentation

9.35.2.1 ~IAlgorithmContext()

```
virtual nvinfer1::IAlgorithmContext::~~IAlgorithmContext ( ) [protected], [virtual], [default],
[noexcept]
```

9.35.3 Member Function Documentation

9.35.3.1 getDimensions()

```
Dims nvinfer1::IAlgorithmContext::getDimensions (
    int32_t index,
    OptProfileSelector select ) const [inline], [noexcept]
```

Get the minimum / optimum / maximum dimensions for input or output tensor.

Parameters

<i>index</i>	Index of the input or output of the algorithm. Incremental numbers assigned to indices of inputs and the outputs.
<i>select</i>	Which of the minimum, optimum, or maximum dimensions to be queried.

9.35.3.2 getName()

```
char const * nvinfer1::IAlgorithmContext::getName ( ) const [inline], [noexcept]
```

Return name of the algorithm node. This is a unique identifier for the [IAlgorithmContext](#).

9.35.3.3 getNbInputs()

```
int32_t nvinfer1::IAlgorithmContext::getNbInputs ( ) const [inline], [noexcept]
```

Return number of inputs of the algorithm.

9.35.3.4 getNbOutputs()

```
int32_t nvinfer1::IAlgorithmContext::getNbOutputs ( ) const [inline], [noexcept]
```

Return number of outputs of the algorithm.

9.35.4 Member Data Documentation

9.35.4.1 mImpl

```
apiv::VAlgorithmContext* nvinfer1::IAlgorithmContext::mImpl [protected]
```

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

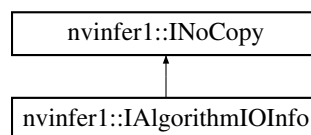
- [NvInfer.h](#)

9.36 nvinfer1::IAlgorithmIOInfo Class Reference

Carries information about input or output of the algorithm. [IAlgorithmIOInfo](#) for all the input and output along with [IAlgorithmVariant](#) denotes the variation of algorithm and can be used to select or reproduce an algorithm using [IAlgorithmSelector::selectAlgorithms\(\)](#).

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

Inheritance diagram for nvinfer1::IAlgorithmIOInfo:



Public Member Functions

- [TensorFormat](#) [getTensorFormat](#) () const noexcept
Return TensorFormat of the input/output of algorithm.
- [DataType](#) [getDataType](#) () const noexcept
Return DataType of the input/output of algorithm.
- [Dims](#) [getStrides](#) () const noexcept
Return strides of the input/output tensor of algorithm.

Protected Member Functions

- virtual [~IAlgorithmIOInfo](#) () noexcept=default

Protected Attributes

- apiv::VAlgorithmIOInfo * [mImpl](#)

9.36.1 Detailed Description

Carries information about input or output of the algorithm. [IAlgorithmIOInfo](#) for all the input and output along with [IAlgorithmVariant](#) denotes the variation of algorithm and can be used to select or reproduce an algorithm using [IAlgorithmSelector::selectAlgorithms\(\)](#).

See also

[IAlgorithmVariant](#), [IAlgorithm](#), [IAlgorithmSelector::selectAlgorithms\(\)](#)

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.36.2 Constructor & Destructor Documentation

9.36.2.1 ~IAlgorithmIOInfo()

```
virtual nvinfer1::IAlgorithmIOInfo::~~IAlgorithmIOInfo ( ) [protected], [virtual], [default],
[noexcept]
```

9.36.3 Member Function Documentation

9.36.3.1 getDataTypes()

```
DataType nvinfer1::IAlgorithmIOInfo::getDataTypes ( ) const [inline], [noexcept]
```

Return DataTypes of the input/output of algorithm.

9.36.3.2 getStrides()

```
Dims nvinfer1::IAlgorithmIOInfo::getStrides ( ) const [inline], [noexcept]
```

Return strides of the input/output tensor of algorithm.

9.36.3.3 getTensorFormat()

```
TensorFormat nvinfer1::IAlgorithmIOInfo::getTensorFormat ( ) const [inline], [noexcept]
```

Return TensorFormat of the input/output of algorithm.

9.36.4 Member Data Documentation

9.36.4.1 mImpl

```
apiv::VAlgorithmIOInfo* nvinfer1::IAlgorithmIOInfo::mImpl [protected]
```

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

- [NvInfer.h](#)

9.37 nvinfer1::IAlgorithmSelector Class Reference

Interface implemented by application for selecting and reporting algorithms of a layer provided by the builder.

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

Public Member Functions

- virtual int32_t [selectAlgorithms](#) ([IAlgorithmContext](#) const &context, [IAlgorithm](#) const *const *choices, int32_t nbChoices, int32_t *selection) noexcept=0
Select Algorithms for a layer from the given list of algorithm choices.
- virtual void [reportAlgorithms](#) ([IAlgorithmContext](#) const *const *algoContexts, [IAlgorithm](#) const *const *algo↵ Choices, int32_t nbAlgorithms) noexcept=0
Called by TensorRT to report choices it made.
- virtual [~IAlgorithmSelector](#) () noexcept=default

9.37.1 Detailed Description

Interface implemented by application for selecting and reporting algorithms of a layer provided by the builder.

Note

A layer in context of algorithm selection may be different from [ILayer](#) in [INetworkDefinition](#). For example, an algorithm might be implementing a conglomeration of multiple [ILayers](#) in [INetworkDefinition](#).

9.37.2 Constructor & Destructor Documentation

9.37.2.1 ~IAlgorithmSelector()

```
virtual nvinfer1::IAlgorithmSelector::~~IAlgorithmSelector ( ) [virtual], [default], [noexcept]
```

9.37.3 Member Function Documentation

9.37.3.1 reportAlgorithms()

```
virtual void nvinfer1::IAlgorithmSelector::reportAlgorithms (
    IAlgorithmContext const *const * algoContexts,
    IAlgorithm const *const * algoChoices,
    int32_t nbAlgorithms ) [pure virtual], [noexcept]
```

Called by TensorRT to report choices it made.

Note

For a given optimization profile, this call comes after all calls to `selectAlgorithms`. `algoChoices[i]` is the choice that TensorRT made for `algoContexts[i]`, for `i` in `[0, nbAlgorithms-1]`

Parameters

<i>algoContexts</i>	The list of all algorithm contexts.
<i>algoChoices</i>	The list of algorithm choices made by TensorRT
<i>nbAlgorithms</i>	The size of <code>algoContexts</code> as well as <code>algoChoices</code> .

9.37.3.2 selectAlgorithms()

```
virtual int32_t nvinfer1::IAlgorithmSelector::selectAlgorithms (
    IAlgorithmContext const & context,
    IAlgorithm const *const * choices,
    int32_t nbChoices,
    int32_t * selection ) [pure virtual], [noexcept]
```

Select Algorithms for a layer from the given list of algorithm choices.

Returns

The number of choices selected from [0, nbChoices-1].

Parameters

<i>context</i>	The context for which the algorithm choices are valid.
<i>choices</i>	The list of algorithm choices to select for implementation of this layer.
<i>nbChoices</i>	Number of algorithm choices.
<i>selection</i>	The user writes indices of selected choices in to selection buffer which is of size nbChoices.

Note

TensorRT uses its default algorithm selection to choose from the list provided. If return value is 0, TensorRT's default algorithm selection is used unless [BuilderFlag::kREJECT_EMPTY_ALGORITHMS](#) (or the deprecated [BuilderFlag::kSTRICT_TYPES](#)) is set. The list of choices is valid only for this specific algorithm context.

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

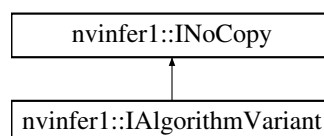
- [NvInfer.h](#)

9.38 nvinfer1::IAlgorithmVariant Class Reference

provides a unique 128-bit identifier, which along with the input and output information denotes the variation of algorithm and can be used to select or reproduce an algorithm, using [IAlgorithmSelector::selectAlgorithms\(\)](#)

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

Inheritance diagram for nvinfer1::IAlgorithmVariant:



Public Member Functions

- `int64_t getImplementation ()` const noexcept
Return implementation of the algorithm.
- `int64_t getTactic ()` const noexcept
Return tactic of the algorithm.

Protected Member Functions

- virtual `~IAlgorithmVariant ()` noexcept=default

Protected Attributes

- `apiv::VAlgorithmVariant * mImpl`

9.38.1 Detailed Description

provides a unique 128-bit identifier, which along with the input and output information denotes the variation of algorithm and can be used to select or reproduce an algorithm, using [IAlgorithmSelector::selectAlgorithms\(\)](#)

See also

[IAlgorithmIOInfo](#), [IAlgorithm](#), [IAlgorithmSelector::selectAlgorithms\(\)](#)

Note

A single implementation can have multiple tactics.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.38.2 Constructor & Destructor Documentation

9.38.2.1 ~IAlgorithmVariant()

```
virtual nvinfer1::IAlgorithmVariant::~~IAlgorithmVariant ( ) [protected], [virtual], [default],
[noexcept]
```

9.38.3 Member Function Documentation

9.38.3.1 `getImplementation()`

```
int64_t nvinfer1::IAlgorithmVariant::getImplementation ( ) const [inline], [noexcept]
```

Return implementation of the algorithm.

9.38.3.2 `getTactic()`

```
int64_t nvinfer1::IAlgorithmVariant::getTactic ( ) const [inline], [noexcept]
```

Return tactic of the algorithm.

9.38.4 Member Data Documentation

9.38.4.1 `mImpl`

```
apiv::VAlgorithmVariant* nvinfer1::IAlgorithmVariant::mImpl [protected]
```

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

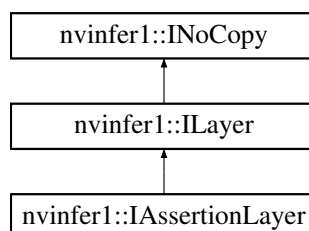
- [NvInfer.h](#)

9.39 `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=default

Protected Attributes

- apiv::VAssertionLayer * [mImpl](#)

9.39.1 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.

9.39.2 Constructor & Destructor Documentation

9.39.2.1 ~IAssertionLayer()

```
virtual nvinfer1::IAssertionLayer::~~IAssertionLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.39.3 Member Function Documentation

9.39.3.1 getMessage()

```
char const * nvinfer1::IAssertionLayer::getMessage ( ) const [inline], [noexcept]
```

Return the assertion message.

See also

[setMessage\(\)](#)

9.39.3.2 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\(\)](#)

9.39.4 Member Data Documentation

9.39.4.1 mImpl

```
apiv::VAssertionLayer* nvinfer1::IAssertionLayer::mImpl [protected]
```

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

- [NvInfer.h](#)

9.40 nvcaffeparser1:IBinaryProtoBlob Class Reference

Object used to store and query data extracted from a binaryproto file using the [ICaffeParser](#).

```
#include <NvCaffeParser.h>
```

Public Member Functions

- virtual void const * [getData](#) () noexcept=0
- virtual [nvinfer1::Dims4](#) [getDimensions](#) () noexcept=0
- virtual [nvinfer1::DataType](#) [getDataType](#) () noexcept=0
- virtual [TRT_DEPRECATED](#) void [destroy](#) () noexcept=0
- virtual [~IBinaryProtoBlob](#) () noexcept=default

9.40.1 Detailed Description

Object used to store and query data extracted from a binaryproto file using the [ICaffeParser](#).

See also

[nvcaffeparser1::ICaffeParser](#)

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.40.2 Constructor & Destructor Documentation

9.40.2.1 ~IBinaryProtoBlob()

```
virtual nvcaffeparser1::IBinaryProtoBlob::~IBinaryProtoBlob ( ) [virtual], [default], [noexcept]
```

9.40.3 Member Function Documentation

9.40.3.1 destroy()

```
virtual TRT\_DEPRECATED void nvcaffeparser1::IBinaryProtoBlob::destroy ( ) [pure virtual], [noexcept]
```

Deprecated Deprecated in TensorRT 8.0. Superseded by `delete`.

Warning

Calling `destroy` on a managed pointer will result in a double-free error.

9.40.3.2 `getData()`

```
virtual void const * nvcaffeparser1::IBinaryProtoBlob::getData ( ) [pure virtual], [noexcept]
```

9.40.3.3 `getDataType()`

```
virtual nvinfer1::DataType nvcaffeparser1::IBinaryProtoBlob::getDataType ( ) [pure virtual],  
[noexcept]
```

9.40.3.4 `getDimensions()`

```
virtual nvinfer1::Dims4 nvcaffeparser1::IBinaryProtoBlob::getDimensions ( ) [pure virtual], [noexcept]
```

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

- [NvCaffeParser.h](#)

9.41 `nvcaffeparser1::IBlobNameToTensor` Class Reference

Object used to store and query Tensors after they have been extracted from a Caffe model using the [ICaffeParser](#).

```
#include <NvCaffeParser.h>
```

Public Member Functions

- virtual [nvinfer1::ITensor](#) * [find](#) (char const *name) const noexcept=0
Given a blob name, returns a pointer to a ITensor object.

Protected Member Functions

- virtual [~IBlobNameToTensor](#) ()

9.41.1 Detailed Description

Object used to store and query Tensors after they have been extracted from a Caffe model using the [ICaffeParser](#).

Note

The lifetime of [IBlobNameToTensor](#) is the same as the lifetime of its parent [ICaffeParser](#).

See also

[nvcaffeparser1::ICaffeParser](#)

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.41.2 Constructor & Destructor Documentation**9.41.2.1 ~IBlobNameToTensor()**

```
virtual nvcaffeparser1::IBlobNameToTensor::~~IBlobNameToTensor ( ) [inline], [protected], [virtual]
```

9.41.3 Member Function Documentation**9.41.3.1 find()**

```
virtual nvinfer1::ITensor * nvcaffeparser1::IBlobNameToTensor::find (
    char const * name ) const [pure virtual], [noexcept]
```

Given a blob name, returns a pointer to a ITensor object.

Parameters

<i>name</i>	Caffe blob name for which the user wants the corresponding ITensor.
-------------	---

Returns

ITensor* corresponding to the queried name. If no such ITensor exists, then nullptr is returned.

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

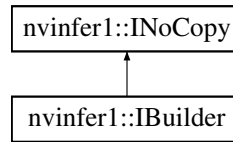
- [NvCaffeParser.h](#)

9.42 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=default`
- `TRT_DEPRECATED` void `setMaxBatchSize (int32_t batchSize) noexcept`
Set the maximum batch size. This has no effect for networks created with explicit batch dimension mode.
- `TRT_DEPRECATED` int32_t `getMaxBatchSize () const noexcept`
Get the maximum batch size.
- bool `platformHasFastFp16 () const noexcept`
Determine whether the platform has fast native fp16.
- bool `platformHasFastInt8 () const noexcept`
Determine whether the platform has fast native int8.
- `TRT_DEPRECATED` void `destroy () noexcept`
Destroy this object.
- 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.
- `TRT_DEPRECATED` `nvinfer1::ICudaEngine` * `buildEngineWithConfig (INetworkDefinition &network, IBuilderConfig &config) noexcept`
Builds an engine for the given INetworkDefinition and given IBuilderConfig.
- `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.
- bool `platformHasTf32 () const noexcept`
Determine whether the platform has TF32 support.
- `nvinfer1::IHostMemory` * `buildSerializedNetwork (INetworkDefinition &network, IBuilderConfig &config) noexcept`

Builds and serializes 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

9.42.1 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.

9.42.2 Constructor & Destructor Documentation

9.42.2.1 ~IBuilder()

```
virtual nvinfer1::IBuilder::~~IBuilder ( ) [virtual], [default], [noexcept]
```

9.42.3 Member Function Documentation

9.42.3.1 buildEngineWithConfig()

```
TRT_DEPRECATED nvinfer1::ICudaEngine * nvinfer1::IBuilder::buildEngineWithConfig (
    INetworkDefinition & network,
    IBuilderConfig & config ) [inline], [noexcept]
```

Builds an engine for the given [INetworkDefinition](#) and given [IBuilderConfig](#).

It enables the builder to build multiple engines based on the same network definition, but with different builder configurations.

Note

This function will synchronize the cuda stream returned by `config.getProfileStream()` before returning.

Deprecated Deprecated in TensorRT 8.0. Superseded by [IBuilder::buildSerializedNetwork\(\)](#).

9.42.3.2 buildSerializedNetwork()

```
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

<i>network</i>	Network definition.
<i>config</i>	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](#)

9.42.3.3 createBuilderConfig()

```
nvinfer1::IBuilderConfig * nvinfer1::IBuilder::createBuilderConfig ( ) [inline], [noexcept]
```

Create a builder configuration object.

See also

[IBuilderConfig](#)

9.42.3.4 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 dynamic shapes and explicit batch dimensions when used with [NetworkDefinitionCreationFlag::kEXPLICIT_BATCH](#) flag. Creating a network without [NetworkDefinitionCreationFlag::kEXPLICIT_BATCH](#) flag has been deprecated.

Parameters

<i>flags</i>	Bitset of NetworkDefinitionCreationFlags specifying network properties combined with bitwise OR. e.g., $1U << \text{NetworkDefinitionCreationFlag::kEXPLICIT_BATCH}$
--------------	---

See also

[INetworkDefinition](#), [NetworkDefinitionCreationFlags](#)

9.42.3.5 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 `setShapeValues()` 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](#)

9.42.3.6 destroy()

```
TRT_DEPRECATED void nvinfer1::IBuilder::destroy ( ) [inline], [noexcept]
```

Destroy this object.

Deprecated Deprecated in TensorRT 8.0. Superseded by `delete`.

Warning

Calling `destroy` on a managed pointer will result in a double-free error.

9.42.3.7 getErrorRecorder()

```
IErrRecorder * 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 [IErrRecorder](#) object that has been registered.

See also

[setErrorRecorder\(\)](#)

9.42.3.8 getLogger()

```
ILogger * nvinfer1::IBuilder::getLogger ( ) const [inline], [noexcept]
```

get the logger with which the builder was created

Returns

the logger

9.42.3.9 getMaxBatchSize()

```
TRT_DEPRECATED int32_t nvinfer1::IBuilder::getMaxBatchSize ( ) const [inline], [noexcept]
```

Get the maximum batch size.

Returns

The maximum batch size.

Deprecated Deprecated in TensorRT 8.4.

See also

[setMaxBatchSize\(\)](#)

[getMaxDLABatchSize\(\)](#)

9.42.3.10 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.

9.42.3.11 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\(\)](#)

9.42.3.12 getNbDLACores()

```
int32_t nvinfer1::IBuilder::getNbDLACores ( ) const [inline], [noexcept]
```

Return the number of DLA engines available to this builder.

9.42.3.13 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.

9.42.3.14 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

<i>network</i>	The network definition to check for configuration compliance.
<i>config</i>	The configuration of the builder to use when checking <i>network</i> .

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

This function will synchronize the cuda stream returned by `config.getProfileStream()` before returning.

This function is only supported in NVIDIA Drive(R) products.

9.42.3.15 platformHasFastFp16()

```
bool nvinfer1::IBuilder::platformHasFastFp16 ( ) const [inline], [noexcept]
```

Determine whether the platform has fast native fp16.

9.42.3.16 platformHasFastInt8()

```
bool nvinfer1::IBuilder::platformHasFastInt8 ( ) const [inline], [noexcept]
```

Determine whether the platform has fast native int8.

9.42.3.17 platformHasTf32()

```
bool nvinfer1::IBuilder::platformHasTf32 ( ) const [inline], [noexcept]
```

Determine whether the platform has TF32 support.

9.42.3.18 reset()

```
void nvinfer1::IBuilder::reset ( ) [inline], [noexcept]
```

Resets the builder state to default values.

9.42.3.19 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

<i>recorder</i>	The error recorder to register with this interface.
-----------------	---

See also

[getErrorRecorder\(\)](#)

9.42.3.20 setGpuAllocator()

```
void nvinfer1::IBuilder::setGpuAllocator (
    IAllocator * allocator ) [inline], [noexcept]
```

Set the GPU allocator.

Parameters

<i>allocator</i>	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: uses cudaMalloc/cudaFree.

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.

9.42.3.21 setMaxBatchSize()

```
TRT_DEPRECATED void nvinfer1::IBuilder::setMaxBatchSize (
    int32_t batchSize ) [inline], [noexcept]
```

Set the maximum batch size. This has no effect for networks created with explicit batch dimension mode.

Parameters

<i>batchSize</i>	The maximum batch size which can be used at execution time, and also the batch size for which the engine will be optimized.
------------------	---

Deprecated Deprecated in TensorRT 8.4.

See also

[getMaxBatchSize\(\)](#)

9.42.3.22 setMaxThreads()

```
bool nvinfer1::IBuilder::setMaxThreads (
    int32_t maxThreads ) [inline], [noexcept]
```

Set the maximum number of threads.

Parameters

<i>maxThreads</i>	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.

9.42.4 Member Data Documentation

9.42.4.1 mImpl

```
apiv::VBuilder* nvinfer1::IBuilder::mImpl [protected]
```

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

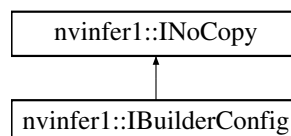
- [NvInfer.h](#)

9.43 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=default
- virtual [TRT_DEPRECATED](#) void [setMinTimingIterations](#) (int32_t minTiming) noexcept
Set the number of minimization iterations used when timing layers.
- virtual [TRT_DEPRECATED](#) int32_t [getMinTimingIterations](#) () const noexcept
Query the number of minimization iterations.
- 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 [setInt8Calibrator](#) ([IInt8Calibrator](#) *calibrator) noexcept
Set Int8 Calibration interface.
- [IInt8Calibrator](#) * [getInt8Calibrator](#) () const noexcept
Get Int8 Calibration interface.
- [TRT_DEPRECATED](#) void [setMaxWorkspaceSize](#) (std::size_t workspaceSize) noexcept
Set the maximum workspace size.
- [TRT_DEPRECATED](#) std::size_t [getMaxWorkspaceSize](#) () const noexcept
Get the maximum workspace size.
- 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.
- [TRT_DEPRECATED](#) void [destroy](#) () noexcept
Delete this IBuilderConfig.
- 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](#).
- void [setAlgorithmSelector](#) ([IAlgorithmSelector](#) *selector) noexcept
Set Algorithm Selector.
- [IAlgorithmSelector](#) * [getAlgorithmSelector](#) () const noexcept
Get Algorithm Selector.
- bool [setCalibrationProfile](#) ([IOptimizationProfile](#) const *profile) noexcept
Add a calibration profile.
- [IOptimizationProfile](#) const * [getCalibrationProfile](#) () noexcept
Get the current calibration profile.
- void [setQuantizationFlags](#) ([QuantizationFlags](#) flags) noexcept
Set the quantization flags.
- [QuantizationFlags](#) [getQuantizationFlags](#) () const noexcept
Get the quantization flags.
- void [clearQuantizationFlag](#) ([QuantizationFlag](#) flag) noexcept
clear a quantization flag.
- void [setQuantizationFlag](#) ([QuantizationFlag](#) flag) noexcept
Set a single quantization flag.
- bool [getQuantizationFlag](#) ([QuantizationFlag](#) flag) const noexcept
Returns true if the quantization flag is set.
- 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 forward-compatible engines.
- char const * [getPluginToSerialize](#) (int32_t index) const noexcept
Get the plugin library path to be serialized with forward-compatible engines.
- int32_t [getNbPluginsToSerialize](#) () const noexcept
Get the number of plugin library paths to be serialized with forward-compatible engines.

Protected Attributes

- apiv::VBuilderConfig * [mImpl](#)

Additional Inherited Members

9.43.1 Detailed Description

Holds properties for configuring a builder to produce an engine.

See also

[BuilderFlags](#)

9.43.2 Constructor & Destructor Documentation

9.43.2.1 ~IBuilderConfig()

```
virtual nvinfer1::IBuilderConfig::~~IBuilderConfig ( ) [virtual], [default], [noexcept]
```

9.43.3 Member Function Documentation

9.43.3.1 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

<i>profile</i>	The new optimization profile, which must satisfy <code>profile->isValid() == true</code>
----------------	---

Returns

The index of the optimization profile (starting from 0) if the input is valid, or -1 if the input is not valid.

9.43.3.2 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.

9.43.3.3 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\(\)](#)

9.43.3.4 clearQuantizationFlag()

```
void nvinfer1::IBuilderConfig::clearQuantizationFlag (
    QuantizationFlag flag ) [inline], [noexcept]
```

clear a quantization flag.

Clears the quantization flag from the enabled quantization flags.

See also

[setQuantizationFlags\(\)](#)

9.43.3.5 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.

Parameters

<i>blob</i>	A pointer to the raw data that contains serialized timing cache
<i>size</i>	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

9.43.3.6 destroy()

```
TRT_DEPRECATED void nvinfer1::IBuilderConfig::destroy ( ) [inline], [noexcept]
```

Delete this [IBuilderConfig](#).

De-allocates any internally allocated memory.

Deprecated Deprecated in TensorRT 8.0. Superseded by `delete`.

Warning

Calling `destroy` on a managed pointer will result in a double-free error.

9.43.3.7 getAlgorithmSelector()

```
IAlgorithmSelector * nvinfer1::IBuilderConfig::getAlgorithmSelector ( ) const [inline], [noexcept]
```

Get Algorithm Selector.

9.43.3.8 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\(\)](#)

9.43.3.9 `getBuilderOptimizationLevel()`

```
int32_t nvinfer1::IBuilderConfig::getBuilderOptimizationLevel ( ) [inline], [noexcept]
```

Get builder optimization level.

Returns

the current builder optimization level

See also

[setBuilderOptimizationLevel](#)

9.43.3.10 `getCalibrationProfile()`

```
IOptimizationProfile const * nvinfer1::IBuilderConfig::getCalibrationProfile ( ) [inline], [noexcept]
```

Get the current calibration profile.

Returns

A pointer to the current calibration profile or nullptr if calibration profile is unset.

9.43.3.11 `getDefaultDeviceType()`

```
DeviceType nvinfer1::IBuilderConfig::getDefaultDeviceType ( ) const [inline], [noexcept]
```

Get the default DeviceType which was set by `setDefaultDeviceType`.

By default it returns `DeviceType::kGPU`.

9.43.3.12 `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.

9.43.3.13 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.

9.43.3.14 getEngineCapability()

```
EngineCapability nvinfer1::IBuilderConfig::getEngineCapability ( ) const [inline], [noexcept]
```

Query EngineCapability flow configured for the builder.

By default it returns [EngineCapability::kSTANDARD](#).

See also

[setEngineCapability\(\)](#)

9.43.3.15 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.

9.43.3.16 `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\(\)](#)

9.43.3.17 `getHardwareCompatibilityLevel()`

```
HardwareCompatibilityLevel nvinfer1::IBuilderConfig::getHardwareCompatibilityLevel ( ) const [inline],  
[noexcept]
```

Get the hardware compatibility level.

Returns

hardwareCompatibilityLevel The level of hardware compatibility.

See also

[setHardwareCompatiblityLevel\(\)](#)

9.43.3.18 `getInt8Calibrator()`

```
IInt8Calibrator * nvinfer1::IBuilderConfig::getInt8Calibrator ( ) const [inline], [noexcept]
```

Get Int8 Calibration interface.

9.43.3.19 getMaxWorkspaceSize()

`TRT_DEPRECATED` `std::size_t nvinfer1::IBuilderConfig::getMaxWorkspaceSize () const [inline], [noexcept]`

Get the maximum workspace size.

By default the workspace size is the size of total global memory in the device.

Returns

The maximum workspace size.

See also

[setMaxWorkspaceSize\(\)](#)

Deprecated Deprecated in TensorRT 8.3. Superseded by [IBuilderConfig::getMemoryPoolLimit\(\)](#) with [MemoryPoolType::kWORKSPACE](#)

9.43.3.20 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

<i>pool</i>	The memory pool to get the limit for.
-------------	---------------------------------------

Returns

The size of the memory limit, in bytes, for the corresponding pool.

See also

[setMemoryPoolLimit](#)

9.43.3.21 `getMinTimingIterations()`

```
virtual TRT\_DEPRECATED int32_t nvinfer1::IBuilderConfig::getMinTimingIterations ( ) const [inline],  
[virtual], [noexcept]
```

Query the number of minimization iterations.

By default the minimum number of iterations is 1.

See also

[setMinTimingIterations\(\)](#)

Deprecated Deprecated in TensorRT 8.4. Superseded by [getAvgTimingIterations\(\)](#).

9.43.3.22 `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 been defined (or zero, if none has been defined yet).

Returns

The number of the optimization profiles.

9.43.3.23 `getNbPluginsToSerialize()`

```
int32_t nvinfer1::IBuilderConfig::getNbPluginsToSerialize ( ) const [inline], [noexcept]
```

Get the number of plugin library paths to be serialized with forward-compatible engines.

Returns

Number of paths plugin libraries which will be serialized.

9.43.3.24 getPluginToSerialize()

```
char const * nvinfer1::IBuilderConfig::getPluginToSerialize (
    int32_t index ) const [inline], [noexcept]
```

Get the plugin library path to be serialized with forward-compatible engines.

Returns

A path of plugin library which will be serialized.

9.43.3.25 getPreviewFeature()

```
bool nvinfer1::IBuilderConfig::getPreviewFeature (
    PreviewFeature feature ) const [inline], [noexcept]
```

Get status of preview feature.

Parameters

<i>feature</i>	the feature to query
----------------	----------------------

Returns

true if the `feature` is enabled, false otherwise

See also

[PreviewFeature](#), [setPreviewFeature](#)

9.43.3.26 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\(\)](#)

9.43.3.27 `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](#)

9.43.3.28 `getQuantizationFlag()`

```
bool nvinfer1::IBuilderConfig::getQuantizationFlag (
    QuantizationFlag flag ) const [inline], [noexcept]
```

Returns true if the quantization flag is set.

See also

[getQuantizationFlags\(\)](#)

Returns

True if quantization flag is set, false if unset.

9.43.3.29 `getQuantizationFlags()`

```
QuantizationFlags nvinfer1::IBuilderConfig::getQuantizationFlags ( ) const [inline], [noexcept]
```

Get the quantization flags.

Returns

The quantization flags as a bitmask.

See also

[setQuantizationFlag\(\)](#)

9.43.3.30 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

9.43.3.31 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](#)

9.43.3.32 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\(\)](#)

9.43.3.33 reset()

```
void nvinfer1::IBuilderConfig::reset ( ) [inline], [noexcept]
```

Resets the builder configuration to defaults.

Useful for initializing a builder config object to its original state.

9.43.3.34 resetDeviceType()

```
void nvinfer1::IBuilderConfig::resetDeviceType (
    ILayer const * layer ) [inline], [noexcept]
```

reset the DeviceType for this layer

See also

[setDeviceType\(\)](#) [getDeviceType\(\)](#) [isDeviceTypeSet\(\)](#)

9.43.3.35 setAlgorithmSelector()

```
void nvinfer1::IBuilderConfig::setAlgorithmSelector (
    IAlgorithmSelector * selector ) [inline], [noexcept]
```

Set Algorithm Selector.

Parameters

<i>selector</i>	The algorithm selector to be set in the build config.
-----------------	---

9.43.3.36 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\(\)](#)

9.43.3.37 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 2. Valid values include integers from 0 to the maximum optimization level, which is currently 4. Setting it to greater than the maximum level results in behavior identical to the maximum level.

Parameters

<i>level</i>	The optimization level to set to. Must be non-negative.
--------------	---

See also

[getBuilderOptimizationLevel](#)

9.43.3.38 setCalibrationProfile()

```
bool nvinfer1::IBuilderConfig::setCalibrationProfile (
    IOptimizationProfile const * profile ) [inline], [noexcept]
```

Add a calibration profile.

Calibration optimization profile must be set if int8 calibration is used to set scales for a network with runtime dimensions.

Parameters

<i>profile</i>	The new calibration profile, which must satisfy <code>profile->isValid() == true</code> or be nullptr. MIN and MAX values will be overwritten by kOPT.
----------------	---

Returns

True if the calibration profile was set correctly.

9.43.3.39 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\(\)](#)

9.43.3.40 setDeviceType()

```
void nvinfer1::IBuilderConfig::setDeviceType (
    ILayer const * layer,
    DeviceType deviceType ) [inline], [noexcept]
```

Set the device that this layer must execute on.

Parameters

<i>deviceType</i>	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 kSAFE.GPU.

See also

[getDeviceType\(\)](#)

9.43.3.41 setDLACore()

```
void nvinfer1::IBuilderConfig::setDLACore (
    int32_t dlaCore ) [inline], [noexcept]
```

Sets the DLA core used by the network. Defaults to -1.

Parameters

<i>dlaCore</i>	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\(\)](#)

9.43.3.42 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.

9.43.3.43 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\(\)](#)

9.43.3.44 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

<i>builderFlags</i>	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\(\)](#)

9.43.3.45 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

<i>hardwareCompatibilityLevel</i>	The level of hardware compatibility.
-----------------------------------	--------------------------------------

9.43.3.46 setInt8Calibrator()

```
void nvinfer1::IBuilderConfig::setInt8Calibrator (
    IInt8Calibrator * calibrator ) [inline], [noexcept]
```

Set Int8 Calibration interface.

The calibrator is to minimize the information loss during the INT8 quantization process.

9.43.3.47 setMaxWorkspaceSize()

```
TRT_DEPRECATED void nvinfer1::IBuilderConfig::setMaxWorkspaceSize (
    std::size_t workspaceSize ) [inline], [noexcept]
```

Set the maximum workspace size.

Parameters

<i>workspaceSize</i>	The maximum GPU temporary memory which the engine can use at execution time.
----------------------	--

See also

[getMaxWorkspaceSize\(\)](#)

Deprecated Deprecated in TensorRT 8.3. Superseded by [IBuilderConfig::setMemoryPoolLimit\(\)](#) with [MemoryPoolType::kWORKSPACE](#).

9.43.3.48 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

<i>pool</i>	The memory pool to limit the available memory for.
<i>poolSize</i>	The size of the pool in bytes.

See also

[getMemoryPoolLimit](#), [MemoryPoolType](#)

9.43.3.49 setMinTimingIterations()

```
virtual TRT\_DEPRECATED void nvinfer1::IBuilderConfig::setMinTimingIterations (
    int32_t minTiming ) [inline], [virtual], [noexcept]
```

Set the number of minimization 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 minimization. The builder may sometimes run layers for more iterations to improve timing accuracy if this parameter is set to a small value and the runtime of the layer is short.

See also

[getMinTimingIterations\(\)](#)

Deprecated Deprecated in TensorRT 8.4. Superseded by [setAvgTimingIterations\(\)](#).

9.43.3.50 setPluginsToSerialize()

```
void nvinfer1::IBuilderConfig::setPluginsToSerialize (
    char const *const * paths,
    int32_t nbPaths ) [inline], [noexcept]
```

Set the plugin libraries to be serialized with forward-compatible engines.

Parameters

<i>paths</i>	The paths of plugin libraries.
<i>nbPaths</i>	The number of paths.

9.43.3.51 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

<i>feature</i>	the feature to enable / disable
<i>enable</i>	true for enable, false for disable

See also

[PreviewFeature](#), [getPreviewFeature](#)

9.43.3.52 setProfileStream()

```
void nvinfer1::IBuilderConfig::setProfileStream (
    const cudaStream_t stream ) [inline], [noexcept]
```

Set the cuda stream that is used to profile this network.

Parameters

<i>stream</i>	The cuda stream used for profiling by the builder.
---------------	--

See also

[getProfileStream\(\)](#)

9.43.3.53 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](#)

9.43.3.54 setQuantizationFlag()

```
void nvinfer1::IBuilderConfig::setQuantizationFlag (
    QuantizationFlag flag ) [inline], [noexcept]
```

Set a single quantization flag.

Add the input quantization flag to the already enabled quantization flags.

See also

[setQuantizationFlags\(\)](#)

9.43.3.55 setQuantizationFlags()

```
void nvinfer1::IBuilderConfig::setQuantizationFlags (
    QuantizationFlags flags ) [inline], [noexcept]
```

Set the quantization flags.

The flags are listed in the QuantizationFlag enum. The flags set configuration options to quantize the network in int8.

Parameters

<i>flags</i>	The quantization flags.
--------------	-------------------------

Note

This function will override the previous set flags, rather than bitwise ORing the new flag.

See also

[getQuantizationFlags\(\)](#)

9.43.3.56 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 cublas and cublasLt as tactic sources, use a value of:

```
1U << static_cast<uint32_t>(TacticSource::kCUBLAS) | 1U << static_cast<uint32_t>(TacticSource::kCUBLAS←
.LT)
```

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.

9.43.3.57 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

<i>cache</i>	the timing cache to be used
<i>ignoreMismatch</i>	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.

9.43.4 Member Data Documentation

9.43.4.1 mImpl

```
apiv::VBuilderConfig* nvinfer1::IBuilderConfig::mImpl [protected]
```

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

- [NvInfer.h](#)

9.44 nvcaffeparser1::ICaffeParser Class Reference

Class used for parsing Caffe models.

```
#include <NvCaffeParser.h>
```

Public Member Functions

- virtual [IBlobNameToTensor](#) const * [parse](#) (char const *deploy, char const *model, [nvinfer1::INetworkDefinition](#) &network, [nvinfer1::DataType](#) weightType) noexcept=0
Parse a prototxt file and a binaryproto Caffe model to extract network definition and weights associated with the network, respectively.
- virtual [IBlobNameToTensor](#) const * [parseBuffers](#) (uint8_t const *deployBuffer, std::size_t deployLength, uint8_t const *modelBuffer, std::size_t modelLength, [nvinfer1::INetworkDefinition](#) &network, [nvinfer1::DataType](#) weightType) noexcept=0
Parse a deploy prototxt and a binaryproto Caffe model from memory buffers to extract network definition and weights associated with the network, respectively.
- virtual [IBinaryProtoBlob](#) * [parseBinaryProto](#) (char const *fileName) noexcept=0
Parse and extract data stored in binaryproto file.
- virtual void [setProtobufBufferSize](#) (size_t size) noexcept=0
Set buffer size for the parsing and storage of the learned model.
- virtual [TRT_DEPRECATED](#) void [destroy](#) () noexcept=0
Destroy this ICaffeParser object.
- virtual void [setPluginFactoryV2](#) ([IPluginFactoryV2](#) *factory) noexcept=0
Set the IPluginFactoryV2 used to create the user defined pluginV2 objects.
- virtual void [setPluginNamespace](#) (char const *libNamespace) noexcept=0
Set the namespace used to lookup and create plugins in the network.
- virtual [~ICaffeParser](#) () noexcept=default
- virtual void [setErrorRecorder](#) ([nvinfer1::IErrorRecorder](#) *recorder) noexcept=0
Set the ErrorRecorder for this interface.
- virtual [nvinfer1::IErrorRecorder](#) * [getErrorRecorder](#) () const noexcept=0
get the ErrorRecorder assigned to this interface.

9.44.1 Detailed Description

Class used for parsing Caffe models.

Allows users to export models trained using Caffe to TRT.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.44.2 Constructor & Destructor Documentation

9.44.2.1 ~ICaffeParser()

```
virtual nvcaffeparser1::ICaffeParser::~~ICaffeParser ( ) [virtual], [default], [noexcept]
```

9.44.3 Member Function Documentation

9.44.3.1 destroy()

```
virtual TRT\_DEPRECATED void nvcaffeparser1::ICaffeParser::destroy ( ) [pure virtual], [noexcept]
```

Destroy this [ICaffeParser](#) object.

Deprecated Deprecated in TensorRT 8.0. Superseded by `delete`.

Warning

Calling `destroy` on a managed pointer will result in a double-free error.

9.44.3.2 getErrorRecorder()

```
virtual nvinfer1::IErrorRecorder * nvcaffeparser1::ICaffeParser::getErrorRecorder ( ) const [pure virtual], [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\(\)](#)

9.44.3.3 parse()

```
virtual IBlobNameToTensor const * nvcaffeparser1::ICaffeParser::parse (
    char const * deploy,
    char const * model,
    nvinfer1::INetworkDefinition & network,
    nvinfer1::DataType weightType ) [pure virtual], [noexcept]
```

Parse a `prototxt` file and a `binaryproto` Caffe model to extract network definition and weights associated with the network, respectively.

Parameters

<i>deploy</i>	The plain text, prototxt file used to define the network definition.
<i>model</i>	The binaryproto Caffe model that contains the weights associated with the network.
<i>network</i>	Network in which the CaffeParser will fill the layers.
<i>weightType</i>	The type to which the weights will transformed.

Returns

A pointer to an [IBlobNameToTensor](#) object that contains the extracted data.

See also

[nvcaffeparser1::IBlobNameToTensor](#)

9.44.3.4 parseBinaryProto()

```
virtual IBinaryProtoBlob * nvcaffeparser1::ICaffeParser::parseBinaryProto (
    char const * fileName ) [pure virtual], [noexcept]
```

Parse and extract data stored in binaryproto file.

The binaryproto file contains data stored in a binary blob. [parseBinaryProto\(\)](#) converts it to an [IBinaryProtoBlob](#) object which gives the user access to the data and meta-data about data.

Parameters

<i>fileName</i>	Path to file containing binary proto.
-----------------	---------------------------------------

Returns

A pointer to an [IBinaryProtoBlob](#) object that contains the extracted data.

See also

[nvcaffeparser1::IBinaryProtoBlob](#)

9.44.3.5 parseBuffers()

```
virtual IBlobNameToTensor const * nvcaffeparser1::ICaffeParser::parseBuffers (
    uint8_t const * deployBuffer,
    std::size_t deployLength,
    uint8_t const * modelBuffer,
    std::size_t modelLength,
    nvinfer1::INetworkDefinition & network,
    nvinfer1::DataType weightType ) [pure virtual], [noexcept]
```

Parse a deploy prototxt and a binaryproto Caffe model from memory buffers to extract network definition and weights associated with the network, respectively.

Parameters

<i>deployBuffer</i>	The plain text deploy prototxt used to define the network definition.
<i>deployLength</i>	The length of the deploy buffer.
<i>modelBuffer</i>	The binaryproto Caffe memory buffer that contains the weights associated with the network.
<i>modelLength</i>	The length of the model buffer.
<i>network</i>	Network in which the CaffeParser will fill the layers.
<i>weightType</i>	The type to which the weights will transformed.

Returns

A pointer to an [IBlobNameToTensor](#) object that contains the extracted data.

See also

[nvcaffeparser1::IBlobNameToTensor](#)

9.44.3.6 setErrorRecorder()

```
virtual void nvcaffeparser1::ICaffeParser::setErrorRecorder (
    nvinfer1::IErrorRecorder * 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.

If an error recorder is not set, messages will be sent to the global log stream.

Parameters

<i>recorder</i>	The error recorder to register with this interface.
-----------------	---

See also

[getErrorRecorder\(\)](#)

9.44.3.7 setPluginFactoryV2()

```
virtual void nvcaffeparser1::ICaffeParser::setPluginFactoryV2 (
    IPluginFactoryV2 * factory ) [pure virtual], [noexcept]
```

Set the [IPluginFactoryV2](#) used to create the user defined pluginV2 objects.

Parameters

<i>factory</i>	Pointer to an instance of the user implementation of IPluginFactoryV2 .
----------------	---

9.44.3.8 setPluginNamespace()

```
virtual void nvcaffeparser1::ICaffeParser::setPluginNamespace (
    char const * libNamespace ) [pure virtual], [noexcept]
```

Set the namespace used to lookup and create plugins in the network.

9.44.3.9 setProtobufBufferSize()

```
virtual void nvcaffeparser1::ICaffeParser::setProtobufBufferSize (
    size_t size ) [pure virtual], [noexcept]
```

Set buffer size for the parsing and storage of the learned model.

Parameters

<i>size</i>	The size of the buffer specified as the number of bytes.
-------------	--

Note

Default size is 2^{30} bytes.

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

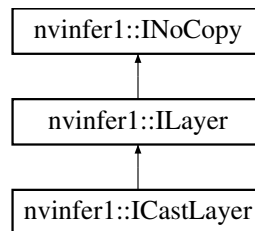
- [NvCaffeParser.h](#)

9.45 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=default

Protected Attributes

- apiv::VCastLayer * [mImpl](#)

9.45.1 Detailed Description

A cast layer in a network.

This layer casts a given tensor to the datatype specified by toType.

9.45.2 Constructor & Destructor Documentation

9.45.2.1 ~ICastLayer()

```
virtual nvinfer1::ICastLayer::~~ICastLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.45.3 Member Function Documentation

9.45.3.1 getToType()

```
DataType nvinfer1::ICastLayer::getToType ( ) const [inline], [noexcept]
```

Return cast layer output type.

9.45.3.2 setToType()

```
void nvinfer1::ICastLayer::setToType (
    DataType toType ) [inline], [noexcept]
```

Set cast layer output type.

9.45.4 Member Data Documentation

9.45.4.1 mImpl

```
apiv::VCastLayer* nvinfer1::ICastLayer::mImpl [protected]
```

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

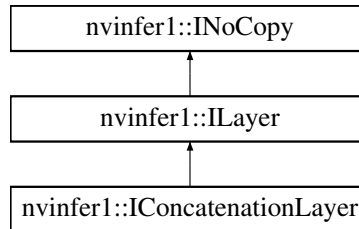
- [NvInfer.h](#)

9.46 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=default

Protected Attributes

- apiv::VConcatenationLayer * [mImpl](#)

9.46.1 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.

9.46.2 Constructor & Destructor Documentation

9.46.2.1 ~IConcatenationLayer()

```
virtual nvinfer1::IConcatenationLayer::~~IConcatenationLayer ( ) [protected], [virtual], [default],
[noexcept]
```

9.46.3 Member Function Documentation

9.46.3.1 getAxis()

```
int32_t nvinfer1::IConcatenationLayer::getAxis ( ) const [inline], [noexcept]
```

Get the axis along which concatenation occurs.

See also

[setAxis\(\)](#)

9.46.3.2 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. For implicit batch mode, the number of tensor dimensions does NOT include the implicit batch dimension.

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

<i>axis</i>	The axis along which concatenation occurs.
-------------	--

9.46.4 Member Data Documentation

9.46.4.1 mImpl

apiv::VConcatenationLayer* nvinfer1::IConcatenationLayer::mImpl [protected]

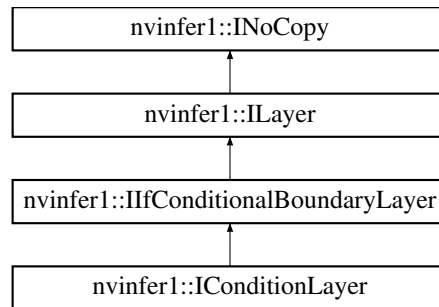
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.47 nvinfer1::IConditionLayer Class Reference

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

Inheritance diagram for nvinfer1::IConditionLayer:



Protected Member Functions

- virtual [~IConditionLayer](#) () noexcept=default

Protected Attributes

- apiv::VConditionLayer * [mImpl](#)

Additional Inherited Members

9.47.1 Detailed Description

This layer represents a condition input to an [IIfConditional](#).

9.47.2 Constructor & Destructor Documentation

9.47.2.1 ~IConditionLayer()

```
virtual nvinfer1::IConditionLayer::~~IConditionLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.47.3 Member Data Documentation

9.47.3.1 mImpl

```
apiv::VConditionLayer* nvinfer1::IConditionLayer::mImpl [protected]
```

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

- [NvInfer.h](#)

9.48 nvinfer1::consistency::IConsistencyChecker Class Reference

Validates a serialized engine blob.

```
#include <NvInferConsistency.h>
```

Public Member Functions

- bool [validate](#) () const noexcept
Check that a blob that was input to createConsistencyChecker method represents a valid engine.
- virtual [~IConsistencyChecker](#) ()=default
De-allocates any internally allocated memory.

Protected Member Functions

- [IConsistencyChecker](#) ()=default
- [IConsistencyChecker](#) ([IConsistencyChecker](#) const &other)=delete
- [IConsistencyChecker](#) & operator= ([IConsistencyChecker](#) const &other)=delete
- [IConsistencyChecker](#) ([IConsistencyChecker](#) &&other)=delete
- [IConsistencyChecker](#) & operator= ([IConsistencyChecker](#) &&other)=delete

Protected Attributes

- apiv::VConsistencyChecker * [mImpl](#)

9.48.1 Detailed Description

Validates a serialized engine blob.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.48.2 Constructor & Destructor Documentation**9.48.2.1 ~IConsistencyChecker()**

```
virtual nvinfer1::consistency::IConsistencyChecker::~IConsistencyChecker ( ) [virtual], [default]
```

De-allocates any internally allocated memory.

9.48.2.2 IConsistencyChecker() [1/3]

```
nvinfer1::consistency::IConsistencyChecker::IConsistencyChecker ( ) [protected], [default]
```

9.48.2.3 IConsistencyChecker() [2/3]

```
nvinfer1::consistency::IConsistencyChecker::IConsistencyChecker (
    IConsistencyChecker const & other ) [protected], [delete]
```

9.48.2.4 IConsistencyChecker() [3/3]

```
nvinfer1::consistency::IConsistencyChecker::IConsistencyChecker (
    IConsistencyChecker && other ) [protected], [delete]
```

9.48.3 Member Function Documentation**9.48.3.1 operator=() [1/2]**

```
IConsistencyChecker & nvinfer1::consistency::IConsistencyChecker::operator= (
    IConsistencyChecker && other ) [protected], [delete]
```


9.48.3.2 operator=() [2/2]

```
IConsistencyChecker & nvinfer1::consistency::IConsistencyChecker::operator= (
    IConsistencyChecker const & other ) [protected], [delete]
```

9.48.3.3 validate()

```
bool nvinfer1::consistency::IConsistencyChecker::validate ( ) const [inline], [noexcept]
```

Check that a blob that was input to createConsistencyChecker method represents a valid engine.

Returns

true if the original blob encoded an engine that belongs to valid engine domain with target capability [EngineCapability::kSAFETY](#), false otherwise.

9.48.4 Member Data Documentation

9.48.4.1 mImpl

```
apiv::VConsistencyChecker* nvinfer1::consistency::IConsistencyChecker::mImpl [protected]
```

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

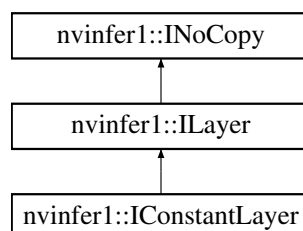
- [NvInferConsistency.h](#)

9.49 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](#) dimensions) noexcept
Set the dimensions for the layer.
- [Dims](#) [getDimensions](#) () const noexcept
Get the dimensions for the layer.

Protected Member Functions

- virtual [~IConstantLayer](#) () noexcept=default

Protected Attributes

- apiv::VConstantLayer * [mImpl](#)

9.49.1 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.

9.49.2 Constructor & Destructor Documentation

9.49.2.1 [~IConstantLayer](#)()

```
virtual nvinfer1::IConstantLayer::~~IConstantLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.49.3 Member Function Documentation

9.49.3.1 `getDimensions()`

```
Dims nvinfer1::IConstantLayer::getDimensions ( ) const [inline], [noexcept]
```

Get the dimensions for the layer.

Returns

the dimensions for the layer

See also

[getDimensions](#)

9.49.3.2 `getWeights()`

```
Weights nvinfer1::IConstantLayer::getWeights ( ) const [inline], [noexcept]
```

Get the weights for the layer.

See also

[setWeights](#)

9.49.3.3 `setDimensions()`

```
void nvinfer1::IConstantLayer::setDimensions (
    Dims dimensions ) [inline], [noexcept]
```

Set the dimensions for the layer.

Parameters

<i>dimensions</i>	The dimensions of the layer
-------------------	-----------------------------

See also

[setDimensions](#)

9.49.3.4 setWeights()

```
void nvinfer1::IConstantLayer::setWeights (
    Weights weights ) [inline], [noexcept]
```

Set the weights for the layer.

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.

See also

[getWeights\(\)](#)

9.49.4 Member Data Documentation

9.49.4.1 mImpl

```
apiv::VConstantLayer* nvinfer1::IConstantLayer::mImpl [protected]
```

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

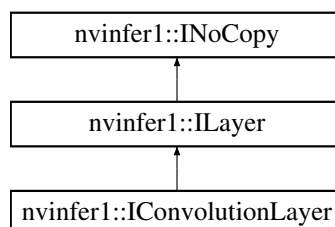
- [NvInfer.h](#)

9.50 nvinfer1::IConvolutionLayer Class Reference

A convolution layer in a network definition.

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

Inheritance diagram for `nvinfer1::IConvolutionLayer`:



Public Member Functions

- [TRT_DEPRECATED](#) void [setKernelSize](#) ([DimsHW](#) kernelSize) noexcept
Set the HW kernel size of the convolution.
- [TRT_DEPRECATED](#) [DimsHW](#) [getKernelSize](#) () const noexcept
Get the HW kernel size of the convolution.
- void [setNbOutputMaps](#) (int32_t nbOutputMaps) noexcept
Set the number of output maps for the convolution.
- int32_t [getNbOutputMaps](#) () const noexcept
Get the number of output maps for the convolution.
- [TRT_DEPRECATED](#) void [setStride](#) ([DimsHW](#) stride) noexcept
Get the stride of the convolution.
- [TRT_DEPRECATED](#) [DimsHW](#) [getStride](#) () const noexcept
Get the stride of the convolution.
- [TRT_DEPRECATED](#) void [setPadding](#) ([DimsHW](#) padding) noexcept
Set the padding of the convolution.
- [TRT_DEPRECATED](#) [DimsHW](#) [getPadding](#) () const noexcept
Get the padding of the convolution. If the padding is asymmetric, the pre-padding is returned.
- void [setNbGroups](#) (int32_t nbGroups) noexcept
Set the number of groups for a convolution.
- int32_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.
- [TRT_DEPRECATED](#) void [setDilation](#) ([DimsHW](#) dilation) noexcept
Set the dilation for a convolution.
- [TRT_DEPRECATED](#) [DimsHW](#) [getDilation](#) () const noexcept
Get the dilation for a convolution.
- void [setPrePadding](#) ([Dims](#) padding) noexcept
Set the multi-dimension pre-padding of the convolution.
- [Dims](#) [getPrePadding](#) () const noexcept
Get the pre-padding.
- void [setPostPadding](#) ([Dims](#) 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](#) 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](#) 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](#) 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](#) 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=default

Protected Attributes

- apiv::VConvolutionLayer * [mImpl](#)

9.50.1 Detailed Description

A convolution layer in a network definition.

This layer performs a correlation operation between 3-dimensional filter with a 4-dimensional tensor to produce another 4-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.

9.50.2 Constructor & Destructor Documentation

9.50.2.1 ~IConvolutionLayer()

```
virtual nvinfer1::IConvolutionLayer::~~IConvolutionLayer ( ) [protected], [virtual], [default],  
[noexcept]
```

9.50.3 Member Function Documentation

9.50.3.1 getBiasWeights()

```
Weights nvinfer1::IConvolutionLayer::getBiasWeights ( ) const [inline], [noexcept]
```

Get the bias weights for the convolution.

See also

[setBiasWeights\(\)](#)

9.50.3.2 getDilation()

```
TRT_DEPRECATED DimsHW nvinfer1::IConvolutionLayer::getDilation ( ) const [inline], [noexcept]
```

Get the dilation for a convolution.

See also

[setDilation\(\)](#)

Deprecated Superseded by `getDilationNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.50.3.3 getDilationNd()

```
Dims nvinfer1::IConvolutionLayer::getDilationNd ( ) const [inline], [noexcept]
```

Get the multi-dimension dilation of the convolution.

See also

[setDilation\(\)](#)

9.50.3.4 getKernelSize()

```
TRT_DEPRECATED DimsHW nvinfer1::IConvolutionLayer::getKernelSize ( ) const [inline], [noexcept]
```

Get the HW kernel size of the convolution.

See also

[setKernelSize\(\)](#)

Deprecated Superseded by getKernelSizeNd. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.50.3.5 getKernelSizeNd()

```
Dims nvinfer1::IConvolutionLayer::getKernelSizeNd ( ) const [inline], [noexcept]
```

Get the multi-dimension kernel size of the convolution.

See also

[setKernelSizeNd\(\)](#)

9.50.3.6 getKernelWeights()

```
Weights nvinfer1::IConvolutionLayer::getKernelWeights ( ) const [inline], [noexcept]
```

Get the kernel weights of the convolution.

See also

[setKernelWeights\(\)](#)

9.50.3.7 getNbGroups()

```
int32_t nvinfer1::IConvolutionLayer::getNbGroups ( ) const [inline], [noexcept]
```

Get the number of groups of the convolution.

See also

[setNbGroups\(\)](#)

9.50.3.8 getNbOutputMaps()

```
int32_t nvinfer1::IConvolutionLayer::getNbOutputMaps ( ) const [inline], [noexcept]
```

Get the number of output maps for the convolution.

See also

[setNbOutputMaps\(\)](#)

9.50.3.9 getPadding()

```
TRT_DEPRECATED DimsHW nvinfer1::IConvolutionLayer::getPadding ( ) const [inline], [noexcept]
```

Get the padding of the convolution. If the padding is asymmetric, the pre-padding is returned.

See also

[setPadding\(\)](#)

Deprecated Superseded by `getPaddingNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.50.3.10 getPaddingMode()

```
PaddingMode nvinfer1::IConvolutionLayer::getPaddingMode ( ) const [inline], [noexcept]
```

Get the padding mode.

Default: `kEXPLICIT_ROUND_DOWN`

See also

[setPaddingMode\(\)](#)

9.50.3.11 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\(\)](#)

9.50.3.12 getPostPadding()

```
Dims nvinfer1::IConvolutionLayer::getPostPadding ( ) const [inline], [noexcept]
```

Get the post-padding.

See also

[setPostPadding\(\)](#)

9.50.3.13 getPrePadding()

```
Dims nvinfer1::IConvolutionLayer::getPrePadding ( ) const [inline], [noexcept]
```

Get the pre-padding.

See also

[setPrePadding\(\)](#)

9.50.3.14 getStride()

```
TRT_DEPRECATED DimsHW nvinfer1::IConvolutionLayer::getStride ( ) const [inline], [noexcept]
```

Get the stride of the convolution.

Deprecated Superseded by `getStrideNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.50.3.15 getStrideNd()

```
Dims nvinfer1::IConvolutionLayer::getStrideNd ( ) const [inline], [noexcept]
```

Get the multi-dimension stride of the convolution.

See also

[setStrideNd\(\)](#)

9.50.3.16 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\(\)](#)

9.50.3.17 setDilation()

```
TRT_DEPRECATED void nvinfer1::IConvolutionLayer::setDilation (
    DimsHW dilation ) [inline], [noexcept]
```

Set the dilation for a convolution.

Default: (1,1)

If executing this layer on DLA, both height and width must be in the range [1,32].

See also

[getDilation\(\)](#)

Deprecated Superseded by `setDilationNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.50.3.18 setDilationNd()

```
void nvinfer1::IConvolutionLayer::setDilationNd (
    Dims 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

[getDilation\(\)](#)

9.50.3.19 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

<i>index</i>	the index of the input to modify.
<i>tensor</i>	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\(\)](#)

9.50.3.20 setKernelSize()

```
TRT_DEPRECATED void nvinfer1::IConvolutionLayer::setKernelSize (
    DimsHW kernelSize ) [inline], [noexcept]
```

Set the HW kernel size of the convolution.

If executing this layer on DLA, both height and width of kernel size must be in the range [1,32].

See also

[getKernelSize\(\)](#)

Deprecated Superseded by `setKernelSizeNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.50.3.21 `setKernelSizeNd()`

```
void nvinfer1::IConvolutionLayer::setKernelSizeNd (
    Dims 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\(\)](#)

9.50.3.22 `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\(\)](#)

9.50.3.23 setNbGroups()

```
void nvinfer1::IConvolutionLayer::setNbGroups (
    int32_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\(\)](#)

9.50.3.24 setNbOutputMaps()

```
void nvinfer1::IConvolutionLayer::setNbOutputMaps (
    int32_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\(\)](#)

9.50.3.25 setPadding()

```
TRT_DEPRECATED void nvinfer1::IConvolutionLayer::setPadding (
    DimsHW padding ) [inline], [noexcept]
```

Set the padding of the convolution.

The input will be zero-padded by this number of elements in the height and width directions. Padding is symmetric.

Default: (0,0)

If executing this layer on DLA, both height and width of padding must be in the range [0,31], and the padding size must be less than the kernel size.

See also

[getPadding\(\)](#)

Deprecated Superseded by setPaddingNd. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.50.3.26 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\(\)](#)

9.50.3.27 setPaddingNd()

```
void nvinfer1::IConvolutionLayer::setPaddingNd (
    Dims 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\(\)](#)

9.50.3.28 setPostPadding()

```
void nvinfer1::IConvolutionLayer::setPostPadding (
    Dims 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\(\)](#)

9.50.3.29 setPrePadding()

```
void nvinfer1::IConvolutionLayer::setPrePadding (
    Dims 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\(\)](#)

9.50.3.30 setStride()

```
TRT_DEPRECATED void nvinfer1::IConvolutionLayer::setStride (
    DimsHW stride ) [inline], [noexcept]
```

Get the stride of the convolution.

Default: (1,1)

If executing this layer on DLA, both height and width of stride must be in the range [1,8].

See also

[getStride\(\)](#)

Deprecated Superseded by `setStrideNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.50.3.31 setStrideNd()

```
void nvinfer1::IConvolutionLayer::setStrideNd (
    Dims 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\(\)](#) [setStride\(\)](#) [getStride\(\)](#)

9.50.4 Member Data Documentation

9.50.4.1 mImpl

```
apiv::VConvolutionLayer* nvinfer1::IConvolutionLayer::mImpl [protected]
```

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

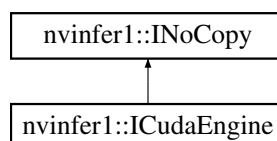
- [NvInfer.h](#)

9.51 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=default
- [TRT_DEPRECATED](#) int32_t [getNbBindings](#) () const noexcept
Get the number of binding indices.
- [TRT_DEPRECATED](#) int32_t [getBindingIndex](#) (char const *name) const noexcept
Retrieve the binding index for a named tensor.
- [TRT_DEPRECATED](#) char const * [getBindingName](#) (int32_t bindingIndex) const noexcept
Retrieve the name corresponding to a binding index.
- [TRT_DEPRECATED](#) bool [bindingIsInput](#) (int32_t bindingIndex) const noexcept
Determine whether a binding is an input binding.
- [TRT_DEPRECATED](#) Dims [getBindingDimensions](#) (int32_t bindingIndex) const noexcept
Get the dimensions of a binding.
- Dims [getTensorShape](#) (char const *tensorName) const noexcept
Get shape of an input or output tensor.
- [TRT_DEPRECATED](#) DataType [getBindingDataType](#) (int32_t bindingIndex) const noexcept
Determine the required data type for a buffer from its binding index.
- DataType [getTensorDataType](#) (char const *tensorName) const noexcept
Determine the required data type for a buffer from its tensor name.
- [TRT_DEPRECATED](#) int32_t [getMaxBatchSize](#) () const noexcept
Get the maximum batch size which can be used for inference. Should only be called if the engine is built from an [INetworkDefinition](#) with implicit batch dimension mode.
- 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](#) () noexcept
Create an execution context.
- [TRT_DEPRECATED](#) void [destroy](#) () noexcept
Destroy this object;.
- [TRT_DEPRECATED](#) TensorLocation [getLocation](#) (int32_t bindingIndex) const noexcept
Get location of binding.
- 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.
- IExecutionContext * [createExecutionContextWithoutDeviceMemory](#) () noexcept
create an execution context without any device memory allocated
- size_t [getDeviceMemorySize](#) () const noexcept
Return the amount of device memory required by an execution context.
- bool [isRefittable](#) () const noexcept
Return true if an engine can be refit.
- [TRT_DEPRECATED](#) int32_t [getBindingBytesPerComponent](#) (int32_t bindingIndex) const noexcept
Return the number of bytes per component of an element.
- int32_t [getTensorBytesPerComponent](#) (char const *tensorName) const noexcept

Return the number of bytes per component of an element, or -1 if the provided name does not map to an input or output tensor.

- [TRT_DEPRECATED](#) `int32_t getBindingComponentsPerElement (int32_t bindingIndex) const noexcept`

Return the number of components included in one element.

- `int32_t getTensorComponentsPerElement (char const *tensorName) const noexcept`

Return the number of components included in one element, or -1 if the provided name does not map to an input or output tensor.

- [TRT_DEPRECATED](#) `TensorFormat getBindingFormat (int32_t bindingIndex) const noexcept`

Return the binding format.

- `TensorFormat getTensorFormat (char const *tensorName) const noexcept`

Return the binding format, or [TensorFormat::kLINEAR](#) if the provided name does not map to an input or output tensor.

- [TRT_DEPRECATED](#) `char const * getBindingFormatDesc (int32_t bindingIndex) const noexcept`

Return the human readable description of the tensor format, or nullptr 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.

- [TRT_DEPRECATED](#) `int32_t getBindingVectorizedDim (int32_t bindingIndex) const noexcept`

Return the dimension index that the buffer is vectorized, or -1 if the name is not found.

- `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.

- `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.

- [TRT_DEPRECATED](#) `Dims getProfileDimensions (int32_t bindingIndex, int32_t profileIndex, OptProfileSelector select) const noexcept`

Get the minimum / optimum / maximum dimensions for a particular input binding under an optimization profile.

- `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.

- [TRT_DEPRECATED](#) `int32_t const * getProfileShapeValues (int32_t profileIndex, int32_t inputIndex, OptProfileSelector select) const noexcept`

Get minimum / optimum / maximum values for an input shape binding under an optimization profile.

- [TRT_DEPRECATED](#) `bool isShapeBinding (int32_t bindingIndex) const noexcept`

True if tensor is required as input for shape calculations or output from them.

- [TRT_DEPRECATED](#) `bool isExecutionBinding (int32_t bindingIndex) const noexcept`

True if pointer to tensor data is required for execution phase, false if nullptr can be supplied.

- `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.

- `bool hasImplicitBatchDimension () const noexcept`

Query whether the engine was built with an implicit batch dimension.

- `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.

Protected Attributes

- `apiv::VCudaEngine` * [mImpl](#)

Additional Inherited Members

9.51.1 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.

9.51.2 Constructor & Destructor Documentation

9.51.2.1 ~ICudaEngine()

```
virtual nvinfer1::ICudaEngine::~~ICudaEngine ( ) [virtual], [default], [noexcept]
```

9.51.3 Member Function Documentation

9.51.3.1 bindingIsInput()

```
TRT\_DEPRECATED bool nvinfer1::ICudaEngine::bindingIsInput (
    int32_t bindingIndex ) const [inline], [noexcept]
```

Determine whether a binding is an input binding.

Parameters

<i>bindingIndex</i>	The binding index.
---------------------	--------------------

Returns

True if the index corresponds to an input binding and the index is in range.

Deprecated Deprecated in TensorRT 8.5. Superseded by [getTensorIOMode\(\)](#).

See also

[getTensorIOMode\(\)](#)

9.51.3.2 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](#).

9.51.3.3 createExecutionContext()

```
IEExecutionContext * nvinfer1::ICudaEngine::createExecutionContext ( ) [inline], [noexcept]
```

Create an execution context.

The execution context created will call `setOptimizationProfile(0)` implicitly if there are no other execution contexts assigned to optimization profile 0. This functionality is deprecated in TensorRT 8.6 and will instead default all optimization profiles to 0 starting in TensorRT 9.0. If an error recorder has been set for the engine, it will also be passed to the execution context.

See also

[IEExecutionContext](#).

[IEExecutionContext::setOptimizationProfile\(\)](#)

9.51.3.4 createExecutionContextWithoutDeviceMemory()

```
IEExecutionContext * nvinfer1::ICudaEngine::createExecutionContextWithoutDeviceMemory ( ) [inline],
[noexcept]
```

create an execution context without any device memory allocated

The memory for execution of this device context must be supplied by the application.

9.51.3.5 destroy()

```
TRT_DEPRECATED void nvinfer1::ICudaEngine::destroy ( ) [inline], [noexcept]
```

Destroy this object;.

Deprecated Deprecated in TRT 8.0. Superseded by `delete`.

Warning

Calling `destroy` on a managed pointer will result in a double-free error.

9.51.3.6 getBindingBytesPerComponent()

```
TRT_DEPRECATED int32_t nvinfer1::ICudaEngine::getBindingBytesPerComponent (
    int32_t bindingIndex ) const [inline], [noexcept]
```

Return the number of bytes per component of an element.

The vector component size is returned if `getBindingVectorizedDim()` != -1.

Parameters

<i>bindingIndex</i>	The binding Index.
---------------------	--------------------

Deprecated Deprecated in TensorRT 8.5. Superseded by `getTensorBytesPerComponent()`.

See also

[getBindingVectorizedDim\(\)](#)
[getTensorBytesPerComponent\(\)](#)

9.51.3.7 `getBindingComponentsPerElement()`

```
TRT_DEPRECATED int32_t nvinfer1::ICudaEngine::getBindingComponentsPerElement (
    int32_t bindingIndex ) const [inline], [noexcept]
```

Return the number of components included in one element.

The number of elements in the vectors is returned if `getBindingVectorizedDim() != -1`.

Parameters

<i>bindingIndex</i>	The binding Index.
---------------------	--------------------

Deprecated Deprecated in TensorRT 8.5. Superseded by `getTensorComponentsPerElement()`.

See also

[`getBindingVectorizedDim\(\)`](#)

9.51.3.8 `getBindingDataType()`

```
TRT_DEPRECATED DataType nvinfer1::ICudaEngine::getBindingDataType (
    int32_t bindingIndex ) const [inline], [noexcept]
```

Determine the required data type for a buffer from its binding index.

Parameters

<i>bindingIndex</i>	The binding index.
---------------------	--------------------

Returns

The type of the data in the buffer.

Deprecated Deprecated in TensorRT 8.5. Superseded by `getTensorDataType()`.

See also

[`getTensorDataType\(\)`](#)

9.51.3.9 getBindingDimensions()

```
TRT_DEPRECATED Dims nvinfer1::ICudaEngine::getBindingDimensions (
    int32_t bindingIndex ) const [inline], [noexcept]
```

Get the dimensions of a binding.

Parameters

<i>bindingIndex</i>	The binding index.
---------------------	--------------------

Returns

The dimensions of the binding if the index is in range, otherwise [Dims\(\)](#). Has -1 for any dimension that varies within the optimization profile.

For example, suppose an [INetworkDefinition](#) has an input with shape [-1,-1] that becomes a binding b in the engine. If the associated optimization profile specifies that b has minimum dimensions as [6,9] and maximum dimensions [7,9], `getBindingDimensions(b)` returns [-1,9], despite the second dimension being dynamic in the [INetworkDefinition](#).

Because each optimization profile has separate bindings, the returned value can differ across profiles. Consider another binding b' for the same network input, but for another optimization profile. If that other profile specifies minimum dimensions [5,8] and maximum dimensions [5,9], `getBindingDimensions(b')` returns [5,-1].

Deprecated Deprecated in TensorRT 8.5. Superseded by [getTensorShape\(\)](#).

See also

[getTensorShape\(\)](#)

9.51.3.10 getBindingFormat()

```
TRT_DEPRECATED TensorFormat nvinfer1::ICudaEngine::getBindingFormat (
    int32_t bindingIndex ) const [inline], [noexcept]
```

Return the binding format.

Parameters

<i>bindingIndex</i>	The binding Index.
---------------------	--------------------

Deprecated Deprecated in TensorRT 8.5. Superseded by [getTensorFormat\(\)](#).

See also

[getTensorFormat\(\)](#)

9.51.3.11 getBindingFormatDesc()

```
TRT_DEPRECATED char const * nvinfer1::ICudaEngine::getBindingFormatDesc (
    int32_t bindingIndex ) const [inline], [noexcept]
```

Return the human readable description of the tensor format, or nullptr 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

<i>bindingIndex</i>	The binding Index.
---------------------	--------------------

Deprecated Deprecated in TensorRT 8.5. Superseded by [getTensorFormatDesc\(\)](#).

See also

[getTensorFormatDesc\(\)](#)

9.51.3.12 getBindingIndex()

```
TRT_DEPRECATED int32_t nvinfer1::ICudaEngine::getBindingIndex (
    char const * name ) const [inline], [noexcept]
```

Retrieve the binding index for a named tensor.

[IExecutionContext::enqueueV2\(\)](#) and [IExecutionContext::executeV2\(\)](#) require an array of buffers.

Engine bindings map from tensor names to indices in this array. Binding indices are assigned at engine build time, and take values in the range [0 ... n-1] where n is the total number of inputs and outputs.

To get the binding index of the name in an optimization profile with index $k > 0$, mangle the name by appending "[profile k]", as described for method [getBindingName\(\)](#).

Parameters

<i>name</i>	The tensor name.
-------------	------------------

Returns

The binding index for the named tensor, or -1 if the provided name does not map to an input or output tensor.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

Deprecated Deprecated in TensorRT 8.5. Superseded by name-based methods. Use them instead of binding-index based methods.

See also

[getNbBindings\(\)](#) [getBindingName\(\)](#)

9.51.3.13 getBindingName()

```
TRT_DEPRECATED char const * nvinfer1::ICudaEngine::getBindingName (
    int32_t bindingIndex ) const [inline], [noexcept]
```

Retrieve the name corresponding to a binding index.

This is the reverse mapping to that provided by [getBindingIndex\(\)](#).

For optimization profiles with an index $k > 0$, the name is mangled by appending " [profile k]", with k written in decimal. For example, if the tensor in the [INetworkDefinition](#) had the name "foo", and bindingIndex refers to that tensor in the optimization profile with index 3, getBindingName returns "foo [profile 3]".

Parameters

<i>bindingIndex</i>	The binding index.
---------------------	--------------------

Returns

The name corresponding to the index, or nullptr if the index is out of range.

Deprecated Deprecated in TensorRT 8.5. Superseded by name-based methods. Use them instead of binding-index based methods.

See also

[getBindingIndex\(\)](#)

9.51.3.14 getBindingVectorizedDim()

```
TRT_DEPRECATED int32_t nvinfer1::ICudaEngine::getBindingVectorizedDim (
    int32_t bindingIndex ) const [inline], [noexcept]
```

Return the dimension index that the buffer is vectorized, or -1 is the name is not found.

Specifically -1 is returned if scalars per vector is 1.

Parameters

<i>bindingIndex</i>	The binding Index.
---------------------	--------------------

Deprecated Deprecated in TensorRT 8.5. Superseded by [getTensorVectorizedDim\(\)](#).

See also

[getTensorVectorizedDim\(\)](#)

9.51.3.15 getDeviceMemorySize()

```
size_t nvinfer1::ICudaEngine::getDeviceMemorySize ( ) const [inline], [noexcept]
```

Return the amount of device memory required by an execution context.

See also

[IExecutionContext::setDeviceMemory\(\)](#)

9.51.3.16 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.

9.51.3.17 getErrorRecorder()

```
IErrRecorder * 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 [IErrRecorder](#) object that has been registered.

See also

[setErrorRecorder\(\)](#)

9.51.3.18 getHardwareCompatibilityLevel()

```
HardwareCompatibilityLevel nvinfer1::ICudaEngine::getHardwareCompatibilityLevel ( ) const [inline], [noexcept]
```

Return the hardware compatibility level of this engine.

Returns

hardwareCompatibilityLevel The level of hardware compatibility.

This is only supported for Ampere and newer architectures.

9.51.3.19 getIOTensorName()

```
char const * nvinfer1::ICudaEngine::getIOTensorName (
    int32_t index ) const [inline], [noexcept]
```

Return name of an IO tensor.

Parameters

<i>index</i>	value between 0 and getNbIOTensors()-1
--------------	--

See also

[getNbIOTensors\(\)](#)

9.51.3.20 getLocation()

```
TRT_DEPRECATED TensorLocation nvinfer1::ICudaEngine::getLocation (
    int32_t bindingIndex ) const [inline], [noexcept]
```

Get location of binding.

This lets you know whether the binding should be a pointer to device or host memory.

Parameters

<i>bindingIndex</i>	The binding index.
---------------------	--------------------

Returns

The location of the bound tensor with given index.

Deprecated Deprecated in TensorRT 8.5. Superseded by [getTensorLocation\(\)](#).

See also

[ITensor::setLocation\(\)](#) [ITensor::getLocation\(\)](#)
[getTensorLocation\(\)](#)

9.51.3.21 getMaxBatchSize()

```
TRT_DEPRECATED int32_t nvinfer1::ICudaEngine::getMaxBatchSize ( ) const [inline], [noexcept]
```

Get the maximum batch size which can be used for inference. Should only be called if the engine is built from an [INetworkDefinition](#) with implicit batch dimension mode.

Returns

The maximum batch size for this engine.

Warning

For an engine built from an [INetworkDefinition](#) with explicit batch dimension mode, this will always return 1.

Deprecated Deprecated in TensorRT 8.4.

9.51.3.22 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.

9.51.3.23 getNbBindings()

```
TRT_DEPRECATED int32_t nvinfer1::ICudaEngine::getNbBindings ( ) const [inline], [noexcept]
```

Get the number of binding indices.

There are separate binding indices for each optimization profile. This method returns the total over all profiles. If the engine has been built for K profiles, the first [getNbBindings\(\)](#) / K bindings are used by profile number 0, the following [getNbBindings\(\)](#) / K bindings are used by profile number 1 etc.

Deprecated Deprecated in TensorRT 8.5. Superseded by [getNbIOTensors](#).

See also

[getBindingIndex\(\)](#)

9.51.3.24 `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\(\)](#)

9.51.3.25 `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.

9.51.3.26 `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::setOptimizationProfile\(\)](#)

9.51.3.27 `getProfileDimensions()`

```
TRT_DEPRECATED Dims nvinfer1::ICudaEngine::getProfileDimensions (
    int32_t bindingIndex,
    int32_t profileIndex,
    OptProfileSelector select ) const [inline], [noexcept]
```

Get the minimum / optimum / maximum dimensions for a particular input binding under an optimization profile.

Parameters

<i>bindingIndex</i>	The input binding index, which must belong to the given profile, or be between 0 and bindingsPerProfile-1 as described below.
<i>profileIndex</i>	The profile index, which must be between 0 and getNbOptimizationProfiles() -1.
<i>select</i>	Whether to query the minimum, optimum, or maximum dimensions for this binding.

Returns

The minimum / optimum / maximum dimensions for this binding in this profile. If the profileIndex or bindingIndex are invalid, return [Dims](#) with nbDims=-1.

For backwards compatibility with earlier versions of TensorRT, if the bindingIndex does not belong to the current optimization profile, but is between 0 and bindingsPerProfile-1, where bindingsPerProfile = [getNbBindings\(\)](#)/[getNbOptimizationProfiles\(\)](#), then a corrected bindingIndex is used instead, computed by:

```
profileIndex * bindingsPerProfile + bindingIndex % bindingsPerProfile
```

Otherwise the bindingIndex is considered invalid.

Deprecated Deprecated in TensorRT 8.5. Superseded by [getProfileShape\(\)](#).

See also

[getProfileShape\(\)](#)

9.51.3.28 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

<i>tensorName</i>	The name of an input tensor.
<i>profileIndex</i>	The profile index, which must be between 0 and getNbOptimizationProfiles() -1.
<i>select</i>	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.

9.51.3.29 `getProfileShapeValues()`

```
TRT_DEPRECATED int32_t const * nvinfer1::ICudaEngine::getProfileShapeValues (
    int32_t profileIndex,
    int32_t inputIndex,
    OptProfileSelector select ) const [inline], [noexcept]
```

Get minimum / optimum / maximum values for an input shape binding under an optimization profile.

Parameters

<i>profileIndex</i>	The profile index (must be between 0 and <code>getNbOptimizationProfiles()-1</code>)
<i>inputIndex</i>	The input index (must be between 0 and <code>getNbBindings() - 1</code>)
<i>select</i>	Whether to query the minimum, optimum, or maximum shape values for this binding.

Returns

If the binding is an input shape binding, return a pointer to an array that has the same number of elements as the corresponding tensor, i.e. 1 if `dims.nbDims == 0`, or `dims.d[0]` if `dims.nbDims == 1`, where `dims = getBindingDimensions(inputIndex)`. The array contains the elementwise minimum / optimum / maximum values for this shape binding under the profile. If either of the indices is out of range, or if the binding is not an input shape binding, return `nullptr`.

For backwards compatibility with earlier versions of TensorRT, a bindingIndex that does not belong to the profile is corrected as described for `getProfileDimensions()`.

Deprecated Deprecated in TensorRT 8.5. Superseded by `getShapeValues()`. Difference between Execution and shape tensor is superficial since TensorRT 8.5.

See also

[getProfileDimensions\(\)](#) [getShapeValues\(\)](#)

9.51.3.30 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\(\)](#)

9.51.3.31 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 [IBuilderConfig::setTacticSources\(\)](#). 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\(\)](#)

9.51.3.32 getTensorBytesPerComponent()

```
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 provided name does not map to an input or output tensor.

The vector component size is returned if [getTensorVectorizedDim\(\)](#) != -1.

Parameters

<i>tensorName</i>	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.

See also

[getTensorVectorizedDim\(\)](#)

9.51.3.33 `getTensorComponentsPerElement()`

```
int32_t nvinfer1::ICudaEngine::getTensorComponentsPerElement (
    char const * tensorName ) const [inline], [noexcept]
```

Return the number of components included in one element, or -1 if the provided name does not map to an input or output tensor.

The number of elements in the vectors is returned if [getTensorVectorizedDim\(\)](#) != -1.

Parameters

<i>tensorName</i>	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.

See also

[getTensorVectorizedDim\(\)](#)

9.51.3.34 `getTensorDataType()`

```
DataType nvinfer1::ICudaEngine::getTensorDataType (
    char const * tensorName ) const [inline], [noexcept]
```

Determine the required data type for a buffer from its tensor name.

Parameters

<i>tensorName</i>	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.

9.51.3.35 getTensorFormat()

```
TensorFormat nvinfer1::ICudaEngine::getTensorFormat (
    char const * tensorName ) const [inline], [noexcept]
```

Return the binding format, or [TensorFormat::kLINEAR](#) if the provided name does not map to an input or output tensor.

Parameters

<i>tensorName</i>	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.

9.51.3.36 getTensorFormatDesc()

```
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

<i>tensorName</i>	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.

9.51.3.37 getTensorIOMode()

```
TensorIOMode nvinfer1::ICudaEngine::getTensorIOMode (
    char const * tensorName ) const [inline], [noexcept]
```

Determine whether a tensor is an input or output tensor.

Parameters

<i>tensorName</i>	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.

9.51.3.38 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

<i>tensorName</i>	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.

9.51.3.39 getTensorShape()

```
Dims nvinfer1::ICudaEngine::getTensorShape (
    char const * tensorName ) const [inline], [noexcept]
```

Get shape of an input or output tensor.

Parameters

<i>tensorName</i>	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.

9.51.3.40 getTensorVectorizedDim()

```
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

<i>tensorName</i>	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.

9.51.3.41 hasImplicitBatchDimension()

```
bool nvinfer1::ICudaEngine::hasImplicitBatchDimension ( ) const [inline], [noexcept]
```

Query whether the engine was built with an implicit batch dimension.

Returns

True if tensors have implicit batch dimension, false otherwise.

This is an engine-wide property. Either all tensors in the engine have an implicit batch dimension or none of them do.

[hasImplicitBatchDimension\(\)](#) is true if and only if the [INetworkDefinition](#) from which this engine was built was created with [createNetworkV2\(\)](#) without [NetworkDefinitionCreationFlag::kEXPLICIT_BATCH](#) flag.

See also

[createNetworkV2](#)

9.51.3.42 isExecutionBinding()

```
TRT\_DEPRECATED bool nvinfer1::ICudaEngine::isExecutionBinding (
    int32_t bindingIndex ) const [inline], [noexcept]
```

True if pointer to tensor data is required for execution phase, false if nullptr can be supplied.

For example, if a network uses an input tensor with binding *i* ONLY as the "reshape dimensions" input of [IShuffleLayer](#), then [isExecutionBinding\(i\)](#) is false, and a nullptr can be supplied for it when calling [IExecutionContext::execute](#) or [IExecutionContext::enqueue](#).

Deprecated No name-based equivalent replacement. Use [getTensorLocation\(\)](#) instead to know the location of tensor data. Distinction between execution binding and shape binding is superficial since TensorRT 8.5.

See also

[isShapeBinding\(\)](#) [getTensorLocation\(\)](#)

9.51.3.43 isRefittable()

```
bool nvinfer1::ICudaEngine::isRefittable ( ) const [inline], [noexcept]
```

Return true if an engine can be refit.

See also

[nvinfer1::createInferRefitter\(\)](#)

9.51.3.44 isShapeBinding()

```
TRT_DEPRECATED bool nvinfer1::ICudaEngine::isShapeBinding (
    int32_t bindingIndex ) const [inline], [noexcept]
```

True if tensor is required as input for shape calculations or output from them.

TensorRT evaluates a network in two phases:

1. Compute shape information required to determine memory allocation requirements and validate that runtime sizes make sense.
2. Process tensors on the device.

Some tensors are required in phase 1. These tensors are called "shape tensors", and always have type Int32 and no more than one dimension. These tensors are not always shapes themselves, but might be used to calculate tensor shapes for phase 2.

isShapeBinding(i) returns true if the tensor is a required input or an output computed in phase 1. isExecutionBinding(i) returns true if the tensor is a required input or an output computed in phase 2.

For example, if a network uses an input tensor with binding i as an addend to an [IElementWiseLayer](#) that computes the "reshape dimensions" for [IShuffleLayer](#), then isShapeBinding(i) == true.

It's possible to have a tensor be required by both phases. For instance, a tensor can be used for the "reshape dimensions" and as the indices for an [IGatherLayer](#) collecting floating-point data.

It's also possible to have a tensor be required by neither phase, but nonetheless shows up in the engine's inputs. For example, if an input tensor is used only as an input to [IShapeLayer](#), only its shape matters and its values are irrelevant.

Deprecated Use name-based [isShapeInferenceIO\(\)](#) instead to know whether a tensor is a shape tensor.

See also

[isExecutionBinding\(\)](#) [isShapeInferenceIO\(\)](#)

9.51.3.45 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".

9.51.3.46 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\(\)](#)

9.51.3.47 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

<i>recorder</i>	The error recorder to register with this interface.
-----------------	---

See also

[getErrorRecorder\(\)](#)

9.51.4 Member Data Documentation

9.51.4.1 mImpl

```
apiv::VCudaEngine* nvinfer1::ICudaEngine::mImpl [protected]
```

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

- [NvInferRuntime.h](#)

9.52 nvinfer1::safe::ICudaEngine Class Reference

A functionally safe engine for executing inference on a built network.

```
#include <NvInferSafeRuntime.h>
```

Public Member Functions

- virtual [TRT_DEPRECATED](#) std::int32_t [getNbBindings](#) () const noexcept=0
Get the number of binding indices.
- virtual [TRT_DEPRECATED](#) std::int32_t [getBindingIndex](#) ([AsciiChar](#) const *const name) const noexcept=0
Retrieve the binding index for a named tensor.
- virtual [TRT_DEPRECATED](#) [AsciiChar](#) const * [getBindingName](#) (std::int32_t const bindingIndex) const noexcept=0
Retrieve the name corresponding to a binding index.
- virtual [TRT_DEPRECATED](#) bool [bindingIsInput](#) (std::int32_t const bindingIndex) const noexcept=0
Determine whether a binding is an input binding.
- virtual [TRT_DEPRECATED](#) [Dims](#) [getBindingDimensions](#) (std::int32_t const bindingIndex) const noexcept=0
Get the dimensions of a binding.
- virtual [TRT_DEPRECATED](#) [DataType](#) [getBindingDataType](#) (std::int32_t const bindingIndex) const noexcept=0
Determine the required data type for a buffer from its binding index.
- virtual [IExecutionContext](#) * [createExecutionContext](#) () noexcept=0
Create an execution context.

- virtual [IExecutionContext](#) * [createExecutionContextWithoutDeviceMemory](#) () noexcept=0
Create an execution context without any device memory allocated.
- virtual [size_t](#) [getDeviceMemorySize](#) () const noexcept=0
Return the amount of device memory required by an execution context.
- virtual [TRT_DEPRECATED](#) [std::int32_t](#) [getBindingBytesPerComponent](#) ([std::int32_t](#) const bindingIndex) const noexcept=0
Return the number of bytes per component of an element.
- virtual [TRT_DEPRECATED](#) [std::int32_t](#) [getBindingComponentsPerElement](#) ([std::int32_t](#) const bindingIndex) const noexcept=0
Return the number of components included in one element.
- virtual [TRT_DEPRECATED](#) [TensorFormat](#) [getBindingFormat](#) ([std::int32_t](#) const bindingIndex) const noexcept=0
Return the binding format.
- virtual [TRT_DEPRECATED](#) [std::int32_t](#) [getBindingVectorizedDim](#) ([std::int32_t](#) const bindingIndex) const noexcept=0
Return the dimension index that the buffer is vectorized.
- virtual [AsciiChar](#) const * [getName](#) () const noexcept=0
Returns the name of the network associated with the engine.
- 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.
- [ICudaEngine](#) ()=default
- virtual [~ICudaEngine](#) () noexcept=default
- [ICudaEngine](#) ([ICudaEngine](#) const &)=delete
- [ICudaEngine](#) ([ICudaEngine](#) &&)=delete
- [ICudaEngine](#) & operator= ([ICudaEngine](#) const &) &=delete
- [ICudaEngine](#) & operator= ([ICudaEngine](#) &&) &=delete
- virtual [Dims](#) [getTensorShape](#) ([AsciiChar](#) const *tensorName) const noexcept=0
Get extent of an input or output tensor.
- virtual [DataType](#) [getTensorDataType](#) ([AsciiChar](#) const *tensorName) const noexcept=0
Determine the required data type for a buffer from its tensor name.
- virtual [TensorIOMode](#) [getTensorIOMode](#) ([AsciiChar](#) const *tensorName) const noexcept=0
Determine whether a tensor is an input or output tensor.
- virtual [std::int32_t](#) [getTensorBytesPerComponent](#) ([AsciiChar](#) const *tensorName) const noexcept=0
Return the number of bytes per component of an element.
- virtual [std::int32_t](#) [getTensorComponentsPerElement](#) ([AsciiChar](#) const *tensorName) const noexcept=0
Return the number of components included in one element.
- virtual [TensorFormat](#) [getTensorFormat](#) ([AsciiChar](#) const *tensorName) const noexcept=0
Return the tensor format.
- virtual [std::int32_t](#) [getTensorVectorizedDim](#) ([AsciiChar](#) const *tensorName) const noexcept=0
Return the dimension index along which buffer is vectorized.
- virtual [std::int32_t](#) [getNbIOTensors](#) () const noexcept=0
Return the number of input and output tensors for the network from which the engine was built.
- virtual [AsciiChar](#) const * [getIOTensorName](#) ([std::int32_t](#) const index) const noexcept=0
Return the name of an IO tensor.

9.52.1 Detailed Description

A functionally safe engine 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.

9.52.2 Constructor & Destructor Documentation

9.52.2.1 ICudaEngine() [1/3]

```
nvinfer1::safe::ICudaEngine::ICudaEngine ( ) [default]
```

9.52.2.2 ~ICudaEngine()

```
virtual nvinfer1::safe::ICudaEngine::~~ICudaEngine ( ) [virtual], [default], [noexcept]
```

9.52.2.3 ICudaEngine() [2/3]

```
nvinfer1::safe::ICudaEngine::ICudaEngine (
    ICudaEngine const & ) [delete]
```

9.52.2.4 ICudaEngine() [3/3]

```
nvinfer1::safe::ICudaEngine::ICudaEngine (
    ICudaEngine && ) [delete]
```

9.52.3 Member Function Documentation

9.52.3.1 bindingIsInput()

```
virtual TRT\_DEPRECATED bool nvinfer1::safe::ICudaEngine::bindingIsInput (
    std::int32_t const bindingIndex ) const [pure virtual], [noexcept]
```

Determine whether a binding is an input binding.

Parameters

<i>bindingIndex</i>	The binding index.
---------------------	--------------------

Returns

True if the index corresponds to an input binding and the index is in range.

Deprecated Deprecated in TensorRT 8.5. Superseded by `tensorIOMode()`.

See also

`safe::ICudaEngine::tensorIOMode()`

Usage considerations

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

9.52.3.2 createExecutionContext()

```
virtual IExecutionContext * nvinfer1::safe::ICudaEngine::createExecutionContext ( ) [pure virtual],
[noexcept]
```

Create an execution context.

See also

[safe::IExecutionContext](#).

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes; if `createExecutionContext` fails, users should treat this as a critical error and not perform any subsequent TensorRT operations apart from outputting the error logs.

9.52.3.3 createExecutionContextWithoutDeviceMemory()

```
virtual IExecutionContext * nvinfer1::safe::ICudaEngine::createExecutionContextWithoutDeviceMemory
( ) [pure virtual], [noexcept]
```

Create an execution context without any device memory allocated.

The memory for execution of this device context must be supplied by the application.

See also

[getDeviceMemorySize\(\)](#) [safe::IExecutionContext::setDeviceMemory\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes; if createExecutionContext fails, users should treat this as a critical error and not perform any subsequent TensorRT operations apart from outputting the error logs.

9.52.3.4 getBindingBytesPerComponent()

```
virtual TRT_DEPRECATED std::int32_t nvinfer1::safe::ICudaEngine::getBindingBytesPerComponent (
    std::int32_t const bindingIndex ) const [pure virtual], [noexcept]
```

Return the number of bytes per component of an element.

The vector component size is returned if [getBindingVectorizedDim\(\)](#) != -1.

Parameters

<i>bindingIndex</i>	The binding Index.
---------------------	--------------------

Deprecated Deprecated in TensorRT 8.5. Superseded by [getTensorBytesPerComponent\(\)](#).

See also

[safe::ICudaEngine::getTensorBytesPerComponent\(\)](#)

Usage considerations

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

9.52.3.5 getBindingComponentsPerElement()

```
virtual TRT\_DEPRECATED std::int32_t nvinfer1::safe::ICudaEngine::getBindingComponentsPerElement (
    std::int32_t const bindingIndex ) const [pure virtual], [noexcept]
```

Return the number of components included in one element.

The number of elements in the vectors is returned if [getBindingVectorizedDim\(\)](#) != -1.

Parameters

<i>bindingIndex</i>	The binding Index.
---------------------	--------------------

Deprecated Deprecated in TensorRT 8.5. Superseded by [getTensorComponentsPerElement\(\)](#).

See also

[safe::ICudaEngine::getTensorComponentsPerElement\(\)](#)

Usage considerations

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

9.52.3.6 getBindingDataType()

```
virtual TRT\_DEPRECATED DataType nvinfer1::safe::ICudaEngine::getBindingDataType (
    std::int32_t const bindingIndex ) const [pure virtual], [noexcept]
```

Determine the required data type for a buffer from its binding index.

Parameters

<i>bindingIndex</i>	The binding index.
---------------------	--------------------

Returns

The type of the data in the buffer.

Deprecated Deprecated in TensorRT 8.5. Superseded by [getTensorDataType\(\)](#).

See also

[safe::ICudaEngine::getTensorDataType\(\)](#)

Usage considerations

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

9.52.3.7 getBindingDimensions()

```
virtual TRT\_DEPRECATED Dims nvinfer1::safe::ICudaEngine::getBindingDimensions (
    std::int32_t const bindingIndex ) const [pure virtual], [noexcept]
```

Get the dimensions of a binding.

Parameters

<i>bindingIndex</i>	The binding index.
---------------------	--------------------

Returns

The dimensions of the binding if the index is in range, otherwise [Dims\(\)](#)

Deprecated Deprecated in TensorRT 8.5. Superseded by [getTensorShape\(\)](#).

See also

[safe::ICudaEngine::getTensorShape\(\)](#)

Usage considerations

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

9.52.3.8 getBindingFormat()

```
virtual TRT\_DEPRECATED TensorFormat nvinfer1::safe::ICudaEngine::getBindingFormat (
    std::int32_t const bindingIndex ) const [pure virtual], [noexcept]
```

Return the binding format.

Parameters

<i>bindingIndex</i>	The binding Index.
---------------------	--------------------

Deprecated Deprecated in TensorRT 8.5. Superseded by [getTensorFormat\(\)](#).

See also

[safe::ICudaEngine::getTensorFormat\(\)](#)

Usage considerations

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

9.52.3.9 getBindingIndex()

```
virtual TRT\_DEPRECATED std::int32_t nvinfer1::safe::ICudaEngine::getBindingIndex (
    AsciiChar const *const name ) const [pure virtual], [noexcept]
```

Retrieve the binding index for a named tensor.

[safe::IExecutionContext::enqueueV2\(\)](#) requires an array of buffers. Engine bindings map from tensor names to indices in this array. Binding indices are assigned at engine build time, and take values in the range [0 ... n-1] where n is the total number of inputs and outputs.

Warning

Strings passed to the runtime must be 1024 characters or less including NULL terminator and must be NULL terminated.

Parameters

<i>name</i>	The tensor name.
-------------	------------------

Returns

The binding index for the named tensor, or -1 if the name is not found.

Deprecated Deprecated in TensorRT 8.5. Superseded by name-based methods. Use them instead of binding-index based methods.

Usage considerations

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

9.52.3.10 getBindingName()

```
virtual TRT\_DEPRECATED AsciiChar const * nvinfer1::safe::ICudaEngine::getBindingName (
    std::int32_t const bindingIndex ) const [pure virtual], [noexcept]
```

Retrieve the name corresponding to a binding index.

This is the reverse mapping to that provided by [getBindingIndex\(\)](#).

Parameters

<i>bindingIndex</i>	The binding index.
---------------------	--------------------

Returns

The name corresponding to the index, or nullptr if the index is out of range.

Deprecated Deprecated in TensorRT 8.5. Superseded by name-based methods. Use them instead of binding-index based methods.

See also

[getBindingIndex\(\)](#)

Usage considerations

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

9.52.3.11 getBindingVectorizedDim()

```
virtual TRT\_DEPRECATED std::int32_t nvinfer1::safe::ICudaEngine::getBindingVectorizedDim (
    std::int32_t const bindingIndex ) const [pure virtual], [noexcept]
```

Return the dimension index that the buffer is vectorized.

Specifically -1 is returned if scalars per vector is 1.

Parameters

<i>bindingIndex</i>	The binding Index.
---------------------	--------------------

Deprecated Deprecated in TensorRT 8.5. Superseded by [getTensorVectorizedDim\(\)](#).

See also

[safe::ICudaEngine::getTensorVectorizedDim\(\)](#)

Usage considerations

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

9.52.3.12 `getDeviceMemorySize()`

```
virtual size_t nvinfer1::safe::ICudaEngine::getDeviceMemorySize ( ) const [pure virtual], [noexcept]
```

Return the amount of device memory required by an execution context.

See also

[safe::IExecutionContext::setDeviceMemory\(\)](#)

Usage considerations

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

9.52.3.13 getErrorRecorder()

```
virtual IErrorRecorder * nvinfer1::safe::ICudaEngine::getErrorRecorder ( ) const [pure virtual],
[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 reporter has not been inherited from the [IRuntime](#), and setErrorReporter() has not been called.

Returns

A pointer to the [IErrorRecorder](#) object that has been registered.

See also

[setErrorRecorder\(\)](#)

Usage considerations

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

9.52.3.14 getIOTensorName()

```
virtual AsciiChar const * nvinfer1::safe::ICudaEngine::getIOTensorName (
    std::int32_t const index ) const [pure virtual], [noexcept]
```

Return the name of an IO tensor.

If the index does not fall between 0 and [getNbIOTensors\(\)-1](#), the function will fail with an error code of [ErrorCode::KINVALID_ARGUMENT\(3\)](#) that is emitted to the registered [IErrorRecorder](#).

Parameters

<i>index</i>	The value that falls between 0 and getNbIOTensors()-1 .
--------------	---

Returns

The name of an IO tensor. nullptr will be returned if the index does not fall between 0 and [getNbIOTensors\(\)-1](#).

See also

[getNbIOTensors\(\)](#)

Usage considerations

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

9.52.3.15 `getName()`

```
virtual AsciiChar const * nvinfer1::safe::ICudaEngine::getName ( ) const [pure virtual], [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.

Usage considerations

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

9.52.3.16 getNbBindings()

```
virtual TRT\_DEPRECATED std::int32_t nvinfer1::safe::ICudaEngine::getNbBindings ( ) const [pure virtual], [noexcept]
```

Get the number of binding indices.

Returns

The number of binding indices.

Deprecated Deprecated in TensorRT 8.5. Superseded by getNbIOTensors.

See also

[getBindingIndex\(\)](#)

Usage considerations

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

9.52.3.17 getNbIOTensors()

```
virtual std::int32_t nvinfer1::safe::ICudaEngine::getNbIOTensors ( ) const [pure virtual], [noexcept]
```

Return the number of input and output tensors for the network from which the engine was built.

Returns

The number of IO tensors.

See also

[getIOTensorName\(\)](#)

Usage considerations

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

9.52.3.18 getTensorBytesPerComponent()

```
virtual std::int32_t nvinfer1::safe::ICudaEngine::getTensorBytesPerComponent ( AsciiChar const * tensorName ) const [pure virtual], [noexcept]
```

Return the number of bytes per component of an element.

Parameters

<i>tensorName</i>	The name of an input or output tensor.
-------------------	--

Warning

The string *tensorName* must be 1024 characters or less including NULL terminator and must be NULL terminated.

Returns

The vector component size. 0 will be returned if (1) name is not the name of an input or output tensor, or (2) name is nullptr, or (3) name exceeds the string length limit, or (4) the tensor of given name is not vectorized.

See also

[safe::ICudaEngine::getTensorVectorizedDim\(\)](#)

Usage considerations

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

9.52.3.19 getTensorComponentsPerElement()

```
virtual std::int32_t nvinfer1::safe::ICudaEngine::getTensorComponentsPerElement (
    AsciiChar const * tensorName ) const [pure virtual], [noexcept]
```

Return the number of components included in one element.

Parameters

<i>tensorName</i>	The name of an input or output tensor.
-------------------	--

Warning

The string *tensorName* must be 1024 characters or less including NULL terminator and must be NULL terminated.

Returns

The vector component size. -1 will be returned if (1) name is not the name of an input or output tensor, or (2) name is nullptr, or (3) name exceeds the string length limit, or (4) the tensor of given name is not vectorized.

See also

[safe::ICudaEngine::getTensorVectorizedDim\(\)](#)

Usage considerations

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

9.52.3.20 getTensorDataType()

```
virtual DataType nvinfer1::safe::ICudaEngine::getTensorDataType (
    AsciiChar const * tensorName ) const [pure virtual], [noexcept]
```

Determine the required data type for a buffer from its tensor name.

Parameters

<i>tensorName</i>	The name of an input or output tensor.
-------------------	--

Warning

The string tensorName must be 1024 characters or less including NULL terminator and must be NULL terminated.

Returns

The type of the data in the buffer. [DataType::kFLOAT](#) will be returned if (1) name is not the name of an input or output tensor, or (2) name is nullptr, or (3) name exceeds the string length limit.

Usage considerations

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

9.52.3.21 getTensorFormat()

```
virtual TensorFormat nvinfer1::safe::ICudaEngine::getTensorFormat (
    AsciiChar const * tensorName ) const    [pure virtual], [noexcept]
```

Return the tensor format.

Parameters

<i>tensorName</i>	The name of an input or output tensor.
-------------------	--

Warning

The string *tensorName* must be 1024 characters or less including NULL terminator and must be NULL terminated.

Returns

The tensor format. [TensorFormat::kLINEAR](#) will be returned if (1) name is not the name of an input or output tensor, or (2) name is nullptr, or (3) name exceeds the string length limit.

Usage considerations

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

9.52.3.22 getTensorIOMode()

```
virtual TensorIOMode nvinfer1::safe::ICudaEngine::getTensorIOMode (
    AsciiChar const * tensorName ) const    [pure virtual], [noexcept]
```

Determine whether a tensor is an input or output tensor.

Parameters

<i>tensorName</i>	The name of an input or output tensor.
-------------------	--

Warning

The string *tensorName* must be 1024 characters or less including NULL terminator and must be NULL terminated.

Returns

kINPUT if tensorName is an input, kOUTPUT if tensorName is an output, or kNONE if neither.

Usage considerations

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

9.52.3.23 getTensorShape()

```
virtual Dims nvinfer1::safe::ICudaEngine::getTensorShape (
    AsciiChar const * tensorName ) const [pure virtual], [noexcept]
```

Get extent of an input or output tensor.

Parameters

<i>tensorName</i>	The name of an input or output tensor.
-------------------	--

Warning

The string tensorName must be 1024 characters or less including NULL terminator and must be NULL terminated.

Returns

Extent of the tensor. `Dims{-1, {}}` will be returned if (1) name is not the name of an input or output tensor, or (2) name is nullptr, or (3) name exceeds the string length limit.

Usage considerations

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

9.52.3.24 getTensorVectorizedDim()

```
virtual std::int32_t nvinfer1::safe::ICudaEngine::getTensorVectorizedDim (
    AsciiChar const * tensorName ) const [pure virtual], [noexcept]
```

Return the dimension index along which buffer is vectorized.

Specifically -1 is returned if scalars per vector is 1.

Parameters

<i>tensorName</i>	The name of an input or output tensor.
-------------------	--

Warning

The string `tensorName` must be 1024 characters or less including NULL terminator and must be NULL terminated.

Returns

The dimension index along which the buffer is vectorized. -1 will be returned if (1) name is not the name of an input or output tensor, or (2) name is nullptr, or (3) name exceeds the string length limit, or (4) the tensor of given name is not vectorized.

Usage considerations

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

9.52.3.25 operator=() [1/2]

```
ICudaEngine & nvinfer1::safe::ICudaEngine::operator= (
    ICudaEngine && ) & [delete]
```

9.52.3.26 operator=() [2/2]

```
ICudaEngine & nvinfer1::safe::ICudaEngine::operator= (
    ICudaEngine const & ) & [delete]
```

9.52.3.27 setErrorRecorder()

```
virtual void nvinfer1::safe::ICudaEngine::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

<i>recorder</i>	The error recorder to register with this interface.
-----------------	---

See also

[getErrorRecorder\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

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

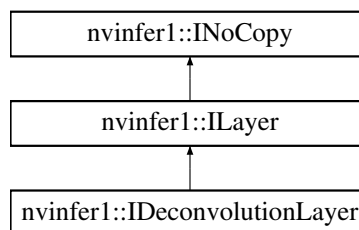
- [NvInferSafeRuntime.h](#)

9.53 nvinfer1::IDeconvolutionLayer Class Reference

A deconvolution layer in a network definition.

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

Inheritance diagram for nvinfer1::IDeconvolutionLayer:



Public Member Functions

- **TRT_DEPRECATED** void [setKernelSize](#) ([DimsHW](#) kernelSize) noexcept
Set the HW kernel size of the convolution.
- **TRT_DEPRECATED** [DimsHW](#) [getKernelSize](#) () const noexcept
Get the HW kernel size of the deconvolution.
- void [setNbOutputMaps](#) (int32_t nbOutputMaps) noexcept
Set the number of output feature maps for the deconvolution.
- int32_t [getNbOutputMaps](#) () const noexcept
Get the number of output feature maps for the deconvolution.

- [TRT_DEPRECATED](#) void [setStride](#) ([DimsHW](#) stride) noexcept
Set the stride of the deconvolution.
- [TRT_DEPRECATED](#) [DimsHW](#) [getStride](#) () const noexcept
Get the stride of the deconvolution.
- [TRT_DEPRECATED](#) void [setPadding](#) ([DimsHW](#) padding) noexcept
Set the padding of the deconvolution.
- [TRT_DEPRECATED](#) [DimsHW](#) [getPadding](#) () const noexcept
Get the padding of the deconvolution.
- void [setNbGroups](#) (int32_t nbGroups) noexcept
Set the number of groups for a deconvolution.
- int32_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](#) padding) noexcept
Set the multi-dimension pre-padding of the deconvolution.
- [Dims](#) [getPrePadding](#) () const noexcept
Get the pre-padding.
- void [setPostPadding](#) ([Dims](#) 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](#) 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](#) 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](#) 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](#) 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=default

Protected Attributes

- apiv::VDeconvolutionLayer * [mImpl](#)

9.53.1 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.

9.53.2 Constructor & Destructor Documentation

9.53.2.1 ~IDeconvolutionLayer()

```
virtual nvinfer1::IDeconvolutionLayer::~~IDeconvolutionLayer ( ) [protected], [virtual], [default],  
[noexcept]
```

9.53.3 Member Function Documentation

9.53.3.1 getBiasWeights()

```
Weights nvinfer1::IDeconvolutionLayer::getBiasWeights ( ) const [inline], [noexcept]
```

Get the bias weights for the deconvolution.

See also

[getBiasWeights\(\)](#)

9.53.3.2 getDilationNd()

```
Dims nvinfer1::IDeconvolutionLayer::getDilationNd ( ) const [inline], [noexcept]
```

Get the multi-dimension dilation of the deconvolution.

See also

[setDilationNd\(\)](#)

9.53.3.3 getKernelSize()

```
TRT_DEPRECATED DimsHW nvinfer1::IDeconvolutionLayer::getKernelSize ( ) const [inline], [noexcept]
```

Get the HW kernel size of the deconvolution.

See also

[setKernelSize\(\)](#)

Deprecated Superseded by `getKernelSizeNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.53.3.4 getKernelSizeNd()

```
Dims nvinfer1::IDeconvolutionLayer::getKernelSizeNd ( ) const [inline], [noexcept]
```

Get the multi-dimension kernel size of the deconvolution.

See also

[setKernelSizeNd\(\)](#)

9.53.3.5 getKernelWeights()

```
Weights nvinfer1::IDeconvolutionLayer::getKernelWeights ( ) const [inline], [noexcept]
```

Get the kernel weights for the deconvolution.

See also

[setNbGroups\(\)](#)

9.53.3.6 getNbGroups()

```
int32_t nvinfer1::IDeconvolutionLayer::getNbGroups ( ) const [inline], [noexcept]
```

Get the number of groups for a deconvolution.

See also

[setNbGroups\(\)](#)

9.53.3.7 getNbOutputMaps()

```
int32_t nvinfer1::IDeconvolutionLayer::getNbOutputMaps ( ) const [inline], [noexcept]
```

Get the number of output feature maps for the deconvolution.

See also

[setNbOutputMaps\(\)](#)

9.53.3.8 getPadding()

```
TRT_DEPRECATED DimsHW nvinfer1::IDeconvolutionLayer::getPadding ( ) const [inline], [noexcept]
```

Get the padding of the deconvolution.

Default: (0, 0)

See also

[setPadding\(\)](#)

Deprecated Superseded by `getPaddingNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.53.3.9 `getPaddingMode()`

```
PaddingMode nvinfer1::IDeconvolutionLayer::getPaddingMode ( ) const [inline], [noexcept]
```

Get the padding mode.

Default: kEXPLICIT_ROUND_DOWN

See also

[setPaddingMode\(\)](#)

9.53.3.10 `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\(\)](#)

9.53.3.11 `getPostPadding()`

```
Dims nvinfer1::IDeconvolutionLayer::getPostPadding ( ) const [inline], [noexcept]
```

Get the padding.

See also

[setPostPadding\(\)](#)

9.53.3.12 `getPrePadding()`

```
Dims nvinfer1::IDeconvolutionLayer::getPrePadding ( ) const [inline], [noexcept]
```

Get the pre-padding.

See also

[setPrePadding\(\)](#)

9.53.3.13 getStride()

```
TRT_DEPRECATED DimsHW nvinfer1::IDeconvolutionLayer::getStride ( ) const [inline], [noexcept]
```

Get the stride of the deconvolution.

Default: (1,1)

Deprecated Superseded by getStrideNd. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.53.3.14 getStrideNd()

```
Dims nvinfer1::IDeconvolutionLayer::getStrideNd ( ) const [inline], [noexcept]
```

Get the multi-dimension stride of the deconvolution.

See also

[setStrideNd\(\)](#)

9.53.3.15 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\(\)](#)

9.53.3.16 setDilationNd()

```
void nvinfer1::IDeconvolutionLayer::setDilationNd (
    Dims dilation ) [inline], [noexcept]
```

Set the multi-dimension dilation of the deconvolution.

Default: (1, 1, ..., 1)

See also

[getDilationNd\(\)](#)

9.53.3.17 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

<i>index</i>	the index of the input to modify.
<i>tensor</i>	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\(\)](#)

9.53.3.18 setKernelSize()

```
TRT_DEPRECATED void nvinfer1::IDeconvolutionLayer::setKernelSize (
    DimsHW kernelSize ) [inline], [noexcept]
```

Set the HW kernel size of the convolution.

If executing this layer on DLA, both height and width of kernel size 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

[getKernelSize\(\)](#)

Deprecated Superseded by `setKernelSizeNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.53.3.19 setKernelSizeNd()

```
void nvinfer1::IDeconvolutionLayer::setKernelSizeNd (
    Dims 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\(\)](#) [setKernelSize\(\)](#) [getKernelSize\(\)](#)

9.53.3.20 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\(\)](#)

9.53.3.21 setNbGroups()

```
void nvinfer1::IDeconvolutionLayer::setNbGroups (
    int32_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\(\)](#)

9.53.3.22 setNbOutputMaps()

```
void nvinfer1::IDeconvolutionLayer::setNbOutputMaps (
    int32_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\(\)](#)

9.53.3.23 setPadding()

```
TRT_DEPRECATED void nvinfer1::IDeconvolutionLayer::setPadding (
    DimsHW padding ) [inline], [noexcept]
```

Set the padding of the deconvolution.

The output will be trimmed by this number of elements on each side in the height and width directions. 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)

If executing this layer on DLA, both height and width of padding must be 0.

See also

[getPadding\(\)](#)

Deprecated Superseded by setPaddingNd. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.53.3.24 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\(\)](#)

9.53.3.25 setPaddingNd()

```
void nvinfer1::IDeconvolutionLayer::setPaddingNd (
    Dims 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\(\)](#)

9.53.3.26 setPostPadding()

```
void nvinfer1::IDeconvolutionLayer::setPostPadding (
    Dims 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)

If executing this layer on DLA, padding must be 0.

See also

[getPostPadding\(\)](#)

9.53.3.27 setPrePadding()

```
void nvinfer1::IDeconvolutionLayer::setPrePadding (
    Dims 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)

If executing this layer on DLA, padding must be 0.

See also

[getPrePadding\(\)](#)

9.53.3.28 setStride()

```
TRT_DEPRECATED void nvinfer1::IDeconvolutionLayer::setStride (
    DimsHW stride ) [inline], [noexcept]
```

Set the stride of the deconvolution.

If executing this layer on DLA, there is one restriction: 1) 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

[getStride\(\)](#)

Deprecated Superseded by `setStrideNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.53.3.29 setStrideNd()

```
void nvinfer1::IDeconvolutionLayer::setStrideNd (
    Dims 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\(\)](#) [setStride\(\)](#) [getStride\(\)](#)

9.53.4 Member Data Documentation

9.53.4.1 mImpl

```
apiv::VDeconvolutionLayer* nvinfer1::IDeconvolutionLayer::mImpl [protected]
```

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

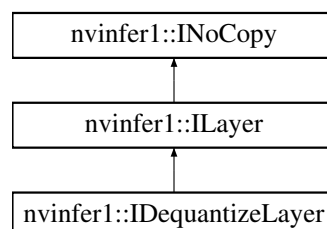
- [NvInfer.h](#)

9.54 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.

Protected Member Functions

- virtual `~IDequantizeLayer ()` noexcept=default

Protected Attributes

- `apiv::VDequantizeLayer * mImpl`

9.54.1 Detailed Description

A Dequantize layer in a network definition.

This layer accepts a signed 8-bit integer input tensor, and uses the configured scale and zeroPt inputs to dequantize the input according to: $\text{output} = (\text{input} - \text{zeroPt}) * \text{scale}$

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 `DataType::kINT8`. `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-channel quantization. All `scale` coefficients must have positive values. 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 `DataType::kFLOAT` or `DataType::kHALF`. The input type, if constrained, must be constrained to `DataType::kINT8`. The output size is the same as the input size. The quantization axis is in reference to the input tensor's dimensions.

`IDequantizeLayer` only supports `DataType::kINT8` precision and will default to this precision during instantiation. `IDequantizeLayer` only supports `DataType::kFLOAT` or `DataType::kHALF` 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`: $\text{output}[n,c,h,w] = (\text{input}[n,c,h,w] - \text{zeroPt}) * \text{scale}$

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`: $\text{output}[k,c,r,s] = (\text{input}[k,c,r,s] - \text{zeroPt}[k]) * \text{scale}[k]$

Note

Only symmetric quantization is supported.

Currently the only allowed build-time constant `scale` and `\zeroPt` subgraphs are:

1. Constant -> Quantize
2. Constant -> Cast -> Quantize

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.54.2 Constructor & Destructor Documentation

9.54.2.1 ~IDequantizeLayer()

```
virtual nvinfer1::IDequantizeLayer::~~IDequantizeLayer ( ) [protected], [virtual], [default],  
[noexcept]
```

9.54.3 Member Function Documentation

9.54.3.1 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.

9.54.3.2 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).

9.54.4 Member Data Documentation

9.54.4.1 mImpl

```
apiv::VDequantizeLayer* nvinfer1::IDequantizeLayer::mImpl [protected]
```

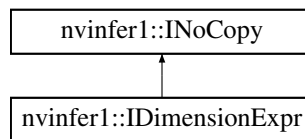
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.55 nvinfer1::IDimensionExpr Class Reference

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

Inheritance diagram for nvinfer1::IDimensionExpr:



Public Member Functions

- bool [isConstant](#) () const noexcept
Return true if expression is a build-time constant.
- int32_t [getConstantValue](#) () const noexcept

Protected Member Functions

- virtual [~IDimensionExpr](#) () noexcept=default

Protected Attributes

- apiv::VDimensionExpr * [mImpl](#)

9.55.1 Detailed Description

An [IDimensionExpr](#) represents an integer expression constructed from constants, input dimensions, and binary operations. These expressions can be used in overrides of [IPluginV2DynamicExt::getOutputDimensions](#) 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](#)

9.55.2 Constructor & Destructor Documentation

9.55.2.1 ~IDimensionExpr()

```
virtual nvinfer1::IDimensionExpr::~IDimensionExpr ( ) [protected], [virtual], [default], [noexcept]
```

9.55.3 Member Function Documentation

9.55.3.1 getConstantValue()

```
int32_t nvinfer1::IDimensionExpr::getConstantValue ( ) const [inline], [noexcept]
```

If [isConstant\(\)](#), returns value of the constant. If ![isConstant\(\)](#), return `std::numeric_limits<int32_t>::min()`.

9.55.3.2 isConstant()

```
bool nvinfer1::IDimensionExpr::isConstant ( ) const [inline], [noexcept]
```

Return true if expression is a build-time constant.

9.55.4 Member Data Documentation

9.55.4.1 mImpl

```
apiv::VDimensionExpr* nvinfer1::IDimensionExpr::mImpl [protected]
```

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

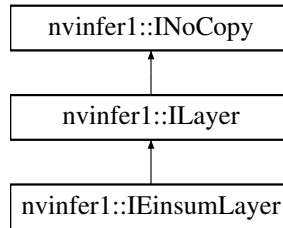
- [NvInferRuntime.h](#)

9.56 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=default

Protected Attributes

- apiv::VEinsumLayer * [mImpl](#)

9.56.1 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

Note

TensorRT does not support ellipsis, diagonal operations or more than two inputs for Einsum.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.56.2 Constructor & Destructor Documentation

9.56.2.1 ~IEinsumLayer()

```
virtual nvinfer1::IEinsumLayer::~~IEinsumLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.56.3 Member Function Documentation

9.56.3.1 getEquation()

```
char const * nvinfer1::IEinsumLayer::getEquation ( ) const [inline], [noexcept]
```

Return the equation.

See also

[setEquation\(\)](#)

9.56.3.2 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\(\)](#)

9.56.4 Member Data Documentation

9.56.4.1 mImpl

```
apiv::VEinsumLayer* nvinfer1::IEinsumLayer::mImpl [protected]
```

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

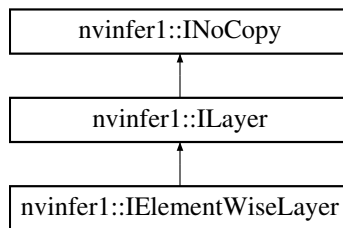
- [NvInfer.h](#)

9.57 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=default

Protected Attributes

- apiv::VElementWiseLayer * [mImpl](#)

9.57.1 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.

9.57.2 Constructor & Destructor Documentation

9.57.2.1 ~IElementWiseLayer()

```
virtual nvinfer1::IElementWiseLayer::~~IElementWiseLayer ( ) [protected], [virtual], [default],
[noexcept]
```

9.57.3 Member Function Documentation

9.57.3.1 getOperation()

```
ElementWiseOperation nvinfer1::IElementWiseLayer::getOperation ( ) const [inline], [noexcept]
```

Get the binary operation for the layer.

See also

[setOperation\(\)](#), [ElementWiseOperation](#)
[setBiasWeights\(\)](#)

9.57.3.2 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\(\)](#)

9.57.4 Member Data Documentation

9.57.4.1 mImpl

```
apiv::VElementWiseLayer* nvinfer1::IElementWiseLayer::mImpl [protected]
```

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

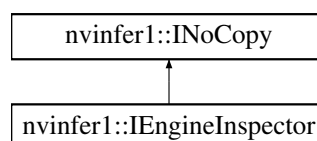
- [NvInfer.h](#)

9.58 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=default
- 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

9.58.1 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](#)

9.58.2 Constructor & Destructor Documentation

9.58.2.1 ~IEngineInspector()

```
virtual nvinfer1::IEngineInspector::~~IEngineInspector ( ) [virtual], [default], [noexcept]
```

9.58.3 Member Function Documentation

9.58.3.1 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

<i>layerIndex</i>	the index of the layer. It must lie in range [0, engine.getNbLayers()).
<i>format</i>	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](#)

9.58.3.2 getErrorRecorder()

```
IErrRecorder * 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\(\)](#)

9.58.3.3 `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\(\)](#)

9.58.3.4 `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

<i>layerIndex</i>	the index of the layer. It must lie in range [0, engine.getNbLayers()).
<i>format</i>	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](#)

9.58.3.5 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

<i>recorder</i>	The error recorder to register with this interface.
-----------------	---

See also

[getErrorRecorder\(\)](#)

9.58.3.6 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.

9.58.4 Member Data Documentation

9.58.4.1 mImpl

```
apiv::VEngineInspector* nvinfer1::IEngineInspector::mImpl [protected]
```

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

- [NvInferRuntime.h](#)

9.59 nvinfer1::IErrorRecorder Class Reference

Reference counted application-implemented error reporting interface for TensorRT objects.

```
#include <NvInferRuntimeCommon.h>
```

Public Types

- using [ErrorDesc](#) = char const *
- using [RefCount](#) = int32_t

Public Member Functions

- [IErrorRecorder](#) ()=default
- virtual [~IErrorRecorder](#) () noexcept=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}

9.59.1 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 1024 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 `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.

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.

9.59.2 Member Typedef Documentation

9.59.2.1 ErrorDesc

```
using nvinfer1::IErrorRecorder::ErrorDesc = char const*
```

A typedef of a C-style string for reporting error descriptions.

9.59.2.2 RefCount

```
using nvinfer1::IErrorRecorder::RefCount = int32_t
```

A typedef of a 32bit integer for reference counting.

9.59.3 Constructor & Destructor Documentation

9.59.3.1 IErrorRecorder()

```
nvinfer1::IErrorRecorder::IErrorRecorder ( ) [default]
```

9.59.3.2 ~IErrorRecorder()

```
virtual nvinfer1::IErrorRecorder::~~IErrorRecorder ( ) [virtual], [default], [noexcept]
```

9.59.4 Member Function Documentation

9.59.4.1 clear()

```
virtual void nvinfer1::IErrorRecorder::clear ( ) [pure virtual], [noexcept]
```

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, the next call to `getNbErrors` will return zero.

See also

[getNbErrors](#)

Usage considerations

- 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.

9.59.4.2 decRefCount()

```
virtual RefCount nvinfer1::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.

Usage considerations

- 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.

9.59.4.3 getErrorCode()

```
virtual ErrorCode nvinfer1::IErrorRecorder::getErrorCode (
    int32_t errorIdx ) const [pure virtual], [noexcept]
```

Returns the ErrorCode enumeration.

Parameters

<i>errorIdx</i>	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.

See also

[getErrorDesc](#), [ErrorCode](#)

Usage considerations

- 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.

9.59.4.4 getErrorDesc()

```
virtual ErrorDesc nvinfer1::IErrorRecorder::getErrorDesc (
    int32_t errorIdx ) const [pure virtual], [noexcept]
```

Returns a null-terminated C-style string description of the error.

Parameters

<i>errorIdx</i>	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 may be truncated. The format of the string is "<EnumAsStr> - <Description>".

Returns

Returns a string representation of the error along with a description of the error.

See also

[getErrorCode](#)

Usage considerations

- 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.

9.59.4.5 getNbErrors()

```
virtual int32_t nvinfer1::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 monotonically increase until [clear\(\)](#) is called.

Returns

Returns the number of errors detected, or 0 if there are no errors.

See also

[clear](#)

Usage considerations

- 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.

9.59.4.6 hasOverflowed()

```
virtual bool nvinfer1::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.

Usage considerations

- 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.

9.59.4.7 incRefCount()

```
virtual RefCount nvinfer1::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.

Usage considerations

- 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.

9.59.4.8 reportError()

```
virtual bool nvinfer1::IErrorRecorder::reportError (
    ErrorCode val,
    ErrorDesc desc ) [pure virtual], [noexcept]
```

Report an error to the error recorder with the corresponding enum and description.

Parameters

<i>val</i>	The error code enum that is being reported.
<i>desc</i>	The string description of the error.

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.

Usage considerations

- 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.

9.59.5 Member Data Documentation

9.59.5.1 kMAX_DESC_LENGTH

```
constexpr size_t nvinfer1::IErrorRecorder::kMAX_DESC_LENGTH {127U} [static], [constexpr]
```

The length limit for an error description, excluding the '\0' string terminator.

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

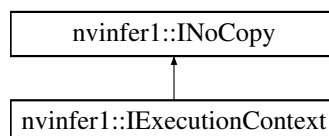
- [NvInferRuntimeCommon.h](#)

9.60 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=default
- [TRT_DEPRECATED](#) bool [execute](#) (int32_t batchSize, void *const *bindings) noexcept
Synchronously execute inference on a batch.
- [TRT_DEPRECATED](#) bool [enqueue](#) (int32_t batchSize, void *const *bindings, cudaStream_t stream, cudaEvent_t *inputConsumed) noexcept
Asynchronously execute inference on a batch.
- 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.
- [TRT_DEPRECATED](#) void [destroy](#) () noexcept
Destroy this object.
- 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.
- [TRT_DEPRECATED](#) Dims [getStrides](#) (int32_t bindingIndex) const noexcept
Return the strides of the buffer for the given binding.
- Dims [getTensorStrides](#) (char const *tensorName) const noexcept
Return the strides of the buffer for the given tensor name.
- [TRT_DEPRECATED](#) bool [setOptimizationProfile](#) (int32_t profileIndex) noexcept
Select an optimization profile for the current context.
- int32_t [getOptimizationProfile](#) () const noexcept
Get the index of the currently selected optimization profile.
- [TRT_DEPRECATED](#) bool [setBindingDimensions](#) (int32_t bindingIndex, Dims dimensions) noexcept
Set the dynamic dimensions of an input binding.
- bool [setInputShape](#) (char const *tensorName, Dims const &dims) noexcept
Set shape of given input.
- [TRT_DEPRECATED](#) Dims [getBindingDimensions](#) (int32_t bindingIndex) const noexcept
Get the dynamic dimensions of a binding.
- Dims [getTensorShape](#) (char const *tensorName) const noexcept
Return the shape of the given input or output.
- [TRT_DEPRECATED](#) bool [setInputShapeBinding](#) (int32_t bindingIndex, int32_t const *data) noexcept
Set values of input tensor required by shape calculations.
- [TRT_DEPRECATED](#) bool [getShapeBinding](#) (int32_t bindingIndex, int32_t *data) const noexcept
Get values of an input tensor required for shape calculations or an output tensor produced by shape calculations.
- bool [allInputDimensionsSpecified](#) () const noexcept

- Whether all dynamic dimensions of input tensors have been specified.*

 - bool [allInputShapesSpecified](#) () const noexcept
- Whether all input shape bindings 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 inference a network.*

 - [TRT_DEPRECATED](#) bool [enqueueV2](#) (void *const *bindings, cudaStream_t stream, cudaEvent_t *inputConsumed) noexcept
- Asynchronously execute inference.*

 - 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 [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.*

 - 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
Asynchronously execute inference.
- 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.

Protected Attributes

- apiv::VExecutionContext * [mImpl](#)

Additional Inherited Members

9.60.1 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.

9.60.2 Constructor & Destructor Documentation

9.60.2.1 ~IExecutionContext()

```
virtual nvinfer1::IExecutionContext::~IExecutionContext ( ) [virtual], [default], [noexcept]
```

9.60.3 Member Function Documentation

9.60.3.1 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 [setBindingDimensions\(\)](#).

Trivially true if network has no dynamically shaped input tensors.

Does not work with name-base interfaces eg. [IExecutionContext::setInputShape\(\)](#). Use [IExecutionContext::inferShapes\(\)](#) instead.

See also

[setBindingDimensions\(bindingIndex,dimensions\)](#)

9.60.3.2 allInputShapesSpecified()

```
bool nvinfer1::IExecutionContext::allInputShapesSpecified ( ) const [inline], [noexcept]
```

Whether all input shape bindings have been specified.

Returns

True if all input shape bindings have been specified by [setInputShapeBinding\(\)](#).

Trivially true if network has no input shape bindings.

Does not work with name-base interfaces eg. [IExecutionContext::setInputShape\(\)](#). Use [IExecutionContext::inferShapes\(\)](#) instead.

See also

[isShapeBinding\(bindingIndex\)](#)

9.60.3.3 destroy()

```
TRT_DEPRECATED void nvinfer1::IExecutionContext::destroy ( ) [inline], [noexcept]
```

Destroy this object.

Deprecated Deprecated in TRT 8.0. Superseded by `delete`.

Warning

Calling `destroy` on a managed pointer will result in a double-free error.

9.60.3.4 enqueue()

```
TRT_DEPRECATED bool nvinfer1::IExecutionContext::enqueue (
    int32_t batchSize,
    void *const * bindings,
    cudaStream_t stream,
    cudaEvent_t * inputConsumed ) [inline], [noexcept]
```

Asynchronously execute inference on a batch.

This method requires an array of input and output buffers. The mapping from tensor names to indices can be queried using [ICudaEngine::getBindingIndex\(\)](#)

Parameters

<i>batchSize</i>	The batch size. This is at most the max batch size value supplied to the builder when the engine was built. If the network is created with NetworkDefinitionCreationFlag::kEXPLICIT_BATCH flag, please use enqueueV3() instead, and this batchSize argument has no effect.
<i>bindings</i>	An array of pointers to input and output buffers for the network.
<i>stream</i>	A cuda stream on which the inference kernels will be enqueued.
<i>inputConsumed</i>	An optional event which will be signaled when the input buffers can be refilled with new data.

Returns

True if the kernels were enqueued successfully.

Deprecated Deprecated in TensorRT 8.4. Superseded by [enqueueV2\(\)](#) if the network is created with [NetworkDefinitionCreationFlag::kEXPLICIT_BATCH](#) flag.

See also

[ICudaEngine::getBindingIndex\(\)](#) [ICudaEngine::getMaxBatchSize\(\)](#)

Warning

Calling [enqueue\(\)](#) in from the same [IExecutionContext](#) object with different CUDA streams concurrently results in undefined behavior. To perform inference concurrently in multiple streams, use one execution context per stream.

This function will trigger layer resource updates if [hasImplicitBatchDimension\(\)](#) returns true and batchSize changes between subsequent calls, possibly resulting in performance bottlenecks.

9.60.3.5 enqueueV2()

```
TRT_DEPRECATED bool nvinfer1::IExecutionContext::enqueueV2 (
    void *const * bindings,
    cudaStream_t stream,
    cudaEvent_t * inputConsumed ) [inline], [noexcept]
```

Asynchronously execute inference.

This method requires an array of input and output buffers. The mapping from tensor names to indices can be queried using [ICudaEngine::getBindingIndex\(\)](#). This method only works for execution contexts built with full dimension networks.

Parameters

<i>bindings</i>	An array of pointers to input and output buffers for the network.
<i>stream</i>	A cuda stream on which the inference kernels will be enqueued
<i>inputConsumed</i>	An optional event which will be signaled when the input buffers can be refilled with new data

Returns

True if the kernels were enqueued successfully.

Deprecated Superseded by [enqueueV3\(\)](#). Deprecated in TensorRT 8.5

See also

[ICudaEngine::getBindingIndex\(\)](#) [ICudaEngine::getMaxBatchSize\(\)](#) [IExecutionContext::enqueueV3\(\)](#)

Note

Calling [enqueueV2\(\)](#) with a stream in CUDA graph capture mode has a known issue. If dynamic shapes are used, the first [enqueueV2\(\)](#) call after a [setInputShapeBinding\(\)](#) call will cause failure in stream capture due to resource allocation. Please call [enqueueV2\(\)](#) once before capturing the graph.

Warning

Calling [enqueueV2\(\)](#) in from the same [IExecutionContext](#) object with different CUDA streams concurrently results in undefined behavior. To perform inference concurrently in multiple streams, use one execution context per stream.

9.60.3.6 enqueueV3()

```
bool nvinfer1::IExecutionContext::enqueueV3 (
    cudaStream_t stream ) [inline], [noexcept]
```

Asynchronously execute inference.

Parameters

<i>stream</i>	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.

9.60.3.7 execute()

```
TRT_DEPRECATED bool nvinfer1::IExecutionContext::execute (
    int32_t batchSize,
    void *const * bindings ) [inline], [noexcept]
```

Synchronously execute inference on a batch.

This method requires an array of input and output buffers. The mapping from tensor names to indices can be queried using `ICudaEngine::getBindingIndex()`

Parameters

<i>batchSize</i>	The batch size. This is at most the max batch size value supplied to the builder when the engine was built. If the network is created with <code>NetworkDefinitionCreationFlag::kEXPLICIT_BATCH</code> flag, please use <code>executeV2()</code> instead, and this <code>batchSize</code> argument has no effect.
<i>bindings</i>	An array of pointers to input and output buffers for the network.

Returns

True if execution succeeded.

Deprecated Deprecated in TensorRT 8.4. Superseded by `executeV2()` if the network is created with `NetworkDefinitionCreationFlag::kEXPLICIT_BATCH` flag.

Warning

This function will trigger layer resource updates if `hasImplicitBatchDimension()` returns true and `batchSize` changes between subsequent calls, possibly resulting in performance bottlenecks.

See also

`ICudaEngine::getBindingIndex()` `ICudaEngine::getMaxBatchSize()`

9.60.3.8 executeV2()

```
bool nvinfer1::IExecutionContext::executeV2 (
    void *const * bindings ) [inline], [noexcept]
```

Synchronously execute inference a network.

This method requires an array of input and output buffers. The mapping from tensor names to indices can be queried using [ICudaEngine::getBindingIndex\(\)](#). This method only works for execution contexts built with full dimension networks.

Parameters

<i>bindings</i>	An array of pointers to input and output buffers for the network.
-----------------	---

Returns

True if execution succeeded.

See also

[ICudaEngine::getBindingIndex\(\)](#) [ICudaEngine::getMaxBatchSize\(\)](#)

9.60.3.9 getBindingDimensions()

```
TRT_DEPRECATED Dims nvinfer1::IExecutionContext::getBindingDimensions (
    int32_t bindingIndex ) const [inline], [noexcept]
```

Get the dynamic dimensions of a binding.

If the engine was built with an implicit batch dimension, same as [ICudaEngine::getBindingDimensions](#).

If [setBindingDimensions\(\)](#) has been called on this binding (or if there are no dynamic dimensions), all dimensions will be positive. Otherwise, it is necessary to call [setBindingDimensions\(\)](#) before [enqueueV2\(\)](#) or [executeV2\(\)](#) may be called.

If the bindingIndex is out of range, an invalid [Dims](#) with nbDims == -1 is returned. The same invalid [Dims](#) will be returned if the engine was not built with an implicit batch dimension and if the execution context is not currently associated with a valid optimization profile (i.e. if [getOptimizationProfile\(\)](#) returns -1).

If [ICudaEngine::bindingIsInput\(bindingIndex\)](#) is false, then both [allInputDimensionsSpecified\(\)](#) and [allInputShapesSpecified\(\)](#) must be true before calling this method.

Returns

Currently selected binding dimensions

For backwards compatibility with earlier versions of TensorRT, a bindingIndex that does not belong to the current profile is corrected as described for [ICudaEngine::getProfileDimensions](#).

Deprecated Deprecated in TensorRT 8.5. Superseded by [getTensorShape\(\)](#).

See also

[ICudaEngine::getProfileDimensions\(\)](#)
[getTensorShape\(\)](#)

9.60.3.10 getDebugSync()

```
bool nvinfer1::IExecutionContext::getDebugSync ( ) const [inline], [noexcept]
```

Get the debug sync flag.

See also

[setDebugSync\(\)](#)

9.60.3.11 getEngine()

```
ICudaEngine const & nvinfer1::IExecutionContext::getEngine ( ) const [inline], [noexcept]
```

Get the associated engine.

See also

[ICudaEngine](#)

9.60.3.12 getEnqueueEmitsProfile()

```
bool nvinfer1::IExecutionContext::getEnqueueEmitsProfile ( ) const [inline], [noexcept]
```

Get the enqueueEmitsProfile state.

Returns

The enqueueEmitsProfile state.

See also

[IExecutionContext::setEnqueueEmitsProfile\(\)](#)

9.60.3.13 getErrorRecorder()

```
IErrRecorder * 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 [IErrRecorder](#) object that has been registered.

See also

[setErrorRecorder\(\)](#)

9.60.3.14 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.

9.60.3.15 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

<i>tensorName</i>	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.

9.60.3.16 getName()

```
char const * nvinfer1::IExecutionContext::getName ( ) const [inline], [noexcept]
```

Return the name of the execution context.

See also

[setName\(\)](#)

9.60.3.17 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\(\)](#)

9.60.3.18 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 [enqueueV2\(\)](#)/[enqueueV3\(\)](#)/[executeV2\(\)](#) will fail until a valid profile index has been set. This behavior is deprecated in TensorRT 8.6 and in TensorRT 9.0, all profiles will default to optimization profile 0 and -1 will no longer be returned.

9.60.3.19 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](#)

9.60.3.20 getOutputTensorAddress()

```
void * nvinfer1::IExecutionContext::getOutputTensorAddress (
    char const * tensorName ) const [inline], [noexcept]
```

Get memory address for given output.

Parameters

<i>tensorName</i>	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\(\)](#)

9.60.3.21 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](#)

9.60.3.22 getProfiler()

```
IProfiler * nvinfer1::IExecutionContext::getProfiler ( ) const [inline], [noexcept]
```

Get the profiler.

See also

[IProfiler setProfiler\(\)](#)

9.60.3.23 getShapeBinding()

```
TRT_DEPRECATED bool nvinfer1::IExecutionContext::getShapeBinding (
    int32_t bindingIndex,
    int32_t * data ) const [inline], [noexcept]
```

Get values of an input tensor required for shape calculations or an output tensor produced by shape calculations.

Parameters

<i>bindingIndex</i>	index of an input or output tensor for which <code>ICudaEngine::isShapeBinding(bindingIndex)</code> is true.
<i>data</i>	pointer to where values will be written. The number of values written is the product of the dimensions returned by <code>getBindingDimensions(bindingIndex)</code> .

If `ICudaEngine::bindingIsInput(bindingIndex)` is false, then both `allInputDimensionsSpecified()` and `allInputShapesSpecified()` must be true before calling this method. The method will also fail if no valid optimization profile has been set for the current execution context, i.e. if `getOptimizationProfile()` returns -1.

Deprecated Deprecated in TensorRT 8.5. Superseded by `getTensorAddress()` or `getOutputTensorAddress()`.

See also

`isShapeBinding()` `getTensorAddress()` `getOutputTensorAddress()`

9.60.3.24 `getStrides()`

```
TRT_DEPRECATED Dims nvinfer1::IExecutionContext::getStrides (
    int32_t bindingIndex ) const [inline], [noexcept]
```

Return the strides of the buffer for the given binding.

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 `bindingIndex` is invalid or there are dynamic dimensions that have not been set yet, returns `Dims` with `Dims::nbDims = -1`.

Parameters

<i>bindingIndex</i>	The binding index.
---------------------	--------------------

Deprecated Deprecated in TensorRT 8.5. Superseded by `getTensorStrides()`.

See also

`getTensorStrides()`

9.60.3.25 getTemporaryStorageAllocator()

```
IGpuAllocator * nvinfer1::IExecutionContext::getTemporaryStorageAllocator ( ) const [inline],
[noexcept]
```

Get allocator set by `setTemporaryStorageAllocator`.

Returns a nullptr if a nullptr was passed with `setTemporaryStorageAllocator()`.

9.60.3.26 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

<i>tensorName</i>	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\(\)](#)

9.60.3.27 getTensorShape()

```
Dims nvinfer1::IExecutionContext::getTensorShape (
    char const * tensorName ) const [inline], [noexcept]
```

Return the shape of the given input or output.

Parameters

<i>tensorName</i>	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.

9.60.3.28 `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

<i>tensorName</i>	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.

9.60.3.29 inferShapes()

```
int32_t nvinfer1::IExecutionContext::inferShapes (
    int32_t nbMaxNames,
    char const ** tensorNames ) [inline], [noexcept]
```

Run shape calculations.

Parameters

<i>nbMaxNames</i>	Maximum number of names to write to <i>tensorNames</i> . When the return value is a positive value <i>n</i> and <i>tensorNames</i> != nullptr, the names of min(<i>n</i> , <i>nbMaxNames</i>) insufficiently specified input tensors are written to <i>tensorNames</i> .
<i>tensorNames</i>	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.

9.60.3.30 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\(\)](#)

9.60.3.31 setBindingDimensions()

```
TRT_DEPRECATED bool nvinfer1::IExecutionContext::setBindingDimensions (
    int32_t bindingIndex,
    Dims dimensions ) [inline], [noexcept]
```

Set the dynamic dimensions of an input binding.

Parameters

<i>bindingIndex</i>	index of an input tensor whose dimensions must be compatible with the network definition (i.e. only the wildcard dimension -1 can be replaced with a new dimension ≥ 0).
<i>dimensions</i>	specifies the dimensions of the input tensor. It must be in the valid range for the currently selected optimization profile, and the corresponding engine must not be safety-certified.

This method requires the engine to be built without an implicit batch dimension. This method will fail unless a valid optimization profile is defined for the current execution context ([getOptimizationProfile\(\)](#) must not be -1).

For all dynamic non-output bindings (which have at least one wildcard dimension of -1), this method needs

to be called before either [enqueueV2\(\)](#) or [executeV2\(\)](#) may be called. This can be checked using the method [allInputDimensionsSpecified\(\)](#).

Warning

This function will trigger layer resource updates on the next call of [enqueueV2\(\)/executeV2\(\)](#), possibly resulting in performance bottlenecks, if the dimensions are different than the previous set dimensions.

Returns

false if an error occurs (e.g. `bindingIndex` is out of range for the currently selected optimization profile or binding dimension is inconsistent with min-max range of the optimization profile), else true. Note that the network can still be invalid for certain combinations of input shapes that lead to invalid output shapes. To confirm the correctness of the network input shapes, check whether the output binding has valid dimensions using [getBindingDimensions\(\)](#) on the output `bindingIndex`.

Deprecated Deprecated in TensorRT 8.5. Superseded by [setInputShape\(\)](#).

See also

[setInputShape\(\)](#)

9.60.3.32 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 [enqueueV2\(\)/enqueueV3\(\)](#).

See also

[getDebugSync\(\)](#)

9.60.3.33 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 at least that returned by `getDeviceMemorySize()`. Setting memory to nullptr is acceptable if `getDeviceMemorySize()` returns 0. If using `enqueueV2()/enqueueV3()` to run the network, the memory is in use from the invocation of `enqueueV2()/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 during this time will result in undefined behavior.

See also

[ICudaEngine::getDeviceMemorySize\(\)](#) [ICudaEngine::createExecutionContextWithoutDeviceMemory\(\)](#)

9.60.3.34 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\(\)](#)

9.60.3.35 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

<i>recorder</i>	The error recorder to register with this interface.
-----------------	---

See also

[getErrorRecorder\(\)](#)

9.60.3.36 setInputConsumedEvent()

```
bool nvinfer1::IExecutionContext::setInputConsumedEvent (
    cudaEvent_t event ) [inline], [noexcept]
```

Mark input as consumed.

Parameters

<i>event</i>	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.

9.60.3.37 setInputShape()

```
bool nvinfer1::IExecutionContext::setInputShape (
    char const * tensorName,
    Dims const & dims ) [inline], [noexcept]
```

Set shape of given input.

Parameters

<i>tensorName</i>	The name of an input tensor.
<i>dims</i>	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.

9.60.3.38 `setInputShapeBinding()`

```
TRT_DEPRECATED bool nvinfer1::IExecutionContext::setInputShapeBinding (
    int32_t bindingIndex,
    int32_t const * data ) [inline], [noexcept]
```

Set values of input tensor required by shape calculations.

Parameters

<i>bindingIndex</i>	index of an input tensor for which <code>ICudaEngine::isShapeBinding(bindingIndex)</code> and <code>ICudaEngine::bindingIsInput(bindingIndex)</code> are both true.
<i>data</i>	pointer to values of the input tensor. The number of values should be the product of the dimensions returned by <code>getBindingDimensions(bindingIndex)</code> .

If `ICudaEngine::isShapeBinding(bindingIndex)` and `ICudaEngine::bindingIsInput(bindingIndex)` are both true, this method must be called before `enqueueV2()` or `executeV2()` may be called. This method will fail unless a valid optimization profile is defined for the current execution context (`getOptimizationProfile()` must not be -1).

Warning

This function will trigger layer resource updates on the next call of `enqueueV2()/executeV2()`, possibly resulting in performance bottlenecks, if the shapes are different than the previous set shapes.

Returns

false if an error occurs (e.g. `bindingIndex` is out of range for the currently selected optimization profile or shape data is inconsistent with min-max range of the optimization profile), else true. Note that the network can still be invalid for certain combinations of input shapes that lead to invalid output shapes. To confirm the correctness of the network input shapes, check whether the output binding has valid dimensions using `getBindingDimensions()` on the output `bindingIndex`.

Deprecated Deprecated in TensorRT 8.5. Superseded by `setInputTensorAddress()` or `setTensorAddress()`.

See also

[setInputTensorAddress\(\)](#) [setTensorAddress\(\)](#)

9.60.3.39 setInputTensorAddress()

```
bool nvinfer1::IExecutionContext::setInputTensorAddress (
    char const * tensorName,
    void const * data ) [inline], [noexcept]
```

Set memory address for given input.

Parameters

<i>tensorName</i>	The name of an input tensor.
<i>data</i>	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\(\)](#)

9.60.3.40 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\(\)](#)

9.60.3.41 `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 `enqueueV2/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

<i>verbosity</i>	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\(\)](#)

9.60.3.42 `setOptimizationProfile()`

```
TRT_DEPRECATED bool nvinfer1::IExecutionContext::setOptimizationProfile (
    int32_t profileIndex ) [inline], [noexcept]
```

Select an optimization profile for the current context.

Parameters

<i>profileIndex</i>	Index of the profile. It must lie between 0 and getEngine().getNbOptimizationProfiles() - 1
---------------------	---

The selected profile will be used in subsequent calls to [executeV2\(\)/enqueueV2\(\)/enqueueV3\(\)](#).

When an optimization profile is switched via this API, TensorRT may enqueue GPU memory copy operations required to set up the new profile during the subsequent [enqueueV2\(\)/enqueueV3\(\)](#) operations. To avoid these calls during [enqueueV2\(\)/enqueueV3\(\)](#), use [setOptimizationProfileAsync\(\)](#) instead.

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 (this is particularly the case for all safe engines).

[setOptimizationProfile\(\)](#) must be called before calling [setBindingDimensions\(\)](#) and [setInputShapeBinding\(\)](#) for all dynamic input tensors or input shape tensors, which in turn must be called before [executeV2\(\)/enqueueV2\(\)/enqueueV3\(\)](#).

Warning

This function will trigger layer resource updates on the next call of [enqueueV2\(\)/enqueueV3\(\)/executeV2\(\)](#), possibly resulting in performance bottlenecks.

Returns

true if the call succeeded, else false (e.g. input out of range)

Deprecated Superseded by [setOptimizationProfileAsync](#). Deprecated prior to TensorRT 8.0 and will be removed in 9.0.

See also

[ICudaEngine::getNbOptimizationProfiles\(\)](#) [IExecutionContext::setOptimizationProfileAsync\(\)](#)

9.60.3.43 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

<i>profileIndex</i>	Index of the profile. The value must lie between 0 and getEngine().getNbOptimizationProfiles() - 1
<i>stream</i>	A cuda stream on which the cudaMemcpyAsync s may 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.

The selected profile will be used in subsequent calls to [executeV2\(\)/enqueueV2\(\)/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](#). If no execution context is assigned optimization profile 0 and a new context is created for an engine, [setOptimizationProfile\(0\)](#) is called implicitly. This functionality is deprecated in TensorRT 8.6 and will instead default all optimization profiles to 0 starting in TensorRT 9.0.

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 [setBindingDimensions\(\)](#) and [setInputShapeBinding\(\)](#) for all dynamic input tensors or input shape tensors, which in turn must be called before [executeV2\(\)/enqueueV2\(\)/enqueueV3\(\)](#).

Warning

This function will trigger layer resource updates on the next call of [enqueueV2\(\)/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\(\)](#)

[IExecutionContext::setOptimizationProfile\(\)](#)

9.60.3.44 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

<i>tensorName</i>	The name of an output tensor.
<i>outputAllocator</i>	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](#)

9.60.3.45 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

<i>size</i>	the size of persistent cache limitation in bytes. The default is 0 Bytes.
-------------	---

See also

[getPersistentCacheLimit](#)

9.60.3.46 setProfiler()

```
void nvinfer1::IExecutionContext::setProfiler (
    IProfiler * profiler ) [inline], [noexcept]
```

Set the profiler.

See also

[IProfiler getProfiler\(\)](#)

9.60.3.47 setTemporaryStorageAllocator()

```
bool nvinfer1::IExecutionContext::setTemporaryStorageAllocator (
    I_gpuAllocator * 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

<i>allocator</i>	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\(\)](#)

9.60.3.48 setTensorAddress()

```
bool nvinfer1::IExecutionContext::setTensorAddress (
    char const * tensorName,
    void * data ) [inline], [noexcept]
```

Set memory address for given input or output tensor.

Parameters

<i>tensorName</i>	The name of an input or output tensor.
<i>data</i>	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 [IOOutputAllocator](#) to set it later.

If the TensorLocation of the tensor is kHOST, the pointer must point to a host buffer of sufficient size. For shape tensors, the only supported data type is int32_t. 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 [IOOutputAllocator](#).

If getTensorShape(name) reports a -1 for any dimension of an output after all input shapes have been set, then to find out the dimensions, use [setOutputAllocator\(\)](#) to associate an [IOOutputAllocator](#) 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\(\)](#) [getTensorShape\(\)](#) [setOutputAllocator\(\)](#) [IOutputAllocator](#)

9.60.4 Member Data Documentation

9.60.4.1 mImpl

```
apiv::VExecutionContext* nvinfer1::IExecutionContext::mImpl [protected]
```

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

- [NvInferRuntime.h](#)

9.61 nvinfer1::safe::IExecutionContext Class Reference

Functionally safe context for executing inference using an engine.

```
#include <NvInferSafeRuntime.h>
```

Public Member Functions

- virtual [ICudaEngine](#) const & [getEngine](#) () const noexcept=0
Get the associated engine.
- virtual void [setName](#) ([AsciiChar](#) const *const name) noexcept=0
Set the name of the execution context.
- virtual [AsciiChar](#) const * [getName](#) () const noexcept=0
Return the name of the execution context.
- virtual void [setDeviceMemory](#) (void *const memory) noexcept=0
Set the device memory for use by this execution context.
- virtual [TRT_DEPRECATED](#) [Dims](#) [getStrides](#) (std::int32_t const bindingIndex) const noexcept=0
Return the strides of the buffer for the given binding.
- 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 `TRT_DEPRECATED` bool `enqueueV2` (void *const *const bindings, cudaStream_t const stream, cudaEvent_t const *const inputConsumed) noexcept=0
Asynchronously execute inference on a batch.
- `ExecutionContext` ()=default
- virtual `~ExecutionContext` () noexcept=default
- `ExecutionContext` (`ExecutionContext` const &)=delete
- `ExecutionContext` (`ExecutionContext` &&)=delete
- `ExecutionContext` & operator= (`ExecutionContext` const &) &=delete
- `ExecutionContext` & operator= (`ExecutionContext` &&) &=delete
- virtual void `setErrorBuffer` (`FloatingPointErrorInformation` *const buffer) noexcept=0
Set error buffer output for floating point errors.
- virtual `FloatingPointErrorInformation` * `getErrorBuffer` () const noexcept=0
Get error buffer output for floating point errors.
- virtual `Dims` `getTensorStrides` (`AsciiChar` const *tensorName) const noexcept=0
Return the strides of the buffer for the given tensor name.
- virtual bool `setInputTensorAddress` (`AsciiChar` const *tensorName, void const *data) noexcept=0
Set memory address for given input tensor.
- virtual bool `setOutputTensorAddress` (`AsciiChar` const *tensorName, void *data) noexcept=0
Set memory address for given output tensor.
- virtual bool `setInputConsumedEvent` (cudaEvent_t event) noexcept=0
Mark input as consumed.
- virtual cudaEvent_t `getInputConsumedEvent` () const noexcept=0
Return the event associated with consuming the input.
- virtual void const * `getInputTensorAddress` (`AsciiChar` const *tensorName) const noexcept=0
Get memory address for given input tensor.
- virtual void * `getOutputTensorAddress` (`AsciiChar` const *tensorName) const noexcept=0
Get memory address for given output tensor.
- virtual bool `enqueueV3` (cudaStream_t stream) noexcept=0
Asynchronously execute inference.

9.61.1 Detailed Description

Functionally safe context for executing inference using an engine.

Multiple safe execution contexts may exist for one `safe::ICudaEngine` instance, allowing the same engine to be used for the execution of multiple inputs simultaneously.

Warning

Do not call the APIs of the same `ExecutionContext` from multiple threads at any given time. Each concurrent execution must have its own instance of an `ExecutionContext`.

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.61.2 Constructor & Destructor Documentation

9.61.2.1 IExecutionContext() [1/3]

```
nvinfer1::safe::IExecutionContext::IExecutionContext ( ) [default]
```

9.61.2.2 ~IExecutionContext()

```
virtual nvinfer1::safe::IExecutionContext::~~IExecutionContext ( ) [virtual], [default], [noexcept]
```

9.61.2.3 IExecutionContext() [2/3]

```
nvinfer1::safe::IExecutionContext::IExecutionContext (
    IExecutionContext const & ) [delete]
```

9.61.2.4 IExecutionContext() [3/3]

```
nvinfer1::safe::IExecutionContext::IExecutionContext (
    IExecutionContext && ) [delete]
```

9.61.3 Member Function Documentation**9.61.3.1 enqueueV2()**

```
virtual TRT\_DEPRECATED bool nvinfer1::safe::IExecutionContext::enqueueV2 (
    void *const *const bindings,
    cudaStream_t const stream,
    cudaEvent_t const *const inputConsumed ) [pure virtual], [noexcept]
```

Asynchronously execute inference on a batch.

This method requires an array of input and output buffers. The mapping from tensor names to indices can be queried using [safe::ICudaEngine::getBindingIndex\(\)](#). This method only works for an execution context built from a network without an implicit batch dimension.

Parameters

<i>bindings</i>	An array of pointers to input and output buffers for the network.
<i>stream</i>	A cuda stream on which the inference kernels will be enqueued.
<i>inputConsumed</i>	An optional event which will be signaled when the input buffers can be refilled with new data.

Returns

True if the kernels were enqueued successfully.

Deprecated Deprecated in TensorRT 8.5. Superseded by [enqueueV3\(\)](#).

See also

[safe::IExecutionContext::enqueueV3\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.61.3.2 enqueueV3()

```
virtual bool nvinfer1::safe::IExecutionContext::enqueueV3 (
    cudaStream_t stream ) [pure virtual], [noexcept]
```

Asynchronously execute inference.

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.

Parameters

<i>stream</i>	A cuda stream on which the inference kernels will be enqueued.
---------------	--

Returns

True on success, false if any execution error occurred.

Usage considerations

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

9.61.3.3 getEngine()

```
virtual ICudaEngine const & nvinfer1::safe::IExecutionContext::getEngine ( ) const [pure virtual],  
[noexcept]
```

Get the associated engine.

See also

[safe::ICudaEngine](#)

Usage considerations

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

9.61.3.4 getErrorBuffer()

```
virtual FloatingPointErrorInformation * nvinfer1::safe::IExecutionContext::getErrorBuffer ( )  
const [pure virtual], [noexcept]
```

Get error buffer output for floating point errors.

Returns

Pointer to device memory to use as floating point error buffer or nullptr if not set.

See also

[setErrorBuffer\(\)](#)

Usage considerations

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

9.61.3.5 `getErrorRecorder()`

```
virtual IErrorRecorder * nvinfer1::safe::IExecutionContext::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.

Returns

A pointer to the [IErrorRecorder](#) object that has been registered.

See also

[setErrorRecorder\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.61.3.6 `getInputConsumedEvent()`

```
virtual cudaEvent_t nvinfer1::safe::IExecutionContext::getInputConsumedEvent ( ) const [pure virtual], [noexcept]
```

Return the event associated with consuming the input.

Returns

The cuda event, nullptr will be returned if the event is not set yet.

Usage considerations

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

9.61.3.7 `getInputTensorAddress()`

```
virtual void const * nvinfer1::safe::IExecutionContext::getInputTensorAddress (
    AsciiChar const * tensorName ) const [pure virtual], [noexcept]
```

Get memory address for given input tensor.

Parameters

<i>tensorName</i>	The name of an input tensor.
-------------------	------------------------------

Warning

The string `tensorName` must be 1024 characters or less including NULL terminator and must be NULL terminated.

Returns

The memory address for the given input tensor. `nullptr` will be returned if (1) name is not the name of an input tensor, or (2) name is `nullptr`, or (3) name exceeds the string length limit, or (4) the memory address for the given input tensor is not set yet.

Usage considerations

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

9.61.3.8 getName()

```
virtual AsciiChar const * nvinfer1::safe::IExecutionContext::getName ( ) const [pure virtual],
[noexcept]
```

Return the name of the execution context.

See also

[setName\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.61.3.9 getOutputTensorAddress()

```
virtual void * nvinfer1::safe::IExecutionContext::getOutputTensorAddress (
    AsciiChar const * tensorName ) const [pure virtual], [noexcept]
```

Get memory address for given output tensor.

Parameters

<i>tensorName</i>	The name of an output tensor.
-------------------	-------------------------------

Warning

The string `tensorName` must be 1024 characters or less including NULL terminator and must be NULL terminated.

Returns

Raw output data pointer (void*) for given output tensor, return nullptr if (1) name is not the name of an output tensor, or (2) name is nullptr, or (3) name exceeds the string length limit, or (4) the memory address for the given output tensor is not set yet.

Usage considerations

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

9.61.3.10 getStrides()

```
virtual TRT\_DEPRECATED Dims nvinfer1::safe::IExecutionContext::getStrides (
    std::int32_t const bindingIndex ) const [pure virtual], [noexcept]
```

Return the strides of the buffer for the given binding.

Parameters

<i>bindingIndex</i>	The binding index.
---------------------	--------------------

Deprecated Deprecated in TensorRT 8.5. Superseded by [getTensorStrides\(\)](#).

See also

[safe::IExecutionContext::getTensorStrides\(\)](#)

Usage considerations

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

9.61.3.11 getTensorStrides()

```
virtual Dims nvinfer1::safe::IExecutionContext::getTensorStrides (
    AsciiChar const * tensorName ) const [pure virtual], [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.

Parameters

<i>tensorName</i>	The name of an input or output tensor.
-------------------	--

Warning

The string `tensorName` must be 1024 characters or less including NULL terminator and must be NULL terminated.

Returns

The strides of the buffer for the given tensor name. `Dims{-1, {}}` will be returned if (1) name is not the name of an input or output tensor, or (2) name is nullptr, or (3) name exceeds the string length limit.

Usage considerations

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

9.61.3.12 operator=() [1/2]

```
IExecutionContext & nvinfer1::safe::IExecutionContext::operator= (
    IExecutionContext && ) & [delete]
```

9.61.3.13 operator=() [2/2]

```
IExecutionContext & nvinfer1::safe::IExecutionContext::operator= (
    IExecutionContext const & ) & [delete]
```

9.61.3.14 setDeviceMemory()

```
virtual void nvinfer1::safe::IExecutionContext::setDeviceMemory (
    void *const memory ) [pure virtual], [noexcept]
```

Set the device memory for use by this execution context.

If using [enqueueV2\(\)](#) to run the network, The memory is in use from the invocation of [enqueueV2\(\)](#) until network execution is complete. Releasing or otherwise using the memory for other purposes during this time will result in undefined behavior.

Warning

Do not release or use for other purposes the memory set here during network execution.

See also

[safe::ICudaEngine::getDeviceMemorySize\(\)](#) [safe::ICudaEngine::createExecutionContextWithoutDeviceMemory\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.61.3.15 setErrorBuffer()

```
virtual void nvinfer1::safe::IExecutionContext::setErrorBuffer (
    FloatingPointErrorInformation *const buffer ) [pure virtual], [noexcept]
```

Set error buffer output for floating point errors.

The error buffer output must be allocated in device memory and will be used for subsequent calls to [enqueueV2](#). Checking the contents of the error buffer after inference is the responsibility of the application. The pointer passed here must have alignment adequate for the [FloatingPointErrorInformation](#) struct.

Warning

Do not release or use the contents of the error buffer for any other purpose before synchronizing on the CUDA stream passed to [enqueueV2](#).

Parameters

<i>buffer</i>	The device memory to use as floating point error buffer
---------------	---

See also

[getErrorBuffer\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.61.3.16 setErrorRecorder()

```
virtual void nvinfer1::safe::IExecutionContext::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

<i>recorder</i>	The error recorder to register with this interface.
-----------------	---

See also

[getErrorRecorder\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.61.3.17 setInputConsumedEvent()

```
virtual bool nvinfer1::safe::IExecutionContext::setInputConsumedEvent (
    cudaEvent_t event ) [pure virtual], [noexcept]
```

Mark input as consumed.

Passing event==nullptr removes whatever event was set, if any.

Parameters

<i>event</i>	The cuda event that is triggered after all input tensors have been consumed.
--------------	--

Returns

True on success, false if error occurred.

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.61.3.18 setInputTensorAddress()

```
virtual bool nvinfer1::safe::IExecutionContext::setInputTensorAddress (
    AsciiChar const * tensorName,
    void const * data ) [pure virtual], [noexcept]
```

Set memory address for given input tensor.

An address defaults to nullptr.

Before calling [enqueueV3\(\)](#), each input must have a non-null address.

Parameters

<i>tensorName</i>	The name of an input tensor.
<i>data</i>	The pointer (void const*) to the const data owned by the user.

Warning

The string `tensorName` must be 1024 characters or less including NULL terminator and must be NULL terminated.

The pointer must have at least 256-byte alignment.

Returns

True on success, false if (1) name is not the name of an input tensor, or (2) name is nullptr, or (3) name exceeds the string length limit, or (4) pointer to the const data is nullptr or not aligned.

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.61.3.19 setName()

```
virtual void nvinfer1::safe::IExecutionContext::setName (
    AsciiChar const *const name ) [pure virtual], [noexcept]
```

Set the name of the execution context.

This method copies the name string.

Warning

Strings passed to the runtime must be 1024 characters or less including NULL terminator and must be NULL terminated.

See also

[getName\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.61.3.20 setOutputTensorAddress()

```
virtual bool nvinfer1::safe::IExecutionContext::setOutputTensorAddress (
    AsciiChar const * tensorName,
    void * data ) [pure virtual], [noexcept]
```

Set memory address for given output tensor.

An address defaults to nullptr.

Before calling [enqueueV3\(\)](#), each output must have a non-null address.

Parameters

<i>tensorName</i>	The name of an output tensor.
<i>data</i>	The pointer (void*) to the data owned by the user.

Warning

The string `tensorName` must be 1024 characters or less including NULL terminator and must be NULL terminated.

The pointer must have at least 256-byte alignment.

Returns

True on success. Return false if (1) name is not the name of an output tensor, or (2) name is nullptr, or (3) name exceeds the string length limit, or (4) pointer to data is nullptr or not aligned.

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

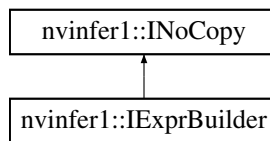
The documentation for this class was generated from the following file:

- [NvInferSafeRuntime.h](#)

9.62 nvinfer1::IExprBuilder Class Reference

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

Inheritance diagram for `nvinfer1::IExprBuilder`:

**Public Member Functions**

- `IDimensionExpr` const * `constant` (int32_t value) noexcept
Return pointer to IDimensionExp for given value.
- `IDimensionExpr` const * `operation` (`DimensionOperation` op, `IDimensionExpr` const &first, `IDimensionExpr` const &second) noexcept

Protected Member Functions

- virtual `~IExprBuilder` () noexcept=default

Protected Attributes

- `apiv::VExprBuilder * mImpl`

9.62.1 Detailed Description

Object for constructing [IDimensionExpr](#).

There is no public way to construct an [IExprBuilder](#). It appears as an argument to method [IPluginV2DynamicExt::getOutputDimensions\(\)](#). 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\(\)](#) returns.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

See also

[IDimensionExpr](#)

9.62.2 Constructor & Destructor Documentation

9.62.2.1 ~IExprBuilder()

```
virtual nvinfer1::IExprBuilder::~~IExprBuilder ( ) [protected], [virtual], [default], [noexcept]
```

9.62.3 Member Function Documentation

9.62.3.1 constant()

```
IDimensionExpr const * nvinfer1::IExprBuilder::constant (
    int32_t value ) [inline], [noexcept]
```

Return pointer to [IDimensionExp](#) for given value.

9.62.3.2 operation()

```
IDimensionExpr const * nvinfer1::IExprBuilder::operation (
    DimensionOperation op,
    IDimensionExpr const & first,
    IDimensionExpr const & second ) [inline], [noexcept]
```

Return pointer to IDimensionExpr that represents the given operation applied to first and second. Returns nullptr if op is not a valid DimensionOperation.

9.62.4 Member Data Documentation

9.62.4.1 mImpl

```
apiv::VExprBuilder* nvinfer1::IExprBuilder::mImpl [protected]
```

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

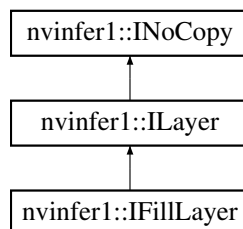
- [NvInferRuntime.h](#)

9.63 nvinfer1::IFillLayer Class Reference

Generate an output tensor with specified mode.

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

Inheritance diagram for nvinfer1::IFillLayer:



Public Member Functions

- void [setDimensions](#) ([Dims](#) 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 [setInput](#) (int32_t index, [ITensor](#) &tensor) noexcept
replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual [~IFillLayer](#) () noexcept=default

Protected Attributes

- apiv::VFillLayer * [mImpl](#)

9.63.1 Detailed Description

Generate an output tensor with specified mode.

The fill layer has two variants, static and dynamic. Static fill specifies its parameters at layer creation time via [Dims](#) and the get/set accessor functions of the [IFillLayer](#). Dynamic fill specifies one or more of its parameters as ITensors, by using [ILayer::setInput](#) to add a corresponding input. The corresponding static parameter is used if an input is missing or null.

The shape of the output is specified by the parameter `Dimension`, or if non-null and present, the first input, which must be a 1D Int32 shape tensor. Thus an application can determine if the [IFillLayer](#) has a dynamic output shape based on whether it has a non-null first input.

Alpha and Beta are treated differently based on the Fill Operation specified. See details in [IFillLayer::setAlpha\(\)](#), [IFillLayer::setBeta\(\)](#), and [IFillLayer::setInput\(\)](#).

A fill layer can produce a shape tensor if the following restrictions are met:

- The FillOperation is kLinspace.
- The output is an Int32 or Float tensor within the volume limit of a shape tensor.
- There is at most one input, and if so, that input is input 0.
- If input 0 exists, the length of the output tensor 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.

9.63.2 Constructor & Destructor Documentation

9.63.2.1 ~IFillLayer()

```
virtual nvinfer1::IFillLayer::~~IFillLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.63.3 Member Function Documentation

9.63.3.1 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](#)

9.63.3.2 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](#)

9.63.3.3 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](#)

9.63.3.4 getOperation()

```
FillOperation nvinfer1::IFillLayer::getOperation ( ) const [inline], [noexcept]
```

Get the fill operation for the layer.

See also

[setOperation\(\)](#), [FillOperation](#)

9.63.3.5 setAlpha()

```
void nvinfer1::IFillLayer::setAlpha (
    double alpha ) [inline], [noexcept]
```

Set the alpha parameter.

Parameters

<i>alpha</i>	has different meanings for each operator:
--------------	---

Operation | Usage kLinspace | the start value, defaults to 0.0; kRandomUniform | the minimum value, defaults to 0.0; kRandomNormal | the mean of the normal distribution, default is 0.0;

If a second input had been used to create this layer, that input is reset to null by this method.

See also

[getAlpha](#)

9.63.3.6 setBeta()

```
void nvinfer1::IFillLayer::setBeta (
    double beta ) [inline], [noexcept]
```

Set the beta parameter.

Parameters

<i>beta</i>	has different meanings for each operator:
-------------	---

Operation | Usage kLinspace | the delta value, defaults to 1.0; kRandomUniform | the maximal value, defaults to 1.0; kRandomNormal | the standard deviation of the normal distribution, default is 1.0;

If a third input had been used to create this layer, that input is reset to null by this method.

See also

[getBeta](#)

9.63.3.7 setDimensions()

```
void nvinfer1::IFillLayer::setDimensions (
    Dims dimensions ) [inline], [noexcept]
```

Set the output tensor's dimensions.

Parameters

<i>dimensions</i>	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](#)

9.63.3.8 setInput()

```
void nvinfer1::ILayer::setInput (
    int32_t index,
    ITensor & tensor ) [inline], [noexcept]
```

replace an input of this layer with a specific tensor.

Parameters

<i>index</i>	the index of the input to set.
<i>tensor</i>	the new input tensor

Indices for kLinspace are described as:

- 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.

Indices for kRandomUniform are described as:

- 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.

Indices for kRandomNormal are described as:

- 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.

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.

9.63.3.9 setOperation()

```
void nvinfer1::IFillLayer::setOperation (
    FillOperation op ) [inline], [noexcept]
```

Set the fill operation for the layer.

See also

[getOperation\(\)](#), [FillOperation](#)

9.63.4 Member Data Documentation

9.63.4.1 mImpl

```
apiv::VFillLayer* nvinfer1::IFillLayer::mImpl [protected]
```

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

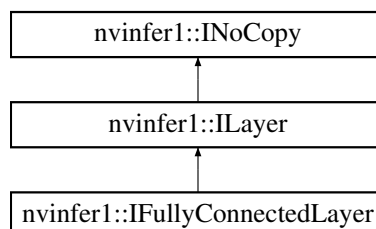
- [NvInfer.h](#)

9.64 nvinfer1::IFullyConnectedLayer Class Reference

A fully connected layer in a network definition. This layer expects an input tensor of three or more non-batch dimensions. The input is automatically reshaped into an $M \times V$ tensor X , where V is a product of the last three dimensions and M is a product of the remaining dimensions (where the product over 0 dimensions is defined as 1). For example:

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

Inheritance diagram for `nvinfer1::IFullyConnectedLayer`:



Public Member Functions

- void [setNbOutputChannels](#) (int32_t nbOutputs) noexcept
Set the number of output channels K from the fully connected layer.
- int32_t [getNbOutputChannels](#) () const noexcept
Get the number of output channels K from the fully connected layer.
- void [setKernelWeights](#) ([Weights](#) weights) noexcept
Set the kernel weights, given as a $K \times C$ matrix in row-major order.
- [Weights](#) [getKernelWeights](#) () const noexcept
Get the kernel weights.
- void [setBiasWeights](#) ([Weights](#) weights) noexcept
Set the bias weights.
- [Weights](#) [getBiasWeights](#) () const noexcept
Get the bias weights.
- void [setInput](#) (int32_t index, [ITensor](#) &tensor) noexcept
Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual [~IFullyConnectedLayer](#) () noexcept=default

Protected Attributes

- apiv::VFullyConnectedLayer * [mImpl](#)

9.64.1 Detailed Description

A fully connected layer in a network definition. This layer expects an input tensor of three or more non-batch dimensions. The input is automatically reshaped into an $M \times V$ tensor X , where V is a product of the last three dimensions and M is a product of the remaining dimensions (where the product over 0 dimensions is defined as 1). For example:

- If the input tensor has shape $\{C, H, W\}$, then the tensor is reshaped into $\{1, C * H * W\}$.
- If the input tensor has shape $\{P, C, H, W\}$, then the tensor is reshaped into $\{P, C * H * W\}$.

The layer then performs the following operation:

```

$$Y := \text{matmul}(X, W^T) + \text{bias}$$

```

Where X is the $M \times V$ tensor defined above, W is the $K \times V$ weight tensor of the layer, and bias is a row vector size K that is broadcasted to $M \times K$. K is the number of output channels, and configurable via [setNbOutputChannels\(\)](#). If bias is not specified, it is implicitly 0.

The $M \times K$ result Y is then reshaped such that the last three dimensions are $\{K, 1, 1\}$ and the remaining dimensions match the dimensions of the input tensor. For example:

- If the input tensor has shape $\{C, H, W\}$, then the output tensor will have shape $\{K, 1, 1\}$.
- If the input tensor has shape $\{P, C, H, W\}$, then the output tensor will have shape $\{P, K, 1, 1\}$.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

Deprecated Deprecated in TensorRT 8.4. Superseded by [IMatrixMultiplyLayer](#).

9.64.2 Constructor & Destructor Documentation

9.64.2.1 ~IFullyConnectedLayer()

```
virtual nvinfer1::IFullyConnectedLayer::~~IFullyConnectedLayer ( ) [protected], [virtual], [default],
[noexcept]
```

9.64.3 Member Function Documentation

9.64.3.1 getBiasWeights()

```
Weights nvinfer1::IFullyConnectedLayer::getBiasWeights ( ) const [inline], [noexcept]
```

Get the bias weights.

See also

[setBiasWeightsWeights\(\)](#)

9.64.3.2 getKernelWeights()

```
Weights nvinfer1::IFullyConnectedLayer::getKernelWeights ( ) const [inline], [noexcept]
```

Get the kernel weights.

See also

[setKernelWeights\(\)](#)

9.64.3.3 getNbOutputChannels()

```
int32_t nvinfer1::IFullyConnectedLayer::getNbOutputChannels ( ) const [inline], [noexcept]
```

Get the number of output channels K from the fully connected layer.

See also

[setNbOutputChannels\(\)](#)

9.64.3.4 setBiasWeights()

```
void nvinfer1::IFullyConnectedLayer::setBiasWeights (
    Weights weights ) [inline], [noexcept]
```

Set the bias weights.

Bias is optional. To omit bias, set the count value in the weights structure to zero.

See also

[getBiasWeightsWeights\(\)](#)

9.64.3.5 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

<i>index</i>	the index of the input to modify.
<i>tensor</i>	the new input tensor

Only index 0 (data input) is valid, unless explicit-quantization mode is enabled. In explicit-quantization mode, input with index 1 is the kernel-weights tensor, if present. The kernel-weights tensor must be a build-time constant (computable at build-time via constant-folding) and an output of a dequantize layer. If input index 1 is used then the kernel-weights parameter must be set to empty [Weights](#).

See also

[getKernelWeights\(\)](#), [setKernelWeights\(\)](#)

The indices are as follows:

- 0: The input activation tensor.
- 1: The kernel weights tensor (a constant tensor).

If this function is called with the value 1, then the function [getNbInputs\(\)](#) changes from returning 1 to 2.

9.64.3.6 [setKernelWeights\(\)](#)

```
void nvinfer1::IFullyConnectedLayer::setKernelWeights (
    Weights weights ) [inline], [noexcept]
```

Set the kernel weights, given as a $K \times C$ matrix in row-major order.

See also

[getKernelWeights\(\)](#)

9.64.3.7 [setNbOutputChannels\(\)](#)

```
void nvinfer1::IFullyConnectedLayer::setNbOutputChannels (
    int32_t nbOutputs ) [inline], [noexcept]
```

Set the number of output channels K from the fully connected layer.

If executing this layer on DLA, number of output channels must in the range [1,8192].

See also

[getNbOutputChannels\(\)](#)

9.64.4 Member Data Documentation

9.64.4.1 mImpl

```
apiv::VFullyConnectedLayer* nvinfer1::IFullyConnectedLayer::mImpl [protected]
```

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

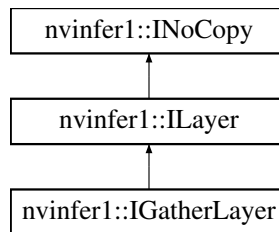
- [NvInfer.h](#)

9.65 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. The gathering of indexing starts from the dimension of data[NbElementWiseDims:]. The NbElementWiseDims must be less than the Rank of the data input.
- 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=default

Protected Attributes

- `apiv::VGatherLayer * mImpl`

9.65.1 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 $r \geq 1$ 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`: $q \geq 0$
 - `GatherMode::kND`: $q \geq 1$ 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`: $s = q + r - 1 - \text{nbElementwiseDims}$
 - `GatherMode::kND`: $s = q + r - \text{indices.d}[q-1] - 1 - \text{nbElementwiseDims}$
 - `GatherMode::kELEMENT`: $s = q = r$. The output can be a shape tensor only if the mode is `GatherMode::kDEFAULT`.

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`.

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**: $0 \leq \text{nbElementWiseDims} < \text{rank}(\text{Data}) - 1$. 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.
- Only mode **GatherMode::kDEFAULT** supports an implicit batch dimensions or broadcast on the elementwise dimensions.
- 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.

9.65.2 Constructor & Destructor Documentation

9.65.2.1 ~IGatherLayer()

```
virtual nvinfer1::IGatherLayer::~IGatherLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.65.3 Member Function Documentation

9.65.3.1 `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\(\)](#)

9.65.3.2 `getMode()`

```
GatherMode nvinfer1::IGatherLayer::getMode ( ) const [inline], [noexcept]
```

Get the gather mode.

See also

[setMode\(\)](#)

9.65.3.3 `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\(\)](#)

9.65.3.4 `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\(\)](#)

9.65.3.5 setMode()

```
void nvinfer1::IGatherLayer::setMode (
    GatherMode mode ) [inline], [noexcept]
```

Set the gather mode.

See also

[getMode\(\)](#)

9.65.3.6 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

<i>elementWiseDims</i>	number of dims to be handled as elementwise.
------------------------	--

Default: 0

The value of nbElementWiseDims and GatherMode are checked during network validation:

[GatherMode::kDEFAULT](#): nbElementWiseDims must be 0 if there is an implicit batch dimension. It can be 0 or 1 if there is not an implicit batch dimension. [GatherMode::kND](#): nbElementWiseDims can be between 0 and one less than rank(data). [GatherMode::kELEMENT](#): nbElementWiseDims must be 0

See also

[getNbElementWiseDims\(\)](#)

9.65.4 Member Data Documentation

9.65.4.1 mImpl

```
apiv::VGatherLayer* nvinfer1::IGatherLayer::mImpl [protected]
```

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

- [NvInfer.h](#)

9.66 nvinfer1::IGpuAllocator Class Reference

Application-implemented class for controlling allocation on the GPU.

```
#include <NvInferRuntimeCommon.h>
```

Public Member Functions

- virtual void * [allocate](#) (uint64_t const size, uint64_t const alignment, [AllocatorFlags](#) const flags) noexcept=0
- virtual [TRT_DEPRECATED](#) void [free](#) (void *const memory) noexcept=0
- virtual [~IGpuAllocator](#) ()=default
- [IGpuAllocator](#) ()=default
- virtual void * [reallocate](#) (void *, uint64_t, uint64_t) noexcept
- virtual bool [deallocate](#) (void *const memory) noexcept

9.66.1 Detailed Description

Application-implemented class for controlling allocation on the GPU.

9.66.2 Constructor & Destructor Documentation

9.66.2.1 ~IGpuAllocator()

```
virtual nvinfer1::IGpuAllocator::~~IGpuAllocator ( ) [virtual], [default]
```

Destructor declared virtual as general good practice for a class with virtual methods. TensorRT never calls the destructor for an [IGpuAllocator](#) defined by the application.

9.66.2.2 IGpuAllocator()

```
nvinfer1::IGpuAllocator::IGpuAllocator ( ) [default]
```

9.66.3 Member Function Documentation

9.66.3.1 allocate()

```
virtual void * nvinfer1::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

<i>size</i>	The size of the memory required.
<i>alignment</i>	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.
<i>flags</i>	Reserved for future use. In the current release, 0 will be passed.

If an allocation request of size 0 is made, nullptr should be returned.

If an allocation request cannot be satisfied, nullptr should be returned.

Note

The implementation must guarantee thread safety for concurrent allocate/free/reallocate/deallocate requests.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads.

9.66.3.2 deallocate()

```
virtual bool nvinfer1::IGpuAllocator::deallocate (
    void *const memory ) [inline], [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

<i>memory</i>	The acquired memory.
---------------	----------------------

Returns

True if the acquired memory is released successfully.

Note

The implementation must guarantee thread safety for concurrent allocate/free/reallocate/deallocate requests.

If user-implemented [free\(\)](#) might hit an error condition, the user should override [deallocate\(\)](#) as the primary implementation and override [free\(\)](#) to call [deallocate\(\)](#) for backwards compatibility.

See also

[free\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads.

9.66.3.3 free()

```
virtual TRT\_DEPRECATED void nvinfer1::IGpuAllocator::free (
    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

<i>memory</i>	The acquired memory.
---------------	----------------------

Note

The implementation must guarantee thread safety for concurrent allocate/free/reallocate/deallocate requests.

See also

[deallocate\(\)](#)

Deprecated Deprecated in TensorRT 8.0. Superseded by [deallocate](#).

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads.

9.66.3.4 [reallocate\(\)](#)

```
virtual void * nvinfer1::IGpuAllocator::reallocate (
    void * ,
    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

<i>baseAddr</i>	the address of the original allocation.
<i>alignment</i>	The alignment used by the original allocation.
<i>newSize</i>	The new memory size required.

Returns

the address of the reallocated memory

Note

The implementation must guarantee thread safety for concurrent allocate/free/reallocate/deallocate requests.

Usage considerations

- 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:

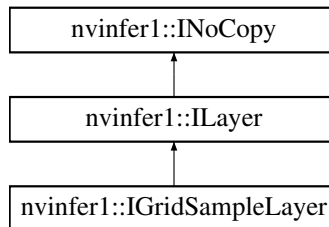
- [NvInferRuntimeCommon.h](#)

9.67 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=default

Protected Attributes

- apiv::VGridSampleLayer * [mImpl](#)

9.67.1 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.

9.67.2 Constructor & Destructor Documentation

9.67.2.1 ~IGridSampleLayer()

```
virtual nvinfer1::IGridSampleLayer::~~IGridSampleLayer ( ) [protected], [virtual], [default],
[noexcept]
```

9.67.3 Member Function Documentation

9.67.3.1 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.

9.67.3.2 `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.

9.67.3.3 `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.

9.67.3.4 `setAlignCorners()`

```
void nvinfer1::IGridSampleLayer::setAlignCorners (
    bool alignCorners ) [inline], [noexcept]
```

Set the align corners mode.

See also

[getAlignCorners\(\)](#)

9.67.3.5 setInterpolationMode()

```
void nvinfer1::IGridSampleLayer::setInterpolationMode (
    InterpolationMode mode ) [inline], [noexcept]
```

Set the grid sample interpolation mode.

See also

[getInterpolationMode\(\)](#)

9.67.3.6 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.

9.67.4 Member Data Documentation

9.67.4.1 mImpl

```
apiv::VGridSampleLayer* nvinfer1::IGridSampleLayer::mImpl [protected]
```

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

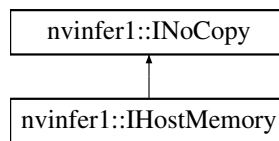
- [NvInfer.h](#)

9.68 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=default
- 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.
- `TRT_DEPRECATED` void `destroy ()` noexcept

Protected Attributes

- apiv::VHostMemory * `mImpl`

Additional Inherited Members

9.68.1 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.

9.68.2 Constructor & Destructor Documentation

9.68.2.1 ~IHostMemory()

```
virtual nvinfer1::IHostMemory::~~IHostMemory ( ) [virtual], [default], [noexcept]
```

9.68.3 Member Function Documentation

9.68.3.1 data()

```
void * nvinfer1::IHostMemory::data ( ) const [inline], [noexcept]
```

A pointer to the raw data that is owned by the library.

9.68.3.2 destroy()

```
TRT_DEPRECATED void nvinfer1::IHostMemory::destroy ( ) [inline], [noexcept]
```

Destroy the allocated memory.

Deprecated Deprecated in TRT 8.0. Superseded by delete.

Warning

Calling destroy on a managed pointer will result in a double-free error.

9.68.3.3 size()

```
std::size_t nvinfer1::IHostMemory::size ( ) const [inline], [noexcept]
```

The size in bytes of the data that was allocated.

9.68.3.4 type()

```
DataType nvinfer1::IHostMemory::type ( ) const [inline], [noexcept]
```

The type of the memory that was allocated.

9.68.4 Member Data Documentation

9.68.4.1 mImpl

```
apiv::VHostMemory* nvinfer1::IHostMemory::mImpl [protected]
```

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

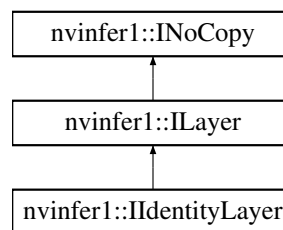
- [NvInferRuntime.h](#)

9.69 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=default

Protected Attributes

- apiv::VIdentityLayer * [mImpl](#)

Additional Inherited Members

9.69.1 Detailed Description

A layer that represents the identity function.

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`, is recognized as incorrect as of TensorRT 8.4, but is retained for API compatibility within TensorRT 8.x releases. In a future major release the behavior will change to record an error if the network output tensor type is incompatible with the layer output type. E.g., implicit conversion from kFLOAT to kINT32 will not be allowed, and instead such a conversion will require calling `setOutputType(DataType::kINT32)`.

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 ~IIdentityLayer()

```
virtual nvinfer1::IIdentityLayer::~~IIdentityLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.69.3 Member Data Documentation

9.69.3.1 mImpl

```
apiv::VIdentityLayer* nvinfer1::IIdentityLayer::mImpl [protected]
```

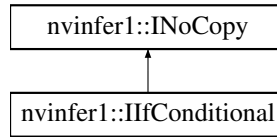
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.70 nvinfer1::IIfConditional Class Reference

```
#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=default`

Protected Attributes

- `apiv::VIfConditional * mImpl`

9.70.1 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 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 true/false subgraphs are the same.

9.70.2 Constructor & Destructor Documentation

9.70.2.1 ~IIfConditional()

```
virtual nvinfer1::IIfConditional::~~IIfConditional ( ) [protected], [virtual], [default], [noexcept]
```

9.70.3 Member Function Documentation

9.70.3.1 addInput()

```
IIfConditionalInputLayer * nvinfer1::IIfConditional::addInput (
    ITensor & input ) [inline], [noexcept]
```

Add an If-conditional input.

Parameters

<i>input</i>	An input to the conditional that can be used by either or both of the conditional's subgraphs.
--------------	--

See also

[IIfConditionalInputLayer](#)

9.70.3.2 addOutput()

```
IIfConditionalOutputLayer * nvinfer1::IIfConditional::addOutput (
    ITensor & trueSubgraphOutput,
    ITensor & falseSubgraphOutput ) [inline], [noexcept]
```

Add an If-conditional output.

Parameters

<i>trueSubgraphOutput</i>	The output of the subgraph executed when the conditional evaluates to true.
<i>falseSubgraphOutput</i>	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.

See also

[IIfConditionalOutputLayer](#)

9.70.3.3 getName()

```
char const * nvinfer1::IIfConditional::getName ( ) const [inline], [noexcept]
```

Return the name of the conditional.

See also

[setName\(\)](#)

9.70.3.4 setCondition()

```
IConditionLayer * nvinfer1::IIfConditional::setCondition (
    ITensor & condition ) [inline], [noexcept]
```

Set the condition tensor for this If-Conditional construct.

Parameters

<i>condition</i>	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](#)

9.70.3.5 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\(\)](#)

9.70.4 Member Data Documentation

9.70.4.1 mImpl

```
apiv::VIfConditional* nvinfer1::IIfConditional::mImpl [protected]
```

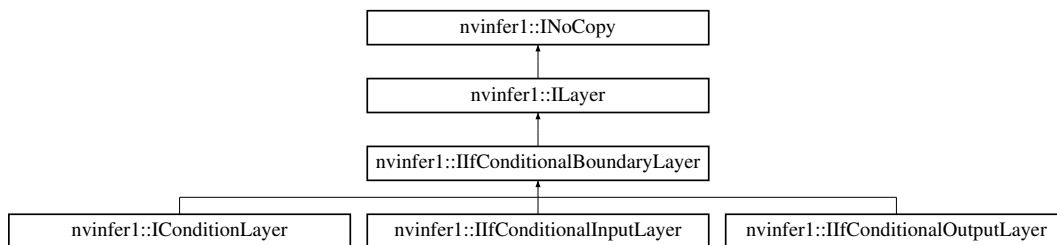
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.71 nvinfer1::IIfConditionalBoundaryLayer Class Reference

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

Inheritance diagram for nvinfer1::IIfConditionalBoundaryLayer:



Public Member Functions

- [IIfConditional](#) * [getConditional](#) () const noexcept
Return pointer to the [IIfConditional](#) associated with this boundary layer.

Protected Member Functions

- virtual [~IIfConditionalBoundaryLayer](#) () noexcept=default

Protected Attributes

- `apiv::VConditionalBoundaryLayer * mBoundary`

9.71.1 Detailed Description

This is a base class for Conditional boundary layers.

Boundary layers are used to demarcate the boundaries of Conditionals.

9.71.2 Constructor & Destructor Documentation

9.71.2.1 `~IIfConditionalBoundaryLayer()`

```
virtual nvinfer1::IIfConditionalBoundaryLayer::~~IIfConditionalBoundaryLayer ( ) [protected],  
[virtual], [default], [noexcept]
```

9.71.3 Member Function Documentation

9.71.3.1 `getConditional()`

```
IIfConditional * nvinfer1::IIfConditionalBoundaryLayer::getConditional ( ) const [inline], [noexcept]
```

Return pointer to the [IIfConditional](#) associated with this boundary layer.

9.71.4 Member Data Documentation

9.71.4.1 `mBoundary`

```
apiv::VConditionalBoundaryLayer* nvinfer1::IIfConditionalBoundaryLayer::mBoundary [protected]
```

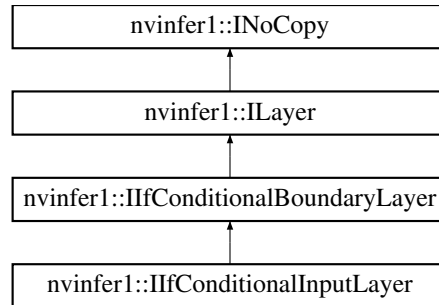
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.72 nvinfer1::IIfConditionalInputLayer Class Reference

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

Inheritance diagram for nvinfer1::IIfConditionalInputLayer:



Protected Member Functions

- virtual [~IIfConditionalInputLayer](#) () noexcept=default

Protected Attributes

- apiv::VConditionalInputLayer * [mImpl](#)

Additional Inherited Members

9.72.1 Detailed Description

This layer represents an input to an [IIfConditional](#).

9.72.2 Constructor & Destructor Documentation

9.72.2.1 ~IIfConditionalInputLayer()

```
virtual nvinfer1::IIfConditionalInputLayer::~~IIfConditionalInputLayer ( ) [protected], [virtual],
[default], [noexcept]
```

9.72.3 Member Data Documentation

9.72.3.1 mImpl

```
apiv::VConditionalInputLayer* nvinfer1::IIfConditionalInputLayer::mImpl [protected]
```

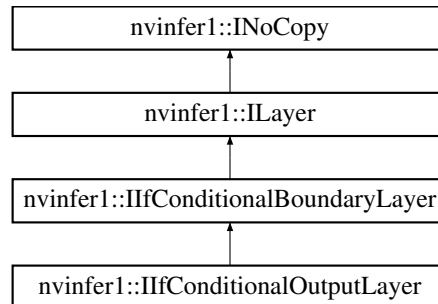
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.73 nvinfer1::IIfConditionalOutputLayer Class Reference

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

Inheritance diagram for nvinfer1::IIfConditionalOutputLayer:



Protected Member Functions

- virtual [~IIfConditionalOutputLayer](#) () noexcept=default

Protected Attributes

- apiv::VConditionalOutputLayer * [mImpl](#)

Additional Inherited Members

9.73.1 Detailed Description

This layer represents an output of an [IIfConditional](#).

An [IIfConditionalOutputLayer](#) has exactly one output.

9.73.2 Constructor & Destructor Documentation

9.73.2.1 ~IIfConditionalOutputLayer()

```
virtual nvinfer1::IIfConditionalOutputLayer::~~IIfConditionalOutputLayer ( ) [protected], [virtual],
[default], [noexcept]
```

9.73.3 Member Data Documentation

9.73.3.1 mImpl

```
apiv::VConditionalOutputLayer* nvinfer1::IIfConditionalOutputLayer::mImpl [protected]
```

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

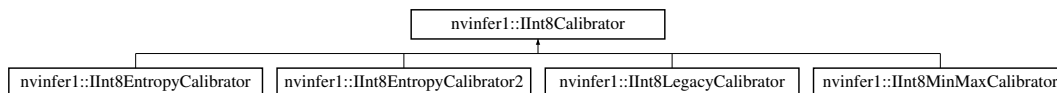
- [NvInfer.h](#)

9.74 nvinfer1::IInt8Calibrator Class Reference

Application-implemented interface for calibration.

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

Inheritance diagram for nvinfer1::IInt8Calibrator:



Public Member Functions

- virtual int32_t [getBatchSize](#) () const noexcept=0
Get the batch size used for calibration batches.
- virtual bool [getBatch](#) (void *bindings[], char const *names[], int32_t nbBindings) noexcept=0
Get a batch of input for calibration.
- virtual void const * [readCalibrationCache](#) (std::size_t &length) noexcept=0
Load a calibration cache.
- virtual void [writeCalibrationCache](#) (void const *ptr, std::size_t length) noexcept=0
Save a calibration cache.
- virtual [CalibrationAlgoType](#) [getAlgorithm](#) () noexcept=0
Get the algorithm used by this calibrator.
- virtual [~IInt8Calibrator](#) () noexcept=default

9.74.1 Detailed Description

Application-implemented interface for calibration.

Calibration is a step performed by the builder when deciding suitable scale factors for 8-bit inference.

It must also provide a method for retrieving representative images which the calibration process can use to examine the distribution of activations. It may optionally implement a method for caching the calibration result for reuse on subsequent runs.

9.74.2 Constructor & Destructor Documentation

9.74.2.1 ~IInt8Calibrator()

```
virtual nvinfer1::IInt8Calibrator::~~IInt8Calibrator ( ) [virtual], [default], [noexcept]
```

9.74.3 Member Function Documentation

9.74.3.1 getAlgorithm()

```
virtual CalibrationAlgoType nvinfer1::IInt8Calibrator::getAlgorithm ( ) [pure virtual], [noexcept]
```

Get the algorithm used by this calibrator.

Returns

The algorithm used by the calibrator.

Implemented in [nvinfer1::IInt8EntropyCalibrator](#), [nvinfer1::IInt8EntropyCalibrator2](#), [nvinfer1::IInt8MinMaxCalibrator](#), and [nvinfer1::IInt8LegacyCalibrator](#).

9.74.3.2 getBatch()

```
virtual bool nvinfer1::IInt8Calibrator::getBatch (
    void * bindings[],
    char const * names[],
    int32_t nbBindings ) [pure virtual], [noexcept]
```

Get a batch of input for calibration.

The batch size of the input must match the batch size returned by [getBatchSize\(\)](#).

Parameters

<i>bindings</i>	An array of pointers to device memory that must be updated to point to device memory containing each network input data.
<i>names</i>	The names of the network input for each pointer in the binding array.
<i>nbBindings</i>	The number of pointers in the bindings array.

Returns

False if there are no more batches for calibration.

See also

[getBatchSize\(\)](#)

9.74.3.3 getBatchSize()

```
virtual int32_t nvinfer1::IInt8Calibrator::getBatchSize ( ) const [pure virtual], [noexcept]
```

Get the batch size used for calibration batches.

Returns

The batch size.

9.74.3.4 readCalibrationCache()

```
virtual void const * nvinfer1::IInt8Calibrator::readCalibrationCache (
    std::size_t & length ) [pure virtual], [noexcept]
```

Load a calibration cache.

Calibration is potentially expensive, so it can be useful to generate the calibration data once, then use it on subsequent builds of the network. The cache includes the regression cutoff and quantile values used to generate it, and will not be used if these do not batch the settings of the current calibrator. However, the network should also be recalibrated if its structure changes, or the input data set changes, and it is the responsibility of the application to ensure this.

Parameters

<i>length</i>	The length of the cached data, that should be set by the called function. If there is no data, this should be zero.
---------------	---

Returns

A pointer to the cache, or nullptr if there is no data.

9.74.3.5 writeCalibrationCache()

```
virtual void nvinfer1::IInt8Calibrator::writeCalibrationCache (
    void const * ptr,
    std::size_t length ) [pure virtual], [noexcept]
```

Save a calibration cache.

Parameters

<i>ptr</i>	A pointer to the data to cache.
<i>length</i>	The length in bytes of the data to cache.

See also

[readCalibrationCache\(\)](#)

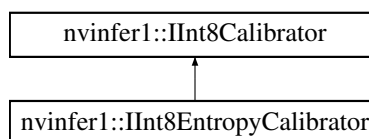
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.75 nvinfer1::IInt8EntropyCalibrator Class Reference

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

Inheritance diagram for nvinfer1::IInt8EntropyCalibrator:



Public Member Functions

- [CalibrationAlgoType getAlgorithm \(\)](#) noexcept override
- [virtual ~IInt8EntropyCalibrator \(\)](#) noexcept=default

9.75.1 Detailed Description

Entropy calibrator. This is the Legacy Entropy calibrator. It is less complicated than the legacy calibrator and produces better results.

9.75.2 Constructor & Destructor Documentation

9.75.2.1 ~IInt8EntropyCalibrator()

```
virtual nvinfer1::IInt8EntropyCalibrator::~~IInt8EntropyCalibrator ( ) [virtual], [default], [noexcept]
```

9.75.3 Member Function Documentation

9.75.3.1 getAlgorithm()

```
CalibrationAlgoType nvinfer1::IInt8EntropyCalibrator::getAlgorithm ( ) [inline], [override],  
[virtual], [noexcept]
```

Signal that this is the entropy calibrator.

Implements [nvinfer1::IInt8Calibrator](#).

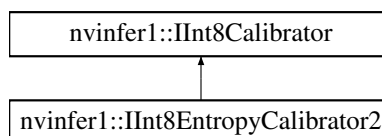
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.76 nvinfer1::IInt8EntropyCalibrator2 Class Reference

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

Inheritance diagram for nvinfer1::IInt8EntropyCalibrator2:



Public Member Functions

- [CalibrationAlgoType](#) `getAlgorithm()` noexcept override
- virtual `~IInt8EntropyCalibrator2()` noexcept=default

9.76.1 Detailed Description

Entropy calibrator 2. This is the preferred calibrator. This is the required calibrator for DLA, as it supports per activation tensor scaling.

9.76.2 Constructor & Destructor Documentation

9.76.2.1 `~IInt8EntropyCalibrator2()`

```
virtual nvinfer1::IInt8EntropyCalibrator2::~~IInt8EntropyCalibrator2 ( ) [virtual], [default],
[noexcept]
```

9.76.3 Member Function Documentation

9.76.3.1 `getAlgorithm()`

```
CalibrationAlgoType nvinfer1::IInt8EntropyCalibrator2::getAlgorithm ( ) [inline], [override],
[virtual], [noexcept]
```

Signal that this is the entropy calibrator 2.

Implements [nvinfer1::IInt8Calibrator](#).

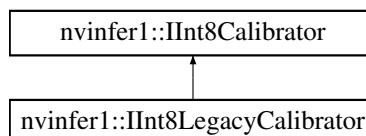
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.77 nvinfer1::IInt8LegacyCalibrator Class Reference

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

Inheritance diagram for `nvinfer1::IInt8LegacyCalibrator`:



Public Member Functions

- [CalibrationAlgoType](#) `getAlgorithm()` noexcept override
- virtual double `getQuantile()` const noexcept=0
The quantile (between 0 and 1) that will be used to select the region maximum when the quantile method is in use.
- virtual double `getRegressionCutoff()` const noexcept=0
The fraction (between 0 and 1) of the maximum used to define the regression cutoff when using regression to determine the region maximum.
- virtual void const * `readHistogramCache` (std::size_t &length) noexcept=0
Load a histogram.
- virtual void `writeHistogramCache` (void const *ptr, std::size_t length) noexcept=0
Save a histogram cache.
- virtual `~IInt8LegacyCalibrator()` noexcept=default

9.77.1 Detailed Description

Legacy calibrator left for backward compatibility with TensorRT 2.0. This calibrator requires user parameterization, and is provided as a fallback option if the other calibrators yield poor results.

9.77.2 Constructor & Destructor Documentation

9.77.2.1 `~IInt8LegacyCalibrator()`

```
virtual nvinfer1::IInt8LegacyCalibrator::~~IInt8LegacyCalibrator ( ) [virtual], [default], [noexcept]
```

9.77.3 Member Function Documentation

9.77.3.1 `getAlgorithm()`

```
CalibrationAlgoType nvinfer1::IInt8LegacyCalibrator::getAlgorithm ( ) [inline], [override], [virtual], [noexcept]
```

Signal that this is the legacy calibrator.

Implements [nvinfer1::IInt8Calibrator](#).

9.77.3.2 getQuantile()

```
virtual double nvinfer1::IInt8LegacyCalibrator::getQuantile ( ) const [pure virtual], [noexcept]
```

The quantile (between 0 and 1) that will be used to select the region maximum when the quantile method is in use.

See the user guide for more details on how the quantile is used.

9.77.3.3 getRegressionCutoff()

```
virtual double nvinfer1::IInt8LegacyCalibrator::getRegressionCutoff ( ) const [pure virtual],  
[noexcept]
```

The fraction (between 0 and 1) of the maximum used to define the regression cutoff when using regression to determine the region maximum.

See the user guide for more details on how the regression cutoff is used

9.77.3.4 readHistogramCache()

```
virtual void const * nvinfer1::IInt8LegacyCalibrator::readHistogramCache (  
    std::size_t & length ) [pure virtual], [noexcept]
```

Load a histogram.

Histogram generation is potentially expensive, so it can be useful to generate the histograms once, then use them when exploring the space of calibrations. The histograms should be regenerated if the network structure changes, or the input data set changes, and it is the responsibility of the application to ensure this.

Parameters

<i>length</i>	The length of the cached data, that should be set by the called function. If there is no data, this should be zero.
---------------	---

Returns

A pointer to the cache, or nullptr if there is no data.

9.77.3.5 writeHistogramCache()

```
virtual void nvinfer1::IInt8LegacyCalibrator::writeHistogramCache (  
    void const * ptr,  
    std::size_t length ) [pure virtual], [noexcept]
```

Save a histogram cache.

Parameters

<i>ptr</i>	A pointer to the data to cache.
<i>length</i>	The length in bytes of the data to cache.

See also

[readHistogramCache\(\)](#)

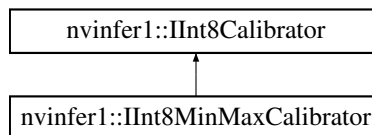
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.78 nvinfer1::IInt8MinMaxCalibrator Class Reference

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

Inheritance diagram for nvinfer1::IInt8MinMaxCalibrator:



Public Member Functions

- [CalibrationAlgoType getAlgorithm\(\)](#) noexcept override
- virtual [~IInt8MinMaxCalibrator\(\)](#) noexcept=default

9.78.1 Detailed Description

MinMax Calibrator. It supports per activation tensor scaling.

9.78.2 Constructor & Destructor Documentation

9.78.2.1 ~IInt8MinMaxCalibrator()

```
virtual nvinfer1::IInt8MinMaxCalibrator::~~IInt8MinMaxCalibrator ( ) [virtual], [default], [noexcept]
```

9.78.3 Member Function Documentation

9.78.3.1 getAlgorithm()

```
CalibrationAlgoType nvinfer1::IInt8MinMaxCalibrator::getAlgorithm ( ) [inline], [override], [virtual],
[noexcept]
```

Signal that this is the MinMax Calibrator.

Implements [nvinfer1::IInt8Calibrator](#).

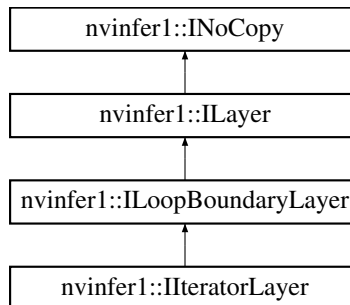
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.79 nvinfer1::IIteratorLayer Class Reference

```
#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
- bool [getReverse](#) () const noexcept
True if and only if reversing input.

Protected Member Functions

- virtual [~IteratorLayer](#) () noexcept=default

Protected Attributes

- apiv::VIteratorLayer * [mImpl](#)

9.79.1 Constructor & Destructor Documentation

9.79.1.1 ~IteratorLayer()

```
virtual nvinfer1::IteratorLayer::~~IteratorLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.79.2 Member Function Documentation

9.79.2.1 getAxis()

```
int32_t nvinfer1::IteratorLayer::getAxis ( ) const [inline], [noexcept]
```

Get axis being iterated over.

9.79.2.2 getReverse()

```
bool nvinfer1::IteratorLayer::getReverse ( ) const [inline], [noexcept]
```

True if and only if reversing input.

9.79.2.3 setAxis()

```
void nvinfer1::IteratorLayer::setAxis (
    int32_t axis ) [inline], [noexcept]
```

Set axis to iterate over.

9.79.2.4 setReverse()

```
void nvinfer1::IIteratorLayer::setReverse (
    bool reverse ) [inline], [noexcept]
```

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.

9.79.3 Member Data Documentation

9.79.3.1 mImpl

```
apiv::VIteratorLayer* nvinfer1::IIteratorLayer::mImpl [protected]
```

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

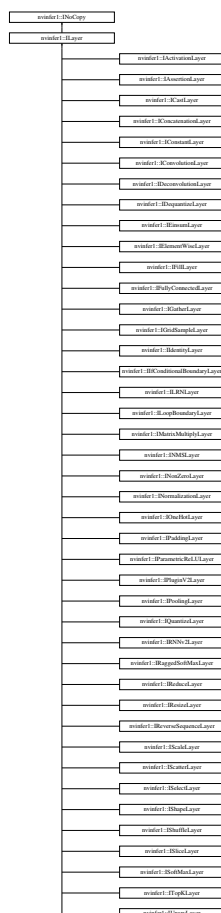
- [NvInfer.h](#)

9.80 nvinfer1::ILayer Class Reference

Base class for all layer classes in a network definition.

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

Inheritance diagram for nvinfer1::ILayer:



- `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.
- void `setPrecision` (`DataType` dataType) noexcept
Set the computational precision of this layer.

- `DataType getPrecision ()` const noexcept
get the computational precision of this layer
- `bool precisionIsSet ()` const noexcept
whether the computational precision has been set for this layer
- `void resetPrecision ()` noexcept
reset the computational precision for this layer
- `void setOutputType (int32_t index, DataType dataType)` noexcept
Set the output type of this layer.
- `DataType getOutputType (int32_t index)` const noexcept
get the output type of this layer
- `bool outputTypeIsSet (int32_t index)` const noexcept
whether the output type has been set for this layer
- `void resetOutputType (int32_t index)` noexcept
reset the output type for this layer

Protected Member Functions

- `virtual ~ILayer ()` noexcept=default

Protected Attributes

- `apiv::VLayer * mLayer`

9.80.1 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.

9.80.2 Constructor & Destructor Documentation

9.80.2.1 ~ILayer()

```
virtual nvinfer1::ILayer::~~ILayer ( ) [protected], [virtual], [default], [noexcept]
```

9.80.3 Member Function Documentation

9.80.3.1 getInput()

```
ITensor * nvinfer1::ILayer::getInput (
    int32_t index ) const [inline], [noexcept]
```

Get the layer input corresponding to the given index.

Parameters

<i>index</i>	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](#) and [IRNNv2Layer](#)).

9.80.3.2 getName()

```
char const * nvinfer1::ILayer::getName ( ) const [inline], [noexcept]
```

Return the name of a layer.

See also

[setName\(\)](#)

9.80.3.3 getNbInputs()

```
int32_t nvinfer1::ILayer::getNbInputs ( ) const [inline], [noexcept]
```

Get the number of inputs of a layer.

9.80.3.4 getNbOutputs()

```
int32_t nvinfer1::ILayer::getNbOutputs ( ) const [inline], [noexcept]
```

Get the number of outputs of a layer.

9.80.3.5 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 ([IRNNv2Layer](#)).

9.80.3.6 getOutputType()

```
DataType nvinfer1::ILayer::getOutputType (
    int32_t index ) const [inline], [noexcept]
```

get the output type of this layer

Parameters

<i>index</i>	the index of the output
--------------	-------------------------

Returns

the output precision. If no precision has been set, [DataType::kFLOAT](#) will be returned, unless the output type is inherently [DataType::kINT32](#).

See also

[getOutputType\(\)](#) [outputTypeIsSet\(\)](#) [resetOutputType\(\)](#)

9.80.3.7 getPrecision()

```
DataType nvinfer1::ILayer::getPrecision ( ) const [inline], [noexcept]
```

get the computational precision of this layer

Returns

the computational precision

See also

[setPrecision\(\)](#) [precisionIsSet\(\)](#) [resetPrecision\(\)](#)

9.80.3.8 getType()

```
LayerType nvinfer1::ILayer::getType ( ) const [inline], [noexcept]
```

Return the type of a layer.

See also

[LayerType](#)

9.80.3.9 outputTypeIsSet()

```
bool nvinfer1::ILayer::outputTypeIsSet (
    int32_t index ) const [inline], [noexcept]
```

whether the output type has been set for this layer

Parameters

<i>index</i>	the index of the output
--------------	-------------------------

Returns

whether the output type has been explicitly set

See also

[setOutputType\(\)](#) [getOutputType\(\)](#) [resetOutputType\(\)](#)

9.80.3.10 precisionIsSet()

```
bool nvinfer1::ILayer::precisionIsSet ( ) const [inline], [noexcept]
```

whether the computational precision has been set for this layer

Returns

whether the computational precision has been explicitly set

See also

[setPrecision\(\)](#) [getPrecision\(\)](#) [resetPrecision\(\)](#)

9.80.3.11 resetOutputType()

```
void nvinfer1::ILayer::resetOutputType (
    int32_t index ) [inline], [noexcept]
```

reset the output type for this layer

Parameters

<i>index</i>	the index of the output
--------------	-------------------------

See also

[setOutputType\(\)](#) [getOutputType\(\)](#) [outputTypeIsSet\(\)](#)

9.80.3.12 resetPrecision()

```
void nvinfer1::ILayer::resetPrecision ( ) [inline], [noexcept]
```

reset the computational precision for this layer

See also

[setPrecision\(\)](#) [getPrecision\(\)](#) [precisionIsSet\(\)](#)

9.80.3.13 setInput()

```
void nvinfer1::ILayer::setInput (
    int32_t index,
    ITensor & tensor ) [inline], [noexcept]
```

Replace an input of this layer with a specific tensor.

Parameters

<i>index</i>	the index of the input to modify.
<i>tensor</i>	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.

9.80.3.14 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\(\)](#)

9.80.3.15 setOutputType()

```
void nvinfer1::ILayer::setOutputType (
    int32_t index,
    DataType dataType ) [inline], [noexcept]
```

Set the output type of this layer.

Setting the output type constrains TensorRT to choose implementations which generate output data with the given type. If it is not set, TensorRT will select output type based on layer computational precision. TensorRT could still choose non-conforming output type based on fastest implementation. To force choosing the requested output type, set exactly one of the following flags, which differ in what happens if no such implementation exists:

- [BuilderFlag::kOBEY_PRECISION_CONSTRAINTS](#) - build fails with an error message.
- [BuilderFlag::kPREFER_PRECISION_CONSTRAINTS](#) - TensorRT falls back to an implementation with a non-conforming output type.

In case layer precision is not specified, or falling back, the output type depends on the chosen implementation, based on performance considerations and the flags specified to the builder.

This method cannot be used to set the data type of the second output tensor of the TopK layer. The data type of the second output tensor of the topK layer is always Int32. Also the output type of all layers that are shape operations must be [DataType::kINT32](#), and all attempts to set the output type to some other data type will be ignored except for issuing an error message.

Note that the layer output type is generally not identical to the data type of the output tensor, as TensorRT may insert implicit reformatting operations to convert the former to the latter. Calling `layer->setOutputType(i, type)` has no effect on the data type of the *i*-th output tensor of layer, and users need to call `layer->getOutput(i)->setType(type)` to change the tensor data type. This is particularly relevant if the tensor is marked as a network output, since only `setType()` [but not `setOutputType()`] will affect the data representation in the corresponding output binding.

Parameters

<i>index</i>	the index of the output to set
<i>dataType</i>	the type of the output

See also

[getOutputType\(\)](#) [outputTypeIsSet\(\)](#) [resetOutputType\(\)](#)

9.80.3.16 setPrecision()

```
void nvinfer1::ILayer::setPrecision (
    DataType dataType ) [inline], [noexcept]
```

Set the computational precision of this layer.

Setting the precision allows TensorRT to choose an implementation which run at this computational precision. TensorRT could still choose a non-conforming fastest implementation that ignores the requested precision. To force choosing an implementation with the requested precision, set exactly one of the following flags, which differ in what happens if no such implementation exists:

- [BuilderFlag::kOBEY_PRECISION_CONSTRAINTS](#) - build fails with an error message.
- [BuilderFlag::kPREFER_PRECISION_CONSTRAINTS](#) - TensorRT falls back to an implementation without the requested precision.

If precision is not set, or falling back, TensorRT will select the layer computational precision and layer input type based on global performance considerations and the flags specified to the builder.

For a [IIdentityLayer](#): If it casts to/from float/half/int8/uint8, the precision must be one of those types, otherwise it must be either the input or output type.

Parameters

<i>dataType</i>	the computational precision.
-----------------	------------------------------

See also

[getPrecision\(\)](#) [precisionIsSet\(\)](#) [resetPrecision\(\)](#)

9.80.4 Member Data Documentation

9.80.4.1 mLayer

```
apiv::VLayer* nvinfer1::ILayer::mLayer [protected]
```

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

- [NvInfer.h](#)

9.81 nvinfer1::ILogger Class Reference

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

```
#include <NvInferRuntimeCommon.h>
```

Public Types

- enum class [Severity](#) : int32_t {
[kINTERNAL_ERROR](#) = 0 , [kERROR](#) = 1 , [kWARNING](#) = 2 , [kINFO](#) = 3 ,
[kVERBOSE](#) = 4 }

Public Member Functions

- virtual void [log](#) ([Severity](#) severity, [AsciiChar](#) const *msg) noexcept=0
- [ILogger](#) ()=default
- virtual [~ILogger](#) ()=default

9.81.1 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 should 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.

9.81.2 Member Enumeration Documentation

9.81.2.1 Severity

```
enum class nvinfer1::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.

9.81.3 Constructor & Destructor Documentation

9.81.3.1 ILogger()

```
nvinfer1::ILogger::ILogger ( ) [default]
```

9.81.3.2 ~ILogger()

```
virtual nvinfer1::ILogger::~ILogger ( ) [virtual], [default]
```

9.81.4 Member Function Documentation

9.81.4.1 log()

```
virtual void nvinfer1::ILogger::log (
    Severity severity,
    AsciiChar const * msg ) [pure virtual], [noexcept]
```

A callback implemented by the application to handle logging messages;

Parameters

<i>severity</i>	The severity of the message.
<i>msg</i>	A null-terminated log message.

Usage considerations

- 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.

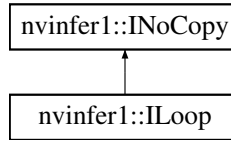
The documentation for this class was generated from the following file:

- [NvInferRuntimeCommon.h](#)

9.82 nvinfer1::ILoop Class Reference

```
#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=default

Protected Attributes

- apiv::VLoop * [mImpl](#)

9.82.1 Detailed Description

Helper for creating a recurrent subgraph.

An [ILoop](#) cannot be added to an [INetworkDefinition](#) where [hasImplicitBatchDimensions\(\)](#) returns true.

9.82.2 Constructor & Destructor Documentation

9.82.2.1 ~ILoop()

```
virtual nvinfer1::ILoop::~~ILoop ( ) [protected], [virtual], [default], [noexcept]
```

9.82.3 Member Function Documentation

9.82.3.1 addIterator()

```
ILayer * 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.

9.82.3.2 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](#).

9.82.3.3 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::setInput\(1,...\)](#) before an Engine can be built.

9.82.3.4 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.

9.82.3.5 getName()

```
char const * nvinfer1::ILoop::getName ( ) const [inline], [noexcept]
```

Return the name of the loop.

See also

[setName\(\)](#)

9.82.3.6 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\(\)](#)

9.82.4 Member Data Documentation

9.82.4.1 mImpl

```
apiv::VLoop* nvinfer1::ILoop::mImpl [protected]
```

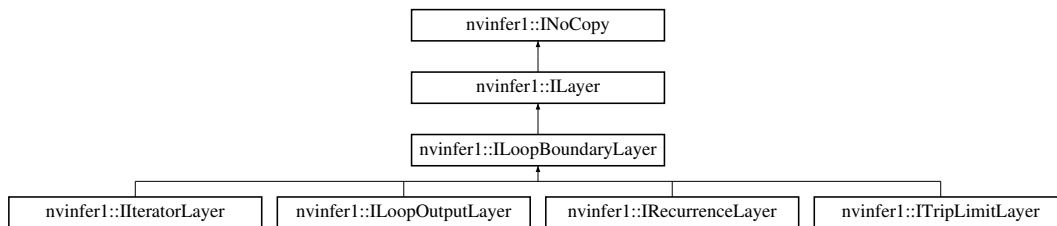
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.83 nvinfer1::ILoopBoundaryLayer Class Reference

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

Inheritance diagram for nvinfer1::ILoopBoundaryLayer:



Public Member Functions

- [ILoop](#) * [getLoop](#) () const noexcept
Return pointer to [ILoop](#) associated with this boundary layer.

Protected Member Functions

- virtual [~ILoopBoundaryLayer](#) () noexcept=default

Protected Attributes

- apiv::VLoopBoundaryLayer * [mBoundary](#)

9.83.1 Constructor & Destructor Documentation

9.83.1.1 ~ILoopBoundaryLayer()

```
virtual nvinfer1::ILoopBoundaryLayer::~~ILoopBoundaryLayer ( ) [protected], [virtual], [default],
[noexcept]
```

9.83.2 Member Function Documentation

9.83.2.1 getLoop()

```
ILoop * nvinfer1::ILoopBoundaryLayer::getLoop ( ) const [inline], [noexcept]
```

Return pointer to [ILoop](#) associated with this boundary layer.

9.83.3 Member Data Documentation

9.83.3.1 mBoundary

```
apiv::VLoopBoundaryLayer* nvinfer1::ILoopBoundaryLayer::mBoundary [protected]
```

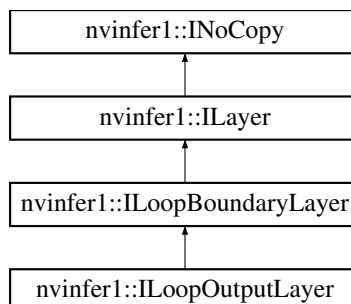
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.84 nvinfer1::ILoopOutputLayer Class Reference

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

Inheritance diagram for nvinfer1::ILoopOutputLayer:



Public Member Functions

- [LoopOutput](#) [getLoopOutput](#) () const noexcept
- void [setAxis](#) (int32_t axis) noexcept
Set where to insert the contatenation 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=default

Protected Attributes

- apiv::VLoopOutputLayer * [mImpl](#)

9.84.1 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 from a [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.

9.84.2 Constructor & Destructor Documentation

9.84.2.1 [~ILoopOutputLayer\(\)](#)

```
virtual nvinferl::ILoopOutputLayer::~~ILoopOutputLayer ( ) [protected], [virtual], [default],
[noexcept]
```

9.84.3 Member Function Documentation

9.84.3.1 `getAxis()`

```
int32_t nvinfer1::ILoopOutputLayer::getAxis ( ) const [inline], [noexcept]
```

Get axis being concatenated over.

9.84.3.2 `getLoopOutput()`

```
LoopOutput nvinfer1::ILoopOutputLayer::getLoopOutput ( ) const [inline], [noexcept]
```

9.84.3.3 `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.

9.84.3.4 `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

<i>index</i>	the index of the input to modify.
<i>tensor</i>	the new input tensor Sets the input tensor for the given index. The index must be 0 for a <code>kLAST_VALUE</code> loop output layer. Loop output layer is converted to a <code>kCONCATENATE</code> or <code>kREVERSE</code> loop output layer by calling <code>setInput</code> with an index 1. A <code>kCONCATENATE</code> or <code>kREVERSE</code> loop output layer cannot be converted back to a <code>kLAST_VALUE</code> loop output layer.

For a kCONCATENATE or kREVERSE loop output layer, the values 0 and 1 are valid. The indices in the k←CONCATENATE 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 Int32 shape tensor.

If this function is called with the value 1, then the function [getNbInputs\(\)](#) changes from returning 1 to 2.

9.84.4 Member Data Documentation

9.84.4.1 mImpl

```
apiv::VLoopOutputLayer* nvinfer1::ILoopOutputLayer::mImpl [protected]
```

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

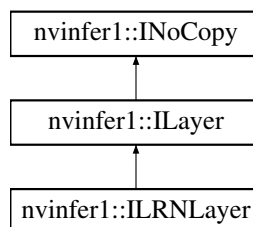
- [NvInfer.h](#)

9.85 nvinfer1::ILRNLayer Class Reference

A LRN layer in a network definition.

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

Inheritance diagram for nvinfer1::ILRNLayer:



Public Member Functions

- void [setWindowSize](#) (int32_t windowSize) noexcept
Set the LRN window size.
- int32_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=default

Protected Attributes

- apiv::VLRNLayer * [mImpl](#)

9.85.1 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.

9.85.2 Constructor & Destructor Documentation

9.85.2.1 ~ILRNLayer()

```
virtual nvinfer1::ILRNLayer::~ILRNLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.85.3 Member Function Documentation

9.85.3.1 `getAlpha()`

```
float nvinfer1::ILRNLayer::getAlpha ( ) const [inline], [noexcept]
```

Get the LRN alpha value.

See also

[setAlpha\(\)](#)

9.85.3.2 `getBeta()`

```
float nvinfer1::ILRNLayer::getBeta ( ) const [inline], [noexcept]
```

Get the LRN beta value.

See also

[setBeta\(\)](#)

9.85.3.3 `getK()`

```
float nvinfer1::ILRNLayer::getK ( ) const [inline], [noexcept]
```

Get the LRN K value.

See also

[setK\(\)](#)

9.85.3.4 getWindowSize()

```
int32_t nvinfer1::ILRNLayer::getWindowSize ( ) const [inline], [noexcept]
```

Get the LRN window size.

See also

[getWindowStride\(\)](#)

9.85.3.5 setAlpha()

```
void nvinfer1::ILRNLayer::setAlpha (
    float alpha ) [inline], [noexcept]
```

Set the LRN alpha value.

The valid range is [-1e20, 1e20].

See also

[getAlpha\(\)](#)

9.85.3.6 setBeta()

```
void nvinfer1::ILRNLayer::setBeta (
    float beta ) [inline], [noexcept]
```

Set the LRN beta value.

The valid range is [0.01, 1e5f].

See also

[getBeta\(\)](#)

9.85.3.7 setK()

```
void nvinfer1::ILRNLayer::setK (
    float k ) [inline], [noexcept]
```

Set the LRN K value.

The valid range is [1e-5, 1e10].

See also

[getK\(\)](#)

9.85.3.8 setWindowSize()

```
void nvinfer1::ILRNLayer::setWindowSize (
    int32_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\(\)](#)

9.85.4 Member Data Documentation

9.85.4.1 mImpl

```
apiv::VLRNLayer* nvinfer1::ILRNLayer::mImpl [protected]
```

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

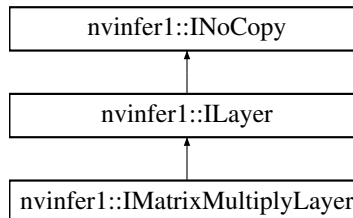
- [NvInfer.h](#)

9.86 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=default

Protected Attributes

- apiv::VMatrixMultiplyLayer * [mImpl](#)

9.86.1 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.

9.86.2 Constructor & Destructor Documentation

9.86.2.1 ~IMatrixMultiplyLayer()

```
virtual nvinfer1::IMatrixMultiplyLayer::~IMatrixMultiplyLayer ( ) [protected], [virtual], [default],
[noexcept]
```

9.86.3 Member Function Documentation

9.86.3.1 getOperation()

```
MatrixOperation nvinfer1::IMatrixMultiplyLayer::getOperation (
    int32_t index ) const [inline], [noexcept]
```

Get the operation for an input tensor.

Parameters

<i>index</i>	Input tensor number (0 or 1).
--------------	-------------------------------

See also

[setOperation\(\)](#)

9.86.3.2 setOperation()

```
void nvinfer1::IMatrixMultiplyLayer::setOperation (
    int32_t index,
    MatrixOperation op ) [inline], [noexcept]
```

Set the operation for an input tensor.

Parameters

<i>index</i>	Input tensor number (0 or 1).
<i>op</i>	New operation.

See also

[getOperation\(\)](#)

9.86.4 Member Data Documentation

9.86.4.1 mImpl

```
apiv::VMatrixMultiplyLayer* nvinfer1::IMatrixMultiplyLayer::mImpl [protected]
```

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

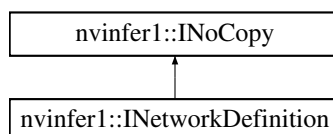
- [NvInfer.h](#)

9.87 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=default
- [ITensor](#) * [addInput](#) (char const *name, [DataType](#) type, [Dims](#) dimensions) noexcept
Add an input tensor to the network.
- void [markOutput](#) ([ITensor](#) &tensor) noexcept
Mark a tensor as a network output.
- [TRT_DEPRECATED IConvolutionLayer](#) * [addConvolution](#) ([ITensor](#) &input, int32_t nbOutputMaps, [DimsHW](#) kernelSize, [Weights](#) kernelWeights, [Weights](#) biasWeights) noexcept
Add a convolution layer to the network.
- [TRT_DEPRECATED IFullyConnectedLayer](#) * [addFullyConnected](#) ([ITensor](#) &input, int32_t nbOutputs, [Weights](#) kernelWeights, [Weights](#) biasWeights) noexcept
Add a fully connected layer to the network.
- [IActivationLayer](#) * [addActivation](#) ([ITensor](#) &input, [ActivationType](#) type) noexcept
Add an activation layer to the network.
- [TRT_DEPRECATED IPoolingLayer](#) * [addPooling](#) ([ITensor](#) &input, [PoolingType](#) type, [DimsHW](#) windowSize) noexcept
Add a pooling layer to the network.
- [ILRNLayer](#) * [addLRN](#) ([ITensor](#) &input, int32_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.
- [TRT_DEPRECATED IDEconvolutionLayer](#) * [addDeconvolution](#) ([ITensor](#) &input, int32_t nbOutputMaps, [DimsHW](#) kernelSize, [Weights](#) kernelWeights, [Weights](#) biasWeights) noexcept
Add a deconvolution 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.
- [TRT_DEPRECATED IPaddingLayer](#) * [addPadding](#) ([ITensor](#) &input, [DimsHW](#) prePadding, [DimsHW](#) postPadding) noexcept
Add a padding 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.
- **TRT_DEPRECATED** void **destroy** () noexcept
*Destroy this **INetworkDefinition** object.*
- **IReduceLayer** * **addReduce** (**ITensor** &input, **ReduceOperation** operation, uint32_t reduceAxes, bool keepDimensions) noexcept
Add a reduce layer to the network.
- **ITopKLayer** * **addTopK** (**ITensor** &input, **TopKOperation** op, int32_t k, uint32_t reduceAxes) 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.
- **INonZeroLayer** * **addNonZero** (**ITensor** &input) noexcept
Add a nonzero layer to the network.
- **IConstantLayer** * **addConstant** (**Dims** dimensions, **Weights** weights) noexcept
Add a constant layer to the network.
- **TRT_DEPRECATED** **IRNNv2Layer** * **addRNNv2** (**ITensor** &input, int32_t layerCount, int32_t hiddenSize, int32_t maxSeqLen, **RNNOperation** op) noexcept
Add an layerCount deep RNN layer to the network with hiddenSize internal states that can take a batch with fixed or variable sequence lengths.
- **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.
- **IPluginV2Layer** * **addPluginV2** (**ITensor** *const *inputs, int32_t nbInputs, **IPluginV2** &plugin) noexcept
*Add a plugin layer to the network using the **IPluginV2** interface.*
- **ISliceLayer** * **addSlice** (**ITensor** &input, **Dims** start, **Dims** size, **Dims** 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.

- bool [hasImplicitBatchDimension](#) () const noexcept
Query whether the network was created with an implicit batch dimension.
- 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, int32_t nbOutputMaps, Dims kernelSize, Weights kernelWeights, Weights biasWeights) noexcept
Add a multi-dimension convolution layer to the network.
- IPoolingLayer * [addPoolingNd](#) (ITensor &input, PoolingType type, Dims windowSize) noexcept
Add a multi-dimension pooling layer to the network.
- IDEconvolutionLayer * [addDeconvolutionNd](#) (ITensor &input, int32_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.
- TRT_DEPRECATED bool [hasExplicitPrecision](#) () const noexcept
True if network is an explicit precision network.
- ILoop * [addLoop](#) () noexcept
Add a loop 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 dimensions, FillOperation op) noexcept
Add a fill layer to the network.
- TRT_DEPRECATED IPaddingLayer * [addPaddingNd](#) (ITensor &input, Dims prePadding, Dims 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) 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) noexcept
Add a quantization layer to the network.
- IIfConditional * [addIfConditional](#) () noexcept

Add an If-conditional 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.

- [INMSLayer](#) * [addNMS](#) ([ITensor](#) &boxes, [ITensor](#) &scores, [ITensor](#) &maxOutputBoxesPerClass) noexcept

Add a non-maximum suppression layer to the network.

- [IRReverseSequenceLayer](#) * [addReverseSequence](#) ([ITensor](#) &input, [ITensor](#) &sequenceLens) noexcept

Add a ReverseSequence layer to the network.

- [INormalizationLayer](#) * [addNormalization](#) ([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

9.87.1 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 either have an implicit batch dimensions, specified at runtime, or all dimensions explicit, full dims mode, in the network definition. The former mode, i.e. the implicit batch size mode, has been deprecated. The function [hasImplicitBatchDimension\(\)](#) can be used to query the mode of the network.

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.

9.87.2 Constructor & Destructor Documentation

9.87.2.1 ~INetworkDefinition()

```
virtual nvinfer1::INetworkDefinition::~~INetworkDefinition ( ) [virtual], [default], [noexcept]
```

9.87.3 Member Function Documentation

9.87.3.1 addActivation()

```
IActivationLayer * nvinfer1::INetworkDefinition::addActivation (
    ITensor & input,
    ActivationType type ) [inline], [noexcept]
```

Add an activation layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
<i>type</i>	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 tensors are not valid input tensors.

Returns

The new activation layer, or nullptr if it could not be created.

9.87.3.2 addAssertion()

```
IAssertionLayer * nvinfer1::INetworkDefinition::addAssertion (
    ITensor & condition,
    char const * message ) [inline], [noexcept]
```

Add an assertion layer to the network.

Parameters

<i>condition</i>	The input tensor to the layer.
<i>message</i>	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.

9.87.3.3 addCast()

```
ICastLayer * nvinfer1::INetworkDefinition::addCast (
    ITensor & input,
    DataType toType ) [inline], [noexcept]
```

Add a cast layer.

Parameters

<i>input</i>	The input tensor to the layer.
<i>toType</i>	The DataType of the output tensor

See also

[ICastLayer](#)

Returns

The new cast layer, or nullptr if it could not be created.

9.87.3.4 addConcatenation()

```
IConcatenationLayer * nvinfer1::INetworkDefinition::addConcatenation (
    ITensor *const * inputs,
    int32_t nbInputs ) [inline], [noexcept]
```

Add a concatenation layer to the network.

Parameters

<i>inputs</i>	The input tensors to the layer.
<i>nbInputs</i>	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.

9.87.3.5 addConstant()

```
IConstantLayer * nvinfer1::INetworkDefinition::addConstant (
    Dims dimensions,
    Weights weights ) [inline], [noexcept]
```

Add a constant layer to the network.

Parameters

<i>dimensions</i>	The dimensions of the constant.
<i>weights</i>	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 tensors in the network have an implicit batch dimension, the constant is broadcast over that dimension.

If a wildcard dimension is used, the volume of the runtime dimensions must equal the number of weights specified.

Warning

[DataType::kUINT8](#) not supported.

9.87.3.6 addConvolution()

```
TRT_DEPRECATED IConvolutionLayer * nvinfer1::INetworkDefinition::addConvolution (
    ITensor & input,
    int32_t nbOutputMaps,
    DimsHW kernelSize,
    Weights kernelWeights,
    Weights biasWeights ) [inline], [noexcept]
```

Add a convolution layer to the network.

Parameters

<i>input</i>	The input tensor to the convolution.
<i>nbOutputMaps</i>	The number of output feature maps for the convolution.
<i>kernelSize</i>	The HW-dimensions of the convolution kernel.
<i>kernelWeights</i>	The kernel weights for the convolution.
<i>biasWeights</i>	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.

Returns

The new convolution layer, or nullptr if it could not be created.

Deprecated Superseded by `addConvolutionNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.87.3.7 addConvolutionNd()

```
IConvolutionLayer * nvinfer1::INetworkDefinition::addConvolutionNd (
    ITensor & input,
    int32_t nbOutputMaps,
    Dims kernelSize,
    Weights kernelWeights,
    Weights biasWeights ) [inline], [noexcept]
```

Add a multi-dimension convolution layer to the network.

Parameters

<i>input</i>	The input tensor to the convolution.
<i>nbOutputMaps</i>	The number of output feature maps for the convolution.
<i>kernelSize</i>	The multi-dimensions of the convolution kernel.
<i>kernelWeights</i>	The kernel weights for the convolution.
<i>biasWeights</i>	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.

9.87.3.8 addDeconvolution()

```
TRT_DEPRECATED IDeconvolutionLayer * nvinfer1::INetworkDefinition::addDeconvolution (
    ITensor & input,
    int32_t nbOutputMaps,
    DimsHW kernelSize,
    Weights kernelWeights,
    Weights biasWeights ) [inline], [noexcept]
```

Add a deconvolution layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
<i>nbOutputMaps</i>	The number of output feature maps.
<i>kernelSize</i>	The HW-dimensions of the deconvolution kernel.
<i>kernelWeights</i>	The kernel weights for the deconvolution.
<i>biasWeights</i>	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.

Returns

The new deconvolution layer, or nullptr if it could not be created.

Deprecated Superseded by addDeconvolutionNd. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.87.3.9 addDeconvolutionNd()

```
IDeconvolutionLayer * nvinfer1::INetworkDefinition::addDeconvolutionNd (
    ITensor & input,
    int32_t nbOutputMaps,
    Dims kernelSize,
    Weights kernelWeights,
    Weights biasWeights ) [inline], [noexcept]
```

Add a multi-dimension deconvolution layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
<i>nbOutputMaps</i>	The number of output feature maps.
<i>kernelSize</i>	The multi-dimensions of the deconvolution kernel.
<i>kernelWeights</i>	The kernel weights for the deconvolution.
<i>biasWeights</i>	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.

9.87.3.10 addDequantize()

```
IDequantizeLayer * nvinfer1::INetworkDefinition::addDequantize (
    ITensor & input,
    ITensor & scale ) [inline], [noexcept]
```

Add a dequantization layer to the network.

Parameters

<i>input</i>	The input tensor to be quantized.
<i>scale</i>	A tensor with the scale value.

See also

[IDequantizeLayer](#)

input tensor data type must be [DataType::kFLOAT](#). *scale* tensor data type must be [DataType::kFLOAT](#). The subgraph which terminates with the *scale* tensor must be a build-time constant.

Returns

The new quantization layer, or nullptr if it could not be created.

9.87.3.11 addEinsum()

```
IEinsumLayer * nvinfer1::INetworkDefinition::addEinsum (
    ITensor *const * inputs,
    int32_t nbInputs,
    char const * equation ) [inline], [noexcept]
```

Add an Einsum layer to the network.

Parameters

<i>inputs</i>	The input tensors to the layer.
<i>nbInputs</i>	The number of input tensors.
<i>equation</i>	The equation of the layer

See also

[IEinsumLayer](#)

Returns

The new Einsum layer, or nullptr if it could not be created.

9.87.3.12 addElementWise()

```
IElementWiseLayer * nvinfer1::INetworkDefinition::addElementWise (
    ITensor & input1,
    ITensor & input2,
    ElementWiseOperation op ) [inline], [noexcept]
```

Add an elementwise layer to the network.

Parameters

<i>input1</i>	The first input tensor to the layer.
<i>input2</i>	The second input tensor to the layer.
<i>op</i>	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.

9.87.3.13 addFill()

```
IFillLayer * nvinfer1::INetworkDefinition::addFill (
    Dims dimensions,
    FillOperation op ) [inline], [noexcept]
```

Add a fill layer to the network.

Parameters

<i>dimensions</i>	The output tensor dimensions.
<i>op</i>	The fill operation that the layer applies.

Warning

For `FillOperation::kLinspace`, `dimensions.nbDims` must be 1.

This layer is non-deterministic across subsequent calls as the same inputs will produce different output tensors if `op` is either `FillOperation::kRandomUniform` or `FillOperation::kRandomNormal` due to random state being shared across calls. The output tensors generated are deterministic when starting from the same initial state.

The network must not have an implicit batch dimension.

See also

[IFillLayer](#)

Returns

The new fill layer, or nullptr if it could not be created.

9.87.3.14 addFullyConnected()

```
TRT_DEPRECATED IFullyConnectedLayer * nvinfer1::INetworkDefinition::addFullyConnected (
    ITensor & input,
    int32_t nbOutputs,
    Weights kernelWeights,
    Weights biasWeights ) [inline], [noexcept]
```

Add a fully connected layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
<i>nbOutputs</i>	The number of outputs of the layer.
<i>kernelWeights</i>	The kernel weights for the fully connected layer.
<i>biasWeights</i>	The bias weights for the fully connected layer. <code>Weights{}</code> represents no bias.

See also

[IFullyConnectedLayer](#)

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.

Returns

The new fully connected layer, or nullptr if it could not be created.

Deprecated Deprecated in TensorRT 8.4. Superseded by [addMatrixMultiply\(\)](#).

9.87.3.15 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

<i>data</i>	The tensor to gather values from.
<i>indices</i>	The tensor to get indices from to populate the output tensor.
<i>axis</i>	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.

9.87.3.16 addGatherV2()

```
IGatherLayer * nvinfer1::INetworkDefinition::addGatherV2 (
    ITensor & data,
    ITensor & indices,
    GatherMode mode ) [inline], [noexcept]
```

Add gather with specified mode, axis=0 and nbElementWiseDims=0.

Parameters

<i>data</i>	The tensor to gather values from.
<i>indices</i>	The tensor to get indices from to populate the output tensor.
<i>mode</i>	The gather mode.

See also

[IGatherLayer](#)

Returns

The new gather layer, or nullptr if it could not be created.

9.87.3.17 addGridSample()

```
IGridSampleLayer * nvinfer1::INetworkDefinition::addGridSample (
    ITensor & input,
    ITensor & grid ) [inline], [noexcept]
```

Add a GridSample layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
<i>grid</i>	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.

9.87.3.18 addIdentity()

```
IIdentityLayer * nvinfer1::INetworkDefinition::addIdentity (
    ITensor & input ) [inline], [noexcept]
```

Add an identity layer.

Parameters

<i>input</i>	The input tensor to the layer.
--------------	--------------------------------

See also

[IIdentityLayer](#)

Returns

The new identity layer, or nullptr if it could not be created.

9.87.3.19 addIfConditional()

```
IIfConditional * nvinfer1::INetworkDefinition::addIfConditional ( ) [inline], [noexcept]
```

Add an If-conditional layer to the network.

An [IIfConditional](#) provides a way to conditionally execute parts of the network.

See also

[IIfConditional](#)

Returns

The new conditional layer, or nullptr if network has an implicit batch dimension or this version of TensorRT does not support conditional execution.

9.87.3.20 addInput()

```
ITensor * nvinfer1::INetworkDefinition::addInput (
    char const * name,
    DataType type,
    Dims dimensions ) [inline], [noexcept]
```

Add an input tensor to the network.

The name of the input tensor is used to find the index into the buffer array for an engine built from the network. The volume must be less than 2^{31} elements.

For networks with an implicit batch dimension, this volume includes the batch dimension with its length set to the maximum batch size. For networks with all explicit dimensions and 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::setBindingDimensions](#). 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

<i>name</i>	The name of the tensor.
<i>type</i>	The type of the data held in the tensor.
<i>dimensions</i>	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.

9.87.3.21 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, or nullptr if network has an implicit batch dimension or this version of TensorRT does not support loops.

The network must not have an implicit batch dimension.

9.87.3.22 addLRN()

```
ILRNLayer * nvinfer1::INetworkDefinition::addLRN (
    ITensor & input,
    int32_t window,
    float alpha,
    float beta,
    float k ) [inline], [noexcept]
```

Add a LRN layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
<i>window</i>	The size of the window.
<i>alpha</i>	The alpha value for the LRN computation.
<i>beta</i>	The beta value for the LRN computation.
<i>k</i>	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.

9.87.3.23 addMatrixMultiply()

```
IMatrixMultiplyLayer * nvinfer1::INetworkDefinition::addMatrixMultiply (
    ITensor & input0,
    MatrixOperation op0,
    ITensor & input1,
    MatrixOperation op1 ) [inline], [noexcept]
```

Add a MatrixMultiply layer to the network.

Parameters

<i>input0</i>	The first input tensor (commonly A).
<i>op0</i>	The operation to apply to input0.
<i>input1</i>	The second input tensor (commonly B).
<i>op1</i>	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.

9.87.3.24 addNMS()

```
INMSLayer * nvinfer1::INetworkDefinition::addNMS (
    ITensor & boxes,
    ITensor & scores,
    ITensor & maxOutputBoxesPerClass ) [inline], [noexcept]
```

Add a non-maximum suppression layer to the network.

Parameters

<i>boxes</i>	The input boxes tensor to the layer.
<i>scores</i>	The input scores tensor to the layer.
<i>maxOutputBoxesPerClass</i>	The input maxOutputBoxesPerClass tensor to the layer.

See also

[INMSLayer](#)

Returns

The new NMS layer, or nullptr if it could not be created.

9.87.3.25 addNonZero()

```
INonZeroLayer * nvinfer1::INetworkDefinition::addNonZero (
    ITensor & input ) [inline], [noexcept]
```

Add a nonzero layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
--------------	--------------------------------

See also

[INonZeroLayer](#)

Returns

The new nonzero layer, or nullptr if it could be created.

9.87.3.26 addNormalization()

```
INormalizationLayer * nvinfer1::INetworkDefinition::addNormalization (
    ITensor & input,
    ITensor & scale,
    ITensor & bias,
    uint32_t axesMask ) [inline], [noexcept]
```

Add a normalization layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
<i>scale</i>	The scale tensor used to scale the normalized output.
<i>bias</i>	The bias tensor used to scale the normalized output.
<i>axesMask</i>	The axes on which to perform mean calculations. The bit in position <i>i</i> of bitmask <i>axesMask</i> corresponds to explicit dimension <i>i</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* are expected to be the same, and must have the same rank and be unidirectionally broadcastable to the shape of *input*.

See also

[INormalizationLayer](#)

Returns

The new normalization layer, or nullptr if it could not be created.

9.87.3.27 addOneHot()

```

IOneHotLayer * nvinfer1::INetworkDefinition::addOneHot (
    ITensor & indices,
    ITensor & values,
    ITensor & depth,
    int32_t axis ) [inline], [noexcept]

```

Add a OneHot layer to the network.

Parameters

<i>indices</i>	- tensor containing indices where on_value should be set.
<i>values</i>	- a 2-element tensor, consisting of [off_value, on_value].
<i>depth</i>	- tensor containing the width of the added one-hot dimension.
<i>axis</i>	- 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.

9.87.3.28 addPadding()

```

TRT_DEPRECATED IPaddingLayer * nvinfer1::INetworkDefinition::addPadding (
    ITensor & input,
    DimsHW prePadding,
    DimsHW postPadding ) [inline], [noexcept]

```

Add a padding layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
<i>prePadding</i>	The padding to apply to the start of the tensor.
<i>postPadding</i>	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.

Deprecated Superseded by `addPaddingNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.87.3.29 addPaddingNd()

```
TRT_DEPRECATED IPaddingLayer * nvinfer1::INetworkDefinition::addPaddingNd (
    ITensor & input,
    Dims prePadding,
    Dims postPadding ) [inline], [noexcept]
```

Add a padding layer to the network. Only 2D padding is currently supported.

Parameters

<i>input</i>	The input tensor to the layer.
<i>prePadding</i>	The padding to apply to the start of the tensor.
<i>postPadding</i>	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.

Deprecated Deprecated in TensorRT 8.0. Superseded by [addSlice\(\)](#).

9.87.3.30 addParametricReLU()

```
IParametricReLULayer * nvinfer1::INetworkDefinition::addParametricReLU (
    ITensor & input,
    ITensor & slope ) [inline], [noexcept]
```

Add a parametric ReLU layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
<i>slope</i>	The slope tensor to the layer. This tensor should be unidirectionally broadcastable to the input tensor.

See also

[IParametricReLULayer](#)

Warning

Int32 tensors are not valid input tensors.

Returns

The new parametric ReLU layer, or nullptr if it could not be created.

9.87.3.31 addPluginV2()

```
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

<i>inputs</i>	The input tensors to the layer.
<i>nbInputs</i>	The number of input tensors.
<i>plugin</i>	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.

9.87.3.32 addPooling()

```
TRT_DEPRECATED IPoolingLayer * nvinfer1::INetworkDefinition::addPooling (
    ITensor & input,
    PoolingType type,
    DimsHW windowSize ) [inline], [noexcept]
```

Add a pooling layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
<i>type</i>	The type of pooling to apply.
<i>windowSize</i>	The size of the pooling window.

See also

[IPoolingLayer](#) [PoolingType](#)

Warning

Int32 tensors are not valid input tensors.

Returns

The new pooling layer, or nullptr if it could not be created.

Deprecated Superseded by `addPoolingNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.87.3.33 addPoolingNd()

```
IPoolingLayer * nvinfer1::INetworkDefinition::addPoolingNd (
    ITensor & input,
    PoolingType type,
    Dims windowSize ) [inline], [noexcept]
```

Add a multi-dimension pooling layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
<i>type</i>	The type of pooling to apply.
<i>windowSize</i>	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.

9.87.3.34 addQuantize()

```
IQuantizeLayer * nvinfer1::INetworkDefinition::addQuantize (
    ITensor & input,
    ITensor & scale ) [inline], [noexcept]
```

Add a quantization layer to the network.

Parameters

<i>input</i>	The input tensor to be quantized.
<i>scale</i>	A tensor with the scale value.

See also

[IQuantizeLayer](#)

input tensor data type must be [DataType::kFLOAT](#). scale tensor data type must be [DataType::kFLOAT](#). The subgraph which terminates with the scale tensor must be a build-time constant.

Returns

The new quantization layer, or nullptr if it could not be created.

9.87.3.35 addRaggedSoftMax()

```
IRaggedSoftMaxLayer * nvinfer1::INetworkDefinition::addRaggedSoftMax (
    ITensor & input,
    ITensor & bounds ) [inline], [noexcept]
```

Add a RaggedSoftMax layer to the network.

Parameters

<i>input</i>	The ZxS input tensor.
<i>bounds</i>	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.

Returns

The new RaggedSoftMax layer, or nullptr if it could not be created.

9.87.3.36 addReduce()

```
IReduceLayer * nvinfer1::INetworkDefinition::addReduce (
    ITensor & input,
    ReduceOperation operation,
    uint32_t reduceAxes,
    bool keepDimensions ) [inline], [noexcept]
```

Add a reduce layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
<i>operation</i>	The reduction operation to perform.
<i>reduceAxes</i>	The reduction dimensions. The bit in position <i>i</i> of bitmask <i>reduceAxes</i> corresponds to explicit dimension <i>i</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.
<i>keepDimensions</i>	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 shape tensor, `ReduceOperation::kAVG` is unsupported.

Returns

The new reduce layer, or nullptr if it could not be created.

9.87.3.37 addResize()

```
IResizeLayer * nvinfer1::INetworkDefinition::addResize (
    ITensor & input ) [inline], [noexcept]
```

Add a resize layer to the network.

Parameters

<i>input</i>	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.

9.87.3.38 addReverseSequence()

```
IReverseSequenceLayer * nvinfer1::INetworkDefinition::addReverseSequence (
    ITensor & input,
    ITensor & sequenceLens ) [inline], [noexcept]
```

Add a ReverseSequence layer to the network.

Parameters

<i>input</i>	The input tensor to the layer. Must have rank ≥ 2 .
<i>sequenceLens</i>	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

[IRReverseSequenceLayer](#)

Returns

The new ReverseSequence layer, or nullptr if it could not be created.

9.87.3.39 addRNNv2()

```
TRT_DEPRECATED IRNNv2Layer * nvinfer1::INetworkDefinition::addRNNv2 (
    ITensor & input,
    int32_t layerCount,
    int32_t hiddenSize,
    int32_t maxSeqLen,
    RNNOperation op ) [inline], [noexcept]
```

Add an `layerCount` deep RNN layer to the network with `hiddenSize` internal states that can take a batch with fixed or variable sequence lengths.

Parameters

<i>input</i>	The input tensor to the layer (see below).
<i>layerCount</i>	The number of layers in the RNN.
<i>hiddenSize</i>	Size of the internal hidden state for each layer.
<i>maxSeqLen</i>	Maximum sequence length for the input.
<i>op</i>	The type of RNN to execute.

By default, the layer is configured with `RNNDirection::kUNIDIRECTION` and `RNNInputMode::kLINEAR`. To change these settings, use `IRNNv2Layer::setDirection()` and `IRNNv2Layer::setInputMode()`.

Weights and biases for the added layer should be set using `IRNNv2Layer::setWeightsForGate()` and `IRNNv2Layer::setBiasForGate()` prior to building an engine using this network.

The input tensors must be of the type `DataType::kFLOAT` or `DataType::kHALF`. The layout of the weights is row major and must be the same datatype as the input tensor. `weights` contain 8 matrices and `bias` contains 8 vectors.

See `IRNNv2Layer::setWeightsForGate()` and `IRNNv2Layer::setBiasForGate()` for details on the required input format for `weights` and `bias`.

The input [ITensor](#) should contain zero or more index dimensions $\{N_1, \dots, N_p\}$, followed by two dimensions, defined as follows:

- `S_max` is the maximum allowed sequence length (number of RNN iterations)
- `E` specifies the embedding length (unless `::kSKIP` is set, in which case it should match `getHiddenSize()`).

By default, all sequences in the input are assumed to be size `maxSeqLen`. To provide explicit sequence lengths for each input sequence in the batch, use [IRNNv2Layer::setSequenceLengths\(\)](#).

The RNN layer outputs up to three tensors.

The first output tensor is the output of the final RNN layer across all timesteps, with dimensions $\{N_1, \dots, N_p, S_{\max}, H\}$:

- $N_1 \dots N_p$ are the index dimensions specified by the input tensor
- `S_max` is the maximum allowed sequence length (number of RNN iterations)
- `H` is an output hidden state (equal to `getHiddenSize()` or `2x getHiddenSize()`)

The second tensor is the final hidden state of the RNN across all layers, and if the RNN is an LSTM (i.e. `getOperation()` is `::kLSTM`), then the third tensor is the final cell state of the RNN across all layers. Both the second and third output tensors have dimensions $\{N_1, \dots, N_p, L, H\}$:

- $N_1 \dots N_p$ are the index dimensions specified by the input tensor
- `L` is the number of layers in the RNN, equal to `getLayerCount()` if `getDirection` is `::kUNIDIRECTION`, and `2x getLayerCount()` if `getDirection` is `::kBIDIRECTION`. In the bi-directional case, layer 1's final forward hidden state is stored in `L = 2*1`, and final backward hidden state is stored in `L = 2*1 + 1`.
- `H` is the hidden state for each layer, equal to `getHiddenSize()`.

See also

[IRNNv2Layer](#)

Deprecated Deprecated prior to TensorRT 8.0 and will be removed in 9.0. Superseded by [INetworkDefinition::addLoop\(\)](#).

Warning

RNN inputs do not support wildcard dimensions or explicit batch size networks.
Int32 tensors are not valid input tensors, only for sequence lengths.

Returns

The new RNN layer, or nullptr if it could not be created.

9.87.3.40 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

<i>input</i>	The input tensor to the layer. This tensor is required to have a minimum of 3 dimensions in implicit batch mode and a minimum of 4 dimensions in explicit batch mode.
<i>mode</i>	The scaling mode.
<i>shift</i>	The shift value.
<i>scale</i>	The scale value.
<i>power</i>	The power value.

If the weights are available, then the size of weights are dependent on the ScaleMode. For ::kUNIFORM, the number of weights equals 1. For ::kCHANNEL, the number of weights equals the channel dimension. For ::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.

9.87.3.41 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

<i>input</i>	The input tensor to the layer.
<i>mode</i>	The scaling mode.
<i>shift</i>	The shift value.
<i>scale</i>	The scale value.
<i>power</i>	The power value.
<i>channelAxis</i>	The channel axis.

If the weights are available, then the size of weights are dependent on the ScaleMode. For `::kUNIFORM`, the number of weights equals 1. For `::kCHANNEL`, the number of weights equals the channel dimension. For `::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 `::kUNIFORM`, the number of weights is equal to 1. For `::kCHANNEL`, the number of weights is C. For `::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.

9.87.3.42 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

<i>input</i>	The input tensor to be updated with additional values.
<i>indices</i>	indices of the elements to be updated.
<i>updates</i>	values to be used for updates.

See also

[IScatterLayer](#)

input tensor data type must be [DataType::kFLOAT](#). indices tensor data type must be [DataType::kINT32](#). updates tensor data type must be [DataType::kFLOAT](#).

Returns

The new Scatter layer, or nullptr if it could not be created.

9.87.3.43 addSelect()

```
ISelectLayer * nvinfer1::INetworkDefinition::addSelect (
    ITensor & condition,
    ITensor & thenInput,
    ITensor & elseInput ) [inline], [noexcept]
```

Add a select layer to the network.

Parameters

<i>condition</i>	The condition tensor to the layer. Must have type DataType::kBOOL .
<i>thenInput</i>	The "then" input tensor to the layer.
<i>elseInput</i>	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 network must not have an implicit batch dimension.

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.

9.87.3.44 addShape()

```
IShapeLayer * nvinfer1::INetworkDefinition::addShape (
    ITensor & input ) [inline], [noexcept]
```

Add a shape layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
--------------	--------------------------------

See also

[IShapeLayer](#)

Warning

addShape is only supported when hasImplicitBatchDimensions is false.
input to addShape cannot contain wildcard dimension values.

Returns

The new shape layer, or nullptr if it could not be created.

9.87.3.45 addShuffle()

```
IShuffleLayer * nvinfer1::INetworkDefinition::addShuffle (
    ITensor & input ) [inline], [noexcept]
```

Add a shuffle layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
--------------	--------------------------------

See also

[IShuffleLayer](#)

Returns

The new shuffle layer, or nullptr if it could not be created.

9.87.3.46 addSlice()

```
ISliceLayer * nvinfer1::INetworkDefinition::addSlice (
    ITensor & input,
```

```

    Dims start,
    Dims size,
    Dims stride ) [inline], [noexcept]

```

Add a slice layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
<i>start</i>	The start offset
<i>size</i>	The output dimension
<i>stride</i>	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.

9.87.3.47 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.

9.87.3.48 addTopK()

```
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.

Parameters

<i>input</i>	The input tensor to the layer.
<i>op</i>	Operation to perform.
<i>k</i>	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.
<i>reduceAxes</i>	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](#)

Warning

Int32 tensors are not valid input tensors.

Returns

The new TopK layer, or nullptr if it could not be created.

9.87.3.49 addUnary()

```
IUnaryLayer * nvinfer1::INetworkDefinition::addUnary (
    ITensor & input,
    UnaryOperation operation ) [inline], [noexcept]
```

Add a unary layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
<i>operation</i>	The operation to apply.

See also

[UnaryLayer](#)

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

9.87.3.50 destroy()

`TRT_DEPRECATED` void nvinfer1::INetworkDefinition::destroy () [inline], [noexcept]

Destroy this [INetworkDefinition](#) object.

Deprecated Deprecated in TensorRT 8.0. Superseded by `delete`.

Warning

Calling destroy on a managed pointer will result in a double-free error.

9.87.3.51 getErrorRecorder()

```
IErrRecorder * 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 [IErrRecorder](#) object that has been registered.

See also

[setErrorRecorder\(\)](#)

9.87.3.52 getInput()

```
ITensor * nvinfer1::INetworkDefinition::getInput (
    int32_t index ) const [inline], [noexcept]
```

Get the input tensor specified by the given index.

Parameters

<i>index</i>	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\(\)](#)

9.87.3.53 getLayer()

```
ILayer * nvinfer1::INetworkDefinition::getLayer (
    int32_t index ) const [inline], [noexcept]
```

Get the layer specified by the given index.

Parameters

<i>index</i>	The index of the layer.
--------------	-------------------------

Returns

The layer, or nullptr if the index is out of range.

See also

[getNbLayers\(\)](#)

9.87.3.54 `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.

9.87.3.55 `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\(\)](#)

9.87.3.56 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\(\)](#)

9.87.3.57 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\(\)](#)

9.87.3.58 getOutput()

```
ITensor * nvinfer1::INetworkDefinition::getOutput (
    int32_t index ) const [inline], [noexcept]
```

Get the output tensor specified by the given index.

Parameters

<i>index</i>	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\(\)](#)

9.87.3.59 hasExplicitPrecision()

`TRT_DEPRECATED` `bool nvinfer1::INetworkDefinition::hasExplicitPrecision () const [inline], [noexcept]`

True if network is an explicit precision network.

Deprecated Deprecated in TensorRT 8.0.

See also

[createNetworkV2](#)

Returns

True if network has explicit precision, false otherwise.

9.87.3.60 hasImplicitBatchDimension()

`bool nvinfer1::INetworkDefinition::hasImplicitBatchDimension () const [inline], [noexcept]`

Query whether the network was created with an implicit batch dimension.

Returns

True if tensors have implicit batch dimension, false otherwise.

This is a network-wide property. Either all tensors in the network have an implicit batch dimension or none of them do.

[hasImplicitBatchDimension\(\)](#) is true if and only if this [INetworkDefinition](#) was created with [createNetworkV2\(\)](#) without [NetworkDefinitionCreationFlag::kEXPLICIT_BATCH](#) flag.

See also

[createNetworkV2](#)

9.87.3.61 markOutput()

`void nvinfer1::INetworkDefinition::markOutput (
 ITensor & tensor) [inline], [noexcept]`

Mark a tensor as a network output.

Parameters

<i>tensor</i>	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.

9.87.3.62 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.

See also

[isShapeBinding\(\)](#), [getShapeBinding\(\)](#)

9.87.3.63 removeTensor()

```
void nvinfer1::INetworkDefinition::removeTensor (
    ITensor & tensor ) [inline], [noexcept]
```

remove a tensor from the network definition.

Parameters

<i>tensor</i>	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()`.

9.87.3.64 `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

<i>recorder</i>	The error recorder to register with this interface.
-----------------	---

See also

[getErrorRecorder\(\)](#)

9.87.3.65 `setName()`

```
void nvinfer1::INetworkDefinition::setName (
    char const * name ) [inline], [noexcept]
```

Sets the name of the network.

Parameters

<i>name</i>	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

9.87.3.66 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

<i>weights</i>	The weights to be named.
<i>name</i>	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.

9.87.3.67 unmarkOutput()

```
void nvinfer1::INetworkDefinition::unmarkOutput (
    ITensor & tensor ) [inline], [noexcept]
```

unmark a tensor as a network output.

Parameters

<i>tensor</i>	The tensor to unmark as an output tensor.
---------------	---

see [markOutput\(\)](#)

9.87.3.68 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.

9.87.4 Member Data Documentation

9.87.4.1 mImpl

```
apiv::VNetworkDefinition* nvinfer1::INetworkDefinition::mImpl [protected]
```

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

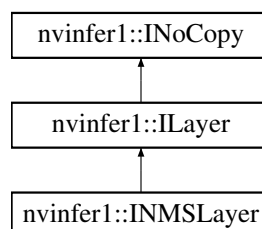
- [NvInfer.h](#)

9.88 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.
- 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=default

Protected Attributes

- apiv::VNMSLayer * [mImpl](#)

9.88.1 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.

For each batch item, the ordering of candidate bounding boxes with the same score is unspecified.

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`. 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.
- **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.

9.88.2 Constructor & Destructor Documentation

9.88.2.1 ~INMSLayer()

```
virtual nvinfer1::INMSLayer::~INMSLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.88.3 Member Function Documentation

9.88.3.1 getBoundingBoxFormat()

```
BoundingBoxFormat nvinfer1::INMSLayer::getBoundingBoxFormat ( ) const [inline], [noexcept]
```

Get the bounding box format parameter for the layer.

See also

[BoundingBoxFormat](#)

[setBoundingBoxFormat\(\)](#)

9.88.3.2 getTopKBoxLimit()

```
int32_t nvinfer1::INMSLayer::getTopKBoxLimit ( ) const [inline], [noexcept]
```

Get the TopK box limit parameter for the layer.

See also

[setTopKBoxLimit\(\)](#)

9.88.3.3 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\(\)](#)

9.88.3.4 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

<i>index</i>	the index of the input to modify.
<i>tensor</i>	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.

9.88.3.5 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\(\)](#)

9.88.4 Member Data Documentation

9.88.4.1 mImpl

```
apiv::VNMSLayer* nvinfer1::INMSLayer::mImpl [protected]
```

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

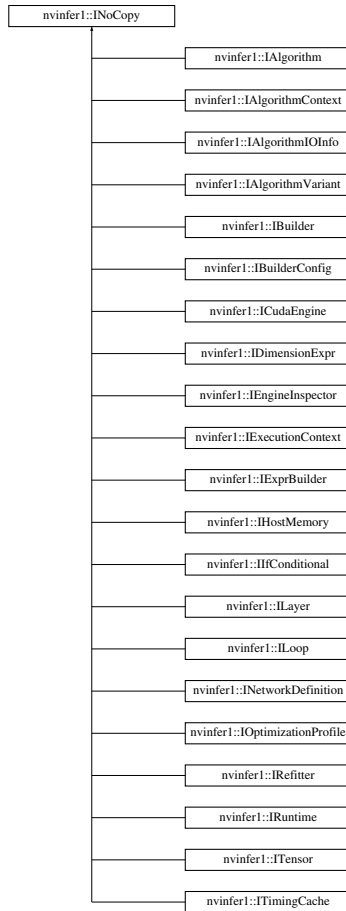
- [NvInfer.h](#)

9.89 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

9.89.1 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.

9.89.2 Constructor & Destructor Documentation

9.89.2.1 INoCopy() [1/3]

```
nvinfer1::INoCopy::INoCopy ( ) [protected], [default]
```

9.89.2.2 ~INoCopy()

```
virtual nvinfer1::INoCopy::~~INoCopy ( ) [protected], [virtual], [default]
```

9.89.2.3 INoCopy() [2/3]

```
nvinfer1::INoCopy::INoCopy (
    INoCopy const & other ) [protected], [delete]
```

9.89.2.4 INoCopy() [3/3]

```
nvinfer1::INoCopy::INoCopy (
    INoCopy && other ) [protected], [delete]
```

9.89.3 Member Function Documentation

9.89.3.1 operator=() [1/2]

```
INoCopy & nvinfer1::INoCopy::operator= (
    INoCopy && other ) [protected], [delete]
```

9.89.3.2 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](#)

9.90 INonZero Class Reference

A NonZero layer in a network.

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

9.90.1 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. 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.

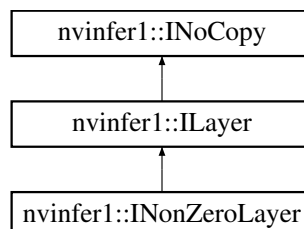
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.91 nvinfer1::INonZeroLayer Class Reference

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

Inheritance diagram for nvinfer1::INonZeroLayer:



Protected Member Functions

- virtual [~INonZeroLayer](#) () noexcept=default

Protected Attributes

- apiv::VNonZeroLayer * [mImpl](#)

Additional Inherited Members

9.91.1 Constructor & Destructor Documentation

9.91.1.1 ~INonZeroLayer()

```
virtual nvinfer1::INonZeroLayer::~~INonZeroLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.91.2 Member Data Documentation

9.91.2.1 mImpl

```
apiv::VNonZeroLayer* nvinfer1::INonZeroLayer::mImpl [protected]
```

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

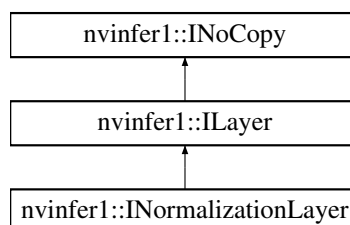
- [NvInfer.h](#)

9.92 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](#) (int32_t nbGroups) noexcept
Set the number of groups used to split the channels in the normalization calculation.
- int32_t [getNbGroups](#) () const noexcept
Get the number of groups used to split the channels for the normalization calculation.
- void [setComputePrecision](#) (DataType type) noexcept
Set the compute precision of this layer.
- DataType [getComputePrecision](#) () const noexcept
Get the compute precision of this layer.

Protected Member Functions

- virtual [~INormalizationLayer](#) () noexcept=default

Protected Attributes

- apiv::VNormalizationLayer * [mImpl](#)

9.92.1 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

$$Y = (X - \text{Mean}(X, \text{axes})) / \text{Sqrt}(\text{Variance}(X) + \text{epsilon}) * S + B$$

Where $\text{Mean}(X, \text{axes})$ is a reduction over a set of axes, and $\text{Variance}(X) = \text{Mean}((X - \text{Mean}(X, \text{axes}))^2, \text{axes})$.

Warning

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 ~INormalizationLayer()

```
virtual nvinfer1::INormalizationLayer::~~INormalizationLayer ( ) [protected], [virtual], [default],  
[noexcept]
```

9.92.3 Member Function Documentation

9.92.3.1 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.

9.92.3.2 getComputePrecision()

```
DataType nvinfer1::INormalizationLayer::getComputePrecision ( ) const [inline], [noexcept]
```

Get the compute precision of this layer.

Returns

The datatype used for the compute precision of this layer.

9.92.3.3 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.

9.92.3.4 getNbGroups()

```
int32_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.

9.92.3.5 setAxes()

```
void nvinfer1::INormalizationLayer::setAxes (
    uint32_t axesMask ) [inline], [noexcept]
```

Set the reduction axes for the normalization calculation.

Parameters

<i>axesMask</i>	The axes used for the normalization calculation.
-----------------	--

9.92.3.6 setComputePrecision()

```
void nvinfer1::INormalizationLayer::setComputePrecision (
    DataType type ) [inline], [noexcept]
```

Set the compute precision of this layer.

Parameters

<i>type</i>	The datatype used for the compute precision of this layer.
-------------	--

By default TensorRT will run the normalization computation in `DataType::kFLOAT32` even in mixed precision mode regardless of any set builder flags to avoid overflow errors. To override this default, use this function to set the desired compute precision.

[setPrecision\(\)](#) and [setOutputPrecision\(\)](#) functions can still be called to control the input and output data types to this layer.

Only `DataType::kFLOAT32` and [DataType::kHALF](#) are valid types for `type`.

9.92.3.7 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

<i>eps</i>	The epsilon value used for the normalization calculation.
------------	---

9.92.3.8 setNbGroups()

```
void nvinfer1::INormalizationLayer::setNbGroups (
    int32_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

<i>nbGroups</i>	The number of groups to split the channels into for the normalization calculation.
-----------------	--

9.92.4 Member Data Documentation

9.92.4.1 mImpl

```
apiv::VNormalizationLayer* nvinfer1::INormalizationLayer::mImpl [protected]
```

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

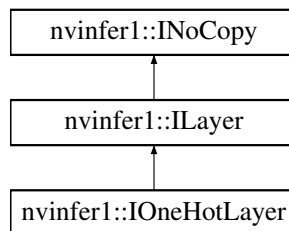
- [NvInfer.h](#)

9.93 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 Attributes

- apiv::VOneHotLayer * [mImpl](#)

Additional Inherited Members

9.93.1 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 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.

- 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. Axis defaults to -1, that is the new dimension in the output is its final dimension. 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.

9.93.2 Member Function Documentation

9.93.2.1 getAxis()

```
int32_t nvinfer1::IOneHotLayer::getAxis ( ) const [inline], [noexcept]
```

Get the value of the axis parameter.

9.93.2.2 setAxis()

```
void nvinfer1::IOneHotLayer::setAxis (
    int32_t axis ) [inline], [noexcept]
```

Set the axis parameter.

See also

[IOneHotLayer](#)

9.93.3 Member Data Documentation

9.93.3.1 mImpl

```
apiv::VOneHotLayer* nvinfer1::IOneHotLayer::mImpl [protected]
```

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

- [NvInfer.h](#)

9.94 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=default
- 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.
- virtual [TRT_DEPRECATED](#) void [destroy](#) () noexcept=0
Destroy IOnnxConfig object.

9.94.1 Detailed Description

Configuration Manager Class.

9.94.2 Member Typedef Documentation

9.94.2.1 Verbosity

`nvonnxparser::IOnnxConfig::Verbosity`

Defines Verbosity level.

9.94.3 Constructor & Destructor Documentation

9.94.3.1 ~IOnnxConfig()

```
virtual nvonnxparser::IOnnxConfig::~IOnnxConfig ( ) [virtual], [default], [noexcept]
```

9.94.4 Member Function Documentation

9.94.4.1 addVerbosity()

```
virtual void nvonnxparser::IOnnxConfig::addVerbosity ( ) [pure virtual], [noexcept]
```

Increase the Verbosity Level.

Returns

The Verbosity Level.

See also

[reduceVerbosity\(\)](#), [setVerbosity\(Verbosity\)](#)

9.94.4.2 destroy()

```
virtual TRT\_DEPRECATED void nvonnxparser::IOnnxConfig::destroy ( ) [pure virtual], [noexcept]
```

Destroy [IOnnxConfig](#) object.

Deprecated Use `delete` instead. Deprecated in TRT 8.0.

Warning

Calling destroy on a managed pointer will result in a double-free error.

9.94.4.3 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\(\)](#)

9.94.4.4 getModelDtype()

```
virtual nvinfer1::DataType nvonnxparser::IOnnxConfig::getModelDtype ( ) const [pure virtual],  
[noexcept]
```

Get the Model Data Type.

Returns

[DataType](#) [nvinfer1::DataType](#)

See also

[setModelDtype\(\)](#) and [#DataType](#)

9.94.4.5 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\(\)](#)

9.94.4.6 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\(\)](#)

9.94.4.7 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\(\)](#)

9.94.4.8 getVerbosityLevel()

```
virtual Verbosity nvonnxparser::IOnnxConfig::getVerbosityLevel ( ) const [pure virtual], [noexcept]
```

Get the Verbosity Level.

Returns

The Verbosity Level.

See also

[addVerbosity\(\)](#), [reduceVerbosity\(\)](#)

9.94.4.9 reduceVerbosity()

```
virtual void nvonnxparser::IOnnxConfig::reduceVerbosity ( ) [pure virtual], [noexcept]
```

Reduce the Verbosity Level.

See also

[addVerbosity\(\)](#), [setVerbosity\(Verbosity\)](#)

9.94.4.10 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

<i>fullTextFileName</i>	Name of the file.
-------------------------	-------------------

See also

[getFullTextFilename\(\)](#)

9.94.4.11 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\(\)](#)

9.94.4.12 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

<i>onnxFilename</i>	The name.
---------------------	-----------

See also

[getModelFileName\(\)](#)

9.94.4.13 setPrintLayerInfo()

```
virtual void nvonnxparser::IOnnxConfig::setPrintLayerInfo (
    bool ) [pure virtual], [noexcept]
```

Set whether the layer information will be printed.

See also

[getPrintLayerInfo\(\)](#)

9.94.4.14 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

<i>textFileName</i>	Name of the file.
---------------------	-------------------

See also

[getTextFilename\(\)](#)

9.94.4.15 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:

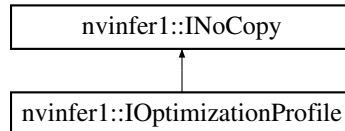
- [NvOnnxConfig.h](#)

9.95 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](#) 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.
- bool [setShapeValues](#) (char const *inputName, [OptProfileSelector](#) select, int32_t const *values, int32_t nbValues) noexcept
Set the minimum / optimum / maximum values for an input shape tensor.
- int32_t [getNbShapeValues](#) (char const *inputName) const noexcept
Get the number of values for an input shape tensor.
- int32_t const * [getShapeValues](#) (char const *inputName, [OptProfileSelector](#) select) const noexcept
Get the minimum / optimum / maximum 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.

Protected Member Functions

- virtual [~IOptimizationProfile](#) () noexcept=default

Protected Attributes

- apiv::VOptimizationProfile * [mImpl](#)

9.95.1 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\(\)](#)

9.95.2 Constructor & Destructor Documentation

9.95.2.1 ~IOptimizationProfile()

```
virtual nvinfer1::IOptimizationProfile::~~IOptimizationProfile ( ) [protected], [virtual], [default],
[noexcept]
```

9.95.3 Member Function Documentation

9.95.3.1 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.

9.95.3.2 `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.

9.95.3.3 `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 [setShapeValues\(\)](#) 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.

9.95.3.4 `getShapeValues()`

```
int32_t const * nvinfer1::IOptimizationProfile::getShapeValues (
    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 [setShapeValues\(\)](#), this returns nullptr.

Warning

The string `inputName` must be null-terminated, and be at most 4096 bytes including the terminator.

9.95.3.5 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.

9.95.3.6 setDimensions()

```
bool nvinfer1::IOptimizationProfile::setDimensions (
    char const * inputName,
    OptProfileSelector select,
    Dims 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

<i>inputName</i>	The input tensor name
<i>select</i>	Whether to set the minimum, optimum, or maximum dimensions
<i>dims</i>	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 be the same.
The string `inputName` must be null-terminated, and be at most 4096 bytes including the terminator.

9.95.3.7 setExtraMemoryTarget()

```
bool nvinfer1::IOptimizationProfile::setExtraMemoryTarget (
    float target ) [inline], [noexcept]
```

Set a target for extra GPU memory that may be used by this profile.

Parameters

<i>target</i>	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 <code>target == 1.0</code> . If <code>target == 0.0</code> , 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.

9.95.3.8 setShapeValues()

```
bool nvinfer1::IOptimizationProfile::setShapeValues (
    char const * inputName,
    OptProfileSelector select,
    int32_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 datatype of `t` is `DataType::kINT32`, 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.

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, and can contain any `int32_t` values appropriate for the network. Shape tensors of other data types (e.g. float) are not supported. 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

<i>inputName</i>	The input tensor name
<i>select</i>	Whether to set the minimum, optimum, or maximum input values.
<i>values</i>	An array of length nbValues containing the minimum, optimum, or maximum shape tensor elements.
<i>nbValues</i>	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.

9.95.4 Member Data Documentation

9.95.4.1 mImpl

```
apiv::VOptimizationProfile* nvinfer1::IOptimizationProfile::mImpl [protected]
```

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

- [NvInferRuntime.h](#)

9.96 nvinfer1::IOutputAllocator Class Reference

Callback from ExecutionContext::enqueueV3()

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

Public Member Functions

- virtual int32_t [getInterfaceVersion](#) () const noexcept
Return the API version of this [IOutputAllocator](#).
- virtual void * [reallocateOutput](#) (char const *tensorName, void *currentMemory, uint64_t size, uint64_t alignment) noexcept=0
Return a pointer to memory for an output tensor, or nullptr if memory cannot be allocated.
- virtual void [notifyShape](#) (char const *tensorName, [Dims](#) const &dims) noexcept=0
Called by TensorRT when the shape of the output tensor is known.
- virtual [~IOutputAllocator](#) ()=default

9.96.1 Detailed Description

Callback from ExecutionContext::enqueueV3()

Clients should override the method [reallocateOutput](#).

See also

[IExecutionContext::enqueueV3\(\)](#)

9.96.2 Constructor & Destructor Documentation

9.96.2.1 ~IOutputAllocator()

```
virtual nvinfer1::IOutputAllocator::~~IOutputAllocator ( ) [virtual], [default]
```

9.96.3 Member Function Documentation

9.96.3.1 `getInterfaceVersion()`

```
virtual int32_t nvinfer1::IOutputAllocator::getInterfaceVersion ( ) const [inline], [virtual],
[noexcept]
```

Return the API version of this [IOutputAllocator](#).

Do not override this method as it is used by the TensorRT library to maintain backwards-compatibility with [IOutputAllocator](#). The value will change if Nvidia adds additional virtual methods to this class.

9.96.3.2 `notifyShape()`

```
virtual void nvinfer1::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

<i>dims</i>	dimensions of the output
<i>tensorName</i>	name of the tensor

9.96.3.3 `reallocateOutput()`

```
virtual void * nvinfer1::IOutputAllocator::reallocateOutput (
    char const * tensorName,
    void * currentMemory,
    uint64_t size,
    uint64_t alignment ) [pure virtual], [noexcept]
```

Return a pointer to memory for an output tensor, or `nullptr` if memory cannot be allocated.

Parameters

<i>tensorName</i>	name of the output tensor.
<i>currentMemory</i>	points to the address set by <code>IExecutionContext::setTensorAddress</code> .
<i>size</i>	number of bytes required. Always positive, even for an empty tensor.
<i>alignment</i>	required alignment of the allocation.

Returns

A pointer to memory to use for the output tensor or nullptr.

If currentMemory is known to be big enough, one option is to return currentMemory.

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.

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

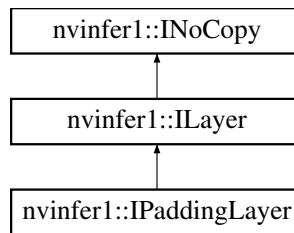
- [NvInferRuntime.h](#)

9.97 nvinfer1::IPaddingLayer Class Reference

Layer that represents a padding operation.

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

Inheritance diagram for nvinfer1::IPaddingLayer:



Public Member Functions

- [TRT_DEPRECATED](#) void [setPrePadding](#) ([DimsHW](#) padding) noexcept
Set the padding that is applied at the start of the tensor.
- [TRT_DEPRECATED](#) [DimsHW](#) [getPrePadding](#) () const noexcept
Get the padding that is applied at the start of the tensor.
- [TRT_DEPRECATED](#) void [setPostPadding](#) ([DimsHW](#) padding) noexcept
Set the padding that is applied at the end of the tensor.
- [TRT_DEPRECATED](#) [DimsHW](#) [getPostPadding](#) () const noexcept
Get the padding that is applied at the end of the tensor.
- void [setPrePaddingNd](#) ([Dims](#) 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](#) 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=default

Protected Attributes

- apiv::VPaddingLayer * [mImpl](#)

9.97.1 Detailed Description

Layer that represents a padding operation.

The padding layer adds zero-padding at the start and end of the input tensor. It only supports padding along the two innermost dimensions. Applying negative padding results in cropping of the input.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.97.2 Constructor & Destructor Documentation

9.97.2.1 ~IPaddingLayer()

```
virtual nvinfer1::IPaddingLayer::~~IPaddingLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.97.3 Member Function Documentation

9.97.3.1 getPostPadding()

```
TRT\_DEPRECATED DimsHW nvinfer1::IPaddingLayer::getPostPadding ( ) const [inline], [noexcept]
```

Get the padding that is applied at the end of the tensor.

See also

[setPostPadding](#)

Deprecated Superseded by `getPostPaddingNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.97.3.2 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](#)

9.97.3.3 getPrePadding()

```
TRT_DEPRECATED DimsHW nvinfer1::IPaddingLayer::getPrePadding ( ) const [inline], [noexcept]
```

Get the padding that is applied at the start of the tensor.

See also

[setPrePadding](#)

Deprecated Superseded by `getPrePaddingNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.97.3.4 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](#)

9.97.3.5 setPostPadding()

```
TRT_DEPRECATED void nvinfer1::IPaddingLayer::setPostPadding (
    DimsHW 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

See also

[getPostPadding](#)

Deprecated Superseded by setPostPaddingNd. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.97.3.6 setPostPaddingNd()

```
void nvinfer1::IPaddingLayer::setPostPaddingNd (
    Dims 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](#)

9.97.3.7 setPrePadding()

```
TRT_DEPRECATED void nvinfer1::IPaddingLayer::setPrePadding (
    DimsHW 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

See also

[getPrePadding](#)

Deprecated Superseded by setPrePaddingNd. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.97.3.8 setPrePaddingNd()

```
void nvinfer1::IPaddingLayer::setPrePaddingNd (
    Dims 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](#)

9.97.4 Member Data Documentation

9.97.4.1 mImpl

```
apiv::VPaddingLayer* nvinfer1::IPaddingLayer::mImpl [protected]
```

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

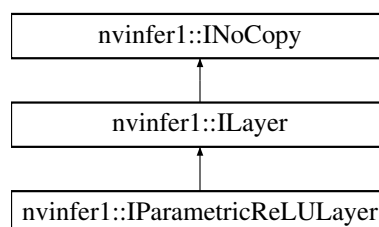
- [NvInfer.h](#)

9.98 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=default

Protected Attributes

- apiv::VParametricReLULayer * [mImpl](#)

Additional Inherited Members

9.98.1 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.

9.98.2 Constructor & Destructor Documentation

9.98.2.1 ~IParametricReLULayer()

```
virtual nvinfer1::IParametricReLULayer::~~IParametricReLULayer ( ) [protected], [virtual], [default],  
[noexcept]
```

9.98.3 Member Data Documentation

9.98.3.1 mImpl

```
apiv::VParametricReLULayer* nvinfer1::IParametricReLULayer::mImpl [protected]
```

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

- [NvInfer.h](#)

9.99 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)=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 is the user responsibility to intercept and report the error. To obtain a better diagnostic, use the [parseFromFile](#) method below.
- virtual bool [parseFromFile](#) (const char *onnxModelFile, int verbosity)=0
Parse an onnx model file, which can be a binary protobuf or a text onnx model calls [parse](#) method inside.
- virtual bool [supportsModel](#) (void const *serialized_onnx_model, size_t serialized_onnx_model_size, [SubGraphCollection_t](#) &sub_graph_collection, const char *model_path=nullptr)=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](#).
- virtual bool [parseWithWeightDescriptors](#) (void const *serialized_onnx_model, size_t serialized_onnx_model_size)=0
Parse a serialized ONNX model into the TensorRT network with consideration of user provided weights.
- virtual bool [supportsOperator](#) (const char *op_name) const =0
Returns whether the specified operator may be supported by the parser.
- virtual [TRT_DEPRECATED](#) void [destroy](#) ()=0
destroy this object
- virtual int [getNbErrors](#) () const =0
Get the number of errors that occurred during prior calls to [parse](#).
- virtual [IParserError](#) const * [getError](#) (int index) const =0
Get an error that occurred during prior calls to [parse](#).
- virtual void [clearErrors](#) ()=0
Clear errors from prior calls to [parse](#).
- virtual [~IParser](#) () noexcept=default

9.99.1 Detailed Description

an object for parsing ONNX models into a TensorRT network definition

9.99.2 Constructor & Destructor Documentation

9.99.2.1 ~IParser()

```
virtual nvonnxparser::IParser::~~IParser ( ) [virtual], [default], [noexcept]
```

9.99.3 Member Function Documentation

9.99.3.1 clearErrors()

```
virtual void nvonnxparser::IParser::clearErrors ( ) [pure virtual]
```

Clear errors from prior calls to parse.

See also

[getNbErrors\(\)](#) [getError\(\)](#) [IParserError](#)

9.99.3.2 destroy()

```
virtual TRT\_DEPRECATED void nvonnxparser::IParser::destroy ( ) [pure virtual]
```

destroy this object

Warning

deprecated and planned on being removed in TensorRT 10.0

9.99.3.3 getError()

```
virtual IParserError const * nvonnxparser::IParser::getError (
    int index ) const [pure virtual]
```

Get an error that occurred during prior calls to parse.

See also

[getNbErrors\(\)](#) [clearErrors\(\)](#) [IParserError](#)

9.99.3.4 getNbErrors()

```
virtual int nvonnxparser::IParser::getNbErrors ( ) const [pure virtual]
```

Get the number of errors that occurred during prior calls to parse.

See also

[getError\(\)](#) [clearErrors\(\)](#) [IParserError](#)

9.99.3.5 parse()

```
virtual bool nvonnxparser::IParser::parse (
    void const * serialized_onnx_model,
    size_t serialized_onnx_model_size,
    const char * model_path = nullptr ) [pure virtual]
```

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 is the user responsibility to intercept and report the error. To obtain a better diagnostic, use the `parseFromFile` method below.

Parameters

<i>serialized_onnx_model</i>	Pointer to the serialized ONNX model
<i>serialized_onnx_model_size</i>	Size of the serialized ONNX model in bytes
<i>model_path</i>	Absolute path to the model file for loading external weights if required

Returns

true if the model was parsed successfully

See also

[getNbErrors\(\)](#) [getError\(\)](#)

9.99.3.6 parseFromFile()

```
virtual bool nvonnxparser::IParser::parseFromFile (
    const char * onnxModelFile,
    int verbosity ) [pure virtual]
```

Parse an onnx model file, which can be a binary protobuf or a text onnx model calls parse method inside.

Parameters

<i>File</i>	name
<i>Verbosity</i>	Level

Returns

true if the model was parsed successfully

9.99.3.7 parseWithWeightDescriptors()

```
virtual bool nvonnxparser::IParser::parseWithWeightDescriptors (
    void const * serialized_onnx_model,
    size_t serialized_onnx_model_size ) [pure virtual]
```

Parse a serialized ONNX model into the TensorRT network with consideration of user provided weights.

Parameters

<i>serialized_onnx_model</i>	Pointer to the serialized ONNX model
<i>serialized_onnx_model_size</i>	Size of the serialized ONNX model in bytes

Returns

true if the model was parsed successfully

See also

[getNbErrors\(\)](#) [getError\(\)](#)

9.99.3.8 supportsModel()

```
virtual bool nvonnxparser::IParser::supportsModel (
    void const * serialized_onnx_model,
    size_t serialized_onnx_model_size,
    SubGraphCollection\_t & sub_graph_collection,
    const char * model_path = nullptr ) [pure virtual]
```

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`.

Parameters

<i>serialized_onnx_model</i>	Pointer to the serialized ONNX model
<i>serialized_onnx_model_size</i>	Size of the serialized ONNX model in bytes
<i>sub_graph_collection</i>	Container to hold supported subgraphs
<i>model_path</i>	Absolute path to the model file for loading external weights if required

Returns

true if the model is supported

9.99.3.9 supportsOperator()

```
virtual bool nvonnxparser::IParser::supportsOperator (
    const char * op_name ) const [pure virtual]
```

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

<i>op_name</i>	The name of the ONNX operator to check for support
----------------	--

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

- [NvOnnxParser.h](#)

9.100 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 const char * [desc](#) () const =0
description of the error

- virtual const char * [file](#) () const =0
source file in which the error occurred
- virtual int [line](#) () const =0
source line at which the error occurred
- virtual const char * [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

Protected Member Functions

- virtual [~IParserError](#) ()

9.100.1 Detailed Description

an object containing information about an error

9.100.2 Constructor & Destructor Documentation

9.100.2.1 ~IParserError()

```
virtual nvonnxparser::IParserError::~~IParserError ( ) [inline], [protected], [virtual]
```

9.100.3 Member Function Documentation

9.100.3.1 code()

```
virtual ErrorCode nvonnxparser::IParserError::code ( ) const [pure virtual]
```

the error code

9.100.3.2 desc()

```
virtual const char * nvonnxparser::IParserError::desc ( ) const [pure virtual]
```

description of the error

9.100.3.3 file()

```
virtual const char * nvonnxparser::IParserError::file ( ) const [pure virtual]
```

source file in which the error occurred

9.100.3.4 func()

```
virtual const char * nvonnxparser::IParserError::func ( ) const [pure virtual]
```

source function in which the error occurred

9.100.3.5 line()

```
virtual int nvonnxparser::IParserError::line ( ) const [pure virtual]
```

source line at which the error occurred

9.100.3.6 node()

```
virtual int nvonnxparser::IParserError::node ( ) const [pure virtual]
```

index of the ONNX model node in which the error occurred

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

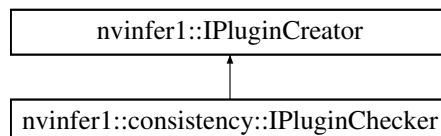
- [NvOnnxParser.h](#)

9.101 nvinfer1::consistency::IPluginChecker Class Reference

Consistency Checker plugin class for user implemented Plugins.

```
#include <NvInferConsistency.h>
```

Inheritance diagram for nvinfer1::consistency::IPluginChecker:



Public Member Functions

- virtual bool [validate](#) (char const *name, void const *serialData, size_t serialLength, [PluginTensorDesc](#) const *in, size_t nbInputs, [PluginTensorDesc](#) const *out, size_t nbOutputs, int64_t workspaceSize) const noexcept=0
Called during [IConsistencyChecker::validate](#). Allows users to provide custom validation of serialized Plugin data. Returns boolean that indicates whether or not the Plugin passed validation.
- [IPluginChecker](#) ()=default
- virtual [~IPluginChecker](#) () override=default

Protected Member Functions

- [IPluginChecker](#) ([IPluginChecker](#) const &)=default
- [IPluginChecker](#) ([IPluginChecker](#) &&)=default
- [IPluginChecker](#) & operator= ([IPluginChecker](#) const &) &=default
- [IPluginChecker](#) & operator= ([IPluginChecker](#) &&) &=default

9.101.1 Detailed Description

Consistency Checker plugin class for user implemented Plugins.

Plugins are a mechanism for applications to implement custom layers. It provides a mechanism to register Consistency plugins and look up the Plugin Registry during validate.

Supported IPlugin interfaces are limited to [IPluginV2IOExt](#) only.

9.101.2 Constructor & Destructor Documentation

9.101.2.1 IPluginChecker() [1/3]

```
nvinfer1::consistency::IPluginChecker::IPluginChecker ( ) [default]
```

9.101.2.2 ~IPluginChecker()

```
virtual nvinfer1::consistency::IPluginChecker::~~IPluginChecker ( ) [override], [virtual], [default]
```


9.101.2.3 IPluginChecker() [2/3]

```
nvinfer1::consistency::IPluginChecker::IPluginChecker (
    IPluginChecker const & ) [protected], [default]
```

9.101.2.4 IPluginChecker() [3/3]

```
nvinfer1::consistency::IPluginChecker::IPluginChecker (
    IPluginChecker && ) [protected], [default]
```

9.101.3 Member Function Documentation**9.101.3.1 operator=() [1/2]**

```
IPluginChecker & nvinfer1::consistency::IPluginChecker::operator= (
    IPluginChecker && ) & [protected], [default]
```

9.101.3.2 operator=() [2/2]

```
IPluginChecker & nvinfer1::consistency::IPluginChecker::operator= (
    IPluginChecker const & ) & [protected], [default]
```

9.101.3.3 validate()

```
virtual bool nvinfer1::consistency::IPluginChecker::validate (
    char const * name,
    void const * serialData,
    size_t serialLength,
    PluginTensorDesc const * in,
    size_t nbInputs,
    PluginTensorDesc const * out,
    size_t nbOutputs,
    int64_t workspaceSize ) const [pure virtual], [noexcept]
```

Called during [IConsistencyChecker::validate](#). Allows users to provide custom validation of serialized Plugin data. Returns boolean that indicates whether or not the Plugin passed validation.

Parameters

<i>name</i>	The plugin name
<i>serialData</i>	The memory that holds the plugin serialized data.
<i>serialLength</i>	The size of the plugin serialized data.
<i>in</i>	The input tensors attributes.
<i>nbInputs</i>	The number of input tensors.
<i>out</i>	The output tensors attributes.
<i>nbOutputs</i>	The number of output tensors.
<i>workspaceSize</i>	The size of workspace provided during enqueue.

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

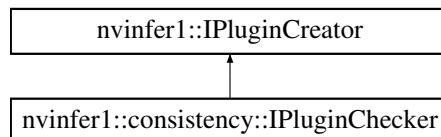
- [NvInferConsistency.h](#)

9.102 nvinfer1::IPluginCreator Class Reference

Plugin creator class for user implemented layers.

```
#include <NvInferRuntimeCommon.h>
```

Inheritance diagram for nvinfer1::IPluginCreator:



Public Member Functions

- virtual int32_t [getTensorRTVersion](#) () const noexcept
Return the version of the API the plugin creator was compiled with.
- 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 needs 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
- virtual `~IPluginCreator ()`=default

9.102.1 Detailed Description

Plugin creator class for user implemented layers.

See also

`IPlugin` and `IPluginFactory`

9.102.2 Constructor & Destructor Documentation

9.102.2.1 IPluginCreator()

```
nvinfer1::IPluginCreator::IPluginCreator ( ) [default]
```

9.102.2.2 ~IPluginCreator()

```
virtual nvinfer1::IPluginCreator::~~IPluginCreator ( ) [virtual], [default]
```

9.102.3 Member Function Documentation

9.102.3.1 createPlugin()

```
virtual IPluginV2 * nvinfer1::IPluginCreator::createPlugin (
    AsciiChar const * name,
    PluginFieldCollection const * fc ) [pure virtual], [noexcept]
```

Return a plugin object. Return nullptr in case of error.

Usage considerations

- 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.

9.102.3.2 deserializePlugin()

```
virtual IPluginV2 * nvinfer1::IPluginCreator::deserializePlugin (
    AsciiChar const * name,
    void const * serialData,
    size_t serialLength ) [pure virtual], [noexcept]
```

Called during deserialization of plugin layer. Return a plugin object.

Usage considerations

- 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.

9.102.3.3 getFieldNames()

```
virtual PluginFieldCollection const * nvinfer1::IPluginCreator::getFieldNames ( ) [pure virtual],
[noexcept]
```

Return a list of fields that needs to be passed to createPlugin.

See also

[PluginFieldCollection](#)

Usage considerations

- 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.

9.102.3.4 getPluginName()

```
virtual AsciiChar const * nvinfer1::IPluginCreator::getPluginName ( ) const [pure virtual], [noexcept]
```

Return the plugin name.

Warning

The string returned must be 1024 bytes or less including the NULL terminator and must be NULL terminated.

Usage considerations

- 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.

9.102.3.5 getPluginNamespace()

```
virtual AsciiChar const * nvinfer1::IPluginCreator::getPluginNamespace ( ) const [pure virtual],  
[noexcept]
```

Return the namespace of the plugin creator object.

Warning

The string returned must be 1024 bytes or less including the NULL terminator and must be NULL terminated.

Usage considerations

- 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.

9.102.3.6 getPluginVersion()

```
virtual AsciiChar const * nvinfer1::IPluginCreator::getPluginVersion ( ) const [pure virtual],  
[noexcept]
```

Return the plugin version.

Warning

The string returned must be 1024 bytes or less including the NULL terminator and must be NULL terminated.

Usage considerations

- 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.

9.102.3.7 getTensorRTVersion()

```
virtual int32_t nvinfer1::IPluginCreator::getTensorRTVersion ( ) const [inline], [virtual], [noexcept]
```

Return the version of the API the plugin creator was compiled with.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, the implementation provided here is safe to call from any thread.

9.102.3.8 setPluginNamespace()

```
virtual void nvinfer1::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.

See also

[IPluginRegistry::registerCreator\(\)](#)

Usage considerations

- 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:

- [NvInferRuntimeCommon.h](#)

9.103 nvcaffeparser1::IPluginFactoryV2 Class Reference

Plugin factory used to configure plugins.

```
#include <NvCaffeParser.h>
```

Public Member Functions

- virtual bool [isPluginV2](#) (char const *layerName) noexcept=0
A user implemented function that determines if a layer configuration is provided by an IPluginV2.
- virtual [nvinfer1::IPluginV2](#) * [createPlugin](#) (char const *layerName, [nvinfer1::Weights](#) const *weights, int32_t nbWeights, char const *libNamespace="") noexcept=0
Creates a plugin.
- virtual [~IPluginFactoryV2](#) () noexcept=default

9.103.1 Detailed Description

Plugin factory used to configure plugins.

9.103.2 Constructor & Destructor Documentation

9.103.2.1 ~IPluginFactoryV2()

```
virtual nvcaffeparser1::IPluginFactoryV2::~~IPluginFactoryV2 ( ) [virtual], [default], [noexcept]
```

9.103.3 Member Function Documentation

9.103.3.1 createPlugin()

```
virtual nvinfer1::IPluginV2 * nvcaffeparser1::IPluginFactoryV2::createPlugin (
    char const * layerName,
    nvinfer1::Weights const * weights,
    int32_t nbWeights,
    char const * libNamespace = "" ) [pure virtual], [noexcept]
```

Creates a plugin.

Parameters

<i>layerName</i>	Name of layer associated with the plugin.
<i>weights</i>	Weights used for the layer.
<i>nbWeights</i>	Number of weights.
<i>libNamespace</i>	Library Namespace associated with the plugin object

9.103.3.2 isPluginV2()

```
virtual bool nvcaffeparser1::IPluginFactoryV2::isPluginV2 (
    char const * layerName ) [pure virtual], [noexcept]
```

A user implemented function that determines if a layer configuration is provided by an IPluginV2.

Parameters

<i>layerName</i>	Name of the layer which the user wishes to validate.
------------------	--

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

- [NvCaffeParser.h](#)

9.104 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 bool [registerCreator](#) ([IPluginCreator](#) &creator, [AsciiChar](#) const *const pluginNamespace) noexcept=0
Register a plugin creator. Returns false if one with same type is already registered.
- virtual [IPluginCreator](#) *const * [getPluginCreatorList](#) (int32_t *const numCreators) const noexcept=0
Return all the registered plugin creators and the number of registered plugin creators. Returns nullptr if none found.
- virtual [IPluginCreator](#) * [getPluginCreator](#) ([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 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 [PluginLibraryHandle](#) [deserializeLibrary](#) (void const *pluginBuffer, size_t size) noexcept=0
Load and register a shared library of plugins from a memory buffer.
- 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 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 [deregisterCreator](#) ([IPluginCreator](#) const &creator) noexcept=0
Deregister a previously registered plugin creator.

Protected Member Functions

- virtual [~IPluginRegistry](#) () noexcept=default

9.104.1 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.

9.104.2 Member Typedef Documentation

9.104.2.1 PluginLibraryHandle

using `nvinfer1::IPluginRegistry::PluginLibraryHandle` = void*

Pointer for plugin library handle.

9.104.3 Constructor & Destructor Documentation

9.104.3.1 ~IPluginRegistry()

```
virtual nvinfer1::IPluginRegistry::~~IPluginRegistry ( ) [protected], [virtual], [default], [noexcept]
```

9.104.4 Member Function Documentation

9.104.4.1 deregisterCreator()

```
virtual bool nvinfer1::IPluginRegistry::deregisterCreator (
    IPluginCreator 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.

Usage considerations

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

9.104.4.2 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

<i>handle</i>	the plugin library handle to deregister.
---------------	--

9.104.4.3 deserializeLibrary()

```
virtual PluginLibraryHandle nvinfer1::IPluginRegistry::deserializeLibrary (
    void const * pluginBuffer,
    size_t size ) [pure virtual], [noexcept]
```

Load and register a shared library of plugins from a memory buffer.

Parameters

<i>pluginBuffer</i>	A pointer to a plugin buffer to deserialize.
<i>size</i>	size of the buffer.

Returns

The loaded plugin library handle. The call will fail and return nullptr if any of the plugins are already registered.

9.104.4.4 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\(\)](#)

Usage considerations

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

9.104.4.5 getPluginCreator()

```
virtual IPluginCreator * nvinfer1::IPluginRegistry::getPluginCreator (
    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.

Warning

The strings pluginName, pluginVersion, and pluginNamespace must be 1024 bytes or less including the NULL terminator and must be NULL terminated.

Usage considerations

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

9.104.4.6 getPluginCreatorList()

```
virtual IPluginCreator *const * nvinfer1::IPluginRegistry::getPluginCreatorList (
    int32_t *const numCreators ) const [pure virtual], [noexcept]
```

Return all the registered plugin creators and the number of registered plugin creators. Returns nullptr if none found.

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.104.4.7 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](#)

9.104.4.8 loadLibrary()

```
virtual PluginLibraryHandle nvinfer1::IPluginRegistry::loadLibrary (  
    AsciiChar const * pluginPath )    [pure virtual], [noexcept]
```

Load and register a shared library of plugins.

Parameters

<i>pluginPath</i>	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.

9.104.4.9 registerCreator()

```
virtual bool nvinfer1::IPluginRegistry::registerCreator (
    IPluginCreator & creator,
    AsciiChar const *const pluginNamespace ) [pure virtual], [noexcept]
```

Register a plugin creator. Returns false if one with same type is already registered.

Warning

The string pluginNamespace must be 1024 bytes or less including the NULL terminator and must be NULL terminated.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes; calls to this method will be synchronized by a mutex.

9.104.4.10 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

<i>recorder</i>	The error recorder to register with this interface.
-----------------	---

See also

[getErrorRecorder\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.104.4.11 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

<i>enabled</i>	The bool variable indicating whether parent search is enabled.
----------------	--

See also

[isParentSearchEnabled](#)

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

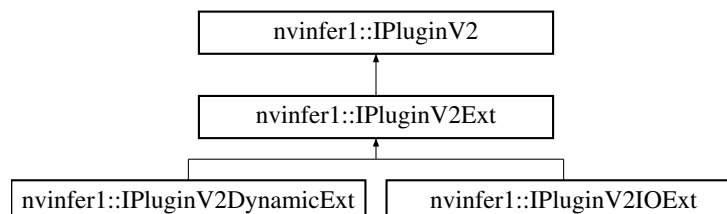
- [NvInferRuntimeCommon.h](#)

9.105 nvinfer1::IPluginV2 Class Reference

Plugin class for user-implemented layers.

```
#include <NvInferRuntimeCommon.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.
- 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 should have the same namespace.
- virtual [AsciiChar](#) const * [getPluginNamespace](#) () const noexcept=0
Return the namespace of the plugin object.

9.105.1 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 [IPluginV2DynamicExt](#) or [IPluginV2IOExt](#) depending on your requirement.

9.105.2 Member Function Documentation

9.105.2.1 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 should be created in an initialized state.

Usage considerations

- 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](#).

9.105.2.2 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

<i>inputDims</i>	The input tensor dimensions.
<i>nbInputs</i>	The number of inputs.
<i>outputDims</i>	The output tensor dimensions.
<i>nbOutputs</i>	The number of outputs.
<i>type</i>	The data type selected for the engine.
<i>format</i>	The format selected for the engine.
<i>maxBatchSize</i>	The maximum batch size.

The dimensions passed here do not include the outermost batch size (i.e. for 2-D 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\(\)](#)

Usage considerations

- 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](#).

9.105.2.3 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.

Usage considerations

- 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.

9.105.2.4 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

<i>batchSize</i>	The number of inputs in the batch.
<i>inputs</i>	The memory for the input tensors.
<i>outputs</i>	The memory for the output tensors.
<i>workspace</i>	Workspace for execution.
<i>stream</i>	The stream in which to execute the kernels.

Returns

0 for success, else non-zero (which will cause engine termination).

Usage considerations

- 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.

9.105.2.5 getNbOutputs()

```
virtual int32_t nvinfer1::IPluginV2::getNbOutputs ( ) const [pure virtual], [noexcept]
```

Get the number of outputs from the layer.

Returns

The number of outputs.

This function is called by the implementations of [INetworkDefinition](#) and [IBuilder](#). In particular, it is called prior to any call to [initialize\(\)](#).

Usage considerations

- 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.

9.105.2.6 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

<i>index</i>	The index of the output tensor.
<i>inputs</i>	The input tensors.
<i>nbInputDims</i>	The number of input tensors.

This function is called by the implementations of [INetworkDefinition](#) and [IBuilder](#). In particular, it is called prior to any call to [initialize\(\)](#).

Usage considerations

- 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 should 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.

9.105.2.7 getPluginNamespace()

```
virtual AsciiChar const * nvinfer1::IPluginV2::getPluginNamespace ( ) const [pure virtual], [noexcept]
```

Return the namespace of the plugin object.

Usage considerations

- 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.

9.105.2.8 `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 1024 bytes or less including the NULL terminator and must be NULL terminated.

Usage considerations

- 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.

9.105.2.9 `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 1024 bytes or less including the NULL terminator and must be NULL terminated.

Usage considerations

- 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.

9.105.2.10 getSerializationSize()

```
virtual size_t nvinfer1::IPluginV2::getSerializationSize ( ) const [pure virtual], [noexcept]
```

Find the size of the serialization buffer required.

Returns

The size of the serialization buffer.

Usage considerations

- 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.

9.105.2.11 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.

Usage considerations

- 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](#).

9.105.2.12 `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 should be sufficient for any batch size up to the maximum.

Returns

The workspace size.

Usage considerations

- 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.

9.105.2.13 `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).

Usage considerations

- 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.

9.105.2.14 `serialize()`

```
virtual void nvinfer1::IPluginV2::serialize (
    void * buffer ) const [pure virtual], [noexcept]
```

Serialize the layer.

Parameters

<i>buffer</i>	A pointer to a buffer to serialize data. Size of buffer must be equal to value returned by <code>getSerializationSize</code> .
---------------	--

See also

[getSerializationSize\(\)](#)

Usage considerations

- 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.

9.105.2.15 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 should have the same namespace.

Parameters

<i>pluginNamespace</i>	The namespace for the plugin object.
------------------------	--------------------------------------

Warning

The string `pluginNamespace` must be 1024 bytes or less including the NULL terminator and must be NULL terminated.

Usage considerations

- 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.

9.105.2.16 supportsFormat()

```
virtual bool nvinfer1::IPluginV2::supportsFormat (
    DataType type,
    PluginFormat format ) const [pure virtual], [noexcept]
```

Check format support.

Parameters

<i>type</i>	DataType requested.
<i>format</i>	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::kCHW32` will not be passed in, this is to keep backward compatibility with TensorRT 5.x series. Use `PluginFormat::kV2IOExt` or `PluginFormat::kV2DynamicExt` for other PluginFormats.

`DataType::kBOOL` and `DataType::kUINT8` are not supported.

Usage considerations

- 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.

9.105.2.17 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\(\)](#)

Usage considerations

- 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.

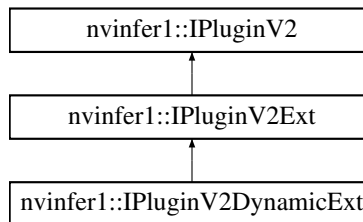
The documentation for this class was generated from the following file:

- [NvInferRuntimeCommon.h](#)

9.106 nvinfer1::IPluginV2DynamicExt Class Reference

```
#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 should also be pre-configured. The returned object should 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

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

9.106.1 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 nvinfer1::DataType getOutputDataType(int32_t index, nvinfer1::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 [getOutputDataType](#), 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](#). Details about the floating-point precision are elicited later by method [supportsFormatCombination](#).

9.106.2 Constructor & Destructor Documentation

9.106.2.1 [~IPluginV2DynamicExt](#)()

```
virtual nvinfer1::IPluginV2DynamicExt::~IPluginV2DynamicExt ( ) [inline], [protected], [virtual], [noexcept]
```

9.106.3 Member Function Documentation

9.106.3.1 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 should also be pre-configured. The returned object should 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.

Usage considerations

- 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](#).

9.106.3.2 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 batch size is changed from previous call of execute()/enqueue() if hasImplicitBatchDimension() returns true.
 - The optimization profile is changed via setOptimizationProfile() or setOptimizationProfileAsync().
 - An input shape binding is changed via setInputShapeBinding().
 - An input execution binding is changed via setBindingDimensions().

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

<i>in</i>	The input tensors attributes that are used for configuration.
<i>nbInputs</i>	Number of input tensors.
<i>out</i>	The output tensors attributes that are used for configuration.
<i>nbOutputs</i>	Number of output tensors.

9.106.3.3 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

<i>inputDesc</i>	how to interpret the memory for the input tensors.
<i>outputDesc</i>	how to interpret the memory for the output tensors.
<i>inputs</i>	The memory for the input tensors.
<i>outputs</i>	The memory for the output tensors.
<i>workspace</i>	Workspace for execution.
<i>stream</i>	The stream in which to execute the kernels.

Returns

0 for success, else non-zero (which will cause engine termination).

9.106.3.4 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

<i>outputIndex</i>	The index of the output tensor
<i>inputs</i>	Expressions for dimensions of the input tensors
<i>nbInputs</i>	The number of input tensors
<i>exprBuilder</i>	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 `getOutputDimensions` can be:

```
DimsExprs output(inputs[0]);
output.d[0] = exprBuilder.operation(DimensionOperation::kSUM, *inputs[0].d[0], *inputs[1].d[0]);
return output;
```

9.106.3.5 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](#).

9.106.3.6 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.

9.106.3.7 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 $0 \leq \text{pos} < \text{nbInputs} + \text{nbOutputs} - 1$.

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[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.

9.106.4 Member Data Documentation

9.106.4.1 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:

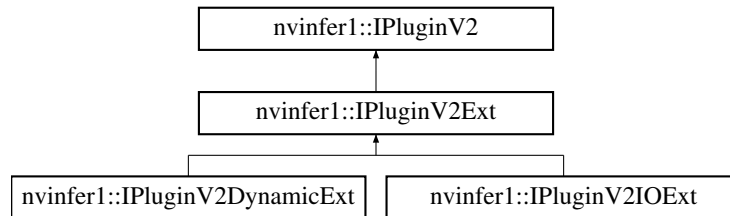
- [NvInferRuntime.h](#)

9.107 nvinfer1::IPluginV2Ext Class Reference

Plugin class for user-implemented layers.

```
#include <NvInferRuntimeCommon.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 bool [isOutputBroadcastAcrossBatch](#) (int32_t outputIndex, bool const *inputIsBroadcasted, int32_t nbInputs) const noexcept=0
Return true if output tensor is broadcast across a batch.
- virtual bool [canBroadcastInputAcrossBatch](#) (int32_t inputIndex) const noexcept=0
Return true if plugin can use input that is broadcast across batch without replication.
- 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 resource.
- 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 should also be pre-configured. The returned object should 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 should not implement this. In a C++11 API it would be override final.

9.107.1 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 batch.

See also

[IPluginV2](#)

Deprecated Deprecated in TensorRT 8.5. Implement `IPluginV2DynamicExt` or `IPluginV2IOExt` depending on your requirement.

9.107.2 Constructor & Destructor Documentation

9.107.2.1 IPluginV2Ext()

```
nvinfer1::IPluginV2Ext::IPluginV2Ext ( ) [default]
```

9.107.2.2 ~IPluginV2Ext()

```
nvinfer1::IPluginV2Ext::~~IPluginV2Ext ( ) [override], [default]
```

9.107.3 Member Function Documentation

9.107.3.1 attachToContext()

```
virtual void nvinfer1::IPluginV2Ext::attachToContext (
    cudnnContext * ,
    cublasContext * ,
    IAllocator * ) [inline], [virtual], [noexcept]
```

Attach the plugin object to an execution context and grant the plugin the access to some context resource.

Parameters

<i>cudnn</i>	The CUDNN context handle of the execution context
<i>cublas</i>	The cublas context handle of the execution context
<i>allocator</i>	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. The plugin can also get context-owned CUDNN and CUBLAS context here.

Note

In the automotive safety context, the CUDNN and CUBLAS parameters will be nullptr because CUDNN and CUBLAS is not used by the safe runtime.

Usage considerations

- 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.

9.107.3.2 canBroadcastInputAcrossBatch()

```
virtual bool nvinfer1::IPluginV2Ext::canBroadcastInputAcrossBatch (
    int32_t inputIndex ) const [pure virtual], [noexcept]
```

Return true if plugin can use input that is broadcast across batch without replication.

Parameters

<i>inputIndex</i>	Index of input that could be broadcast.
-------------------	---

For each input whose tensor is semantically broadcast across a batch, TensorRT calls this method before calling `configurePlugin`. If `canBroadcastInputAcrossBatch` returns true, TensorRT will not replicate the input tensor; i.e., there will be a single copy that the plugin should share across the batch. If it returns false, TensorRT will replicate the input tensor so that it appears like a non-broadcasted tensor.

This method is called only for inputs that can be broadcast.

Usage considerations

- 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.

9.107.3.3 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 should also be pre-configured. The returned object should 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.

Usage considerations

- 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`.

9.107.3.4 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

<i>inputDims</i>	The input tensor dimensions.
<i>nbInputs</i>	The number of inputs.
<i>outputDims</i>	The output tensor dimensions.
<i>nbOutputs</i>	The number of outputs.
<i>inputTypes</i>	The data types selected for the plugin inputs.
<i>outputTypes</i>	The data types selected for the plugin outputs.
<i>inputIsBroadcast</i>	True for each input that the plugin must broadcast across the batch.
<i>outputIsBroadcast</i>	True for each output that TensorRT will broadcast across the batch.
<i>floatFormat</i>	The format selected for the engine for the floating point inputs/outputs.
<i>maxBatchSize</i>	The maximum batch size.

The dimensions passed here do not include the outermost batch size (i.e. for 2-D image networks, they will be 3-dimensional CHW dimensions). When *inputIsBroadcast* or *outputIsBroadcast* is true, the outermost batch size for that input or output should be treated as if it is one. *inputIsBroadcast*[i] is true only if the input is semantically broadcast across the batch and *canBroadcastInputAcrossBatch*(i) returned true. *outputIsBroadcast*[i] is true only if *isOutputBroadcastAcrossBatch*(i) returns true.

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*.

Usage considerations

- 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.

9.107.3.5 `configureWithFormat()`

```
void nvinfer1::IPluginV2Ext::configureWithFormat (
    Dims const * ,
    int32_t ,
    Dims const * ,
    int32_t ,
    DataType ,
    PluginFormat ,
    int32_t ) [inline], [override], [protected], [virtual], [noexcept]
```

Derived classes should not implement this. In a C++11 API it would be `override final`.

Implements [nvinfer1::IPluginV2](#).

9.107.3.6 `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 a 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.

Usage considerations

- 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.

9.107.3.7 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.

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.

See also

[supportsFormat\(\)](#)

Warning

`DataType::kBOOL` and `DataType::kUINT8` are not supported.

Usage considerations

- 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.

9.107.3.8 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`.

Do not override this method as it is used by the TensorRT library to maintain backwards-compatibility with plugins.

Usage considerations

- 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`.

9.107.3.9 isOutputBroadcastAcrossBatch()

```
virtual bool nvinfer1::IPluginV2Ext::isOutputBroadcastAcrossBatch (
    int32_t outputIndex,
    bool const * inputIsBroadcasted,
    int32_t nbInputs ) const [pure virtual], [noexcept]
```

Return true if output tensor is broadcast across a batch.

Parameters

<i>outputIndex</i>	The index of the output
<i>inputIsBroadcasted</i>	The ith element is true if the tensor for the ith input is broadcast across a batch.
<i>nbInputs</i>	The number of inputs

The values in `inputIsBroadcasted` refer to broadcasting at the semantic level, i.e. are unaffected by whether method `canBroadcastInputAcrossBatch` requests physical replication of the values.

Usage considerations

- 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:

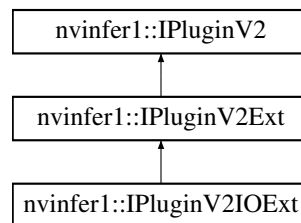
- [NvInferRuntimeCommon.h](#)

9.108 nvinfer1::IPluginV2IOExt Class Reference

Plugin class for user-implemented layers.

```
#include <NvInferRuntimeCommon.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](#).

9.108.1 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](#)

9.108.2 Member Function Documentation

9.108.2.1 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

<i>in</i>	The input tensors attributes that are used for configuration.
<i>nbInput</i>	Number of input tensors.
<i>out</i>	The output tensors attributes that are used for configuration.
<i>nbOutput</i>	Number of output tensors.

Usage considerations

- 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.

9.108.2.2 `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.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, the implementation provided here is safe to call from any thread.

Reimplemented from [nvinfer1::IPluginV2Ext](#).

9.108.2.3 `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 $0 \leq \text{pos} < \text{nbInputs} + \text{nbOutputs} - 1$.

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[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.

Usage considerations

- 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:

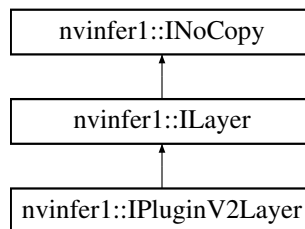
- [NvInferRuntimeCommon.h](#)

9.109 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=default

Protected Attributes

- apiv::VPluginV2Layer * [mImpl](#)

9.109.1 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.

9.109.2 Constructor & Destructor Documentation

9.109.2.1 ~IPluginV2Layer()

```
virtual nvinfer1::IPluginV2Layer::~~IPluginV2Layer ( ) [protected], [virtual], [default], [noexcept]
```

9.109.3 Member Function Documentation

9.109.3.1 getPlugin()

```
IPluginV2 & nvinfer1::IPluginV2Layer::getPlugin ( ) [inline], [noexcept]
```

Get the plugin for the layer.

See also

[IPluginV2](#)

9.109.4 Member Data Documentation

9.109.4.1 mImpl

```
apiv::VPluginV2Layer* nvinfer1::IPluginV2Layer::mImpl [protected]
```

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

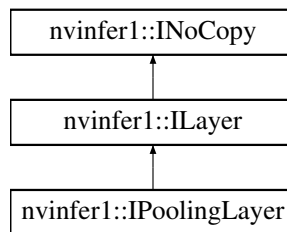
- [NvInfer.h](#)

9.110 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.
- [TRT_DEPRECATED](#) void [setWindowSize](#) ([DimsHW](#) windowSize) noexcept
Set the window size for pooling.
- [TRT_DEPRECATED](#) [DimsHW](#) [getWindowSize](#) () const noexcept
Get the window size for pooling.
- [TRT_DEPRECATED](#) void [setStride](#) ([DimsHW](#) stride) noexcept
Set the stride for pooling.
- [TRT_DEPRECATED](#) [DimsHW](#) [getStride](#) () const noexcept
Get the stride for pooling.
- [TRT_DEPRECATED](#) void [setPadding](#) ([DimsHW](#) padding) noexcept
Set the padding for pooling.

- [TRT_DEPRECATED DimsHW](#) [getPadding \(\)](#) const noexcept
Get the padding for pooling.
- void [setBlendFactor](#) (float blendFactor) noexcept
*Set the blending factor for the max_average_blend mode: $\text{max_average_blendPool} = (1 - \text{blendFactor}) * \text{maxPool} + \text{blendFactor} * \text{avgPool}$ 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: $\text{max_average_blendPool} = (1 - \text{blendFactor}) * \text{maxPool} + \text{blendFactor} * \text{avgPool}$ 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](#) padding) noexcept
Set the multi-dimension pre-padding for pooling.
- [Dims](#) [getPrePadding \(\)](#) const noexcept
Get the pre-padding.
- void [setPostPadding](#) ([Dims](#) 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](#) 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](#) stride) noexcept
Set the multi-dimension stride for pooling.
- [Dims](#) [getStrideNd \(\)](#) const noexcept
Get the multi-dimension stride for pooling.
- void [setPaddingNd](#) ([Dims](#) 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=default

Protected Attributes

- apiv::VPoolingLayer * [mImpl](#)

9.110.1 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.

9.110.2 Constructor & Destructor Documentation

9.110.2.1 ~IPoolingLayer()

```
virtual nvinfer1::IPoolingLayer::~~IPoolingLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.110.3 Member Function Documentation

9.110.3.1 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\(\)](#)

9.110.3.2 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_BLEND`, `blendFactor` is ignored.

See also

[setBlendFactor\(\)](#)

9.110.3.3 `getPadding()`

```
TRT_DEPRECATED DimsHW nvinfer1::IPoolingLayer::getPadding ( ) const [inline], [noexcept]
```

Get the padding for pooling.

Default: 0

See also

[setPadding\(\)](#)

Deprecated Superseded by `getPaddingNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.110.3.4 `getPaddingMode()`

```
PaddingMode nvinfer1::IPoolingLayer::getPaddingMode ( ) const [inline], [noexcept]
```

Get the padding mode.

Default: `kEXPLICIT_ROUND_DOWN`

See also

[setPaddingMode\(\)](#)

9.110.3.5 `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\(\)](#)

9.110.3.6 getPoolingType()

```
PoolingType nvinfer1::IPoolingLayer::getPoolingType ( ) const [inline], [noexcept]
```

Get the type of activation to be performed.

See also

[setPoolingType\(\)](#), [PoolingType](#)

9.110.3.7 getPostPadding()

```
Dims nvinfer1::IPoolingLayer::getPostPadding ( ) const [inline], [noexcept]
```

Get the padding.

See also

[setPostPadding\(\)](#)

9.110.3.8 getPrePadding()

```
Dims nvinfer1::IPoolingLayer::getPrePadding ( ) const [inline], [noexcept]
```

Get the pre-padding.

See also

[setPrePadding\(\)](#)

9.110.3.9 getStride()

```
TRT_DEPRECATED DimsHW nvinfer1::IPoolingLayer::getStride ( ) const [inline], [noexcept]
```

Get the stride for pooling.

See also

[setStride\(\)](#)

Deprecated Superseded by `getStrideNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.110.3.10 getStrideNd()

```
Dims nvinfer1::IPoolingLayer::getStrideNd ( ) const [inline], [noexcept]
```

Get the multi-dimension stride for pooling.

See also

[setStrideNd\(\)](#)

9.110.3.11 getWindowSize()

```
TRT_DEPRECATED DimsHW nvinfer1::IPoolingLayer::getWindowSize ( ) const [inline], [noexcept]
```

Get the window size for pooling.

See also

[setWindowSize\(\)](#)

Deprecated Superseded by `getWindowSizeNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.110.3.12 getWindowSizeNd()

```
Dims nvinfer1::IPoolingLayer::getWindowSizeNd ( ) const [inline], [noexcept]
```

Get the multi-dimension window size for pooling.

See also

[setWindowSizeNd\(\)](#)

9.110.3.13 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

Note

On Xavier, DLA supports only inclusive padding and this must be explicitly set to false.

See also

[getAverageCountExcludesPadding\(\)](#)

9.110.3.14 setBlendFactor()

```
void nvinfer1::IPoolingLayer::setBlendFactor (
    float blendFactor ) [inline], [noexcept]
```

Set the blending factor for the max_average_blend mode: $\text{max_average_blendPool} = (1 - \text{blendFactor}) * \text{maxPool} + \text{blendFactor} * \text{avgPool}$ 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\(\)](#)

9.110.3.15 setPadding()

```
TRT_DEPRECATED void nvinfer1::IPoolingLayer::setPadding (
    DimsHW padding ) [inline], [noexcept]
```

Set the padding for pooling.

Default: 0

If executing this layer on DLA, both height and width of padding must be in the range [0,7].

See also

[getPadding\(\)](#)

Deprecated Superseded by setPaddingNd. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.110.3.16 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\(\)](#)

9.110.3.17 setPaddingNd()

```
void nvinfer1::IPoolingLayer::setPaddingNd (
    Dims 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\(\)](#)

9.110.3.18 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](#)

9.110.3.19 setPostPadding()

```
void nvinfer1::IPoolingLayer::setPostPadding (
    Dims 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\(\)](#)

9.110.3.20 setPrePadding()

```
void nvinfer1::IPoolingLayer::setPrePadding (
    Dims 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\(\)](#)

9.110.3.21 setStride()

```
TRT_DEPRECATED void nvinfer1::IPoolingLayer::setStride (
    DimsHW stride ) [inline], [noexcept]
```

Set the stride for pooling.

Default: 1

If executing this layer on DLA, both height and width of stride must be in the range [1,16].

See also

[getStride\(\)](#)

Deprecated Superseded by `setStrideNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.110.3.22 setStrideNd()

```
void nvinfer1::IPoolingLayer::setStrideNd (
    Dims 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\(\)](#) [setStride\(\)](#) [getStride\(\)](#)

9.110.3.23 setWindowSize()

```
TRT_DEPRECATED void nvinfer1::IPoolingLayer::setWindowSize (
    DimsHW windowSize ) [inline], [noexcept]
```

Set the window size for pooling.

If executing this layer on DLA, both height and width of window size must be in the range [1,8].

See also

[getWindowSize\(\)](#)

Deprecated Superseded by `setWindowSizeNd`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

9.110.3.24 setWindowSizeNd()

```
void nvinfer1::IPoolingLayer::setWindowSizeNd (
    Dims 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\(\)](#)

9.110.4 Member Data Documentation

9.110.4.1 mImpl

```
apiv::VPoolingLayer* nvinfer1::IPoolingLayer::mImpl [protected]
```

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

- [NvInfer.h](#)

9.111 nvinfer1::IProfiler Class Reference

Application-implemented interface for profiling.

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

Public Member Functions

- virtual void [reportLayerTime](#) (char const *layerName, float ms) noexcept=0
Layer time reporting callback.
- virtual [~IProfiler](#) () noexcept

9.111.1 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()/enqueueV2()/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.

9.111.2 Constructor & Destructor Documentation

9.111.2.1 ~IProfiler()

```
virtual nvinfer1::IProfiler::~~IProfiler ( ) [inline], [virtual], [noexcept]
```

9.111.3 Member Function Documentation

9.111.3.1 reportLayerTime()

```
virtual void nvinfer1::IProfiler::reportLayerTime (  
    char const * layerName,  
    float ms ) [pure virtual], [noexcept]
```

Layer time reporting callback.

Parameters

<i>layerName</i>	The name of the layer, set when constructing the network definition.
<i>ms</i>	The time in milliseconds to execute the layer.

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

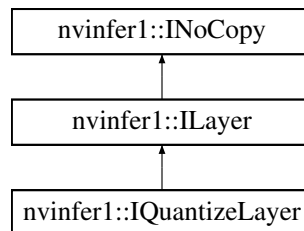
- [NvInferRuntime.h](#)

9.112 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.

Protected Member Functions

- `virtual ~IQuantizeLayer () noexcept=default`

Protected Attributes

- `apiv::VQuantizeLayer * mImpl`

9.112.1 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 to an 8-bit signed integer according to: $\text{output} = \text{clamp}(\text{round}(\text{input} / \text{scale}) + \text{zeroPt})$

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 `zeroPt` 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 `DataType::kINT8`. `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-channel quantization. All `scale` coefficients must have positive values. 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 `DataType::kINT8`. The input type, if constrained, must be constrained to `DataType::kFLOAT` or `DataType::kHALF`. The output size is the same as the input size. The quantization axis is in reference to the input tensor's dimensions.

`IQuantizeLayer` only supports `DataType::kFLOAT` precision and will default to this precision during instantiation. `IQuantizeLayer` only supports `DataType::kINT8` 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`: $\text{output}[n,c,h,w] = \text{clamp}(\text{round}(\text{input}[n,c,h,w] / \text{scale}) + \text{zeroPt})$

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`: $\text{output}[k,c,r,s] = \text{clamp}(\text{round}(\text{input}[k,c,r,s] / \text{scale}[k]) + \text{zeroPt}[k])$

Note

Only symmetric quantization is supported.

Currently the only allowed build-time constant `scale` and `\zeroPt` subgraphs are:

1. Constant -> Quantize
2. Constant -> Cast -> Quantize

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 ~IQuantizeLayer()

```
virtual nvinfer1::IQuantizeLayer::~~IQuantizeLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.112.3 Member Function Documentation

9.112.3.1 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.

9.112.3.2 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 will be ignored if the scale tensor has exactly one coefficient (per-tensor quantization).

9.112.4 Member Data Documentation

9.112.4.1 mImpl

```
apiv::VQuantizeLayer* nvinfer1::IQuantizeLayer::mImpl [protected]
```

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

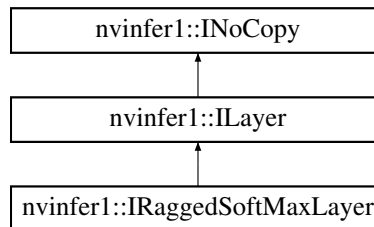
- [NvInfer.h](#)

9.113 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=default

Protected Attributes

- apiv::VRaggedSoftMaxLayer * [mImpl](#)

Additional Inherited Members

9.113.1 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.

9.113.2 Constructor & Destructor Documentation

9.113.2.1 ~IRaggedSoftMaxLayer()

```
virtual nvinfer1::IRaggedSoftMaxLayer::~~IRaggedSoftMaxLayer ( ) [protected], [virtual], [default],
[noexcept]
```

9.113.3 Member Data Documentation

9.113.3.1 mImpl

```
apiv::VRaggedSoftMaxLayer* nvinfer1::IRaggedSoftMaxLayer::mImpl [protected]
```

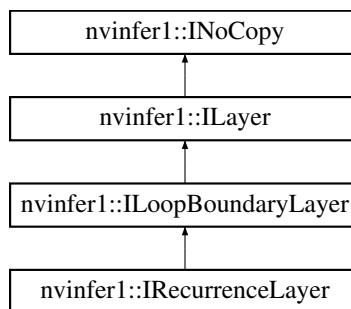
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.114 nvinfer1::IRecurrenceLayer Class Reference

```
#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=default

Protected Attributes

- `apiv::VRecurrenceLayer * mImpl`

9.114.1 Constructor & Destructor Documentation

9.114.1.1 ~IRecurrenceLayer()

```
virtual nvinfer1::IRecurrenceLayer::~~IRecurrenceLayer ( ) [protected], [virtual], [default],
[noexcept]
```

9.114.2 Member Function Documentation

9.114.2.1 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

<i>index</i>	the index of the input to modify.
<i>tensor</i>	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.

9.114.3 Member Data Documentation

9.114.3.1 mImpl

```
apiv::VRecurrenceLayer* nvinfer1::IRecurrenceLayer::mImpl [protected]
```

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

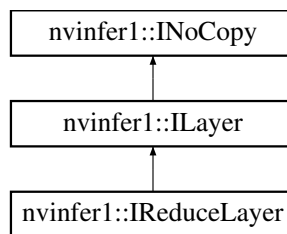
- [NvInfer.h](#)

9.115 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=default

Protected Attributes

- apiv::VReduceLayer * [mImpl](#)

9.115.1 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.

9.115.2 Constructor & Destructor Documentation

9.115.2.1 ~IReduceLayer()

```
virtual nvinfer1::IReduceLayer::~~IReduceLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.115.3 Member Function Documentation

9.115.3.1 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](#)

9.115.3.2 getOperation()

```
ReduceOperation nvinfer1::IReduceLayer::getOperation ( ) const [inline], [noexcept]
```

Get the reduce operation for the layer.

See also

[setOperation\(\)](#), [ReduceOperation](#)

9.115.3.3 getReduceAxes()

```
uint32_t nvinfer1::IReduceLayer::getReduceAxes ( ) const [inline], [noexcept]
```

Get the axes over which to reduce for the layer.

See also

[setReduceAxes](#)

9.115.3.4 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](#)

9.115.3.5 setOperation()

```
void nvinfer1::IReduceLayer::setOperation (
    ReduceOperation op ) [inline], [noexcept]
```

Set the reduce operation for the layer.

See also

[getOperation\(\)](#), [ReduceOperation](#)

9.115.3.6 setReduceAxes()

```
void nvinfer1::IReduceLayer::setReduceAxes (
    uint32_t reduceAxes ) [inline], [noexcept]
```

Set the axes over which to reduce.

See also

[getReduceAxes](#)

9.115.4 Member Data Documentation

9.115.4.1 mImpl

```
apiv::VReduceLayer* nvinfer1::IReduceLayer::mImpl [protected]
```

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

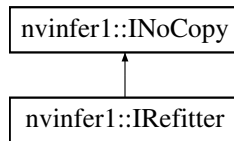
- [NvInfer.h](#)

9.116 nvinfer1::IRefitter Class Reference

Updates weights in an engine.

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

Inheritance diagram for nvinfer1::IRefitter:



Public Member Functions

- virtual [~IRefitter](#) () noexcept=default
- 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
Updates associated engine. Return true if successful.
- 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.
- [TRT_DEPRECATED](#) void [destroy](#) () noexcept
- bool [setDynamicRange](#) (char const *tensorName, float min, float max) noexcept
- float [getDynamicRangeMin](#) (char const *tensorName) const noexcept
Get minimum of dynamic range.
- float [getDynamicRangeMax](#) (char const *tensorName) const noexcept
Get maximum of dynamic range.
- int32_t [getTensorsWithDynamicRange](#) (int32_t size, char const **tensorNames) const noexcept

- *Get names of all tensors that have refittable dynamic ranges.*
- 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.

Protected Attributes

- apiv::VRefitter * [mImpl](#)

Additional Inherited Members

9.116.1 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.

9.116.2 Constructor & Destructor Documentation

9.116.2.1 ~IRefitter()

```
virtual nvinfer1::IRefitter::~~IRefitter ( ) [virtual], [default], [noexcept]
```

9.116.3 Member Function Documentation

9.116.3.1 destroy()

`TRT_DEPRECATED` void nvinfer1::IRefitter::destroy () [inline], [noexcept]

Deprecated Deprecated in TRT 8.0. Superseded by delete.

Warning

Calling destroy on a managed pointer will result in a double-free error.

9.116.3.2 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

<i>size</i>	The number of items that can be safely written to a non-null layerNames or roles.
<i>layerNames</i>	Where to write the layer names.
<i>roles</i>	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.

9.116.3.3 getAllWeights()

```
int32_t nvinfer1::IRefitter::getAllWeights (
    int32_t size,
    char const ** weightsNames ) [inline], [noexcept]
```

Get names of all weights that could be refit.

Parameters

<i>size</i>	The number of weights names that can be safely written to.
<i>weightsNames</i>	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.

9.116.3.4 getDynamicRangeMax()

```
float nvinfer1::IRefitter::getDynamicRangeMax (
    char const * tensorName ) const [inline], [noexcept]
```

Get maximum of dynamic range.

Returns

Maximum of dynamic range.

If the dynamic range was never set, returns the maximum computed during calibration.

Warning

The string *tensorName* must be null-terminated, and be at most 4096 bytes including the terminator.

9.116.3.5 getDynamicRangeMin()

```
float nvinfer1::IRefitter::getDynamicRangeMin (
    char const * tensorName ) const [inline], [noexcept]
```

Get minimum of dynamic range.

Returns

Minimum of dynamic range.

If the dynamic range was never set, returns the minimum computed during calibration.

Warning

The string *tensorName* must be null-terminated, and be at most 4096 bytes including the terminator.

9.116.3.6 `getErrorRecorder()`

```
ILoggerRecorder * 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 [ILoggerRecorder](#) object that has been registered.

See also

[setErrorRecorder\(\)](#)

9.116.3.7 `getLogger()`

```
ILogger * nvinfer1::IRefitter::getLogger ( ) const [inline], [noexcept]
```

get the logger with which the refitter was created

Returns

the logger

9.116.3.8 `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\(\)](#)

9.116.3.9 `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

<i>size</i>	The number of items that can be safely written to a non-null layerNames or roles.
<i>layerNames</i>	Where to write the layer names.
<i>roles</i>	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.

9.116.3.10 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

<i>size</i>	The number of weights names that can be safely written to.
<i>weightsNames</i>	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.

9.116.3.11 getTensorsWithDynamicRange()

```
int32_t nvinfer1::IRefitter::getTensorsWithDynamicRange (
    int32_t size,
    char const ** tensorNames ) const [inline], [noexcept]
```

Get names of all tensors that have refittable dynamic ranges.

Parameters

<i>size</i>	The number of items that can be safely written to a non-null tensorNames.
<i>tensorNames</i>	Where to write the layer names.

Returns

The number of [Weights](#) that could be refit.

If tensorNames!=nullptr, each written pointer points to a string owned by the engine being refit, and becomes invalid when the engine is destroyed.

9.116.3.12 refitCudaEngine()

```
bool nvinfer1::IRefitter::refitCudaEngine ( ) [inline], [noexcept]
```

Updates associated engine. Return true if successful.

Failure occurs if [getMissing\(\)](#) != 0 before the call.

The behavior is undefined if the engine has pending enqueued work.

Extant IExecutionContexts associated with the engine should not be used afterwards. Instead, create new IExecutionContexts after refitting.

9.116.3.13 setDynamicRange()

```
bool nvinfer1::IRefitter::setDynamicRange (
    char const * tensorName,
    float min,
    float max ) [inline], [noexcept]
```

Update dynamic range for a tensor.

Parameters

<i>tensorName</i>	The name of an ITensor in the network.
<i>min</i>	The minimum of the dynamic range for the tensor.
<i>max</i>	The maximum of the dynamic range for the tensor.

Returns

True if successful; false otherwise.

Returns false if there is no Int8 engine tensor derived from a network tensor of that name. If successful, then [getMissing](#) may report that some weights need to be supplied.

Warning

The string `tensorName` must be null-terminated, and be at most 4096 bytes including the terminator.

9.116.3.14 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

<i>recorder</i>	The error recorder to register with this interface.
-----------------	---

See also

[getErrorRecorder\(\)](#)

9.116.3.15 setMaxThreads()

```
bool nvinfer1::IRefitter::setMaxThreads (
    int32_t maxThreads ) [inline], [noexcept]
```

Set the maximum number of threads.

Parameters

<i>maxThreads</i>	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.

9.116.3.16 `setNamedWeights()`

```
bool nvinfer1::IRefitter::setNamedWeights (
    char const * name,
    Weights weights ) [inline], [noexcept]
```

Specify new weights of given name.

Parameters

<i>name</i>	The name of the weights to be refit.
<i>weights</i>	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 number of weights is inconsistent with the original specification.

Modifying the weights before method `refitCudaEngine()` completes will result in undefined behavior.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

9.116.3.17 `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 number of weights is inconsistent with the layer's original specification.

Modifying the weights before method `refit()` completes will result in undefined behavior.

Warning

The string `layerName` must be null-terminated, and be at most 4096 bytes including the terminator.

9.116.4 Member Data Documentation**9.116.4.1 mImpl**

```
apiv::VRefitter* nvinfer1::IRefitter::mImpl [protected]
```

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

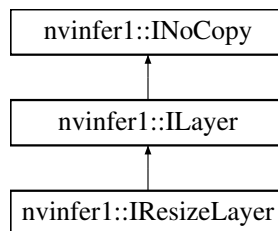
- [NvInferRuntime.h](#)

9.117 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](#) 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](#) ([ResizeMode](#) resizeMode) noexcept
Set resize mode for an input tensor.

- [ResizeMode](#) [getResizeMode](#) () const noexcept
Get resize mode for an input tensor.
- [TRT_DEPRECATED](#) void [setAlignCorners](#) (bool alignCorners) noexcept
Set whether to align corners while resizing.
- [TRT_DEPRECATED](#) bool [getAlignCorners](#) () const noexcept
True if align corners has been set.
- 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=default

Protected Attributes

- [apiv::VResizeLayer](#) * [mImpl](#)

9.117.1 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:

- `ResizeMode::kNEAREST` - resizes innermost m dimensions of N-D, where $0 < m \leq \min(8, N)$ and $N > 0$
- `ResizeMode::kLINEAR` - resizes innermost m dimensions of N-D, where $0 < m \leq \min(3, N)$ and $N > 0$

Default resize mode is `ResizeMode::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 `ResizeMode` 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 $\text{floor}(\text{input dimension} * \text{scale})$. 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.

9.117.2 Constructor & Destructor Documentation

9.117.2.1 ~IResizeLayer()

```
virtual nvinfer1::IResizeLayer::~~IResizeLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.117.3 Member Function Documentation

9.117.3.1 getAlignCorners()

```
TRT\_DEPRECATED bool nvinfer1::IResizeLayer::getAlignCorners ( ) const [inline], [noexcept]
```

True if align corners has been set.

Returns

True if align corners has been set, false otherwise.

Deprecated Deprecated in TensorRT 8.0. Superseded by [IResizeLayer::getCoordinateTransformation\(\)](#).

9.117.3.2 getCoordinateTransformation()

```
ResizeCoordinateTransformation nvinfer1::IResizeLayer::getCoordinateTransformation ( ) const [inline],  
[noexcept]
```

Get coordinate transformation function.

Returns

The coordinate transformation function.

9.117.3.3 getCubicCoeff()

```
float nvinfer1::IResizeLayer::getCubicCoeff ( ) const [inline], [noexcept]
```

Get the coefficient 'A' used in cubic interpolation.

See also

[setCubicCoeff\(\)](#)

9.117.3.4 getExcludeOutside()

```
bool nvinfer1::IResizeLayer::getExcludeOutside ( ) const [inline], [noexcept]
```

Get the state for excluding outside pixels.

See also

[setExcludeOutside\(\)](#)

9.117.3.5 getNearestRounding()

```
ResizeRoundMode nvinfer1::IResizeLayer::getNearestRounding ( ) const [inline], [noexcept]
```

Get rounding mode for nearest neighbor resize.

Returns

The rounding mode.

9.117.3.6 getOutputDimensions()

```
Dims nvinfer1::IResizeLayer::getOutputDimensions ( ) const [inline], [noexcept]
```

Get the output dimensions.

Returns

The output dimensions.

9.117.3.7 getResizeMode()

```
ResizeMode nvinfer1::IResizeLayer::getResizeMode ( ) const [inline], [noexcept]
```

Get resize mode for an input tensor.

Returns

The resize mode.

9.117.3.8 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

<i>size</i>	The number of scales to get. If <code>size != nbScales</code> , no scales will be copied.
<i>scales</i>	Pointer to where to copy the scales. Scales will be copied only if <code>size == nbScales</code> and <code>scales != nullptr</code> .

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.

9.117.3.9 `getSelectorForSinglePixel()`

```
ResizeSelector nvinfer1::IResizeLayer::getSelectorForSinglePixel ( ) const [inline], [noexcept]
```

Get the coordinate selector function when resized to single pixel.

Returns

The selector function.

9.117.3.10 `setAlignCorners()`

```
TRT_DEPRECATED void nvinfer1::IResizeLayer::setAlignCorners (
    bool alignCorners ) [inline], [noexcept]
```

Set whether to align corners while resizing.

If true, the centers of the 4 corner pixels of both input and output tensors are aligned i.e. preserves the values of corner pixels.

Default: false.

Deprecated Deprecated in TensorRT 8.0. Superseded by [IResizeLayer::setCoordinateTransformation\(\)](#).

9.117.3.11 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](#)

9.117.3.12 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.

9.117.3.13 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.

9.117.3.14 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

<i>index</i>	the index of the input to modify.
<i>tensor</i>	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 Int32 shape tensor.

If this function is called with the value 1, then the function `getNbInputs()` changes from returning 1 to 2.

9.117.3.15 `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](#)

9.117.3.16 `setOutputDimensions()`

```
void nvinfer1::IResizeLayer::setOutputDimensions (
    Dims dimensions ) [inline], [noexcept]
```

Set the output dimensions.

Parameters

<i>dimensions</i>	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](#)

9.117.3.17 setResizeMode()

```
void nvinfer1::IResizeLayer::setResizeMode (
    ResizeMode resizeMode ) [inline], [noexcept]
```

Set resize mode for an input tensor.

Supported resize modes are Nearest Neighbor and Linear.

See also

[ResizeMode](#)

9.117.3.18 setScales()

```
void nvinfer1::IResizeLayer::setScales (
    float const * scales,
    int32_t nbScales ) [inline], [noexcept]
```

Set the resize scales.

Parameters

<i>scales</i>	An array of resize scales.
<i>nbScales</i>	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: $\text{outputDims}[i] = \text{floor}(\text{inputDims}[i] * \text{scales}[i])$

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](#)

9.117.3.19 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](#)

9.117.4 Member Data Documentation

9.117.4.1 mImpl

```
apiv::VResizeLayer* nvinfer1::IResizeLayer::mImpl [protected]
```

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

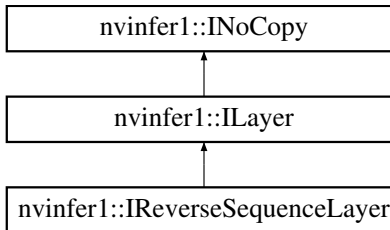
- [NvInfer.h](#)

9.118 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=default

Protected Attributes

- apiv::VReverseSequenceLayer * [mImpl](#)

9.118.1 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.

9.118.2 Constructor & Destructor Documentation

9.118.2.1 ~IReverseSequenceLayer()

```
virtual nvinfer1::IReverseSequenceLayer::~IReverseSequenceLayer ( ) [protected], [virtual], [default],  
[noexcept]
```

9.118.3 Member Function Documentation

9.118.3.1 getBatchAxis()

```
int32_t nvinfer1::IReverseSequenceLayer::getBatchAxis ( ) const [inline], [noexcept]
```

Return the batch axis. Return 1 if no batch axis was set.

See also

[getBatchAxis\(\)](#)

9.118.3.2 getSequenceAxis()

```
int32_t nvinfer1::IReverseSequenceLayer::getSequenceAxis ( ) const [inline], [noexcept]
```

Return the sequence axis. Return 0 if no sequence axis was set.

See also

[getSequenceAxis\(\)](#)

9.118.3.3 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\(\)](#)

9.118.3.4 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\(\)](#)

9.118.4 Member Data Documentation

9.118.4.1 mImpl

```
apiv::VReverseSequenceLayer* nvinfer1::IReverseSequenceLayer::mImpl [protected]
```

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

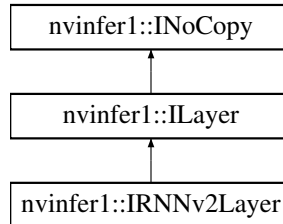
- [NvInfer.h](#)

9.119 nvinfer1::IRNNv2Layer Class Reference

An RNN layer in a network definition, version 2.

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

Inheritance diagram for nvinfer1::IRNNv2Layer:



Public Member Functions

- `int32_t getLayerCount ()` const noexcept
Get the layer count of the RNN.
- `int32_t getHiddenSize ()` const noexcept
Get the hidden size of the RNN.
- `int32_t getMaxSeqLength ()` const noexcept
Get the maximum sequence length of the RNN.
- `int32_t getDataLength ()` const noexcept
Get the embedding length of the RNN.
- `void setSequenceLengths (ITensor &seqLengths)` noexcept
Specify individual sequence lengths in the batch with the ITensor pointed to by seqLengths.
- `ITensor * getSequenceLengths ()` const noexcept
Get the sequence lengths specified for the RNN.
- `void setOperation (RNNOperation op)` noexcept
Set the operation of the RNN layer.
- `RNNOperation getOperation ()` const noexcept
Get the operation of the RNN layer.
- `void setInputMode (RNNInputMode op)` noexcept
Set the input mode of the RNN layer.
- `RNNInputMode getInputMode ()` const noexcept
Get the input mode of the RNN layer.
- `void setDirection (RNNDirection op)` noexcept
Set the direction of the RNN layer.
- `RNNDirection getDirection ()` const noexcept
Get the direction of the RNN layer.
- `void setWeightsForGate (int32_t layerIndex, RNNGateType gate, bool isW, Weights weights)` noexcept
Set the weight parameters for an individual gate in the RNN.
- `Weights getWeightsForGate (int32_t layerIndex, RNNGateType gate, bool isW)` const noexcept
Get the weight parameters for an individual gate in the RNN.
- `void setBiasForGate (int32_t layerIndex, RNNGateType gate, bool isW, Weights bias)` noexcept

- *Set the bias parameters for an individual gate in the RNN.*
Weights `getBiasForGate` (int32_t layerIndex, [RNNGateType](#) gate, bool isW) const noexcept
- *Get the bias parameters for an individual gate in the RNN.*
void `setHiddenState` ([ITensor](#) &hidden) noexcept
- *Set the initial hidden state of the RNN with the provided hidden [ITensor](#).*
ITensor * `getHiddenState` () const noexcept
- *Get the initial hidden state of the RNN.*
void `setCellState` ([ITensor](#) &cell) noexcept
- *Set the initial cell state of the LSTM with the provided cell [ITensor](#).*
ITensor * `getCellState` () const noexcept
- *Get the initial cell state of the RNN.*

Protected Member Functions

- virtual `~IRNNv2Layer` () noexcept=default

Protected Attributes

- apiv::VRNNv2Layer * `mImpl`

9.119.1 Detailed Description

An RNN layer in a network definition, version 2.

This layer supersedes `IRNNLayer`.

Deprecated Deprecated prior to TensorRT 8.0 and will be removed in 9.0. Superseded by `INetworkDefinition::addLoop()`.

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 ~IRNNv2Layer()

```
virtual nvinfer1::IRNNv2Layer::~~IRNNv2Layer ( ) [protected], [virtual], [default], [noexcept]
```

9.119.3 Member Function Documentation

9.119.3.1 `getBiasForGate()`

```
Weights nvinfer1::IRNNv2Layer::getBiasForGate (
    int32_t layerIndex,
    RNNGateType gate,
    bool isW ) const [inline], [noexcept]
```

Get the bias parameters for an individual gate in the RNN.

See also

[`setBiasForGate\(\)`](#)

9.119.3.2 `getCellState()`

```
ITensor * nvinfer1::IRNNv2Layer::getCellState ( ) const [inline], [noexcept]
```

Get the initial cell state of the RNN.

See also

[`setCellState\(\)`](#)

9.119.3.3 `getDataLength()`

```
int32_t nvinfer1::IRNNv2Layer::getDataLength ( ) const [inline], [noexcept]
```

Get the embedding length of the RNN.

9.119.3.4 `getDirection()`

```
RNNDirection nvinfer1::IRNNv2Layer::getDirection ( ) const [inline], [noexcept]
```

Get the direction of the RNN layer.

See also

[`setDirection\(\)`](#), [`RNNDirection`](#)

9.119.3.5 getHiddenSize()

```
int32_t nvinfer1::IRNNv2Layer::getHiddenSize ( ) const [inline], [noexcept]
```

Get the hidden size of the RNN.

9.119.3.6 getHiddenState()

```
ITensor * nvinfer1::IRNNv2Layer::getHiddenState ( ) const [inline], [noexcept]
```

Get the initial hidden state of the RNN.

See also

[setHiddenState\(\)](#)

9.119.3.7 getInputMode()

```
RNNInputMode nvinfer1::IRNNv2Layer::getInputMode ( ) const [inline], [noexcept]
```

Get the input mode of the RNN layer.

See also

[setInputMode\(\)](#), [RNNInputMode](#)

9.119.3.8 getLayerCount()

```
int32_t nvinfer1::IRNNv2Layer::getLayerCount ( ) const [inline], [noexcept]
```

Get the layer count of the RNN.

9.119.3.9 getMaxSeqLength()

```
int32_t nvinfer1::IRNNv2Layer::getMaxSeqLength ( ) const [inline], [noexcept]
```

Get the maximum sequence length of the RNN.

9.119.3.10 getOperation()

```
RNNOperation nvinfer1::IRNNv2Layer::getOperation ( ) const [inline], [noexcept]
```

Get the operation of the RNN layer.

See also

[setOperation\(\)](#), [RNNOperation](#)

9.119.3.11 getSequenceLengths()

```
ITensor * nvinfer1::IRNNv2Layer::getSequenceLengths ( ) const [inline], [noexcept]
```

Get the sequence lengths specified for the RNN.

Returns

nullptr if no sequence lengths were specified, the sequence length data otherwise.

See also

[setSequenceLengths\(\)](#)

9.119.3.12 getWeightsForGate()

```
Weights nvinfer1::IRNNv2Layer::getWeightsForGate (
    int32_t layerIndex,
    RNNGateType gate,
    bool isW ) const [inline], [noexcept]
```

Get the weight parameters for an individual gate in the RNN.

See also

[setWeightsForGate\(\)](#)

9.119.3.13 setBiasForGate()

```
void nvinfer1::IRNNv2Layer::setBiasForGate (
    int32_t layerIndex,
    RNNGateType gate,
    bool isW,
    Weights bias ) [inline], [noexcept]
```

Set the bias parameters for an individual gate in the RNN.

The [DataType](#) for this structure must be `::kFLOAT` or `::kHALF`, and must be the same datatype as the input tensor.

Each bias vector has a fixed size, [getHiddenSize\(\)](#).

Parameters

<i>layerIndex</i>	The index of the layer that contains this gate. See the section Order of weight matrices in <code>IRNNLayer::setWeights()</code> for a description of the layer index.
<i>gate</i>	The name of the gate within the RNN layer. The gate name must correspond to one of the gates used by this layer's RNNOperation .
<i>isW</i>	True if the bias parameters are for the input bias $Wb[g]$ and false if they are for the recurrent input bias $Rb[g]$. See RNNOperation for equations showing how these bias vectors are used in the RNN gate.
<i>bias</i>	The weight structure holding the bias parameters, which should be an array of size getHiddenSize() .

9.119.3.14 setCellState()

```
void nvinfer1::IRNNv2Layer::setCellState (
    ITensor & cell ) [inline], [noexcept]
```

Set the initial cell state of the LSTM with the provided `cell` [ITensor](#).

The `cell` [ITensor](#) should have the dimensions $\{N1, \dots, Np, L, H\}$, where:

- $N1 \dots Np$ are the index dimensions specified by the input tensor
- L is the number of layers in the RNN, equal to [getLayerCount\(\)](#) if `getDirection` is `::kUNIDIRECTION`, and $2 \times$ [getLayerCount\(\)](#) if `getDirection` is `::kBIDIRECTION`. In the bi-directional case, layer 1's final forward hidden state is stored in $L = 2 \times 1$, and final backward hidden state is stored in $L = 2 \times 1 + 1$.
- H is the hidden state for each layer, equal to [getHiddenSize\(\)](#).

It is an error to call `setCellState()` on an RNN layer that is not configured with `RNNOperation::kLSTM`.

9.119.3.15 setDirection()

```
void nvinfer1::IRNNv2Layer::setDirection (
    RNNDirection op ) [inline], [noexcept]
```

Set the direction of the RNN layer.

The direction determines if the RNN is run as a unidirectional(left to right) or bidirectional(left to right and right to left). In the `::kBIDIRECTION` case the output is concatenated together, resulting in output size of $2 \times$ [getHiddenSize\(\)](#).

See also

[getDirection\(\)](#), [RNNDirection](#)

9.119.3.16 setHiddenState()

```
void nvinfer1::IRNNv2Layer::setHiddenState (
    ITensor & hidden ) [inline], [noexcept]
```

Set the initial hidden state of the RNN with the provided hidden [ITensor](#).

The hidden [ITensor](#) should have the dimensions $\{N_1, \dots, N_p, L, H\}$, where:

- $N_1 \dots N_p$ are the index dimensions specified by the input tensor
- L is the number of layers in the RNN, equal to [getLayerCount\(\)](#) if `getDirection` is `::kUNIDIRECTION`, and $2 \times$ [getLayerCount\(\)](#) if `getDirection` is `::kBIDIRECTION`. In the bi-directional case, layer 1's final forward hidden state is stored in $L = 2 \times 1$, and final backward hidden state is stored in $L = 2 \times 1 + 1$.
- H is the hidden state for each layer, equal to [getHiddenSize\(\)](#).

9.119.3.17 setInputMode()

```
void nvinfer1::IRNNv2Layer::setInputMode (
    RNNInputMode op ) [inline], [noexcept]
```

Set the input mode of the RNN layer.

See also

[getInputMode\(\)](#), [RNNInputMode](#)

9.119.3.18 setOperation()

```
void nvinfer1::IRNNv2Layer::setOperation (
    RNNOperation op ) [inline], [noexcept]
```

Set the operation of the RNN layer.

See also

[getOperation\(\)](#), [RNNOperation](#)

9.119.3.19 setSequenceLengths()

```
void nvinfer1::IRNNv2Layer::setSequenceLengths (
    ITensor & seqLengths ) [inline], [noexcept]
```

Specify individual sequence lengths in the batch with the [ITensor](#) pointed to by `seqLengths`.

The `seqLengths` [ITensor](#) should be a $\{N1, ..., Np\}$ tensor, where $N1..Np$ are the index dimensions of the input tensor to the RNN.

If this is not specified, then the RNN layer assumes all sequences are size [getMaxSeqLength\(\)](#).

All sequence lengths in `seqLengths` should be in the range $[1, \text{getMaxSeqLength}())$. Zero-length sequences are not supported.

This tensor must be of type [DataType::kINT32](#).

9.119.3.20 setWeightsForGate()

```
void nvinfer1::IRNNv2Layer::setWeightsForGate (
    int32_t layerIndex,
    RNNGateType gate,
    bool isW,
    Weights weights ) [inline], [noexcept]
```

Set the weight parameters for an individual gate in the RNN.

The [DataType](#) for this structure must be `::kFLOAT` or `::kHALF`, and must be the same datatype as the input tensor.

Each parameter matrix is row-major in memory, and has the following dimensions:

```
Let K := { ::kUNIDIRECTION => 1
           { ::kBIDIRECTION => 2
    l := layer index (as described above)
    H := getHiddenSize()
    E := getDataLength() (the embedding length)
    isW := true if the matrix is an input (W) matrix, and false if
           the matrix is a recurrent input (R) matrix.
if isW:
    if l < K and ::kSKIP:
        (numRows, numCols) := (0, 0) # input matrix is skipped
    elif l < K and ::kLINEAR:
        (numRows, numCols) := (H, E) # input matrix acts on input data size E
    elif l >= K:
        (numRows, numCols) := (H, K * H) # input matrix acts on previous hidden state
else: # not isW
    (numRows, numCols) := (H, H)
```

In other words, the input weights of the first layer of the RNN (if not skipped) transform a [getDataLength\(\)](#)-size column vector into a [getHiddenSize\(\)](#)-size column vector. The input weights of subsequent layers transform a $K \times \text{getHiddenSize}()$ -size column vector into a [getHiddenSize\(\)](#)-size column vector. $K=2$ in the bidirectional case to account for the full hidden state being the concatenation of the forward and backward RNN hidden states.

The recurrent weight matrices for all layers all have shape (H, H) , both in the unidirectional and bidirectional cases. (In the bidirectional case, each recurrent weight matrix for the (forward or backward) RNN cell operates on the previous (forward or backward) RNN cell's hidden state, which is size H).

Parameters

<i>layerIndex</i>	The index of the layer that contains this gate. See the section
<i>gate</i>	The name of the gate within the RNN layer. The gate name must correspond to one of the gates used by this layer's RNNOperation .
<i>isW</i>	True if the weight parameters are for the input matrix $W[g]$ and false if they are for the recurrent input matrix $R[g]$. See RNNOperation for equations showing how these matrices are used in the RNN gate.
<i>weights</i>	The weight structure holding the weight parameters, which are stored as a row-major 2D matrix. See the layout of elements within a weight matrix in <code>IRNNLayer::setWeights()</code> for documentation on the expected dimensions of this matrix.

9.119.4 Member Data Documentation

9.119.4.1 mImpl

```
apiv::VRNNv2Layer* nvinfer1::IRNNv2Layer::mImpl [protected]
```

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

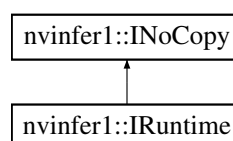
- [NvInfer.h](#)

9.120 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=default
- `TRT_DEPRECATED nvinfer1::ICudaEngine * deserializeCudaEngine (void const *blob, std::size_t size, IPluginFactory *pluginFactory)` noexcept
Deserialize an engine from a stream.
- 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.
- `TRT_DEPRECATED void destroy ()` noexcept
Destroy this object.
- 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 a stream.
- `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

9.120.1 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.

9.120.2 Constructor & Destructor Documentation

9.120.2.1 ~IRuntime()

```
virtual nvinfer1::IRuntime::~~IRuntime ( ) [virtual], [default], [noexcept]
```

9.120.3 Member Function Documentation

9.120.3.1 deserializeCudaEngine() [1/2]

```
ICudaEngine * nvinfer1::IRuntime::deserializeCudaEngine (
    void const * blob,
    std::size_t size ) [inline], [noexcept]
```

Deserialize an engine from a stream.

If an error recorder has been set for the runtime, it will also be passed to the engine.

Parameters

<i>blob</i>	The memory that holds the serialized engine.
<i>size</i>	The size of the memory.

Returns

The engine, or nullptr if it could not be deserialized.

9.120.3.2 deserializeCudaEngine() [2/2]

```
TRT_DEPRECATED nvinfer1::ICudaEngine * nvinfer1::IRuntime::deserializeCudaEngine (
    void const * blob,
    std::size_t size,
    IPluginFactory * pluginFactory ) [inline], [noexcept]
```

Deserialize an engine from a stream.

If an error recorder has been set for the runtime, it will also be passed to the engine.

Parameters

<i>blob</i>	The memory that holds the serialized engine.
<i>size</i>	The size of the memory in bytes.
<i>pluginFactory</i>	The plugin factory, if any plugins are used by the network, otherwise nullptr.

Returns

The engine, or nullptr if it could not be deserialized.

Deprecated Deprecated in TensorRT 8.0.

Warning

IPluginFactory is no longer supported, therefore pluginFactory must be a nullptr.

9.120.3.3 destroy()

```
TRT_DEPRECATED void nvinfer1::IRuntime::destroy ( ) [inline], [noexcept]
```

Destroy this object.

Deprecated Deprecated in TRT 8.0. Superseded by delete.

Warning

Calling destroy on a managed pointer will result in a double-free error.

9.120.3.4 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.

9.120.3.5 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.

9.120.3.6 getErrorRecorder()

```
IErrRecorder * 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 [IErrRecorder](#) object that has been registered.

See also

[setErrorRecorder\(\)](#)

9.120.3.7 getLogger()

```
ILogger * nvinfer1::IRuntime::getLogger ( ) const [inline], [noexcept]
```

get the logger with which the runtime was created

Returns

the logger

9.120.3.8 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\(\)](#)

9.120.3.9 getNbDLACores()

```
int32_t nvinfer1::IRuntime::getNbDLACores ( ) const [inline], [noexcept]
```

Returns number of DLA hardware cores accessible or 0 if DLA is unavailable.

9.120.3.10 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.

9.120.3.11 `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\(\)](#)

9.120.3.12 `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\(\)](#)

9.120.3.13 `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 `IBuilderConfigFlags::kVERSION_COMPATIBLE=true` and `IBuilderConfigFlags::kINCLUDE_LEAN=false`, which was built with the same version of TensorRT as the loaded runtime library.

Parameters

<i>leanPath</i>	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.

9.120.3.14 setDLACore()

```
void nvinfer1::IRuntime::setDLACore (
    int32_t dlaCore ) [inline], [noexcept]
```

Sets the DLA core used by the network. Defaults to -1.

Parameters

<i>dlaCore</i>	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\(\)](#)

9.120.3.15 setEngineHostCodeAllowed()

```
void nvinfer1::IRuntime::setEngineHostCodeAllowed (
    bool allowed ) [inline], [noexcept]
```

Set whether the runtime is allowed to deserialize engines with host executable code.

Parameters

<i>allowed</i>	Whether the runtime is allowed to deserialize engines with host executable code.
----------------	--

The default value is false.

9.120.3.16 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

<i>recorder</i>	The error recorder to register with this interface.
-----------------	---

See also

[getErrorRecorder\(\)](#)

9.120.3.17 setGpuAllocator()

```
void nvinfer1::IRuntime::setGpuAllocator (
    IGpuAllocator * allocator ) [inline], [noexcept]
```

Set the GPU allocator.

Parameters

<i>allocator</i>	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: uses cudaMalloc/cudaFree.

If nullptr is passed, the default allocator will be used.

9.120.3.18 setMaxThreads()

```
bool nvinfer1::IRuntime::setMaxThreads (
    int32_t maxThreads ) [inline], [noexcept]
```

Set the maximum number of threads.

Parameters

<i>maxThreads</i>	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.

9.120.3.19 setTempfileControlFlags()

```
void nvinfer1::IRuntime::setTempfileControlFlags (
    TempfileControlFlags flags ) [inline], [noexcept]
```

Set the tempfile control flags for this runtime.

Parameters

<i>flags</i>	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\(\)](#)

9.120.3.20 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

<i>path</i>	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 \$TMPDIR, then fall back to /tmp
- On Windows, TensorRT will try TEMP%.

See the TensorRT Developer Guide for more information.

The default value is nullptr.

Warning

If path is not nullptr, it should 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 should 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 should be set).

See also

[getTemporaryDirectory\(\)](#)

9.120.4 Member Data Documentation

9.120.4.1 mImpl

```
apiv::VRuntime* nvinfer1::IRuntime::mImpl [protected]
```

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

- [NvInferRuntime.h](#)

9.121 nvinfer1::safe::IRuntime Class Reference

Allows a serialized functionally safe engine to be deserialized.

```
#include <NvInferSafeRuntime.h>
```

Public Member Functions

- virtual [ICudaEngine](#) * [deserializeCudaEngine](#) (void const *const blob, std::size_t const size) noexcept=0
Deserialize an engine from a stream.
- virtual void [setGpuAllocator](#) ([IGpuAllocator](#) *const allocator) noexcept=0
Set the GPU allocator.
- 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.
- [IRuntime](#) ()=default
- virtual [~IRuntime](#) () noexcept=default
- [IRuntime](#) ([IRuntime](#) const &)=delete
- [IRuntime](#) ([IRuntime](#) &&)=delete
- [IRuntime](#) & [operator=](#) ([IRuntime](#) const &) &=delete
- [IRuntime](#) & [operator=](#) ([IRuntime](#) &&) &=delete

9.121.1 Detailed Description

Allows a serialized functionally safe engine to be deserialized.

Warning

In the safety runtime the application is required to set the error reporter for correct error handling.

See also

[setErrorRecorder\(\)](#)

Warning

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 IRuntime() [1/3]

```
nvinfer1::safe::IRuntime::IRuntime ( ) [default]
```

9.121.2.2 ~IRuntime()

```
virtual nvinfer1::safe::IRuntime::~~IRuntime ( ) [virtual], [default], [noexcept]
```

9.121.2.3 IRuntime() [2/3]

```
nvinfer1::safe::IRuntime::IRuntime (
    IRuntime const & ) [delete]
```

9.121.2.4 IRuntime() [3/3]

```
nvinfer1::safe::IRuntime::IRuntime (
    IRuntime && ) [delete]
```

9.121.3 Member Function Documentation

9.121.3.1 deserializeCudaEngine()

```
virtual ICudaEngine * nvinfer1::safe::IRuntime::deserializeCudaEngine (
    void const *const blob,
    std::size_t const size ) [pure virtual], [noexcept]
```

Deserialize an engine from a stream.

If the serialized engine requires plugins the plugin creator must be registered by calling [IPluginRegistry::registerCreator\(\)](#) before calling [deserializeCudaEngine\(\)](#). Every plugin creator registered must have a unique combination of namespace, plugin name, and version.

Parameters

<i>blob</i>	The memory that holds the serialized engine.
<i>size</i>	The size of the memory in bytes.

Returns

The engine, or nullptr if it could not be deserialized.

See also

[IPluginRegistry::registerCreator\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, if called from different instances of [safe::IRuntime](#). Calling `deserializeCudaEngine` of the same safety runtime from multiple threads is not guaranteed to be thread safe.

9.121.3.2 `getErrorRecorder()`

```
virtual IErrorRecorder * nvinfer1::safe::IRuntime::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.

Returns

A pointer to the [IErrorRecorder](#) object that has been registered.

See also

[setErrorRecorder\(\)](#)

Usage considerations

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

9.121.3.3 `operator=()` [1/2]

```
IRuntime & nvinfer1::safe::IRuntime::operator= (  
    IRuntime && ) & [delete]
```

9.121.3.4 operator=() [2/2]

```
IRuntime & nvinfer1::safe::IRuntime::operator= (
    IRuntime const & ) & [delete]
```

9.121.3.5 setErrorRecorder()

```
virtual void nvinfer1::safe::IRuntime::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

<i>recorder</i>	The error recorder to register with this interface.
-----------------	---

See also

[getErrorRecorder\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.121.3.6 setGpuAllocator()

```
virtual void nvinfer1::safe::IRuntime::setGpuAllocator (
    I_gpu_allocator *const allocator ) [pure virtual], [noexcept]
```

Set the GPU allocator.

Parameters

<i>allocator</i>	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: uses cudaMalloc/cudaFree.

If nullptr is passed, the default allocator will be used.

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

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

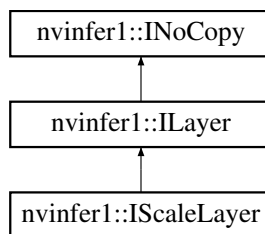
- [NvInferSafeRuntime.h](#)

9.122 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=default

Protected Attributes

- apiv::VScaleLayer * [mImpl](#)

9.122.1 Detailed Description

A Scale layer in a network definition.

This layer applies a per-element computation to its input:

$$\text{output} = (\text{input} * \text{scale} + \text{shift})^{\text{power}}$$

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 in implicit batch mode and a minimum of 4 dimensions in explicit batch mode.

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.

9.122.2 Constructor & Destructor Documentation

9.122.2.1 ~IScaleLayer()

```
virtual nvinfer1::IScaleLayer::~~IScaleLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.122.3 Member Function Documentation

9.122.3.1 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\(\)](#)

9.122.3.2 getMode()

```
ScaleMode nvinfer1::IScaleLayer::getMode ( ) const [inline], [noexcept]
```

Get the scale mode.

See also

[setMode\(\)](#)

9.122.3.3 getPower()

```
Weights nvinfer1::IScaleLayer::getPower ( ) const [inline], [noexcept]
```

Get the power value.

See also

[setPower\(\)](#)

9.122.3.4 `getScale()`

```
Weights nvinfer1::IScaleLayer::getScale ( ) const [inline], [noexcept]
```

Get the scale value.

See also

[setScale\(\)](#)

9.122.3.5 `getShift()`

```
Weights nvinfer1::IScaleLayer::getShift ( ) const [inline], [noexcept]
```

Get the shift value.

See also

[setShift\(\)](#)

9.122.3.6 `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\(\)](#)

9.122.3.7 setMode()

```
void nvinfer1::IScaleLayer::setMode (
    ScaleMode mode ) [inline], [noexcept]
```

Set the scale mode.

See also

[getMode\(\)](#)

9.122.3.8 setPower()

```
void nvinfer1::IScaleLayer::setPower (
    Weights power ) [inline], [noexcept]
```

Set the power value.

See also

[getPower\(\)](#)

9.122.3.9 setScale()

```
void nvinfer1::IScaleLayer::setScale (
    Weights scale ) [inline], [noexcept]
```

Set the scale value.

See also

[getScale\(\)](#)

9.122.3.10 setShift()

```
void nvinfer1::IScaleLayer::setShift (
    Weights shift ) [inline], [noexcept]
```

Set the shift value.

See also

[getShift\(\)](#)

9.122.4 Member Data Documentation

9.122.4.1 mImpl

```
apiv::VScaleLayer* nvinfer1::IScaleLayer::mImpl [protected]
```

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

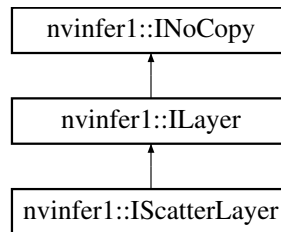
- [NvInfer.h](#)

9.123 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=default

Protected Attributes

- `apiv::VScatterLayer * mImpl`

9.123.1 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 $r \geq 1$ 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 of q depend on the mode: `ScatterMode::kND`: $q \geq 1$ `ScatterMode::kELEMENT`: q must be the same as r
- Updates is a tensor of rank $s \geq 1$ that provides the data to write to Output specified by its corresponding location in Index. Constraints the rank of Updates depend on the mode: `ScatterMode::kND`: $s = r + q - \text{shape}(\text{Indices})[-1] - 1$ `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 `DataType::kINT32`.

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 $\{d_0, \dots, d_{r-1}\}$ and indices dims are $\{i_0, \dots, i_{q-1}\}$, define $k = \text{indices}[q-1]$, it follows that updates dims are $\{i_0, \dots, i_{q-2}, d_k, \dots, d_{r-1}\}$. 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.

9.123.2 Constructor & Destructor Documentation

9.123.2.1 ~IScatterLayer()

```
virtual nvinfer1::IScatterLayer::~~IScatterLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.123.3 Member Function Documentation

9.123.3.1 getAxis()

```
int32_t nvinfer1::IScatterLayer::getAxis ( ) const [inline], [noexcept]
```

Get the axis.

9.123.3.2 getMode()

```
ScatterMode nvinfer1::IScatterLayer::getMode ( ) const [inline], [noexcept]
```

Get the scatter mode.

See also

[setMode\(\)](#)

9.123.3.3 setAxis()

```
void nvinfer1::IScatterLayer::setAxis (
    int32_t axis ) [inline], [noexcept]
```

Set the axis used by ScatterMode::kELEMENTS.

The axis defaults to 0.

9.123.3.4 setMode()

```
void nvinfer1::IScatterLayer::setMode (
    ScatterMode mode ) [inline], [noexcept]
```

Set the scatter mode.

See also

[getMode\(\)](#)

9.123.4 Member Data Documentation

9.123.4.1 mImpl

```
apiv::VScatterLayer* nvinfer1::IScatterLayer::mImpl [protected]
```

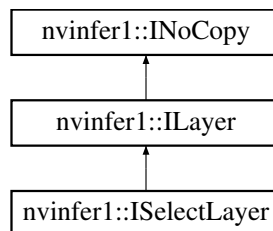
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.124 nvinfer1::ISelectLayer Class Reference

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

Inheritance diagram for nvinfer1::ISelectLayer:



Protected Member Functions

- virtual [~ISelectLayer](#) () noexcept=default

Protected Attributes

- apiv::VSelectLayer * [mImpl](#)

Additional Inherited Members

9.124.1 Detailed Description

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.124.2 Constructor & Destructor Documentation

9.124.2.1 ~ISelectLayer()

```
virtual nvinfer1::ISelectLayer::~ISelectLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.124.3 Member Data Documentation

9.124.3.1 mImpl

```
apiv::VSelectLayer* nvinfer1::ISelectLayer::mImpl [protected]
```

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

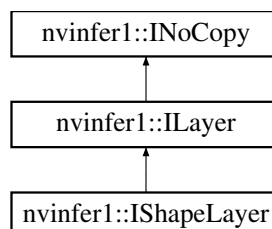
- [NvInfer.h](#)

9.125 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=default

Protected Attributes

- `apiv::VShapeLayer * mImpl`

Additional Inherited Members

9.125.1 Detailed Description

Layer type for getting shape of a tensor.

This layer sets the output to a 1D tensor of type Int32 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 Int32 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.

9.125.2 Constructor & Destructor Documentation

9.125.2.1 ~IShapeLayer()

```
virtual nvinfer1::IShapeLayer::~~IShapeLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.125.3 Member Data Documentation

9.125.3.1 mImpl

```
apiv::VShapeLayer* nvinfer1::IShapeLayer::mImpl [protected]
```

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

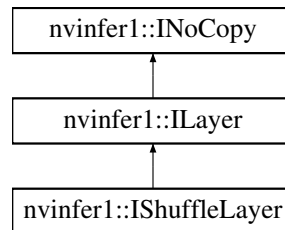
- [NvInfer.h](#)

9.126 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](#) 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=default

Protected Attributes

- apiv::VShuffleLayer * [mImpl](#)

9.126.1 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 Int32 shape tensor, 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.

9.126.2 Constructor & Destructor Documentation

9.126.2.1 ~IShuffleLayer()

```
virtual nvinfer1::IShuffleLayer::~IShuffleLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.126.3 Member Function Documentation

9.126.3.1 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](#)

9.126.3.2 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.

9.126.3.3 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](#)

9.126.3.4 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](#)

9.126.3.5 setFirstTranspose()

```
void nvinfer1::IShuffleLayer::setFirstTranspose (
    Permutation permutation ) [inline], [noexcept]
```

Set the permutation applied by the first transpose operation.

Parameters

<i>permutation</i>	The dimension permutation applied before the reshape.
--------------------	---

The default is the identity permutation.

See also

[getFirstTranspose](#)

9.126.3.6 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

<i>index</i>	the index of the input to modify.
<i>tensor</i>	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 Int32 shape tensor.

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 [setZeroIsPlaceholder\(false\)](#) on the [IShuffleLayer](#).

See also

[setReshapeDimensions](#).

9.126.3.7 setReshapeDimensions()

```
void nvinfer1::IShuffleLayer::setReshapeDimensions (
    Dims dimensions ) [inline], [noexcept]
```

Set the reshaped dimensions.

Parameters

<i>dimensions</i>	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.

If a second input had been used to create this layer, that input is reset to null by this method.

9.126.3.8 setSecondTranspose()

```
void nvinfer1::IShuffleLayer::setSecondTranspose (
    Permutation permutation ) [inline], [noexcept]
```

Set the permutation applied by the second transpose operation.

Parameters

<i>permutation</i>	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](#)

9.126.3.9 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\(\)](#);

9.126.4 Member Data Documentation

9.126.4.1 mImpl

```
apiv::VShuffleLayer* nvinfer1::IShuffleLayer::mImpl [protected]
```

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

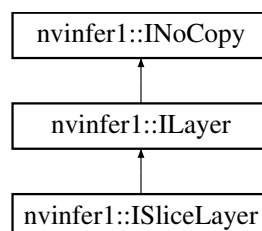
- [NvInfer.h](#)

9.127 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](#) 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](#) size) noexcept
Set the dimensions of the output slice.
- [Dims](#) [getSize](#) () const noexcept
Get dimensions of the output slice.
- void [setStride](#) ([Dims](#) stride) noexcept
Set the stride for computing the output slice data.
- [Dims](#) [getStride](#) () const noexcept
Get the stride for the output slice.
- void [setMode](#) ([SliceMode](#) mode) noexcept
Set the slice mode.
- [SliceMode](#) [getMode](#) () const noexcept
Get the slice mode.
- 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=default

Protected Attributes

- apiv::VSliceLayer * [mImpl](#)

9.127.1 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](#). Dynamic slice specifies one or more of start, size or stride as ITensors, by using [ILayer::setInput](#) to add a second, third, or fourth 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 input (third 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 Int32 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}}

When the sliceMode is kCLAMP or kREFLECT, for each input dimension, if its size is 0 then the corresponding output dimension must be 0 too.

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.
- The number of elements in the output tensor does not exceed $2 * \text{Dims}::\text{MAX_DIMS}$.

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.
- sliceMode is kDEFAULT.
- Strides are 1 for all dimensions.
- Slicing is not performed on the first dimension
- The input tensor has four dimensions

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 ~ISliceLayer()

```
virtual nvinfer1::ISliceLayer::~~ISliceLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.127.3 Member Function Documentation

9.127.3.1 getMode()

```
SliceMode nvinfer1::ISliceLayer::getMode ( ) const [inline], [noexcept]
```

Get the slice mode.

See also

[setMode\(\)](#)

9.127.3.2 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](#)

9.127.3.3 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](#)

9.127.3.4 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](#)

9.127.3.5 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

<i>index</i>	the index of the input to modify.
<i>tensor</i>	the new input tensor

For a slice layer, the values 0-4 are valid. The indices are as follows:

- 0: Tensor to be sliced.
- 1: The start tensor to begin slicing, as a 1D Int32 shape tensor.
- 2: The size tensor of the resulting slice, as a 1D Int32 shape tensor.
- 3: The stride of the slicing operation, as a 1D Int32 shape tensor.
- 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.

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.

9.127.3.6 setMode()

```
void nvinfer1::ISliceLayer::setMode (
    SliceMode mode ) [inline], [noexcept]
```

Set the slice mode.

See also

[getMode\(\)](#)

9.127.3.7 setSize()

```
void nvinfer1::ISliceLayer::setSize (
    Dims size ) [inline], [noexcept]
```

Set the dimensions of the output slice.

Parameters

<i>size</i>	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](#)

9.127.3.8 setStart()

```
void nvinfer1::ISliceLayer::setStart (
    Dims start ) [inline], [noexcept]
```

Set the start offset that the slice layer uses to create the output slice.

Parameters

<i>start</i>	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](#)

9.127.3.9 setStride()

```
void nvinfer1::ISliceLayer::setStride (
    Dims stride ) [inline], [noexcept]
```

Set the stride for computing the output slice data.

Parameters

<i>stride</i>	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](#)

9.127.4 Member Data Documentation

9.127.4.1 mImpl

apiv::VSliceLayer* nvinfer1::ISliceLayer::mImpl [protected]

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

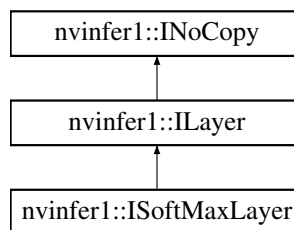
- [NvInfer.h](#)

9.128 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=default

Protected Attributes

- apiv::VSoftMaxLayer * [mImpl](#)

9.128.1 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.

On Xavier, this layer is not supported on DLA. Otherwise, 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.

9.128.2 Constructor & Destructor Documentation

9.128.2.1 ~ISoftMaxLayer()

```
virtual nvinfer1::ISoftMaxLayer::~~ISoftMaxLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.128.3 Member Function Documentation

9.128.3.1 getAxes()

```
uint32_t nvinfer1::ISoftMaxLayer::getAxes ( ) const [inline], [noexcept]
```

Get the axis along which softmax occurs.

See also

[setAxes\(\)](#)

9.128.3.2 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 (three non-batch dimensions).

In implicit mode : Bit 0 corresponds to the C dimension boolean. Bit 1 corresponds to the H dimension boolean. Bit 2 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 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.

In explicit mode : 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 2 with implicit batch mode, set bit 3 with explicit batch mode.

Parameters

<i>axes</i>	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.
-------------	---

9.128.4 Member Data Documentation

9.128.4.1 mImpl

```
apiv::VSoftMaxLayer* nvinfer1::ISoftMaxLayer::mImpl [protected]
```

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

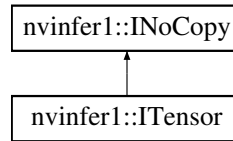
- [NvInfer.h](#)

9.129 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 dimensions) noexcept`
Set the dimensions of a tensor.
- `Dims getDimensions () const noexcept`
Get the dimensions of a tensor.
- `void setType (DataType type) noexcept`
Set the data type of a tensor.
- `DataType getType () const noexcept`
Get the data type of a tensor.
- `bool setDynamicRange (float min, float max) noexcept`
Set dynamic range for the tensor.
- `bool isNetworkInput () const noexcept`
Whether the tensor is a network input.
- `bool isNetworkOutput () const noexcept`
Whether the tensor is a network output.
- `void setBroadcastAcrossBatch (bool broadcastAcrossBatch) noexcept`
Set whether to enable broadcast of tensor across the batch.
- `bool getBroadcastAcrossBatch () const noexcept`
Check if tensor is broadcast across the batch.
- `TensorLocation getLocation () const noexcept`
Get the storage location of a tensor.
- `void setLocation (TensorLocation location) noexcept`
Set the storage location of a tensor.
- `bool dynamicRangeIsSet () const noexcept`
Query whether dynamic range is set.
- `void resetDynamicRange () noexcept`
Undo effect of setDynamicRange.
- `float getDynamicRangeMin () const noexcept`
Get minimum of dynamic range.
- `float getDynamicRangeMax () const noexcept`
Get maximum of dynamic range.
- `void setAllowedFormats (TensorFormats formats) noexcept`
Set allowed formats for this 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=default

Protected Attributes

- apiv::VTensor * [mImpl](#)

9.129.1 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

The volume of the tensor must be less than 2^{31} elements. If the tensor is a shape tensor, its volume must not exceed 64.

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 ~ITensor()

```
virtual nvinfer1::ITensor::~ITensor ( ) [protected], [virtual], [default], [noexcept]
```


9.129.3 Member Function Documentation

9.129.3.1 dynamicRangeIsSet()

```
bool nvinfer1::ITensor::dynamicRangeIsSet ( ) const [inline], [noexcept]
```

Query whether dynamic range is set.

Returns

True if dynamic range is set, false otherwise.

9.129.3.2 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\(\)](#)

9.129.3.3 getBroadcastAcrossBatch()

```
bool nvinfer1::ITensor::getBroadcastAcrossBatch ( ) const [inline], [noexcept]
```

Check if tensor is broadcast across the batch.

When a tensor is broadcast across a batch, it has the same value for every member in the batch. Memory is only allocated once for the single member. If the network is in explicit batch mode, this function returns true if the leading dimension is 1.

Returns

True if tensor is broadcast across the batch, false otherwise.

See also

[setBroadcastAcrossBatch\(\)](#)

9.129.3.4 getDimensionName()

```
char const * nvinfer1::ITensor::getDimensionName (
    int32_t index ) const [inline], [noexcept]
```

Get the name of an input dimension.

Parameters

<i>index</i>	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\(\)](#)

9.129.3.5 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\(\)](#)

9.129.3.6 getDynamicRangeMax()

```
float nvinfer1::ITensor::getDynamicRangeMax ( ) const [inline], [noexcept]
```

Get maximum of dynamic range.

Returns

Maximum of dynamic range, or quiet NaN if range was not set.

9.129.3.7 `getDynamicRangeMin()`

```
float nvinfer1::ITensor::getDynamicRangeMin ( ) const [inline], [noexcept]
```

Get minimum of dynamic range.

Returns

Minimum of dynamic range, or quiet NaN if range was not set.

9.129.3.8 `getLocation()`

```
TensorLocation nvinfer1::ITensor::getLocation ( ) const [inline], [noexcept]
```

Get the storage location of a tensor.

Returns

The location of tensor data.

See also

[setLocation\(\)](#)

9.129.3.9 `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\(\)](#)

9.129.3.10 getType()

```
DataType nvinfer1::ITensor::getType ( ) const [inline], [noexcept]
```

Get the data type of a tensor.

Returns

The data type of the tensor.

See also

[setType\(\)](#)

9.129.3.11 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 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.

If a tensor is an execution tensor and becomes an engine input or output, then [ICudaEngine::isExecutionBinding](#) will be true for that tensor.

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.

9.129.3.12 isNetworkInput()

```
bool nvinfer1::ITensor::isNetworkInput ( ) const [inline], [noexcept]
```

Whether the tensor is a network input.

9.129.3.13 isNetworkOutput()

```
bool nvinfer1::ITensor::isNetworkOutput ( ) const [inline], [noexcept]
```

Whether the tensor is a network output.

9.129.3.14 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`, `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 give 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".

If a tensor is a shape tensor and becomes an engine input or output, then [ICudaEngine::isShapeBinding](#) will be true for that tensor. Such a shape tensor must have type `Int32`.

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\(\)](#), [ICudaEngine::isShapeBinding\(\)](#)

9.129.3.15 resetDynamicRange()

```
void nvinfer1::ITensor::resetDynamicRange ( ) [inline], [noexcept]
```

Undo effect of `setDynamicRange`.

9.129.3.16 setAllowedFormats()

```
void nvinfer1::ITensor::setAllowedFormats (
    TensorFormats formats ) [inline], [noexcept]
```

Set allowed formats for this 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.

Parameters

<i>formats</i>	A bitmask of TensorFormat values that are supported for this tensor.
----------------	--

See also

[ITensor::getAllowedFormats\(\)](#)

[TensorFormats](#)

9.129.3.17 setBroadcastAcrossBatch()

```
void nvinfer1::ITensor::setBroadcastAcrossBatch (
    bool broadcastAcrossBatch ) [inline], [noexcept]
```

Set whether to enable broadcast of tensor across the batch.

When a tensor is broadcast across a batch, it has the same value for every member in the batch. Memory is only allocated once for the single member.

This method is only valid for network input tensors, since the flags of layer output tensors are inferred based on layer inputs and parameters. If this state is modified for a tensor in the network, the states of all dependent tensors will be recomputed. If the tensor is for an explicit batch network, then this function does nothing.

Warning

The broadcast flag is ignored when using explicit batch network mode.

Parameters

<i>broadcastAcrossBatch</i>	Whether to enable broadcast of tensor across the batch.
-----------------------------	---

See also

[getBroadcastAcrossBatch\(\)](#)

9.129.3.18 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

<i>index</i>	index of the dimension
<i>name</i>	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\(\)](#)

9.129.3.19 setDimensions()

```
void nvinfer1::ITensor::setDimensions (
    Dims 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. The volume must be less than 2^{31} elements.

Parameters

<i>dimensions</i>	The dimensions of the tensor.
-------------------	-------------------------------

See also

[getDimensions\(\)](#)

9.129.3.20 setDynamicRange()

```
bool nvinfer1::ITensor::setDynamicRange (
    float min,
    float max ) [inline], [noexcept]
```

Set dynamic range for the tensor.

Currently, only symmetric ranges are supported. Therefore, the larger of the absolute values of the provided bounds is used.

Returns

Whether the dynamic range was set successfully.

Requires that min and max be finite, and min <= max.

9.129.3.21 setLocation()

```
void nvinfer1::ITensor::setLocation (
    TensorLocation location ) [inline], [noexcept]
```

Set the storage location of a tensor.

Parameters

<i>location</i>	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\(\)](#)

9.129.3.22 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.

This method copies the name string.

Parameters

<i>name</i>	The name.
-------------	-----------

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

See also

[getName\(\)](#)

9.129.3.23 setType()

```
void nvinfer1::ITensor::setType (
    DataType type ) [inline], [noexcept]
```

Set the data type of a tensor.

Parameters

<i>type</i>	The data type of the tensor.
-------------	------------------------------

The type is unchanged if the tensor is not a network input tensor, or marked as an output tensor or shape output tensor.

See also

[getType\(\)](#)

9.129.4 Member Data Documentation

9.129.4.1 mImpl

```
apiv::VTensor* nvinfer1::ITensor::mImpl [protected]
```

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

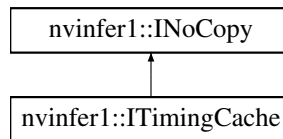
- [NvInfer.h](#)

9.130 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=default
- [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.

Protected Attributes

- apiv::VTimingCache * [mImpl](#)

Additional Inherited Members

9.130.1 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 accelerate the builder wallclock time.

See also

[IBuilderConfig](#)

9.130.2 Constructor & Destructor Documentation

9.130.2.1 ~ITimingCache()

```
virtual nvinfer1::ITimingCache::~~ITimingCache ( ) [virtual], [default], [noexcept]
```

9.130.3 Member Function Documentation

9.130.3.1 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

<i>inputCache</i>	The input timing cache.
<i>ignoreMismatch</i>	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!

9.130.3.2 reset()

```
bool nvinfer1::ITimingCache::reset ( ) [inline], [noexcept]
```

Empty the timing cache.

Returns

True if reset successfully, false otherwise.

9.130.3.3 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](#)

9.130.4 Member Data Documentation

9.130.4.1 mImpl

```
apiv::VTimingCache* nvinfer1::ITimingCache::mImpl [protected]
```

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

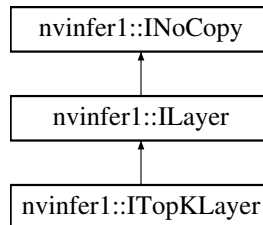
- [NvInfer.h](#)

9.131 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.
- 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=default

Protected Attributes

- apiv::VTopKLayer * [mImpl](#)

9.131.1 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.

9.131.2 Constructor & Destructor Documentation

9.131.2.1 ~ITopKLayer()

```
virtual nvinfer1::ITopKLayer::~~ITopKLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.131.3 Member Function Documentation

9.131.3.1 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\(\)](#)

9.131.3.2 getOperation()

```
TopKOperation nvinfer1::ITopKLayer::getOperation ( ) const [inline], [noexcept]
```

Get the operation for the layer.

See also

[setOperation\(\)](#), [TopKOperation](#)

9.131.3.3 `getReduceAxes()`

```
uint32_t nvinfer1::ITopKLayer::getReduceAxes ( ) const [inline], [noexcept]
```

Get the axes to reduce for the layer.

See also

[setReduceAxes\(\)](#)

9.131.3.4 `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

<i>index</i>	The index of the input to modify.
<i>tensor</i>	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.

9.131.3.5 `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\(\)](#)

9.131.3.6 setOperation()

```
void nvinfer1::ITopKLayer::setOperation (
    TopKOperation op ) [inline], [noexcept]
```

Set the operation for the layer.

See also

[getOperation\(\)](#), [TopKOperation](#)

9.131.3.7 setReduceAxes()

```
void nvinfer1::ITopKLayer::setReduceAxes (
    uint32_t reduceAxes ) [inline], [noexcept]
```

Set which axes to reduce for the layer.

See also

[getReduceAxes\(\)](#)

9.131.4 Member Data Documentation

9.131.4.1 mImpl

```
apiv::VTopKLayer* nvinfer1::ITopKLayer::mImpl [protected]
```

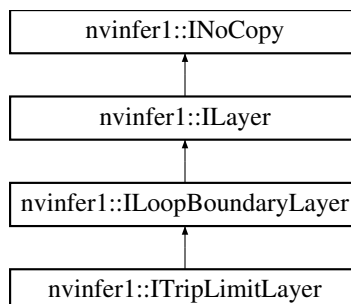
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.132 nvinfer1::ITripLimitLayer Class Reference

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

Inheritance diagram for nvinfer1::ITripLimitLayer:



Public Member Functions

- [TripLimit](#) `getTripLimit ()` const noexcept

Protected Member Functions

- virtual [~ITripLimitLayer](#) () noexcept=default

Protected Attributes

- `apiv::VTripLimitLayer * mImpl`

9.132.1 Constructor & Destructor Documentation

9.132.1.1 ~ITripLimitLayer()

```
virtual nvinfer1::ITripLimitLayer::~~ITripLimitLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.132.2 Member Function Documentation

9.132.2.1 getTripLimit()

```
TripLimit nvinfer1::ITripLimitLayer::getTripLimit ( ) const [inline], [noexcept]
```

9.132.3 Member Data Documentation

9.132.3.1 mImpl

```
apiv::VTripLimitLayer* nvinfer1::ITripLimitLayer::mImpl [protected]
```

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

- [NvInfer.h](#)

9.133 nvuffparser::IUffParser Class Reference

Class used for parsing models described using the UFF format.

```
#include <NvUffParser.h>
```

Public Member Functions

- virtual bool [registerInput](#) (char const *inputName, [nvinfer1::Dims](#) inputDims, [UffInputOrder](#) inputOrder) noexcept=0
Register an input name of a UFF network with the associated Dimensions.
- virtual bool [registerOutput](#) (char const *outputName) noexcept=0
Register an output name of a UFF network.
- virtual bool [parse](#) (char const *file, [nvinfer1::INetworkDefinition](#) &network, [nvinfer1::DataType](#) weightsType=[nvinfer1::DataType::kFLOAT](#)) noexcept=0
Parse a UFF file.
- virtual bool [parseBuffer](#) (char const *buffer, std::size_t size, [nvinfer1::INetworkDefinition](#) &network, [nvinfer1::DataType](#) weightsType=[nvinfer1::DataType::kFLOAT](#)) noexcept=0
Parse a UFF buffer, useful if the file already live in memory.
- virtual [TRT_DEPRECATED](#) void [destroy](#) () noexcept=0
- virtual int32_t [getUffRequiredVersionMajor](#) () noexcept=0
Return Version Major of the UFF.
- virtual int32_t [getUffRequiredVersionMinor](#) () noexcept=0
Return Version Minor of the UFF.
- virtual int32_t [getUffRequiredVersionPatch](#) () noexcept=0
Return Patch Version of the UFF.
- virtual void [setPluginNamespace](#) (char const *libNamespace) noexcept=0
Set the namespace used to lookup and create plugins in the network.
- virtual [~IUffParser](#) () noexcept=default
- virtual void [setErrorRecorder](#) ([nvinfer1::IErrorRecorder](#) *recorder) noexcept=0
Set the ErrorRecorder for this interface.
- virtual [nvinfer1::IErrorRecorder](#) * [getErrorRecorder](#) () const noexcept=0
get the ErrorRecorder assigned to this interface.

9.133.1 Detailed Description

Class used for parsing models described using the UFF format.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.133.2 Constructor & Destructor Documentation

9.133.2.1 ~IUffParser()

```
virtual nvuffparser::IUffParser::~~IUffParser ( ) [virtual], [default], [noexcept]
```

9.133.3 Member Function Documentation

9.133.3.1 destroy()

```
virtual TRT\_DEPRECATED void nvuffparser::IUffParser::destroy ( ) [pure virtual], [noexcept]
```

Deprecated Use delete instead. Deprecated in TRT 8.0.

9.133.3.2 getErrorRecorder()

```
virtual nvinfer1::IErrorRecorder * nvuffparser::IUffParser::getErrorRecorder ( ) const [pure virtual], [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\(\)](#)

9.133.3.3 getUffRequiredVersionMajor()

```
virtual int32_t nvuffparser::IUffParser::getUffRequiredVersionMajor ( ) [pure virtual], [noexcept]
```

Return Version Major of the UFF.

9.133.3.4 getUffRequiredVersionMinor()

```
virtual int32_t nvuffparser::IUffParser::getUffRequiredVersionMinor ( ) [pure virtual], [noexcept]
```

Return Version Minor of the UFF.

9.133.3.5 getUffRequiredVersionPatch()

```
virtual int32_t nvuffparser::IUffParser::getUffRequiredVersionPatch ( ) [pure virtual], [noexcept]
```

Return Patch Version of the UFF.

9.133.3.6 parse()

```
virtual bool nvuffparser::IUffParser::parse (
    char const * file,
    nvinfer1::INetworkDefinition & network,
    nvinfer1::DataType weightsType = nvinfer1::DataType::kFLOAT ) [pure virtual], [noexcept]
```

Parse a UFF file.

Parameters

<i>file</i>	File name of the UFF file.
<i>network</i>	Network in which the UFFParser will fill the layers.
<i>weightsType</i>	The type on which the weights will transformed in.

9.133.3.7 parseBuffer()

```
virtual bool nvuffparser::IUffParser::parseBuffer (
    char const * buffer,
    std::size_t size,
    nvinfer1::INetworkDefinition & network,
    nvinfer1::DataType weightsType = nvinfer1::DataType::kFLOAT ) [pure virtual], [noexcept]
```

Parse a UFF buffer, useful if the file already live in memory.

Parameters

<i>buffer</i>	Buffer of the UFF file.
---------------	-------------------------

Parameters

<i>size</i>	Size of buffer of the UFF file.
<i>network</i>	Network in which the UFFParser will fill the layers.
<i>weightsType</i>	The type on which the weights will transformed in.

9.133.3.8 registerInput()

```
virtual bool nvuffparser::IUffParser::registerInput (
    char const * inputName,
    nvinfer1::Dims inputDims,
    UffInputOrder inputOrder ) [pure virtual], [noexcept]
```

Register an input name of a UFF network with the associated Dimensions.

Parameters

<i>inputName</i>	Input name.
<i>inputDims</i>	Input dimensions.
<i>inputOrder</i>	Input order on which the framework input was originally.

9.133.3.9 registerOutput()

```
virtual bool nvuffparser::IUffParser::registerOutput (
    char const * outputName ) [pure virtual], [noexcept]
```

Register an output name of a UFF network.

Parameters

<i>outputName</i>	Output name.
-------------------	--------------

9.133.3.10 setErrorRecorder()

```
virtual void nvuffparser::IUffParser::setErrorRecorder (
    nvinfer1::IErrorRecorder * 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.

If an error recorder is not set, messages will be sent to the global log stream.

Parameters

<i>recorder</i>	The error recorder to register with this interface.
-----------------	---

See also

[getErrorRecorder\(\)](#)

9.133.3.11 setPluginNamespace()

```
virtual void nvuffparser::IUFFParser::setPluginNamespace (
    char const * libNamespace ) [pure virtual], [noexcept]
```

Set the namespace used to lookup and create plugins in the network.

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

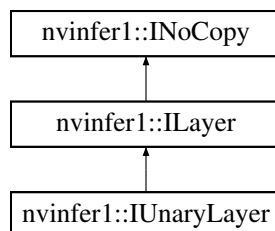
- [NvUffParser.h](#)

9.134 nvinfer1::UnaryLayer Class Reference

Layer that represents an unary operation.

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

Inheritance diagram for `nvinfer1::UnaryLayer`:



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 [~UnaryLayer](#) () noexcept=default

Protected Attributes

- `apiv::UnaryLayer * mImpl`

9.134.1 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.

9.134.2 Constructor & Destructor Documentation

9.134.2.1 [~UnaryLayer](#)()

```
virtual nvinfer1::UnaryLayer::~~UnaryLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.134.3 Member Function Documentation

9.134.3.1 [getOperation](#)()

```
UnaryOperation nvinfer1::UnaryLayer::getOperation ( ) const [inline], [noexcept]
```

Get the unary operation for the layer.

See also

[setOperation\(\)](#), [UnaryOperation](#)

9.134.3.2 setOperation()

```
void nvinfer1::UnaryLayer::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](#)

9.134.4 Member Data Documentation

9.134.4.1 mImpl

```
apiv::VUnaryLayer* nvinfer1::UnaryLayer::mImpl [protected]
```

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

- [NvInfer.h](#)

9.135 nvinfer1::plugin::NMSParameters Struct Reference

The [NMSParameters](#) are used by the BatchedNMSPPlugin for performing the non_max_suppression operation over boxes for object detection networks.

```
#include <NvInferPluginUtils.h>
```

Public Attributes

- bool [shareLocation](#)
- int32_t [backgroundLabelId](#)
- int32_t [numClasses](#)
- int32_t [topK](#)
- int32_t [keepTopK](#)
- float [scoreThreshold](#)
- float [iouThreshold](#)
- bool [isNormalized](#)

9.135.1 Detailed Description

The [NMSParameters](#) are used by the BatchedNMSPPlugin for performing the non_max_suppression operation over boxes for object detection networks.

Parameters

<i>shareLocation</i>	If set to true, the boxes inputs are shared across all classes. If set to false, the boxes input should account for per class box data.
<i>background↔ LabelId</i>	Label ID for the background class. If there is no background class, set it as -1
<i>numClasses</i>	Number of classes in the network.
<i>topK</i>	Number of bounding boxes to be fed into the NMS step.
<i>keepTopK</i>	Number of total bounding boxes to be kept per image after NMS step. Should be less than or equal to the topK value.
<i>scoreThreshold</i>	Scalar threshold for score (low scoring boxes are removed).
<i>iouThreshold</i>	scalar threshold for IOU (new boxes that have high IOU overlap with previously selected boxes are removed).
<i>isNormalized</i>	Set to false, if the box coordinates are not normalized, i.e. not in the range [0,1]. Defaults to false.

9.135.2 Member Data Documentation**9.135.2.1 backgroundLabelId**

```
int32_t nvinfer1::plugin::NMSParameters::backgroundLabelId
```

9.135.2.2 iouThreshold

```
float nvinfer1::plugin::NMSParameters::iouThreshold
```

9.135.2.3 isNormalized

```
bool nvinfer1::plugin::NMSParameters::isNormalized
```

9.135.2.4 keepTopK

```
int32_t nvinfer1::plugin::NMSParameters::keepTopK
```

9.135.2.5 numClasses

```
int32_t nvinfer1::plugin::NMSParameters::numClasses
```

9.135.2.6 scoreThreshold

```
float nvinfer1::plugin::NMSParameters::scoreThreshold
```

9.135.2.7 shareLocation

```
bool nvinfer1::plugin::NMSParameters::shareLocation
```

9.135.2.8 topK

```
int32_t nvinfer1::plugin::NMSParameters::topK
```

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

- [NvInferPluginUtils.h](#)

9.136 nvinfer1::Permutation Struct Reference

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

Public Attributes

- int32_t [order](#) [[Dims::MAX_DIMS](#)]

9.136.1 Member Data Documentation

9.136.1.1 order

```
int32_t nvinfer1::Permutation::order[Dims::MAX_DIMS]
```

The elements of the 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], 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](#)

9.137 nvinfer1::PluginField Class Reference

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

```
#include <NvInferRuntimeCommon.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.

9.137.1 Detailed Description

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

9.137.2 Constructor & Destructor Documentation

9.137.2.1 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]
```

9.137.3 Member Data Documentation

9.137.3.1 data

```
void const* nvinfer1::PluginField::data
```

Plugin field attribute data.

9.137.3.2 length

```
int32_t nvinfer1::PluginField::length
```

Number of data entries in the Plugin attribute.

9.137.3.3 name

```
AsciiChar const* nvinfer1::PluginField::name
```

Plugin field attribute name.

9.137.3.4 type

```
PluginFieldType nvinfer1::PluginField::type
```

Plugin field attribute type.

See also

[PluginFieldType](#)

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

- [NvInferRuntimeCommon.h](#)

9.138 nvinfer1::PluginFieldCollection Struct Reference

Plugin field collection struct.

```
#include <NvInferRuntimeCommon.h>
```

Public Attributes

- `int32_t nbFields`
Number of [PluginField](#) entries.
- `PluginField const * fields`
Pointer to [PluginField](#) entries.

9.138.1 Detailed Description

Plugin field collection struct.

9.138.2 Member Data Documentation

9.138.2.1 fields

```
PluginField const* nvinfer1::PluginFieldCollection::fields
```

Pointer to [PluginField](#) entries.

9.138.2.2 nbFields

```
int32_t nvinfer1::PluginFieldCollection::nbFields
```

Number of [PluginField](#) entries.

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

- [NvInferRuntimeCommon.h](#)

9.139 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 \(\)](#)

9.139.1 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.

9.139.2 Constructor & Destructor Documentation

9.139.2.1 PluginRegistrar()

```
template<typename T >
nvinfer1::PluginRegistrar< T >::PluginRegistrar ( ) [inline]
```

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

- [NvInferRuntime.h](#)

9.140 nvinfer1::safe::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 <NvInferSafeRuntime.h>
```

Public Member Functions

- [PluginRegistrar](#) ()

9.140.1 Detailed Description

```
template<typename T>
class nvinfer1::safe::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.

9.140.2 Constructor & Destructor Documentation

9.140.2.1 PluginRegistrar()

```
template<typename T >
nvinfer1::safe::PluginRegistrar< T >::PluginRegistrar ( ) [inline]
```

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

- [NvInferSafeRuntime.h](#)

9.141 nvinfer1::PluginTensorDesc Struct Reference

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

```
#include <NvInferRuntimeCommon.h>
```

Public Attributes

- [Dims](#) `dims`
Dimensions.
- [DataType](#) `type`
- [TensorFormat](#) `format`
Tensor format.
- float [scale](#)
Scale for INT8 data type.

9.141.1 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](#)

9.141.2 Member Data Documentation

9.141.2.1 dims

`Dims` `nvinfer1::PluginTensorDesc::dims`

Dimensions.

9.141.2.2 format

`TensorFormat` `nvinfer1::PluginTensorDesc::format`

Tensor format.

9.141.2.3 scale

`float` `nvinfer1::PluginTensorDesc::scale`

Scale for INT8 data type.

9.141.2.4 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:

- [NvInferRuntimeCommon.h](#)

9.142 PluginVersion Struct Reference

Definition of plugin versions.

```
#include <NvInferRuntimeCommon.h>
```

9.142.1 Detailed Description

Definition of plugin versions.

Tag for plug-in versions. Used in upper byte of `getTensorRTVersion()`.

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

- [NvInferRuntimeCommon.h](#)

9.143 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). [PriorBoxParameters](#) defines a set of parameters for creating the PriorBox plugin layer. It contains:

```
#include <NvInferPluginUtils.h>
```

Public Attributes

- float * [minSize](#)
- float * [maxSize](#)
- float * [aspectRatios](#)
- int32_t [numMinSize](#)
- int32_t [numMaxSize](#)
- int32_t [numAspectRatios](#)
- bool [flip](#)
- bool [clip](#)
- float [variance](#) [4]
- int32_t [imgH](#)
- int32_t [imgW](#)
- float [stepH](#)
- float [stepW](#)
- float [offset](#)

9.143.1 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. It contains:

Parameters

<i>minSize</i>	Minimum box size in pixels. Can not be nullptr.
<i>maxSize</i>	Maximum box size in pixels. Can be nullptr.
<i>aspectRatios</i>	Aspect ratios of the boxes. Can be nullptr.
<i>numMinSize</i>	Number of elements in minSize. Must be larger than 0.
<i>numMaxSize</i>	Number of elements in maxSize. Can be 0 or same as numMinSize.
<i>numAspectRatios</i>	Number of elements in aspectRatios. Can be 0.
<i>flip</i>	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.
<i>clip</i>	If true, will clip the prior so that it is within [0,1].
<i>variance</i>	Variance for adjusting the prior boxes.
<i>imgH</i>	Image height. If 0, then the H dimension of the data tensor will be used.
<i>imgW</i>	Image width. If 0, then the W dimension of the data tensor will be used.
<i>stepH</i>	Step in H. If 0, then (float)imgH/h will be used where h is the H dimension of the 1st input tensor.
<i>stepW</i>	Step in W. If 0, then (float)imgW/w will be used where w is the W dimension of the 1st input tensor.
<i>offset</i>	Offset to the top left corner of each cell.

9.143.2 Member Data Documentation

9.143.2.1 aspectRatios

```
float * nvinfer1::plugin::PriorBoxParameters::aspectRatios
```

9.143.2.2 clip

```
bool nvinfer1::plugin::PriorBoxParameters::clip
```

9.143.2.3 flip

```
bool nvinfer1::plugin::PriorBoxParameters::flip
```

9.143.2.4 imgH

```
int32_t nvinfer1::plugin::PriorBoxParameters::imgH
```

9.143.2.5 imgW

```
int32_t nvinfer1::plugin::PriorBoxParameters::imgW
```

9.143.2.6 maxSize

```
float * nvinfer1::plugin::PriorBoxParameters::maxSize
```

9.143.2.7 minSize

```
float* nvinfer1::plugin::PriorBoxParameters::minSize
```

9.143.2.8 numAspectRatios

```
int32_t nvinfer1::plugin::PriorBoxParameters::numAspectRatios
```

9.143.2.9 numMaxSize

```
int32_t nvinfer1::plugin::PriorBoxParameters::numMaxSize
```

9.143.2.10 numMinSize

```
int32_t nvinfer1::plugin::PriorBoxParameters::numMinSize
```

9.143.2.11 offset

```
float nvinfer1::plugin::PriorBoxParameters::offset
```

9.143.2.12 stepH

```
float nvinfer1::plugin::PriorBoxParameters::stepH
```

9.143.2.13 stepW

```
float nvinfer1::plugin::PriorBoxParameters::stepW
```

9.143.2.14 variance

```
float nvinfer1::plugin::PriorBoxParameters::variance[4]
```

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

- [NvInferPluginUtils.h](#)

9.144 nvinfer1::plugin::Quadruple Struct Reference

The Permute plugin layer permutes the input tensor by changing the memory order of the data. [Quadruple](#) defines a structure that contains an array of 4 integers. They can represent the permute orders or the strides in each dimension.

```
#include <NvInferPluginUtils.h>
```

Public Attributes

- int32_t [data](#) [4]

9.144.1 Detailed Description

The Permute plugin layer permutes the input tensor by changing the memory order of the data. [Quadruple](#) defines a structure that contains an array of 4 integers. They can represent the permute orders or the strides in each dimension.

9.144.2 Member Data Documentation

9.144.2.1 data

```
int32_t nvinfer1::plugin::Quadruple::data[4]
```

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

- [NvInferPluginUtils.h](#)

9.145 nvinfer1::plugin::RegionParameters Struct Reference

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.

```
#include <NvInferPluginUtils.h>
```

Public Attributes

- int32_t [num](#)
- int32_t [coords](#)
- int32_t [classes](#)
- [softmaxTree](#) * [smTree](#)

9.145.1 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.

Parameters

<i>num</i>	Number of predicted bounding box for each grid cell.
<i>coords</i>	Number of coordinates for a bounding box.
<i>classes</i>	Number of classifications to be predicted.
<i>smTree</i>	Helping structure to do softmax on confidence scores.

9.145.2 Member Data Documentation

9.145.2.1 classes

```
int32_t nvinfer1::plugin::RegionParameters::classes
```

9.145.2.2 coords

```
int32_t nvinfer1::plugin::RegionParameters::coords
```

9.145.2.3 num

```
int32_t nvinfer1::plugin::RegionParameters::num
```

9.145.2.4 smTree

```
softmaxTree* nvinfer1::plugin::RegionParameters::smTree
```

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

- [NvInferPluginUtils.h](#)

9.146 nvinfer1::plugin::RPROIParams Struct Reference

[RPROIParams](#) is used to create the RPROIPlugin instance. It contains:

```
#include <NvInferPluginUtils.h>
```

Public Attributes

- int32_t [poolingH](#)
- int32_t [poolingW](#)
- int32_t [featureStride](#)
- int32_t [preNmsTop](#)
- int32_t [nmsMaxOut](#)
- int32_t [anchorsRatioCount](#)
- int32_t [anchorsScaleCount](#)
- float [iouThreshold](#)
- float [minBoxSize](#)
- float [spatialScale](#)

9.146.1 Detailed Description

[RPROIParams](#) is used to create the RPROIPlugin instance. It contains:

Parameters

<i>poolingH</i>	Height of the output in pixels after ROI pooling on feature map.
<i>poolingW</i>	Width of the output in pixels after ROI pooling on feature map.
<i>featureStride</i>	Feature stride; ratio of input image size to feature map size. Assuming that max pooling layers in the neural network use square filters.
<i>preNmsTop</i>	Number of proposals to keep before applying NMS.
<i>nmsMaxOut</i>	Number of remaining proposals after applying NMS.
<i>anchorsRatioCount</i>	Number of anchor box ratios.
<i>anchorsScaleCount</i>	Number of anchor box scales.
<i>iouThreshold</i>	IoU (Intersection over Union) threshold used for the NMS step.
<i>minBoxSize</i>	Minimum allowed bounding box size before scaling, used for anchor box calculation.
<i>spatialScale</i>	Spatial scale between the input image and the last feature map.

9.146.2 Member Data Documentation

9.146.2.1 anchorsRatioCount

```
int32_t nvinfer1::plugin::RPROIParams::anchorsRatioCount
```

9.146.2.2 anchorsScaleCount

```
int32_t nvinfer1::plugin::RPROIParams::anchorsScaleCount
```

9.146.2.3 featureStride

```
int32_t nvinfer1::plugin::RPROIParams::featureStride
```

9.146.2.4 iouThreshold

```
float nvinfer1::plugin::RPROIParams::iouThreshold
```

9.146.2.5 minBoxSize

```
float nvinfer1::plugin::RPROIParams::minBoxSize
```

9.146.2.6 nmsMaxOut

```
int32_t nvinfer1::plugin::RPROIParams::nmsMaxOut
```

9.146.2.7 poolingH

```
int32_t nvinfer1::plugin::RPROIParams::poolingH
```

9.146.2.8 poolingW

```
int32_t nvinfer1::plugin::RPROIParams::poolingW
```

9.146.2.9 preNmsTop

```
int32_t nvinfer1::plugin::RPROIParams::preNmsTop
```

9.146.2.10 spatialScale

```
float nvinfer1::plugin::RPROIParams::spatialScale
```

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

- [NvInferPluginUtils.h](#)

9.147 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`

9.147.1 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.

9.147.2 Member Data Documentation

9.147.2.1 child

```
int32_t* nvinfer1::plugin::softmaxTree::child
```

9.147.2.2 group

```
int32_t* nvinfer1::plugin::softmaxTree::group
```

9.147.2.3 groupOffset

```
int32_t* nvinfer1::plugin::softmaxTree::groupOffset
```

9.147.2.4 groups

```
int32_t nvinfer1::plugin::softmaxTree::groups
```

9.147.2.5 groupSize

```
int32_t* nvinfer1::plugin::softmaxTree::groupSize
```

9.147.2.6 leaf

```
int32_t* nvinfer1::plugin::softmaxTree::leaf
```

9.147.2.7 n

```
int32_t nvinfer1::plugin::softmaxTree::n
```

9.147.2.8 name

```
char** nvinfer1::plugin::softmaxTree::name
```

9.147.2.9 parent

```
int32_t* nvinfer1::plugin::softmaxTree::parent
```

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

- [NvInferPluginUtils.h](#)

9.148 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.

9.148.1 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`).

9.148.2 Member Data Documentation

9.148.2.1 `count`

```
int64_t nvinfer1::Weights::count
```

The number of weights in the array.

9.148.2.2 `type`

```
DataType nvinfer1::Weights::type
```

The type of the weights.

9.148.2.3 `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 NvCaffeParser.h File Reference

```
#include "NvInfer.h"
```

Classes

- class [nvcaffeparser1::IBlobNameToTensor](#)
Object used to store and query Tensors after they have been extracted from a Caffe model using the [ICaffeParser](#).
- class [nvcaffeparser1::IBinaryProtoBlob](#)
Object used to store and query data extracted from a binaryproto file using the [ICaffeParser](#).
- class [nvcaffeparser1::IPluginFactoryV2](#)
Plugin factory used to configure plugins.
- class [nvcaffeparser1::ICaffeParser](#)
Class used for parsing Caffe models.

Namespaces

- namespace [nvcaffeparser1](#)
The TensorRT Caffe parser API namespace.

Functions

- [ICaffeParser * nvcaffeparser1::createCaffeParser \(\)](#) noexcept
Creates a [ICaffeParser](#) object.
- [void nvcaffeparser1::shutdownProtobufLibrary \(\)](#) noexcept
Shuts down protocol buffers library.

10.1.1 Detailed Description

This is the API for the Caffe Parser

10.2 NvCaffeParser.h

[Go to the documentation of this file.](#)

```

1  /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_CAFFE_PARSER_H
14 #define NV_CAFFE_PARSER_H
15
16 #include "NvInfer.h"
17
18
19 namespace nvcaffeparser1
20 {
21
22     class IBlobNameToTensor
23     {
24     public:
25         virtual nvinfer1::ITensor* find(char const* name) const noexcept = 0;
26
27     protected:
28         virtual ~IBlobNameToTensor() {}
29     };
30
31     class IBinaryProtoBlob
32     {
33     public:
34         virtual void const* getData() noexcept = 0;
35         virtual nvinfer1::Dims4 getDimensions() noexcept = 0;
36         virtual nvinfer1::DataType getDataType() noexcept = 0;
37         TRT_DEPRECATED virtual void destroy() noexcept = 0;
38         virtual ~IBinaryProtoBlob() noexcept = default;
39     };
40
41     class IPluginFactoryV2
42     {
43     public:
44         virtual bool isPluginV2(char const* layerName) noexcept = 0;
45
46         virtual nvinfer1::IPluginV2* createPlugin(char const* layerName, nvinfer1::Weights const* weights,
47             int32_t nbWeights, char const* libNamespace = "") noexcept = 0;
48
49         virtual ~IPluginFactoryV2() noexcept = default;
50     };
51
52     class ICaffeParser
53     {
54     public:
55         virtual IBlobNameToTensor const* parse(char const* deploy, char const* model,
56             nvinfer1::INetworkDefinition& network,
57             nvinfer1::DataType weightType) noexcept = 0;
58
59         virtual IBlobNameToTensor const* parseBuffers(uint8_t const* deployBuffer, std::size_t deployLength,
60             uint8_t const* modelBuffer, std::size_t modelLength, nvinfer1::INetworkDefinition& network,
61             nvinfer1::DataType weightType) noexcept = 0;
62
63         virtual IBinaryProtoBlob* parseBinaryProto(char const* fileName) noexcept = 0;
64
65         virtual void setProtobufBufferSize(size_t size) noexcept = 0;
66
67         TRT_DEPRECATED virtual void destroy() noexcept = 0;
68     };
69
70 }
71
72 
```

```

188
194     virtual void setPluginFactoryV2(IPluginFactoryV2* factory) noexcept = 0;
195
199     virtual void setPluginNamespace(char const* libNamespace) noexcept = 0;
200
201     virtual ~ICaffeParser() noexcept = default;
202
203 public:
218     virtual void setErrorRecorder(nvinfer1::IErrorRecorder* recorder) noexcept = 0;
219
230     virtual nvinfer1::IErrorRecorder* getErrorRecorder() const noexcept = 0;
231 };
232
243 TENSORRTAPI ICaffeParser* createCaffeParser() noexcept;
244
250 TENSORRTAPI void shutdownProtobufLibrary() noexcept;
251 } // namespace nvcaffeparser1
252
257 extern "C" TENSORRTAPI void* createNvCaffeParser_INTERNAL() noexcept;
258 #endif

```

10.3 NvInfer.h File Reference

```

#include "NvInferLegacyDims.h"
#include "NvInferRuntime.h"

```

Classes

- 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::IFullyConnectedLayer](#)
A fully connected layer in a network definition. This layer expects an input tensor of three or more non-batch dimensions. The input is automatically reshaped into an $M \times V$ tensor X , where V is a product of the last three dimensions and M is a product of the remaining dimensions (where the product over 0 dimensions is defined as 1). For example:
- 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.
- 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.
- class [nvinfer1::IGatherLayer](#)
A Gather layer in a network definition. Supports several kinds of gathering.
- class [nvinfer1::IRNNv2Layer](#)
An RNN layer in a network definition, version 2.
- class [nvinfer1::IPluginV2Layer](#)
Layer type for pluginV2.
- class [nvinfer1::IUnaryLayer](#)
Layer that represents an unary operation.
- 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](#)
- class [nvinfer1::IShuffleLayer](#)
Layer type for shuffling data.
- 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.
- class [nvinfer1::ITopKLayer](#)
Layer that represents a TopK reduction.
- 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.
- class [nvinfer1::ILoopBoundaryLayer](#)
- class [nvinfer1::IfConditionalBoundaryLayer](#)
- class [nvinfer1::IConditionLayer](#)

- class [nvinfer1::IfConditionalOutputLayer](#)
- class [nvinfer1::IfConditionalInputLayer](#)
- class [nvinfer1::IfConditional](#)
- class [nvinfer1::IRecurrenceLayer](#)
- class [nvinfer1::ILoopOutputLayer](#)
- class [nvinfer1::ITripLimitLayer](#)
- class [nvinfer1::IteratorLayer](#)
- class [nvinfer1::ILoop](#)
- class [nvinfer1::ISelectLayer](#)
- class [nvinfer1::IAssertionLayer](#)
An assertion layer in a network.
- class [nvinfer1::IFillLayer](#)
Generate an output tensor with specified mode.
- class [nvinfer1::IQuantizeLayer](#)
A Quantize layer in a network definition.
- class [nvinfer1::IDequantizeLayer](#)
A Dequantize layer in a network definition.
- class [nvinfer1::IEinsumLayer](#)
An Einsum layer in a network.
- 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.
- 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::INetworkDefinition](#)
A network definition for input to the builder.
- class [nvinfer1::IInt8Calibrator](#)
Application-implemented interface for calibration.
- class [nvinfer1::IInt8EntropyCalibrator](#)
- class [nvinfer1::IInt8EntropyCalibrator2](#)
- class [nvinfer1::IInt8MinMaxCalibrator](#)
- class [nvinfer1::IInt8LegacyCalibrator](#)
- class [nvinfer1::IAlgorithmIOInfo](#)
Carries information about input or output of the algorithm. [IAlgorithmIOInfo](#) for all the input and output along with [IAlgorithmVariant](#) denotes the variation of algorithm and can be used to select or reproduce an algorithm using [IAlgorithmSelector::selectAlgorithms\(\)](#).
- class [nvinfer1::IAlgorithmVariant](#)
provides a unique 128-bit identifier, which along with the input and output information denotes the variation of algorithm and can be used to select or reproduce an algorithm, using [IAlgorithmSelector::selectAlgorithms\(\)](#)
- class [nvinfer1::IAlgorithmContext](#)
Describes the context and requirements, that could be fulfilled by one or more instances of [IAlgorithm](#).
- class [nvinfer1::IAlgorithm](#)

Describes a variation of execution of a layer. An algorithm is represented by [IAAlgorithmVariant](#) and the [IAAlgorithmIOInfo](#) for each of its inputs and outputs. An algorithm can be selected or reproduced using `AlgorithmSelector::select↔Algorithms()`.”.

- class [nvinfer1::IAAlgorithmSelector](#)
Interface implemented by application for selecting and reporting algorithms of a layer provided by the builder.
- class [nvinfer1::ITimingCache](#)
Class to handle tactic timing info collected from builder.
- struct [nvinfer1::impl::EnumMaxImpl< PreviewFeature >](#)
- struct [nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel >](#)
- class [nvinfer1::IBuilderConfig](#)
Holds properties for configuring a builder to produce an engine.
- class [nvinfer1::IBuilder](#)
Builds an engine from a network definition.

Namespaces

- namespace [nvinfer1](#)
The TensorRT API version 1 namespace.
- namespace [nvinfer1::impl](#)

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::SliceMode](#) = SampleMode
- using [nvinfer1::ResizeMode](#) = InterpolationMode
- using [nvinfer1::QuantizationFlags](#) = uint32_t
Represents one or more QuantizationFlag values using binary OR operations.
- using [nvinfer1::BuilderFlags](#) = uint32_t
Represents one or more QuantizationFlag values using binary OR operations, e.g., `1U << BuilderFlag::kFP16 | 1U << BuilderFlag::kDEBUG`.
- using [nvinfer1::NetworkDefinitionCreationFlags](#) = uint32_t
Represents one or more NetworkDefinitionCreationFlag flags using binary OR operations. e.g., `1U << NetworkDefinitionCreationFlag::kEXPLICIT_BATCH`.

Enumerations

- enum class [nvinfer1::LayerType](#) : int32_t {
[nvinfer1::kCONVOLUTION](#) = 0 , [nvinfer1::kFULLY_CONNECTED](#) = 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::kRNN_V2 = 20 , nvinfer1::kIDENTITY = 21 , nvinfer1::kPLUGIN_V2 = 22 , nvinfer1::kSLICE = 23 ,
nvinfer1::kSHAPE = 24 , nvinfer1::kPARAMETRIC_RELU = 25 , nvinfer1::kRESIZE = 26 , nvinfer1::kTRIP_LIMIT
= 27 ,
nvinfer1::kRECURRENCE = 28 , nvinfer1::kITERATOR = 29 , nvinfer1::kLOOP_OUTPUT = 30 ,
nvinfer1::kSELECT = 31 ,
nvinfer1::kFILL = 32 , nvinfer1::kQUANTIZE = 33 , nvinfer1::kDEQUANTIZE = 34 , nvinfer1::kCONDITION
= 35 ,
nvinfer1::kCONDITIONAL_INPUT = 36 , nvinfer1::kCONDITIONAL_OUTPUT = 37 , nvinfer1::kSCATTER
= 38 , nvinfer1::kEINSUM = 39 ,
nvinfer1::kASSERTION = 40 , nvinfer1::kONE_HOT = 41 , nvinfer1::kNON_ZERO = 42 , nvinfer1::kGRID_SAMPLE
= 43 ,
nvinfer1::kNMS = 44 , nvinfer1::kREVERSE_SEQUENCE = 45 , nvinfer1::kNORMALIZATION = 46 ,
nvinfer1::kCAST = 47 }

```

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 }

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 ,
`nvinfer1::kCAFFE_ROUND_DOWN` = 4 , `nvinfer1::kCAFFE_ROUND_UP` = 5 }

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::RNNOperation` : `int32_t` { `nvinfer1::kRELU` = 0 , `nvinfer1::kTANH` = 1 , `nvinfer1::kLSTM`
= 2 , `nvinfer1::kGRU` = 3 }

Enumerates the RNN operations that may be performed by an RNN layer.

- enum class `nvinfer1::RNNDirection` : `int32_t` { `nvinfer1::kUNIDIRECTION` = 0 , `nvinfer1::kBIDIRECTION` =
1 }

Enumerates the RNN direction that may be performed by an RNN layer.

- enum class `nvinfer1::RNNInputMode` : `int32_t` { `nvinfer1::kLINEAR` = 0 , `nvinfer1::kSKIP` = 1 }

Enumerates the RNN input modes that may occur with an RNN layer.

- enum class `nvinfer1::RNNGateType` : `int32_t` {
`nvinfer1::kINPUT` = 0 , `nvinfer1::kOUTPUT` = 1 , `nvinfer1::kFORGET` = 2 , `nvinfer1::kUPDATE` = 3 ,
`nvinfer1::kRESET` = 4 , `nvinfer1::kCELL` = 5 , `nvinfer1::kHIDDEN` = 6 }

Identifies an individual gate within an RNN cell.

- 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 }

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 }

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

- enum class `nvinfer1::SampleMode` : `int32_t` {
`nvinfer1::kSTRICT_BOUNDS` = 0 , `nvinfer1::kDEFAULT` = `kSTRICT_BOUNDS` , `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` , `nvinfer1::kTRANSPOSE` , `nvinfer1::kVECTOR` }

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 be 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::CalibrationAlgoType` : `int32_t` { `nvinfer1::kLEGACY_CALIBRATION` = 0 ,
`nvinfer1::kENTROPY_CALIBRATION` = 1 , `nvinfer1::kENTROPY_CALIBRATION_2` = 2 , `nvinfer1::kMINMAX_CALIBRATION` = 3 }

Version of calibration algorithm to use.

- enum class `nvinfer1::QuantizationFlag` : `int32_t` { `nvinfer1::kCALIBRATE_BEFORE_FUSION` = 0 }

List of valid flags for quantizing the network to int8.

- enum class `nvinfer1::BuilderFlag` : `int32_t` {
`nvinfer1::kFP16` = 0 , `nvinfer1::kINT8` = 1 , `nvinfer1::kDEBUG` = 2 , `nvinfer1::kGPU_FALLBACK` = 3 ,
`nvinfer1::kSTRICT_TYPES` = 4 , `nvinfer1::kREFIT` = 5 , `nvinfer1::kDISABLE_TIMING_CACHE` = 6 ,
`nvinfer1::kTF32` = 7 ,
`nvinfer1::kSPARSE_WEIGHTS` = 8 , `nvinfer1::kSAFETY_SCOPE` = 9 , `nvinfer1::kOBEY_PRECISION_CONSTRAINTS`
= 10 , `nvinfer1::kPREFER_PRECISION_CONSTRAINTS` = 11 ,
`nvinfer1::kDIRECT_IO` = 12 , `nvinfer1::kREJECT_EMPTY_ALGORITHMS` = 13 , `nvinfer1::kENABLE_TACTIC_HEURISTIC`
= 14 , `nvinfer1::kVERSION_COMPATIBLE` = 15 ,
`nvinfer1::kEXCLUDE_LEAN_RUNTIME` = 16 , `nvinfer1::kFP8` = 17 }

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 }

The type for memory pools used by TensorRT.

- enum class `nvinfer1::PreviewFeature` : `int32_t` { `nvinfer1::kFASTER_DYNAMIC_SHAPES_0805` = 0 ,
`nvinfer1::kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805` = 1 , `nvinfer1::kPROFILE_SHARING_0806`
= 2 , `nvinfer1::kENABLE_MULTI_STREAM_ENGINE_0806` = 3 }

Define preview features.

- enum class `nvinfer1::HardwareCompatibilityLevel` : `int32_t` { `nvinfer1::kNONE` = 0 , `nvinfer1::kAMPERE_PLUS`
= 1 }
- enum class `nvinfer1::NetworkDefinitionCreationFlag` : `int32_t` { `nvinfer1::kEXPLICIT_BATCH` = 0 ,
`nvinfer1::kEXPLICIT_PRECISION` = 1 }

List of immutable network properties expressed at network creation time. NetworkDefinitionCreationFlag is used with createNetworkV2() to specify immutable properties of the network. Creating a network without NetworkDefinitionCreationFlag::kEXPLICIT_BATCH flag has been deprecated.

Functions

- template<> constexpr `int32_t nvinfer1::EnumMax< LayerType >()` noexcept
- template<> constexpr `int32_t nvinfer1::EnumMax< ScaleMode >()` noexcept
- template<> constexpr `int32_t nvinfer1::EnumMax< GatherMode >()` noexcept
- template<> constexpr `int32_t nvinfer1::EnumMax< RNNOperation >()` noexcept
- template<> constexpr `int32_t nvinfer1::EnumMax< RNNDirection >()` noexcept
- template<> constexpr `int32_t nvinfer1::EnumMax< RNNInputMode >()` noexcept
- template<> constexpr `int32_t nvinfer1::EnumMax< RNNGateType >()` noexcept
- template<> constexpr `int32_t nvinfer1::EnumMax< UnaryOperation >()` noexcept
- template<> constexpr `int32_t nvinfer1::EnumMax< ReduceOperation >()` noexcept
- template<> constexpr `int32_t nvinfer1::EnumMax< SampleMode >()` noexcept
- template<> constexpr `int32_t nvinfer1::EnumMax< TopKOperation >()` noexcept
- template<> constexpr `int32_t nvinfer1::EnumMax< MatrixOperation >()` noexcept
- template<> constexpr `int32_t nvinfer1::EnumMax< LoopOutput >()` noexcept
- template<> constexpr `int32_t nvinfer1::EnumMax< TripLimit >()` noexcept
- template<> constexpr `int32_t nvinfer1::EnumMax< FillOperation >()` noexcept
- template<> constexpr `int32_t nvinfer1::EnumMax< ScatterMode >()` noexcept
- template<> constexpr `int32_t nvinfer1::EnumMax< BoundingBoxFormat >()` noexcept
- template<> constexpr `int32_t nvinfer1::EnumMax< CalibrationAlgoType >()` noexcept

- `template<> constexpr int32_t nvinfer1::EnumMax< QuantizationFlag > () noexcept`
- `template<> constexpr int32_t nvinfer1::EnumMax< BuilderFlag > () noexcept`
- `template<> constexpr int32_t nvinfer1::EnumMax< MemoryPoolType > () noexcept`
- `template<> constexpr int32_t nvinfer1::EnumMax< NetworkDefinitionCreationFlag > () noexcept`
- `nvinfer1::IPluginRegistry * nvinfer1::getBuilderPluginRegistry (nvinfer1::EngineCapability capability) noexcept`

Return the plugin registry for the given capability or nullptr if no registry exists.

10.3.1 Detailed Description

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

This is the top-level API file for TensorRT.

10.4 NvInfer.h

Go to the documentation of this file.

```

1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_INFER_H
14 #define NV_INFER_H
15
16 #include "NvInferLegacyDims.h"
17 #include "NvInferRuntime.h"
18
19 //
20
21
22 namespace nvinfer1
23 {
24     enum class LayerType : int32_t
25     {
26         kCONVOLUTION = 0,
27         kFULLY.CONNECTED = 1,
28         kACTIVATION = 2,
29         kPOOLING = 3,
30         kLRN = 4,
31         kSCALE = 5,
32         kSOFTMAX = 6,
33         kDECONVOLUTION = 7,
34         kCONCATENATION = 8,
35         kELEMENTWISE = 9,
36         kPLUGIN = 10,
37         kUNARY = 11,
38         kPADDING = 12,
39         kSHUFFLE = 13,
40         kREDUCE = 14,
41         kTOPK = 15,
42         kGATHER = 16,
43         kMATRIX.MULTIPLY = 17,
44         kRAGGED.SOFTMAX = 18,
45         kCONSTANT = 19,
46         kRNN.V2 = 20,
47         kIDENTITY = 21,
48         kPLUGIN.V2 = 22,
49     };
50 }

```

```

77     kSLICE = 23,
78     kSHAPE = 24,
79     kPARAMETRIC_RELU = 25,
80     kRESIZE = 26,
81     kTRIP_LIMIT = 27,
82     kRECURRENCE = 28,
83     kITERATOR = 29,
84     kLOOP_OUTPUT = 30,
85     kSELECT = 31,
86     kFILL = 32,
87     kQUANTIZE = 33,
88     kDEQUANTIZE = 34,
89     kCONDITION = 35,
90     kCONDITIONAL_INPUT = 36,
91     kCONDITIONAL_OUTPUT = 37,
92     kSCATTER = 38,
93     kEINSUM = 39,
94     kASSERTION = 40,
95     kONE_HOT = 41,
96     kNON_ZERO = 42,
97     kGRID_SAMPLE = 43,
98     kNMS = 44,
99     kREVERSE_SEQUENCE = 45,
100    kNORMALIZATION = 46,
101    kCAST = 47,
102 };
103
104 template <>
105 constexpr inline int32_t EnumMax<LayerType>() noexcept
106 {
107     return 48;
108 }
109
110 using TensorFormats = uint32_t;
111
112 enum class ActivationType : int32_t
113 {
114     kRELU = 0,
115     kSIGMOID = 1,
116     kTANH = 2,
117     kLEAKY_RELU = 3,
118     kELU = 4,
119     kSELU = 5,
120     kSOFTSIGN = 6,
121     kSOFTPLUS = 7,
122     kCLIP = 8,
123     kHARD_SIGMOID = 9,
124     kSCALED_TANH = 10,
125     kTHRESHOLDED_RELU = 11
126 };
127
128 namespace impl
129 {
130     template <>
131     struct EnumMaxImpl<ActivationType>
132     {
133         static constexpr int32_t kVALUE = 12;
134     };
135 } // namespace impl
136
137 class ITensor : public INoCopy
138 {
139 public:
140     void setName(char const* name) noexcept
141     {
142         mImpl->setName(name);
143     }
144
145     char const* getName() const noexcept
146     {
147         return mImpl->getName();
148     }
149
150     void setDimensions(Dims dimensions) noexcept
151     {
152         mImpl->setDimensions(dimensions);
153     }
154
155     Dims getDimensions() const noexcept
156     {
157         return mImpl->getDimensions();
158     }

```

```

238     }
239
250     void setType(DataType type) noexcept
251     {
252         mImpl->setType(type);
253     }
254
262     DataType getType() const noexcept
263     {
264         return mImpl->getType();
265     }
266
277     bool setDynamicRange(float min, float max) noexcept
278     {
279         return mImpl->setDynamicRange(min, max);
280     }
281
285     bool isNetworkInput() const noexcept
286     {
287         return mImpl->isNetworkInput();
288     }
289
293     bool isNetworkOutput() const noexcept
294     {
295         return mImpl->isNetworkOutput();
296     }
297
315     void setBroadcastAcrossBatch(bool broadcastAcrossBatch) noexcept
316     {
317         mImpl->setBroadcastAcrossBatch(broadcastAcrossBatch);
318     }
319
331     bool getBroadcastAcrossBatch() const noexcept
332     {
333         return mImpl->getBroadcastAcrossBatch();
334     }
335
341     TensorLocation getLocation() const noexcept
342     {
343         return mImpl->getLocation();
344     }
345
356     void setLocation(TensorLocation location) noexcept
357     {
358         mImpl->setLocation(location);
359     }
360
366     bool dynamicRangeIsSet() const noexcept
367     {
368         return mImpl->dynamicRangeIsSet();
369     }
370
374     void resetDynamicRange() noexcept
375     {
376         mImpl->resetDynamicRange();
377     }
378
384     float getDynamicRangeMin() const noexcept
385     {
386         return mImpl->getDynamicRangeMin();
387     }
388
394     float getDynamicRangeMax() const noexcept
395     {
396         return mImpl->getDynamicRangeMax();
397     }
398
413     void setAllowedFormats(TensorFormats formats) noexcept
414     {
415         mImpl->setAllowedFormats(formats);
416     }
417
426     TensorFormats getAllowedFormats() const noexcept
427     {
428         return mImpl->getAllowedFormats();
429     }
430
461     bool isShapeTensor() const noexcept
462     {
463         return mImpl->isShapeTensor();
464     }

```

```

465
484     bool isExecutionTensor() const noexcept
485     {
486         return mImpl->isExecutionTensor();
487     }
488
510     void setDimensionName(int32_t index, char const* name) noexcept
511     {
512         mImpl->setDimensionName(index, name);
513     }
514
525     char const* getDimensionName(int32_t index) const noexcept
526     {
527         return mImpl->getDimensionName(index);
528     }
529
530 protected:
531     apiv::VTensor* mImpl;
532     virtual ~ITensor() noexcept = default;
533 };
534
542 class ILayer : public INoCopy
543 {
544 public:
550     LayerType getType() const noexcept
551     {
552         return mLayer->getType();
553     }
554
564     void setName(char const* name) noexcept
565     {
566         mLayer->setName(name);
567     }
568
574     char const* getName() const noexcept
575     {
576         return mLayer->getName();
577     }
578
582     int32_t getNbInputs() const noexcept
583     {
584         return mLayer->getNbInputs();
585     }
586
595     ITensor* getInput(int32_t index) const noexcept
596     {
597         return mLayer->getInput(index);
598     }
599
603     int32_t getNbOutputs() const noexcept
604     {
605         return mLayer->getNbOutputs();
606     }
607
614     ITensor* getOutput(int32_t index) const noexcept
615     {
616         return mLayer->getOutput(index);
617     }
618
631     void setInput(int32_t index, ITensor& tensor) noexcept
632     {
633         return mLayer->setInput(index, tensor);
634     }
635
659     void setPrecision(DataType dataType) noexcept
660     {
661         mLayer->setPrecision(dataType);
662     }
663
671     DataType getPrecision() const noexcept
672     {
673         return mLayer->getPrecision();
674     }
675
683     bool precisionIsSet() const noexcept
684     {
685         return mLayer->precisionIsSet();
686     }
687
693     void resetPrecision() noexcept
694     {

```



```

695     mLayer->resetPrecision();
696 }
697
731 void setOutputType(int32_t index, DataType dataType) noexcept
732 {
733     mLayer->setOutputType(index, dataType);
734 }
735
745 DataType getOutputType(int32_t index) const noexcept
746 {
747     return mLayer->getOutputType(index);
748 }
749
758 bool outputTypeIsSet(int32_t index) const noexcept
759 {
760     return mLayer->outputTypeIsSet(index);
761 }
762
770 void resetOutputType(int32_t index) noexcept
771 {
772     return mLayer->resetOutputType(index);
773 }
774
775 protected:
776     virtual ~ILayer() noexcept = default;
777     apiv::VLayer* mLayer;
778 };
779
1002 enum class PaddingMode : int32_t
1003 {
1004     kEXPLICIT_ROUND_DOWN = 0,
1005     kEXPLICIT_ROUND_UP = 1,
1006     kSAME_UPPER = 2,
1007     kSAME_LOWER = 3,
1008     kCAFFE_ROUND_DOWN = 4,
1009     kCAFFE_ROUND_UP = 5
1010 };
1011
1012 namespace impl
1013 {
1014     template <>
1020 struct EnumMaxImpl<PaddingMode>
1021 {
1022     static constexpr int32_t kVALUE = 6;
1023 };
1024 } // namespace impl
1025
1038 class IConvolutionLayer : public ILayer
1039 {
1040 public:
1050     TRT_DEPRECATED void setKernelSize(DimsHW kernelSize) noexcept
1051     {
1052         mImpl->setKernelSize(kernelSize);
1053     }
1054
1062     TRT_DEPRECATED DimsHW getKernelSize() const noexcept
1063     {
1064         return mImpl->getKernelSize();
1065     }
1066
1074     void setNbOutputMaps(int32_t nbOutputMaps) noexcept
1075     {
1076         mImpl->setNbOutputMaps(nbOutputMaps);
1077     }
1078
1084     int32_t getNbOutputMaps() const noexcept
1085     {
1086         return mImpl->getNbOutputMaps();
1087     }
1088
1100     TRT_DEPRECATED void setStride(DimsHW stride) noexcept
1101     {
1102         mImpl->setStride(stride);
1103     }
1104
1110     TRT_DEPRECATED DimsHW getStride() const noexcept
1111     {
1112         return mImpl->getStride();
1113     }
1114
1130     TRT_DEPRECATED void setPadding(DimsHW padding) noexcept

```

```

1131     {
1132         return mImpl->setPadding(padding);
1133     }
1134
1142     TRT_DEPRECATED DimsHW getPadding() const noexcept
1143     {
1144         return mImpl->getPadding();
1145     }
1146
1162     void setNbGroups(int32_t nbGroups) noexcept
1163     {
1164         mImpl->setNbGroups(nbGroups);
1165     }
1166
1172     int32_t getNbGroups() const noexcept
1173     {
1174         return mImpl->getNbGroups();
1175     }
1176
1186     void setKernelWeights(Weights weights) noexcept
1187     {
1188         mImpl->setKernelWeights(weights);
1189     }
1190
1196     Weights getKernelWeights() const noexcept
1197     {
1198         return mImpl->getKernelWeights();
1199     }
1200
1211     void setBiasWeights(Weights weights) noexcept
1212     {
1213         mImpl->setBiasWeights(weights);
1214     }
1215
1221     Weights getBiasWeights() const noexcept
1222     {
1223         return mImpl->getBiasWeights();
1224     }
1225
1237     TRT_DEPRECATED void setDilation(DimsHW dilation) noexcept
1238     {
1239         return mImpl->setDilation(dilation);
1240     }
1241
1249     TRT_DEPRECATED DimsHW getDilation() const noexcept
1250     {
1251         return mImpl->getDilation();
1252     }
1253
1266     void setPrePadding(Dims padding) noexcept
1267     {
1268         mImpl->setPrePadding(padding);
1269     }
1270
1276     Dims getPrePadding() const noexcept
1277     {
1278         return mImpl->getPrePadding();
1279     }
1280
1293     void setPostPadding(Dims padding) noexcept
1294     {
1295         mImpl->setPostPadding(padding);
1296     }
1297
1303     Dims getPostPadding() const noexcept
1304     {
1305         return mImpl->getPostPadding();
1306     }
1307
1317     void setPaddingMode(PaddingMode paddingMode) noexcept
1318     {
1319         mImpl->setPaddingMode(paddingMode);
1320     }
1321
1329     PaddingMode getPaddingMode() const noexcept
1330     {
1331         return mImpl->getPaddingMode();
1332     }
1333
1342     void setKernelSizeNd(Dims kernelSize) noexcept
1343     {

```

```

1344     mImpl->setKernelSizeNd(kernelSize);
1345 }
1346
1352 Dims getKernelSizeNd() const noexcept
1353 {
1354     return mImpl->getKernelSizeNd();
1355 }
1356
1367 void setStrideNd(Dims stride) noexcept
1368 {
1369     mImpl->setStrideNd(stride);
1370 }
1371
1377 Dims getStrideNd() const noexcept
1378 {
1379     return mImpl->getStrideNd();
1380 }
1381
1395 void setPaddingNd(Dims padding) noexcept
1396 {
1397     mImpl->setPaddingNd(padding);
1398 }
1399
1407 Dims getPaddingNd() const noexcept
1408 {
1409     return mImpl->getPaddingNd();
1410 }
1411
1421 void setDilationNd(Dims dilation) noexcept
1422 {
1423     mImpl->setDilationNd(dilation);
1424 }
1425
1431 Dims getDilationNd() const noexcept
1432 {
1433     return mImpl->getDilationNd();
1434 }
1435
1449 using ILayer::setInput;
1450
1451 protected:
1452     virtual ~IConvolutionLayer() noexcept = default;
1453     apiv::VConvolutionLayer* mImpl;
1454 };
1455
1487 class TRT_DEPRECATED IFullyConnectedLayer : public ILayer
1488 {
1489 public:
1497     void setNbOutputChannels(int32_t nbOutputs) noexcept
1498     {
1499         mImpl->setNbOutputChannels(nbOutputs);
1500     }
1501
1507     int32_t getNbOutputChannels() const noexcept
1508     {
1509         return mImpl->getNbOutputChannels();
1510     }
1511
1517     void setKernelWeights(Weights weights) noexcept
1518     {
1519         mImpl->setKernelWeights(weights);
1520     }
1521
1527     Weights getKernelWeights() const noexcept
1528     {
1529         return mImpl->getKernelWeights();
1530     }
1531
1539     void setBiasWeights(Weights weights) noexcept
1540     {
1541         mImpl->setBiasWeights(weights);
1542     }
1543
1549     Weights getBiasWeights() const noexcept
1550     {
1551         return mImpl->getBiasWeights();
1552     }
1553
1575     using ILayer::setInput;
1576
1577 protected:

```

```

1578     virtual ~IFullyConnectedLayer() noexcept = default;
1579     apiv::VFullyConnectedLayer* mImpl;
1580 };
1581
1582 class IActivationLayer : public ILayer
1583 {
1584 public:
1585     void setActivationType(ActivationType type) noexcept
1586     {
1587         mImpl->setActivationType(type);
1588     }
1589
1590     ActivationType getActivationType() const noexcept
1591     {
1592         return mImpl->getActivationType();
1593     }
1594
1595     void setAlpha(float alpha) noexcept
1596     {
1597         mImpl->setAlpha(alpha);
1598     }
1599
1600     void setBeta(float beta) noexcept
1601     {
1602         mImpl->setBeta(beta);
1603     }
1604
1605     float getAlpha() const noexcept
1606     {
1607         return mImpl->getAlpha();
1608     }
1609
1610     float getBeta() const noexcept
1611     {
1612         return mImpl->getBeta();
1613     }
1614
1615 protected:
1616     virtual ~IActivationLayer() noexcept = default;
1617     apiv::VActivationLayer* mImpl;
1618 };
1619
1620 enum class PoolingType : int32_t
1621 {
1622     kMAX = 0,           // Maximum over elements
1623     kAVERAGE = 1,      // Average over elements. If the tensor is padded, the count includes the
padding
1624     kMAX_AVERAGE_BLEND = 2 // Blending between max and average pooling: (1-blendFactor)*maxPool +
blendFactor*avgPool
1625 };
1626
1627 namespace impl
1628 {
1629     template <>
1630     struct EnumMaxImpl<PoolingType>
1631     {
1632         static constexpr int32_t kVALUE = 3;
1633     };
1634 } // namespace impl
1635
1636 class IPoolingLayer : public ILayer
1637 {
1638 public:
1639     void setPoolingType(PoolingType type) noexcept
1640     {
1641         mImpl->setPoolingType(type);
1642     }
1643
1644     PoolingType getPoolingType() const noexcept
1645     {
1646         return mImpl->getPoolingType();
1647     }
1648
1649     TRT_DEPRECATED void setWindowSize(DimsHW windowSize) noexcept
1650     {
1651         mImpl->setWindowSize(windowSize);
1652     }
1653
1654     TRT_DEPRECATED DimsHW getWindowSize() const noexcept
1655     {
1656         return mImpl->getWindowSize();
1657     }

```

```

1758     }
1759
1771     TRT_DEPRECATED void setStride(DimsHW stride) noexcept
1772     {
1773         mImpl->setStride(stride);
1774     }
1775
1783     TRT_DEPRECATED DimsHW getStride() const noexcept
1784     {
1785         return mImpl->getStride();
1786     }
1787
1799     TRT_DEPRECATED void setPadding(DimsHW padding) noexcept
1800     {
1801         mImpl->setPadding(padding);
1802     }
1803
1813     TRT_DEPRECATED DimsHW getPadding() const noexcept
1814     {
1815         return mImpl->getPadding();
1816     }
1817
1828     void setBlendFactor(float blendFactor) noexcept
1829     {
1830         mImpl->setBlendFactor(blendFactor);
1831     }
1832
1841     float getBlendFactor() const noexcept
1842     {
1843         return mImpl->getBlendFactor();
1844     }
1845
1858     void setAverageCountExcludesPadding(bool exclusive) noexcept
1859     {
1860         mImpl->setAverageCountExcludesPadding(exclusive);
1861     }
1862
1869     bool getAverageCountExcludesPadding() const noexcept
1870     {
1871         return mImpl->getAverageCountExcludesPadding();
1872     }
1873
1887     void setPrePadding(Dims padding) noexcept
1888     {
1889         mImpl->setPrePadding(padding);
1890     }
1891
1897     Dims getPrePadding() const noexcept
1898     {
1899         return mImpl->getPrePadding();
1900     }
1901
1915     void setPostPadding(Dims padding) noexcept
1916     {
1917         mImpl->setPostPadding(padding);
1918     }
1919
1925     Dims getPostPadding() const noexcept
1926     {
1927         return mImpl->getPostPadding();
1928     }
1929
1938     void setPaddingMode(PaddingMode paddingMode) noexcept
1939     {
1940         mImpl->setPaddingMode(paddingMode);
1941     }
1942
1949     PaddingMode getPaddingMode() const noexcept
1950     {
1951         return mImpl->getPaddingMode();
1952     }
1953
1962     void setWindowSizeNd(Dims windowSize) noexcept
1963     {
1964         mImpl->setWindowSizeNd(windowSize);
1965     }
1966
1972     Dims getWindowSizeNd() const noexcept
1973     {
1974         return mImpl->getWindowSizeNd();
1975     }

```

```

1976
1987     void setStrideNd(Dims stride) noexcept
1988     {
1989         mImpl->setStrideNd(stride);
1990     }
1991
1997     Dims getStrideNd() const noexcept
1998     {
1999         return mImpl->getStrideNd();
2000     }
2001
2016     void setPaddingNd(Dims padding) noexcept
2017     {
2018         mImpl->setPaddingNd(padding);
2019     }
2020
2028     Dims getPaddingNd() const noexcept
2029     {
2030         return mImpl->getPaddingNd();
2031     }
2032
2033 protected:
2034     virtual ~IPoolingLayer() noexcept = default;
2035     apiv::VPoolingLayer* mImpl;
2036 };
2037
2047 class ILRNLayer : public ILayer
2048 {
2049 public:
2059     void setWindowSize(int32_t windowSize) noexcept
2060     {
2061         mImpl->setWindowSize(windowSize);
2062     }
2063
2069     int32_t getWindowSize() const noexcept
2070     {
2071         return mImpl->getWindowSize();
2072     }
2073
2080     void setAlpha(float alpha) noexcept
2081     {
2082         mImpl->setAlpha(alpha);
2083     }
2084
2090     float getAlpha() const noexcept
2091     {
2092         return mImpl->getAlpha();
2093     }
2094
2101     void setBeta(float beta) noexcept
2102     {
2103         mImpl->setBeta(beta);
2104     }
2105
2111     float getBeta() const noexcept
2112     {
2113         return mImpl->getBeta();
2114     }
2115
2122     void setK(float k) noexcept
2123     {
2124         mImpl->setK(k);
2125     }
2126
2132     float getK() const noexcept
2133     {
2134         return mImpl->getK();
2135     }
2136
2137 protected:
2138     virtual ~ILRNLayer() noexcept = default;
2139     apiv::VLRNLayer* mImpl;
2140 };
2141
2147 enum class ScaleMode : int32_t
2148 {
2149     kUNIFORM = 0,
2150     kCHANNEL = 1,
2151     kELEMENTWISE = 2
2152 };
2153

```

```

2159 template <>
2160 constexpr inline int32_t EnumMax<ScaleMode>() noexcept
2161 {
2162     return 3;
2163 }
2164
2191 class IScaleLayer : public ILayer
2192 {
2193 public:
2199     void setMode(ScaleMode mode) noexcept
2200     {
2201         mImpl->setMode(mode);
2202     }
2203
2209     ScaleMode getMode() const noexcept
2210     {
2211         return mImpl->getMode();
2212     }
2213
2219     void setShift(Weights shift) noexcept
2220     {
2221         mImpl->setShift(shift);
2222     }
2223
2229     Weights getShift() const noexcept
2230     {
2231         return mImpl->getShift();
2232     }
2233
2239     void setScale(Weights scale) noexcept
2240     {
2241         mImpl->setScale(scale);
2242     }
2243
2249     Weights getScale() const noexcept
2250     {
2251         return mImpl->getScale();
2252     }
2253
2259     void setPower(Weights power) noexcept
2260     {
2261         mImpl->setPower(power);
2262     }
2263
2269     Weights getPower() const noexcept
2270     {
2271         return mImpl->getPower();
2272     }
2273
2284     int32_t getChannelAxis() const noexcept
2285     {
2286         return mImpl->getChannelAxis();
2287     }
2288
2305     void setChannelAxis(int32_t channelAxis) noexcept
2306     {
2307         mImpl->setChannelAxis(channelAxis);
2308     }
2309
2310 protected:
2311     virtual ~IScaleLayer() noexcept = default;
2312     apiv::VScaleLayer* mImpl;
2313 };
2314
2336 class ISoftMaxLayer : public ILayer
2337 {
2338 public:
2369     void setAxes(uint32_t axes) noexcept
2370     {
2371         mImpl->setAxes(axes);
2372     }
2373
2379     uint32_t getAxes() const noexcept
2380     {
2381         return mImpl->getAxes();
2382     }
2383
2384 protected:
2385     virtual ~ISoftMaxLayer() noexcept = default;
2386     apiv::VSoftMaxLayer* mImpl;
2387 };

```

```

2388
2401 class IConcatenationLayer : public ILayer
2402 {
2403 public:
2416     void setAxis(int32_t axis) noexcept
2417     {
2418         mImpl->setAxis(axis);
2419     }
2420
2426     int32_t getAxis() const noexcept
2427     {
2428         return mImpl->getAxis();
2429     }
2430
2431 protected:
2432     virtual ~IConcatenationLayer() noexcept = default;
2433     apiv::VConcatenationLayer* mImpl;
2434 };
2435
2443 class IDeconvolutionLayer : public ILayer
2444 {
2445 public:
2457     TRT_DEPRECATED void setKernelSize(DimsHW kernelSize) noexcept
2458     {
2459         mImpl->setKernelSize(kernelSize);
2460     }
2461
2469     TRT_DEPRECATED DimsHW getKernelSize() const noexcept
2470     {
2471         return mImpl->getKernelSize();
2472     }
2473
2481     void setNbOutputMaps(int32_t nbOutputMaps) noexcept
2482     {
2483         mImpl->setNbOutputMaps(nbOutputMaps);
2484     }
2485
2491     int32_t getNbOutputMaps() const noexcept
2492     {
2493         return mImpl->getNbOutputMaps();
2494     }
2495
2507     TRT_DEPRECATED void setStride(DimsHW stride) noexcept
2508     {
2509         mImpl->setStride(stride);
2510     }
2511
2519     TRT_DEPRECATED DimsHW getStride() const noexcept
2520     {
2521         return mImpl->getStride();
2522     }
2523
2539     TRT_DEPRECATED void setPadding(DimsHW padding) noexcept
2540     {
2541         mImpl->setPadding(padding);
2542     }
2543
2553     TRT_DEPRECATED DimsHW getPadding() const noexcept
2554     {
2555         return mImpl->getPadding();
2556     }
2557
2573     void setNbGroups(int32_t nbGroups) noexcept
2574     {
2575         mImpl->setNbGroups(nbGroups);
2576     }
2577
2583     int32_t getNbGroups() const noexcept
2584     {
2585         return mImpl->getNbGroups();
2586     }
2587
2597     void setKernelWeights(Weights weights) noexcept
2598     {
2599         mImpl->setKernelWeights(weights);
2600     }
2601
2607     Weights getKernelWeights() const noexcept
2608     {
2609         return mImpl->getKernelWeights();
2610     }

```



```

2611
2622 void setBiasWeights(Weights weights) noexcept
2623 {
2624     mImpl->setBiasWeights(weights);
2625 }
2626
2632 Weights getBiasWeights() const noexcept
2633 {
2634     return mImpl->getBiasWeights();
2635 }
2636
2650 void setPrePadding(Dims padding) noexcept
2651 {
2652     mImpl->setPrePadding(padding);
2653 }
2654
2660 Dims getPrePadding() const noexcept
2661 {
2662     return mImpl->getPrePadding();
2663 }
2664
2678 void setPostPadding(Dims padding) noexcept
2679 {
2680     mImpl->setPostPadding(padding);
2681 }
2682
2688 Dims getPostPadding() const noexcept
2689 {
2690     return mImpl->getPostPadding();
2691 }
2692
2702 void setPaddingMode(PaddingMode paddingMode) noexcept
2703 {
2704     mImpl->setPaddingMode(paddingMode);
2705 }
2706
2714 PaddingMode getPaddingMode() const noexcept
2715 {
2716     return mImpl->getPaddingMode();
2717 }
2718
2729 void setKernelSizeNd(Dims kernelSize) noexcept
2730 {
2731     mImpl->setKernelSizeNd(kernelSize);
2732 }
2733
2739 Dims getKernelSizeNd() const noexcept
2740 {
2741     return mImpl->getKernelSizeNd();
2742 }
2743
2756 void setStrideNd(Dims stride) noexcept
2757 {
2758     mImpl->setStrideNd(stride);
2759 }
2760
2766 Dims getStrideNd() const noexcept
2767 {
2768     return mImpl->getStrideNd();
2769 }
2770
2784 void setPaddingNd(Dims padding) noexcept
2785 {
2786     mImpl->setPaddingNd(padding);
2787 }
2788
2796 Dims getPaddingNd() const noexcept
2797 {
2798     return mImpl->getPaddingNd();
2799 }
2800
2812 using ILayer::setInput;
2813
2820 void setDilationNd(Dims dilation) noexcept
2821 {
2822     mImpl->setDilationNd(dilation);
2823 }
2824
2830 Dims getDilationNd() const noexcept
2831 {
2832     return mImpl->getDilationNd();

```

```

2833     }
2834
2835 protected:
2836     virtual ~IDeconvolutionLayer() noexcept = default;
2837     apiv::VDeconvolutionLayer* mImpl;
2838 };
2839
2853 enum class ElementWiseOperation : int32_t
2854 {
2855     kSUM = 0,
2856     kPROD = 1,
2857     kMAX = 2,
2858     kMIN = 3,
2859     kSUB = 4,
2860     kDIV = 5,
2861     kPOW = 6,
2862     kFLOOR_DIV = 7,
2863     kAND = 8,
2864     kOR = 9,
2865     kXOR = 10,
2866     kEQUAL = 11,
2867     kGREATER = 12,
2868     kLESS = 13
2869 };
2870
2871 namespace impl
2872 {
2873     template <>
2874     struct EnumMaxImpl<ElementWiseOperation>
2875     {
2876         static constexpr int32_t kVALUE = 14;
2877     };
2878 } // namespace impl
2884
2904 class IElementWiseLayer : public ILayer
2905 {
2906 public:
2916     void setOperation(ElementWiseOperation op) noexcept
2917     {
2918         return mImpl->setOperation(op);
2919     }
2920
2928     ElementWiseOperation getOperation() const noexcept
2929     {
2930         return mImpl->getOperation();
2931     }
2932
2933 protected:
2934     apiv::VElementWiseLayer* mImpl;
2935     virtual ~IElementWiseLayer() noexcept = default;
2936 };
2937
2943 enum class GatherMode : int32_t
2944 {
2945     kDEFAULT = 0,
2946     kELEMENT = 1,
2947     kND = 2
2948 };
2949
2955 template <>
2956 constexpr inline int32_t EnumMax<GatherMode>() noexcept
2957 {
2958     return 3;
2959 }
2960
3039 class IGatherLayer : public ILayer
3040 {
3041 public:
3051     void setGatherAxis(int32_t axis) noexcept
3052     {
3053         mImpl->setGatherAxis(axis);
3054     }
3055
3062     int32_t getGatherAxis() const noexcept
3063     {
3064         return mImpl->getGatherAxis();
3065     }
3066
3083     void setNbElementWiseDims(int32_t elementWiseDims) noexcept
3084     {
3085         mImpl->setNbElementWiseDims(elementWiseDims);

```

```

3086     }
3087
3093     int32_t getNbElementWiseDims() const noexcept
3094     {
3095         return mImpl->getNbElementWiseDims();
3096     }
3097
3103     void setMode(GatherMode mode) noexcept
3104     {
3105         mImpl->setMode(mode);
3106     }
3107
3113     GatherMode getMode() const noexcept
3114     {
3115         return mImpl->getMode();
3116     }
3117
3118 protected:
3119     apiv::VGatherLayer* mImpl;
3120     virtual ~IGatherLayer() noexcept = default;
3121 };
3122
3202 enum class RNNOperation : int32_t
3203 {
3204     kRELU = 0,
3205     kTANH = 1,
3206     kLSTM = 2,
3207     kGRU = 3
3208 };
3209
3215 template <>
3216 constexpr inline int32_t EnumMax<RNNOperation>() noexcept
3217 {
3218     return 4;
3219 }
3220
3228 enum class RNNDirection : int32_t
3229 {
3230     kUNIDIRECTION = 0,
3231     kBIDIRECTION = 1
3232 };
3233
3239 template <>
3240 constexpr inline int32_t EnumMax<RNNDirection>() noexcept
3241 {
3242     return 2;
3243 }
3244
3260 enum class RNNInputMode : int32_t
3261 {
3262     kLINEAR = 0,
3263     kSKIP = 1
3264 };
3265
3271 template <>
3272 constexpr inline int32_t EnumMax<RNNInputMode>() noexcept
3273 {
3274     return 2;
3275 }
3276
3284 enum class RNNGateType : int32_t
3285 {
3286     kINPUT = 0,
3287     kOUTPUT = 1,
3288     kFORGET = 2,
3289     kUPDATE = 3,
3290     kRESET = 4,
3291     kCELL = 5,
3292     kHIDDEN = 6
3293 };
3294
3300 template <>
3301 constexpr inline int32_t EnumMax<RNNGateType>() noexcept
3302 {
3303     return 7;
3304 }
3305
3318 class TRT_DEPRECATED IRNNv2Layer : public ILayer
3319 {
3320 public:
3321     int32_t getLayerCount() const noexcept

```

```

3322     {
3323         return mImpl->getLayerCount();
3324     }
3325     int32_t getHiddenSize() const noexcept
3326     {
3327         return mImpl->getHiddenSize();
3328     }
3329     int32_t getMaxSeqLength() const noexcept
3330     {
3331         return mImpl->getMaxSeqLength();
3332     }
3333     int32_t getDataLength() const noexcept
3334     {
3335         return mImpl->getDataLength();
3336     }
3337
3352     void setSequenceLengths(ITensor& seqLengths) noexcept
3353     {
3354         return mImpl->setSequenceLengths(seqLengths);
3355     }
3356
3364     ITensor* getSequenceLengths() const noexcept
3365     {
3366         return mImpl->getSequenceLengths();
3367     }
3368
3374     void setOperation(RNNOperation op) noexcept
3375     {
3376         mImpl->setOperation(op);
3377     }
3378
3384     RNNOperation getOperation() const noexcept
3385     {
3386         return mImpl->getOperation();
3387     }
3388
3394     void setInputMode(RNNInputMode op) noexcept
3395     {
3396         mImpl->setInputMode(op);
3397     }
3398
3404     RNNInputMode getInputMode() const noexcept
3405     {
3406         return mImpl->getInputMode();
3407     }
3408
3419     void setDirection(RNNDirection op) noexcept
3420     {
3421         mImpl->setDirection(op);
3422     }
3423
3429     RNNDirection getDirection() const noexcept
3430     {
3431         return mImpl->getDirection();
3432     }
3433
3488     void setWeightsForGate(int32_t layerIndex, RNNGateType gate, bool isW, Weights weights) noexcept
3489     {
3490         mImpl->setWeightsForGate(layerIndex, gate, isW, weights);
3491     }
3492
3498     Weights getWeightsForGate(int32_t layerIndex, RNNGateType gate, bool isW) const noexcept
3499     {
3500         return mImpl->getWeightsForGate(layerIndex, gate, isW);
3501     }
3502
3523     void setBiasForGate(int32_t layerIndex, RNNGateType gate, bool isW, Weights bias) noexcept
3524     {
3525         mImpl->setBiasForGate(layerIndex, gate, isW, bias);
3526     }
3527
3533     Weights getBiasForGate(int32_t layerIndex, RNNGateType gate, bool isW) const noexcept
3534     {
3535         return mImpl->getBiasForGate(layerIndex, gate, isW);
3536     }
3537
3550     void setHiddenState(ITensor& hidden) noexcept
3551     {
3552         mImpl->setHiddenState(hidden);
3553     }
3554

```

```

3560     ITensor* getHiddenState() const noexcept
3561     {
3562         return mImpl->getHiddenState();
3563     }
3564
3565     void setCellState(ITensor& cell) noexcept
3566     {
3567         mImpl->setCellState(cell);
3568     }
3569
3570     ITensor* getCellState() const noexcept
3571     {
3572         return mImpl->getCellState();
3573     }
3574
3575 protected:
3576     apiv::VRNNv2Layer* mImpl;
3577     virtual ~VRNNv2Layer() noexcept = default;
3578 };
3579
3580 class IPluginV2Layer : public ILayer
3581 {
3582 public:
3583     IPluginV2& getPlugin() noexcept
3584     {
3585         return mImpl->getPlugin();
3586     }
3587
3588 protected:
3589     apiv::VPluginV2Layer* mImpl;
3590     virtual ~IPluginV2Layer() noexcept = default;
3591 };
3592
3593 enum class UnaryOperation : int32_t
3594 {
3595     kEXP = 0,
3596     kLOG = 1,
3597     kSQRT = 2,
3598     kRECIP = 3,
3599     kABS = 4,
3600     kNEG = 5,
3601     kSIN = 6,
3602     kCOS = 7,
3603     kTAN = 8,
3604     kSINH = 9,
3605     kCOSH = 10,
3606     kASIN = 11,
3607     kACOS = 12,
3608     kATAN = 13,
3609     kASINH = 14,
3610     kACOSH = 15,
3611     kATANH = 16,
3612     kCEIL = 17,
3613     kFLOOR = 18,
3614     kERF = 19,
3615     kNOT = 20,
3616     kSIGN = 21,
3617     kROUND = 22,
3618     kISINF = 23,
3619 };
3620
3621 template <>
3622 constexpr inline int32_t EnumMax<UnaryOperation>() noexcept
3623 {
3624     return 24;
3625 }
3626
3627 class IUnaryLayer : public ILayer
3628 {
3629 public:
3630     void setOperation(UnaryOperation op) noexcept
3631     {
3632         mImpl->setOperation(op);
3633     }
3634
3635     UnaryOperation getOperation() const noexcept
3636     {
3637         return mImpl->getOperation();
3638     }
3639
3640 protected:

```

```

3713     apiv::VUnaryLayer* mImpl;
3714     virtual ~IUnaryLayer() noexcept = default;
3715 };
3716
3717 enum class ReduceOperation : int32_t
3718 {
3719     kSUM = 0,
3720     kPROD = 1,
3721     kMAX = 2,
3722     kMIN = 3,
3723     kAVG = 4
3724 };
3725
3726 template <>
3727 constexpr inline int32_t EnumMax<ReduceOperation>() noexcept
3728 {
3729     return 5;
3730 }
3731
3732 class IReduceLayer : public ILayer
3733 {
3734 public:
3735     void setOperation(ReduceOperation op) noexcept
3736     {
3737         mImpl->setOperation(op);
3738     }
3739
3740     ReduceOperation getOperation() const noexcept
3741     {
3742         return mImpl->getOperation();
3743     }
3744
3745     void setReduceAxes(uint32_t reduceAxes) noexcept
3746     {
3747         mImpl->setReduceAxes(reduceAxes);
3748     }
3749
3750     uint32_t getReduceAxes() const noexcept
3751     {
3752         return mImpl->getReduceAxes();
3753     }
3754
3755     void setKeepDimensions(bool keepDimensions) noexcept
3756     {
3757         mImpl->setKeepDimensions(keepDimensions);
3758     }
3759
3760     bool getKeepDimensions() const noexcept
3761     {
3762         return mImpl->getKeepDimensions();
3763     }
3764
3765 protected:
3766     apiv::VReduceLayer* mImpl;
3767     virtual ~IReduceLayer() noexcept = default;
3768 };
3769
3770 class IPaddingLayer : public ILayer
3771 {
3772 public:
3773     TRT_DEPRECATED void setPrePadding(DimsHW padding) noexcept
3774     {
3775         mImpl->setPrePadding(padding);
3776     }
3777
3778     TRT_DEPRECATED DimsHW getPrePadding() const noexcept
3779     {
3780         return mImpl->getPrePadding();
3781     }
3782
3783     TRT_DEPRECATED void setPostPadding(DimsHW padding) noexcept
3784     {
3785         mImpl->setPostPadding(padding);
3786     }
3787
3788     TRT_DEPRECATED DimsHW getPostPadding() const noexcept
3789     {
3790         return mImpl->getPostPadding();
3791     }
3792
3793     void setPrePaddingNd(Dims padding) noexcept

```

```

3905     {
3906         mImpl->setPrePaddingNd(padding);
3907     }
3908
3916     Dims getPrePaddingNd() const noexcept
3917     {
3918         return mImpl->getPrePaddingNd();
3919     }
3920
3930     void setPostPaddingNd(Dims padding) noexcept
3931     {
3932         mImpl->setPostPaddingNd(padding);
3933     }
3934
3942     Dims getPostPaddingNd() const noexcept
3943     {
3944         return mImpl->getPostPaddingNd();
3945     }
3946
3947 protected:
3948     apiv::VPaddingLayer* mImpl;
3949     virtual ~IPaddingLayer() noexcept = default;
3950 };
3951
3952 struct Permutation
3953 {
3954     int32_t order[Dims::MAX_DIMS];
3955 };
3956
3975 class IShuffleLayer : public ILayer
3976 {
3977 public:
3978     void setFirstTranspose(Permutation permutation) noexcept
3979     {
3980         mImpl->setFirstTranspose(permutation);
3981     }
3982
3991     Permutation getFirstTranspose() const noexcept
3992     {
3993         return mImpl->getFirstTranspose();
3994     }
3995
4003     void setReshapeDimensions(Dims dimensions) noexcept
4004     {
4005         mImpl->setReshapeDimensions(dimensions);
4006     }
4007
4037     Dims getReshapeDimensions() const noexcept
4038     {
4039         return mImpl->getReshapeDimensions();
4040     }
4041
4047     //
4070     using ILayer::setInput;
4071
4084     void setSecondTranspose(Permutation permutation) noexcept
4085     {
4086         mImpl->setSecondTranspose(permutation);
4087     }
4088
4096     Permutation getSecondTranspose() const noexcept
4097     {
4098         return mImpl->getSecondTranspose();
4099     }
4100
4112     void setZeroIsPlaceholder(bool zeroIsPlaceholder) noexcept
4113     {
4114         return mImpl->setZeroIsPlaceholder(zeroIsPlaceholder);
4115     }
4116
4125     bool getZeroIsPlaceholder() const noexcept
4126     {
4127         return mImpl->getZeroIsPlaceholder();
4128     }
4129
4130 protected:
4131     apiv::VShuffleLayer* mImpl;
4132     virtual ~IShuffleLayer() noexcept = default;
4133 };
4134
4140 enum class SampleMode : int32_t

```

```

4141 {
4142     kSTRICT_BOUNDS = 0,
4143     kDEFAULT_TRT_DEPRECATED_ENUM = kSTRICT_BOUNDS,
4144     kWRAP = 1,
4145     kCLAMP = 2,
4146     kFILL = 3,
4147     kREFLECT = 4,
4150 };
4151
4153 using SliceMode = SampleMode;
4154
4160 template <>
4161 constexpr inline int32_t EnumMax<SampleMode>() noexcept
4162 {
4163     return 5;
4164 }
4165
4208 class ISliceLayer : public ILayer
4209 {
4210 public:
4220     void setStart(Dims start) noexcept
4221     {
4222         mImpl->setStart(start);
4223     }
4224
4235     Dims getStart() const noexcept
4236     {
4237         return mImpl->getStart();
4238     }
4239
4249     void setSize(Dims size) noexcept
4250     {
4251         return mImpl->setSize(size);
4252     }
4253
4264     Dims getSize() const noexcept
4265     {
4266         return mImpl->getSize();
4267     }
4268
4278     void setStride(Dims stride) noexcept
4279     {
4280         mImpl->setStride(stride);
4281     }
4282
4293     Dims getStride() const noexcept
4294     {
4295         return mImpl->getStride();
4296     }
4297
4303     void setMode(SliceMode mode) noexcept
4304     {
4305         mImpl->setMode(mode);
4306     }
4307
4313     SliceMode getMode() const noexcept
4314     {
4315         return mImpl->getMode();
4316     }
4317
4341     using ILayer::setInput;
4342
4343 protected:
4344     apiv::VSliceLayer* mImpl;
4345     virtual ~ISliceLayer() noexcept = default;
4346 };
4347
4360 class IShapeLayer : public ILayer
4361 {
4362 protected:
4363     apiv::VShapeLayer* mImpl;
4364     virtual ~IShapeLayer() noexcept = default;
4365 };
4366
4372 enum class TopKOperation : int32_t
4373 {
4374     kMAX = 0,
4375     kMIN = 1,
4376 };
4377
4383 template <>

```



```

4384 constexpr inline int32_t EnumMax<TopKOperation>() noexcept
4385 {
4386     return 2;
4387 }
4388
4400 class ITopKLayer : public ILayer
4401 {
4402 public:
4408     void setOperation(TopKOperation op) noexcept
4409     {
4410         mImpl->setOperation(op);
4411     }
4412
4418     TopKOperation getOperation() const noexcept
4419     {
4420         return mImpl->getOperation();
4421     }
4422
4432     void setK(int32_t k) noexcept
4433     {
4434         mImpl->setK(k);
4435     }
4436
4446     int32_t getK() const noexcept
4447     {
4448         return mImpl->getK();
4449     }
4450
4456     void setReduceAxes(uint32_t reduceAxes) noexcept
4457     {
4458         mImpl->setReduceAxes(reduceAxes);
4459     }
4460
4466     uint32_t getReduceAxes() const noexcept
4467     {
4468         return mImpl->getReduceAxes();
4469     }
4470
4485     using ILayer::setInput;
4486
4487 protected:
4488     apiv::VTopKLayer* mImpl;
4489     virtual ~ITopKLayer() noexcept = default;
4490 };
4491
4498 enum class MatrixOperation : int32_t
4499 {
4503     kNONE,
4504
4506     kTRANPOSE,
4507
4518     kVECTOR
4519 };
4520
4526 template <>
4527 constexpr inline int32_t EnumMax<MatrixOperation>() noexcept
4528 {
4529     return 3;
4530 }
4531
4557 class IMatrixMultiplyLayer : public ILayer
4558 {
4559 public:
4566     void setOperation(int32_t index, MatrixOperation op) noexcept
4567     {
4568         mImpl->setOperation(index, op);
4569     }
4570
4578     MatrixOperation getOperation(int32_t index) const noexcept
4579     {
4580         return mImpl->getOperation(index);
4581     }
4582
4583 protected:
4584     apiv::VMatrixMultiplyLayer* mImpl;
4585     virtual ~IMatrixMultiplyLayer() noexcept = default;
4586 };
4587
4609 class INonZeroLayer : public ILayer
4610 {
4611 protected:

```

```

4612     virtual ~INonZeroLayer() noexcept = default;
4613     apiv::VNonZeroLayer* mImpl;
4614 };
4615
4630 class IRaggedSoftMaxLayer : public ILayer
4631 {
4632 protected:
4633     apiv::VRaggedSoftMaxLayer* mImpl;
4634     virtual ~IRaggedSoftMaxLayer() noexcept = default;
4635 };
4636
4664 class IIdentityLayer : public ILayer
4665 {
4666 protected:
4667     apiv::VIdentityLayer* mImpl;
4668     virtual ~IIdentityLayer() noexcept = default;
4669 };
4670
4677 class ICastLayer : public ILayer
4678 {
4679 public:
4683     void setType(DataType toType) noexcept
4684     {
4685         mImpl->setType(toType);
4686     }
4687
4691     DataType getToType() const noexcept
4692     {
4693         return mImpl->getToType();
4694     }
4695
4696 protected:
4697     apiv::VCastLayer* mImpl;
4698     virtual ~ICastLayer() noexcept = default;
4699 };
4700
4709 class IConstantLayer : public ILayer
4710 {
4711 public:
4721     void setWeights(Weights weights) noexcept
4722     {
4723         mImpl->setWeights(weights);
4724     }
4725
4731     Weights getWeights() const noexcept
4732     {
4733         return mImpl->getWeights();
4734     }
4735
4743     void setDimensions(Dims dimensions) noexcept
4744     {
4745         mImpl->setDimensions(dimensions);
4746     }
4747
4755     Dims getDimensions() const noexcept
4756     {
4757         return mImpl->getDimensions();
4758     }
4759
4760 protected:
4761     apiv::VConstantLayer* mImpl;
4762     virtual ~IConstantLayer() noexcept = default;
4763 };
4764
4774 class IParametricReLULayer : public ILayer
4775 {
4776 protected:
4777     apiv::VParametricReLULayer* mImpl;
4778     virtual ~IParametricReLULayer() noexcept = default;
4779 };
4780
4786 enum class InterpolationMode : int32_t
4787 {
4788     kNEAREST = 0,
4789     kLINEAR = 1,
4790     kCUBIC = 2
4791 };
4792
4794 using ResizeMode = InterpolationMode;
4795
4796 namespace impl

```

```

4797 {
4803 template <>
4804 struct EnumMaxImpl<InterpolationMode>
4805 {
4806     static constexpr int32_t kVALUE = 3;
4807 };
4808 } // namespace impl
4809
4817 enum class ResizeCoordinateTransformation : int32_t
4818 {
4831     kALIGN_CORNERS = 0,
4832
4839     kASYMMETRIC = 1,
4840
4847     kHALF_PIXEL = 2,
4848 };
4849
4850 namespace impl
4851 {
4857 template <>
4858 struct EnumMaxImpl<ResizeCoordinateTransformation>
4859 {
4860     static constexpr int32_t kVALUE = 3;
4861 };
4862 } // namespace impl
4863
4871 enum class ResizeSelector : int32_t
4872 {
4874     kFORMULA = 0,
4875
4877     kUPPER = 1,
4878 };
4879
4880 namespace impl
4881 {
4887 template <>
4888 struct EnumMaxImpl<ResizeSelector>
4889 {
4890     static constexpr int32_t kVALUE = 2;
4891 };
4892 } // namespace impl
4893
4901 enum class ResizeRoundMode : int32_t
4902 {
4904     kHALF_UP = 0,
4905
4907     kHALF_DOWN = 1,
4908
4910     kFLOOR = 2,
4911
4913     kCEIL = 3,
4914 };
4915
4916 namespace impl
4917 {
4923 template <>
4924 struct EnumMaxImpl<ResizeRoundMode>
4925 {
4926     static constexpr int32_t kVALUE = 4;
4927 };
4928 } // namespace impl
4929
4966 class IResizeLayer : public ILayer
4967 {
4968 public:
4987     void setOutputDimensions(Dims dimensions) noexcept
4988     {
4989         return mImpl->setOutputDimensions(dimensions);
4990     }
4991
4997     Dims getOutputDimensions() const noexcept
4998     {
4999         return mImpl->getOutputDimensions();
5000     }
5001
5027     void setScales(float const* scales, int32_t nbScales) noexcept
5028     {
5029         mImpl->setScales(scales, nbScales);
5030     }
5031
5046     int32_t getScales(int32_t size, float* scales) const noexcept

```

```

5047     {
5048         return mImpl->getScales(size, scales);
5049     }
5050
5051     void setResizeMode(ResizeMode resizeMode) noexcept
5052     {
5053         mImpl->setResizeMode(resizeMode);
5054     }
5055
5056     ResizeMode getResizeMode() const noexcept
5057     {
5058         return mImpl->getResizeMode();
5059     }
5060
5061     TRT_DEPRECATED void setAlignCorners(bool alignCorners) noexcept
5062     {
5063         mImpl->setAlignCorners(alignCorners);
5064     }
5065
5066     TRT_DEPRECATED bool getAlignCorners() const noexcept
5067     {
5068         return mImpl->getAlignCorners();
5069     }
5070
5071     using ILayer::setInput;
5072
5073     void setCoordinateTransformation(ResizeCoordinateTransformation coordTransform) noexcept
5074     {
5075         mImpl->setCoordinateTransformation(coordTransform);
5076     }
5077
5078     ResizeCoordinateTransformation getCoordinateTransformation() const noexcept
5079     {
5080         return mImpl->getCoordinateTransformation();
5081     }
5082
5083     void setSelectorForSinglePixel(ResizeSelector selector) noexcept
5084     {
5085         mImpl->setSelectorForSinglePixel(selector);
5086     }
5087
5088     ResizeSelector getSelectorForSinglePixel() const noexcept
5089     {
5090         return mImpl->getSelectorForSinglePixel();
5091     }
5092
5093     void setNearestRounding(ResizeRoundMode value) noexcept
5094     {
5095         mImpl->setNearestRounding(value);
5096     }
5097
5098     ResizeRoundMode getNearestRounding() const noexcept
5099     {
5100         return mImpl->getNearestRounding();
5101     }
5102
5103     void setCubicCoeff(float A) noexcept
5104     {
5105         mImpl->setCubicCoeff(A);
5106     }
5107
5108     float getCubicCoeff() const noexcept
5109     {
5110         return mImpl->getCubicCoeff();
5111     }
5112
5113     void setExcludeOutside(bool excludeFlag) noexcept
5114     {
5115         mImpl->setExcludeOutside(excludeFlag);
5116     }
5117
5118     bool getExcludeOutside() const noexcept
5119     {
5120         return mImpl->getExcludeOutside();
5121     }
5122
5123 protected:
5124     virtual ~IResizeLayer() noexcept = default;
5125     apiv::VResizeLayer* mImpl;
5126 };

```

```

5256 enum class LoopOutput : int32_t
5257 {
5259     kLAST_VALUE = 0,
5260
5262     kCONCATENATE = 1,
5263
5265     kREVERSE = 2
5266 };
5267
5273 template <>
5274 constexpr inline int32_t EnumMax<LoopOutput>() noexcept
5275 {
5276     return 3;
5277 }
5278
5280 enum class TripLimit : int32_t
5281 {
5282
5283     kCOUNT = 0,
5284     kWHILE = 1
5285 };
5286
5292 template <>
5293 constexpr inline int32_t EnumMax<TripLimit>() noexcept
5294 {
5295     return 2;
5296 }
5297
5298 class ILoop;
5299
5300 class ILoopBoundaryLayer : public ILayer
5301 {
5302 public:
5304     ILoop* getLoop() const noexcept
5305     {
5306         return mBoundary->getLoop();
5307     }
5308
5309 protected:
5310     virtual ~ILoopBoundaryLayer() noexcept = default;
5311     apiv::VLoopBoundaryLayer* mBoundary;
5312 };
5313
5319 class IIfConditionalBoundaryLayer : public ILayer
5320 {
5321 public:
5323     IIfConditional* getConditional() const noexcept
5324     {
5325         return mBoundary->getConditional();
5326     }
5327
5328 protected:
5329     virtual ~IIfConditionalBoundaryLayer() noexcept = default;
5330     apiv::VConditionalBoundaryLayer* mBoundary;
5331 };
5332
5336 class IConditionLayer : public IIfConditionalBoundaryLayer
5337 {
5338 public:
5339 protected:
5340     virtual ~IConditionLayer() noexcept = default;
5341     apiv::VConditionLayer* mImpl;
5342 };
5343
5349 class IIfConditionalOutputLayer : public IIfConditionalBoundaryLayer
5350 {
5351 public:
5352 protected:
5353     virtual ~IIfConditionalOutputLayer() noexcept = default;
5354     apiv::VConditionalOutputLayer* mImpl;
5355 };
5356
5360 class IIfConditionalInputLayer : public IIfConditionalBoundaryLayer
5361 {
5362 public:
5363 protected:
5364     virtual ~IIfConditionalInputLayer() noexcept = default;
5365     apiv::VConditionalInputLayer* mImpl;
5366 };
5367
5389 class IIfConditional : public INoCopy

```

```

5390 {
5391 public:
5401     IConditionLayer* setCondition(ITensor& condition) noexcept
5402     {
5403         return mImpl->setCondition(condition);
5404     }
5405
5417     IIfConditionalOutputLayer* addOutput(ITensor& trueSubgraphOutput, ITensor& falseSubgraphOutput)
5418     noexcept
5419     {
5420         return mImpl->addOutput(trueSubgraphOutput, falseSubgraphOutput);
5421     }
5422
5429     IIfConditionalInputLayer* addInput(ITensor& input) noexcept
5430     {
5431         return mImpl->addInput(input);
5432     }
5433
5444     void setName(char const* name) noexcept
5445     {
5446         mImpl->setName(name);
5447     }
5448
5454     char const* getName() const noexcept
5455     {
5456         return mImpl->getName();
5457     }
5458
5459 protected:
5460     virtual ~IIfConditional() noexcept = default;
5461     apiv::VIfConditional* mImpl;
5462 };
5463
5464
5465 class IRecurrenceLayer : public ILoopBoundaryLayer
5466 {
5467 public:
5473     //
5486     using ILayer::setInput;
5487
5488 protected:
5489     virtual ~IRecurrenceLayer() noexcept = default;
5490     apiv::VRecurrenceLayer* mImpl;
5491 };
5492
5510 class ILoopOutputLayer : public ILoopBoundaryLayer
5511 {
5512 public:
5513     LoopOutput getLoopOutput() const noexcept
5514     {
5515         return mImpl->getLoopOutput();
5516     }
5517
5530     void setAxis(int32_t axis) noexcept
5531     {
5532         mImpl->setAxis(axis);
5533     }
5534
5536     int32_t getAxis() const noexcept
5537     {
5538         return mImpl->getAxis();
5539     }
5540
5546     //
5561     using ILayer::setInput;
5562
5563 protected:
5564     virtual ~ILoopOutputLayer() noexcept = default;
5565     apiv::VLoopOutputLayer* mImpl;
5566 };
5567
5568 class ITripLimitLayer : public ILoopBoundaryLayer
5569 {
5570 public:
5571     TripLimit getTripLimit() const noexcept
5572     {
5573         return mImpl->getTripLimit();
5574     }
5575
5576 protected:
5577     virtual ~ITripLimitLayer() noexcept = default;

```

```

5578     apiv::VTriplimitLayer* mImpl;
5579 };
5580
5581 class IIteratorLayer : public ILoopBoundaryLayer
5582 {
5583 public:
5584     void setAxis(int32_t axis) noexcept
5585     {
5586         mImpl->setAxis(axis);
5587     }
5588
5589     int32_t getAxis() const noexcept
5590     {
5591         return mImpl->getAxis();
5592     }
5593
5594     void setReverse(bool reverse) noexcept
5595     {
5596         mImpl->setReverse(reverse);
5597     }
5598
5599     bool getReverse() const noexcept
5600     {
5601         return mImpl->getReverse();
5602     }
5603
5604 protected:
5605     virtual ~IIteratorLayer() noexcept = default;
5606     apiv::VIteratorLayer* mImpl;
5607 };
5608
5609 class ILoop : public INoCopy
5610 {
5611 public:
5612     IRecurrenceLayer* addRecurrence(ITensor& initialValue) noexcept
5613     {
5614         return mImpl->addRecurrence(initialValue);
5615     }
5616
5617     ITriplimitLayer* addTriplimit(ITensor& tensor, Triplimit limit) noexcept
5618     {
5619         return mImpl->addTriplimit(tensor, limit);
5620     }
5621
5622     IIteratorLayer* addIterator(ITensor& tensor, int32_t axis = 0, bool reverse = false) noexcept
5623     {
5624         return mImpl->addIterator(tensor, axis, reverse);
5625     }
5626
5627     ILoopOutputLayer* addLoopOutput(ITensor& tensor, LoopOutput outputKind, int32_t axis = 0) noexcept
5628     {
5629         return mImpl->addLoopOutput(tensor, outputKind, axis);
5630     }
5631
5632     void setName(char const* name) noexcept
5633     {
5634         mImpl->setName(name);
5635     }
5636
5637     char const* getName() const noexcept
5638     {
5639         return mImpl->getName();
5640     }
5641
5642 protected:
5643     virtual ~ILoop() noexcept = default;
5644     apiv::VLoop* mImpl;
5645 };
5646
5647 class ISelectLayer : public ILayer
5648 {
5649 protected:
5650     virtual ~ISelectLayer() noexcept = default;
5651     apiv::VSelectLayer* mImpl;
5652 };
5653
5654 class IAssertionLayer : public ILayer
5655 {
5656 public:
5657     void setMessage(char const* message) noexcept
5658     {

```

```

5749         mImpl->setMessage(message);
5750     }
5751
5752     char const* getMessage() const noexcept
5753     {
5754         return mImpl->getMessage();
5755     }
5756
5757 protected:
5758     virtual ~IAssertionLayer() noexcept = default;
5759     apiv::VAssertionLayer* mImpl;
5760 };
5761
5762 enum class FillOperation : int32_t
5763 {
5764     kLINSPEACE = 0,
5765     kRANDOM.UNIFORM = 1,
5766     kRANDOM.NORMAL = 2
5767 };
5768
5769 template <>
5770 constexpr inline int32_t EnumMax<FillOperation>() noexcept
5771 {
5772     return 3;
5773 }
5774
5775 class IFillLayer : public ILayer
5776 {
5777 public:
5778     //
5779     void setDimensions(Dims dimensions) noexcept
5780     {
5781         mImpl->setDimensions(dimensions);
5782     }
5783
5784     Dims getDimensions() const noexcept
5785     {
5786         return mImpl->getDimensions();
5787     }
5788
5789     void setOperation(FillOperation op) noexcept
5790     {
5791         mImpl->setOperation(op);
5792     }
5793
5794     FillOperation getOperation() const noexcept
5795     {
5796         return mImpl->getOperation();
5797     }
5798
5799     //
5800     void setAlpha(double alpha) noexcept
5801     {
5802         mImpl->setAlpha(alpha);
5803     }
5804
5805     double getAlpha() const noexcept
5806     {
5807         return mImpl->getAlpha();
5808     }
5809
5810     void setBeta(double beta) noexcept
5811     {
5812         mImpl->setBeta(beta);
5813     }
5814
5815     double getBeta() const noexcept
5816     {
5817         return mImpl->getBeta();
5818     }
5819
5820     using ILayer::setInput;
5821
5822 protected:
5823     virtual ~IFillLayer() noexcept = default;
5824     apiv::VFillLayer* mImpl;
5825 };
5826
5827 class IQuantizeLayer : public ILayer
5828 {

```



```

6036 public:
6045     int32_t getAxis() const noexcept
6046     {
6047         return mImpl->getAxis();
6048     }
6056     void setAxis(int32_t axis) noexcept
6057     {
6058         mImpl->setAxis(axis);
6059     }
6060
6061 protected:
6062     virtual ~IQuantizeLayer() noexcept = default;
6063     apiv::VQuantizeLayer* mImpl;
6064 };
6065
6120 class IDequantizeLayer : public ILayer
6121 {
6122 public:
6131     int32_t getAxis() const noexcept
6132     {
6133         return mImpl->getAxis();
6134     }
6142     void setAxis(int32_t axis) noexcept
6143     {
6144         mImpl->setAxis(axis);
6145     }
6146
6147 protected:
6148     virtual ~IDequantizeLayer() noexcept = default;
6149     apiv::VDequantizeLayer* mImpl;
6150 };
6151
6187 class IEinsumLayer : public ILayer
6188 {
6189 public:
6199     bool setEquation(char const* equation) noexcept
6200     {
6201         return mImpl->setEquation(equation);
6202     }
6203
6209     char const* getEquation() const noexcept
6210     {
6211         return mImpl->getEquation();
6212     }
6213
6214 protected:
6215     virtual ~IEinsumLayer() noexcept = default;
6216     apiv::VEinsumLayer* mImpl;
6217 };
6218
6224 enum class ScatterMode : int32_t
6225 {
6226     kELEMENT = 0,
6227     kND = 1,
6228 };
6229
6235 template <>
6236 constexpr inline int32_t EnumMax<ScatterMode>() noexcept
6237 {
6238     return 2;
6239 }
6240
6297 class IScatterLayer : public ILayer
6298 {
6299 public:
6305     void setMode(ScatterMode mode) noexcept
6306     {
6307         mImpl->setMode(mode);
6308     }
6309
6315     ScatterMode getMode() const noexcept
6316     {
6317         return mImpl->getMode();
6318     }
6319
6325     void setAxis(int32_t axis) noexcept
6326     {
6327         mImpl->setAxis(axis);
6328     }
6329
6333     int32_t getAxis() const noexcept

```

```

6334     {
6335         return mImpl->getAxis();
6336     }
6337
6338 protected:
6339     apiv::VScatterLayer* mImpl;
6340     virtual ~IScatterLayer() noexcept = default;
6341 }; // class IScatterLayer
6342
6370 class IOneHotLayer : public ILayer
6371 {
6372 public:
6373     void setAxis(int32_t axis) noexcept
6374     {
6375         mImpl->setAxis(axis);
6376     }
6377
6378     int32_t getAxis() const noexcept
6379     {
6380         return mImpl->getAxis();
6381     }
6382
6390 protected:
6391     apiv::VOneHotLayer* mImpl;
6392 };
6393
6405 class IGridSampleLayer : public ILayer
6406 {
6407 public:
6408     void setInterpolationMode(InterpolationMode mode) noexcept
6409     {
6410         mImpl->setInterpolationMode(mode);
6411     }
6412
6413     InterpolationMode getInterpolationMode() const noexcept
6414     {
6415         return mImpl->getInterpolationMode();
6416     }
6417
6418     void setAlignCorners(bool alignCorners) noexcept
6419     {
6420         mImpl->setAlignCorners(alignCorners);
6421     }
6422
6423     bool getAlignCorners() const noexcept
6424     {
6425         return mImpl->getAlignCorners();
6426     }
6427
6428     bool setSampleMode(SampleMode mode) noexcept
6429     {
6430         return mImpl->setSampleMode(mode);
6431     }
6432
6433     SampleMode getSampleMode() const noexcept
6434     {
6435         return mImpl->getSampleMode();
6436     }
6437
6438 protected:
6439     apiv::VGridSampleLayer* mImpl;
6440     virtual ~IGridSampleLayer() noexcept = default;
6441 }; // class IGridSampleLayer
6442
6486 enum class BoundingBoxFormat : int32_t
6487 {
6488     kCORNER_PAIRS = 0,
6489     kCENTER_SIZES = 1
6490 };
6491
6492 template <>
6493 constexpr inline int32_t EnumMax<BoundingBoxFormat>() noexcept
6494 {
6495     return 2;
6496 }
6497
6504 class INMSLayer : public ILayer
6505 {
6506 public:
6507     void setBoundingBoxFormat(BoundingBoxFormat fmt) noexcept
6508     {

```

```

6562         mImpl->setBoundingBoxFormat(fmt);
6563     }
6564
6572     BoundingBoxFormat getBoundingBoxFormat() const noexcept
6573     {
6574         return mImpl->getBoundingBoxFormat();
6575     }
6576
6586     void setTopKBoxLimit(int32_t limit) noexcept
6587     {
6588         mImpl->setTopKBoxLimit(limit);
6589     }
6590
6596     int32_t getTopKBoxLimit() const noexcept
6597     {
6598         return mImpl->getTopKBoxLimit();
6599     }
6600
6619     using ILayer::setInput;
6620
6621 protected:
6622     apiv::VNMSLayer* mImpl;
6623     virtual ~INMSLayer() noexcept = default;
6624 }; // class INMSLayer
6625
6637 class IReverseSequenceLayer: public ILayer
6638 {
6639 public:
6648     void setBatchAxis(int32_t batchAxis) noexcept
6649     {
6650         mImpl->setBatchAxis(batchAxis);
6651     }
6652
6658     int32_t getBatchAxis() const noexcept
6659     {
6660         return mImpl->getBatchAxis();
6661     }
6662
6671     void setSequenceAxis(int32_t sequenceAxis) noexcept
6672     {
6673         mImpl->setSequenceAxis(sequenceAxis);
6674     }
6675
6681     int32_t getSequenceAxis() const noexcept
6682     {
6683         return mImpl->getSequenceAxis();
6684     }
6685
6686 protected:
6687     apiv::VReverseSequenceLayer* mImpl;
6688     virtual ~IReverseSequenceLayer() noexcept = default;
6689 }; // class IReverseSequenceLayer
6690
6707
6708 class INormalizationLayer : public ILayer
6709 {
6710 public:
6717     void setEpsilon(float eps) noexcept
6718     {
6719         return mImpl->setEpsilon(eps);
6720     }
6721
6726     float getEpsilon() const noexcept
6727     {
6728         return mImpl->getEpsilon();
6729     }
6730
6735     void setAxes(uint32_t axesMask) noexcept
6736     {
6737         return mImpl->setAxes(axesMask);
6738     }
6739
6744     uint32_t getAxes() const noexcept
6745     {
6746         return mImpl->getAxes();
6747     }
6748
6764     void setNbGroups(int32_t nbGroups) noexcept
6765     {
6766         return mImpl->setNbGroups(nbGroups);
6767     }

```

```

6768
6773     int32_t getNbGroups() const noexcept
6774     {
6775         return mImpl->getNbGroups();
6776     }
6777
6791     void setComputePrecision(DataType type) noexcept
6792     {
6793         return mImpl->setComputePrecision(type);
6794     }
6795
6800     DataType getComputePrecision() const noexcept
6801     {
6802         return mImpl->getComputePrecision();
6803     }
6804
6805 protected:
6806     apiv::VNormalizationLayer* mImpl;
6807     virtual ~INormalizationLayer() noexcept = default;
6808 };
6809
6829 class INetworkDefinition : public INoCopy
6830 {
6831 public:
6832     virtual ~INetworkDefinition() noexcept = default;
6833
6871     ITensor* addInput(char const* name, DataType type, Dims dimensions) noexcept
6872     {
6873         return mImpl->addInput(name, type, dimensions);
6874     }
6875
6885     void markOutput(ITensor& tensor) noexcept
6886     {
6887         mImpl->markOutput(tensor);
6888     }
6889
6908     TRT_DEPRECATED IConvolutionLayer* addConvolution(
6909         ITensor& input, int32_t nbOutputMaps, DimsHW kernelSize, Weights kernelWeights, Weights
        biasWeights) noexcept
6910     {
6911         return mImpl->addConvolution(input, nbOutputMaps, kernelSize, kernelWeights, biasWeights);
6912     }
6913
6931     TRT_DEPRECATED IFullyConnectedLayer* addFullyConnected(
6932         ITensor& input, int32_t nbOutputs, Weights kernelWeights, Weights biasWeights) noexcept
6933     {
6934         return mImpl->addFullyConnected(input, nbOutputs, kernelWeights, biasWeights);
6935     }
6936
6951     IActivationLayer* addActivation(ITensor& input, ActivationType type) noexcept
6952     {
6953         return mImpl->addActivation(input, type);
6954     }
6955
6970     TRT_DEPRECATED IPoolingLayer* addPooling(ITensor& input, PoolingType type, DimsHW windowSize) noexcept
6971     {
6972         return mImpl->addPooling(input, type, windowSize);
6973     }
6974
6989     ILRNLayer* addLRN(ITensor& input, int32_t window, float alpha, float beta, float k) noexcept
6990     {
6991         return mImpl->addLRN(input, window, alpha, beta, k);
6992     }
6993
7016     IScaleLayer* addScale(ITensor& input, ScaleMode mode, Weights shift, Weights scale, Weights power)
        noexcept
7017     {
7018         return mImpl->addScale(input, mode, shift, scale, power);
7019     }
7020
7029     ISoftMaxLayer* addSoftMax(ITensor& input) noexcept
7030     {
7031         return mImpl->addSoftMax(input);
7032     }
7033
7046     IConcatenationLayer* addConcatenation(ITensor* const* inputs, int32_t nbInputs) noexcept
7047     {
7048         return mImpl->addConcatenation(inputs, nbInputs);
7049     }
7050
7069     TRT_DEPRECATED IDEconvolutionLayer* addDeconvolution(

```

```

7070         ITensor& input, int32_t nbOutputMaps, DimsHW kernelSize, Weights kernelWeights, Weights
biasWeights) noexcept
7071     {
7072         return mImpl->addDeconvolution(input, nbOutputMaps, kernelSize, kernelWeights, biasWeights);
7073     }
7074
7097     IElementWiseLayer* addElementWise(ITensor& input1, ITensor& input2, ElementWiseOperation op) noexcept
7098     {
7099         return mImpl->addElementWise(input1, input2, op);
7100     }
7101
7119     IUnaryLayer* addUnary(ITensor& input, UnaryOperation operation) noexcept
7120     {
7121         return mImpl->addUnary(input, operation);
7122     }
7123
7136     TRT_DEPRECATED IPaddingLayer* addPadding(ITensor& input, DimsHW prePadding, DimsHW postPadding)
noexcept
7137     {
7138         return mImpl->addPadding(input, prePadding, postPadding);
7139     }
7140
7150     IShuffleLayer* addShuffle(ITensor& input) noexcept
7151     {
7152         return mImpl->addShuffle(input);
7153     }
7154
7167     IOneHotLayer* addOneHot(ITensor& indices, ITensor& values, ITensor& depth, int32_t axis) noexcept
7168     {
7169         return mImpl->addOneHot(indices, values, depth, axis);
7170     }
7171
7179     int32_t getNbLayers() const noexcept
7180     {
7181         return mImpl->getNbLayers();
7182     }
7183
7193     ILayer* getLayer(int32_t index) const noexcept
7194     {
7195         return mImpl->getLayer(index);
7196     }
7197
7205     int32_t getNbInputs() const noexcept
7206     {
7207         return mImpl->getNbInputs();
7208     }
7209
7221     ITensor* getInput(int32_t index) const noexcept
7222     {
7223         return mImpl->getInput(index);
7224     }
7225
7235     int32_t getNbOutputs() const noexcept
7236     {
7237         return mImpl->getNbOutputs();
7238     }
7239
7251     ITensor* getOutput(int32_t index) const noexcept
7252     {
7253         return mImpl->getOutput(index);
7254     }
7255
7263     TRT_DEPRECATED void destroy() noexcept
7264     {
7265         delete this;
7266     }
7267
7290     IReduceLayer* addReduce(
7291         ITensor& input, ReduceOperation operation, uint32_t reduceAxes, bool keepDimensions) noexcept
7292     {
7293         return mImpl->addReduce(input, operation, reduceAxes, keepDimensions);
7294     }
7295
7325     ITopKLayer* addTopK(ITensor& input, TopKOperation op, int32_t k, uint32_t reduceAxes) noexcept
7326     {
7327         return mImpl->addTopK(input, op, k, reduceAxes);
7328     }
7329
7341     IGatherLayer* addGather(ITensor& data, ITensor& indices, int32_t axis) noexcept
7342     {
7343         return mImpl->addGather(data, indices, axis);

```

```

7344     }
7345
7357     IGatherLayer* addGatherV2(ITensor& data, ITensor& indices, GatherMode mode) noexcept
7358     {
7359         return mImpl->addGatherV2(data, indices, mode);
7360     }
7361
7375     IRaggedSoftMaxLayer* addRaggedSoftMax(ITensor& input, ITensor& bounds) noexcept
7376     {
7377         return mImpl->addRaggedSoftMax(input, bounds);
7378     }
7379
7396     IMatrixMultiplyLayer* addMatrixMultiply(
7397         ITensor& input0, MatrixOperation op0, ITensor& input1, MatrixOperation op1) noexcept
7398     {
7399         return mImpl->addMatrixMultiply(input0, op0, input1, op1);
7400     }
7401
7411     INonZeroLayer* addNonZero(ITensor& input) noexcept
7412     {
7413         return mImpl->addNonZero(input);
7414     }
7415
7438     IConstantLayer* addConstant(Dims dimensions, Weights weights) noexcept
7439     {
7440         return mImpl->addConstant(dimensions, weights);
7441     }
7442
7507     TRT_DEPRECATED IRNNv2Layer* addRNNv2(
7508         ITensor& input, int32_t layerCount, int32_t hiddenSize, int32_t maxSeqLen, RNNOperation op) noexcept
7509     {
7510         return mImpl->addRNNv2(input, layerCount, hiddenSize, maxSeqLen, op);
7511     }
7512
7522     IIdentityLayer* addIdentity(ITensor& input) noexcept
7523     {
7524         return mImpl->addIdentity(input);
7525     }
7526
7537     ICastLayer* addCast(ITensor& input, DataType toType) noexcept
7538     {
7539         return mImpl->addCast(input, toType);
7540     }
7541
7552     void removeTensor(ITensor& tensor) noexcept
7553     {
7554         mImpl->removeTensor(tensor);
7555     }
7556
7564     void unmarkOutput(ITensor& tensor) noexcept
7565     {
7566         mImpl->unmarkOutput(tensor);
7567     }
7568
7583     IPluginV2Layer* addPluginV2(ITensor* const* inputs, int32_t nbInputs, IPluginV2& plugin) noexcept
7584     {
7585         return mImpl->addPluginV2(inputs, nbInputs, plugin);
7586     }
7587
7602     ISliceLayer* addSlice(ITensor& input, Dims start, Dims size, Dims stride) noexcept
7603     {
7604         return mImpl->addSlice(input, start, size, stride);
7605     }
7606
7626     void setName(char const* name) noexcept
7627     {
7628         mImpl->setName(name);
7629     }
7630
7640     char const* getName() const noexcept
7641     {
7642         return mImpl->getName();
7643     }
7644
7658     IShapeLayer* addShape(ITensor& input) noexcept
7659     {
7660         return mImpl->addShape(input);
7661     }
7662
7676     bool hasImplicitBatchDimension() const noexcept
7677     {

```

```

7678         return mImpl->hasImplicitBatchDimension();
7679     }
7680
7694     bool markOutputForShapes(ITensor& tensor) noexcept
7695     {
7696         return mImpl->markOutputForShapes(tensor);
7697     }
7698
7706     bool unmarkOutputForShapes(ITensor& tensor) noexcept
7707     {
7708         return mImpl->unmarkOutputForShapes(tensor);
7709     }
7710
7724     IParametricReLULayer* addParametricReLU(ITensor& input, ITensor& slope) noexcept
7725     {
7726         return mImpl->addParametricReLU(input, slope);
7727     }
7728
7746     IConvolutionLayer* addConvolutionNd(
7747         ITensor& input, int32_t nbOutputMaps, Dims kernelSize, Weights kernelWeights, Weights biasWeights)
7748     noexcept
7749     {
7750         return mImpl->addConvolutionNd(input, nbOutputMaps, kernelSize, kernelWeights, biasWeights);
7751     }
7752
7766     IPoolingLayer* addPoolingNd(ITensor& input, PoolingType type, Dims windowSize) noexcept
7767     {
7768         return mImpl->addPoolingNd(input, type, windowSize);
7769     }
7770
7785     //
7788     IDeconvolutionLayer* addDeconvolutionNd(
7789         ITensor& input, int32_t nbOutputMaps, Dims kernelSize, Weights kernelWeights, Weights biasWeights)
7790     noexcept
7791     {
7792         return mImpl->addDeconvolutionNd(input, nbOutputMaps, kernelSize, kernelWeights, biasWeights);
7793     }
7794
7824     IScaleLayer* addScaleNd(
7825         ITensor& input, ScaleMode mode, Weights shift, Weights scale, Weights power, int32_t channelAxis)
7826     noexcept
7827     {
7828         return mImpl->addScaleNd(input, mode, shift, scale, power, channelAxis);
7829     }
7830
7840     IResizeLayer* addResize(ITensor& input) noexcept
7841     {
7842         return mImpl->addResize(input);
7843     }
7844
7854     TRT_DEPRECATED bool hasExplicitPrecision() const noexcept
7855     {
7856         return mImpl->hasExplicitPrecision();
7857     }
7858
7870     ILoop* addLoop() noexcept
7871     {
7872         return mImpl->addLoop();
7873     }
7874
7910     ISelectLayer* addSelect(ITensor& condition, ITensor& thenInput, ITensor& elseInput) noexcept
7911     {
7912         return mImpl->addSelect(condition, thenInput, elseInput);
7913     }
7914
7927     IAssertionLayer* addAssertion(ITensor& condition, char const* message) noexcept
7928     {
7929         return mImpl->addAssertion(condition, message);
7930     }
7931
7950     IFillLayer* addFill(Dims dimensions, FillOperation op) noexcept
7951     {
7952         return mImpl->addFill(dimensions, op);
7953     }
7954
7967     TRT_DEPRECATED IPaddingLayer* addPaddingNd(ITensor& input, Dims prePadding, Dims postPadding) noexcept
7968     {
7969         return mImpl->addPaddingNd(input, prePadding, postPadding);
7970     }
7971
7990     bool setWeightsName(Weights weights, char const* name) noexcept

```

```

7991     {
7992         return mImpl->setWeightsName(weights, name);
7993     }
7994
8006     //
8009     void setErrorRecorder(IErrorRecorder* recorder) noexcept
8010     {
8011         mImpl->setErrorRecorder(recorder);
8012     }
8013
8024     IErrorRecorder* getErrorRecorder() const noexcept
8025     {
8026         return mImpl->getErrorRecorder();
8027     }
8028
8043     IDequantizeLayer* addDequantize(ITensor& input, ITensor& scale) noexcept
8044     {
8045         return mImpl->addDequantize(input, scale);
8046     }
8047
8063     IScatterLayer* addScatter(ITensor& data, ITensor& indices, ITensor& updates, ScatterMode mode) noexcept
8064     {
8065         return mImpl->addScatter(data, indices, updates, mode);
8066     }
8067
8082     IQuantizeLayer* addQuantize(ITensor& input, ITensor& scale) noexcept
8083     {
8084         return mImpl->addQuantize(input, scale);
8085     }
8086
8097     IIfConditional* addIfConditional() noexcept
8098     {
8099         return mImpl->addIfConditional();
8100     }
8101
8111     IEinsumLayer* addEinsum(ITensor* const* inputs, int32_t nbInputs, char const* equation) noexcept
8112     {
8113         return mImpl->addEinsum(inputs, nbInputs, equation);
8114     }
8115
8127     IGridSampleLayer* addGridSample(ITensor& input, ITensor& grid) noexcept
8128     {
8129         return mImpl->addGridSample(input, grid);
8130     }
8131
8145     INMSLayer* addNMS(ITensor& boxes, ITensor& scores, ITensor& maxOutputBoxesPerClass) noexcept
8146     {
8147         return mImpl->addNMS(boxes, scores, maxOutputBoxesPerClass);
8148     }
8149
8162     IReverseSequenceLayer* addReverseSequence(ITensor& input, ITensor& sequenceLens) noexcept
8163     {
8164         return mImpl->addReverseSequence(input, sequenceLens);
8165     }
8166
8188     INormalizationLayer* addNormalization(
8189         ITensor& input, ITensor& scale, ITensor& bias, uint32_t axesMask) noexcept
8190     {
8191         return mImpl->addNormalization(input, scale, bias, axesMask);
8192     }
8193
8194 protected:
8195     apiv::VNetworkDefinition* mImpl;
8196 };
8197
8203 enum class CalibrationAlgoType : int32_t
8204 {
8205     kLEGACY_CALIBRATION = 0,
8206     kENTROPY_CALIBRATION = 1,
8207     kENTROPY_CALIBRATION_2 = 2,
8208     kMINMAX_CALIBRATION = 3,
8209 };
8210
8216 template <>
8217 constexpr inline int32_t EnumMax<CalibrationAlgoType>() noexcept
8218 {
8219     return 4;
8220 }
8221
8233 class IInt8Calibrator
8234 {

```



```

8235 public:
8241     virtual int32_t getBatchSize() const noexcept = 0;
8242
8256     virtual bool getBatch(void* bindings[], char const* names[], int32_t nbBindings) noexcept = 0;
8257
8272     virtual void const* readCalibrationCache(std::size_t& length) noexcept = 0;
8273
8282     virtual void writeCalibrationCache(void const* ptr, std::size_t length) noexcept = 0;
8283
8289     virtual CalibrationAlgoType getAlgorithm() noexcept = 0;
8290
8291     virtual ~IInt8Calibrator() noexcept = default;
8292 };
8293
8298 class IInt8EntropyCalibrator : public IInt8Calibrator
8299 {
8300 public:
8304     CalibrationAlgoType getAlgorithm() noexcept override
8305     {
8306         return CalibrationAlgoType::kENTROPY_CALIBRATION;
8307     }
8308
8309     virtual ~IInt8EntropyCalibrator() noexcept = default;
8310 };
8311
8316 class IInt8EntropyCalibrator2 : public IInt8Calibrator
8317 {
8318 public:
8322     CalibrationAlgoType getAlgorithm() noexcept override
8323     {
8324         return CalibrationAlgoType::kENTROPY_CALIBRATION_2;
8325     }
8326
8327     virtual ~IInt8EntropyCalibrator2() noexcept = default;
8328 };
8329
8333 class IInt8MinMaxCalibrator : public IInt8Calibrator
8334 {
8335 public:
8339     CalibrationAlgoType getAlgorithm() noexcept override
8340     {
8341         return CalibrationAlgoType::kMINMAX_CALIBRATION;
8342     }
8343
8344     virtual ~IInt8MinMaxCalibrator() noexcept = default;
8345 };
8346
8351 class IInt8LegacyCalibrator : public IInt8Calibrator
8352 {
8353 public:
8357     CalibrationAlgoType getAlgorithm() noexcept override
8358     {
8359         return CalibrationAlgoType::kLEGACY_CALIBRATION;
8360     }
8361
8368     virtual double getQuantile() const noexcept = 0;
8369
8376     virtual double getRegressionCutoff() const noexcept = 0;
8377
8390     virtual void const* readHistogramCache(std::size_t& length) noexcept = 0;
8391
8400     virtual void writeHistogramCache(void const* ptr, std::size_t length) noexcept = 0;
8401
8402     virtual ~IInt8LegacyCalibrator() noexcept = default;
8403 };
8404
8415 class IAlgorithmIOInfo : public INoCopy
8416 {
8417 public:
8421     TensorFormat getTensorFormat() const noexcept
8422     {
8423         return mImpl->getTensorFormat();
8424     }
8425
8429     DataType getDataType() const noexcept
8430     {
8431         return mImpl->getDataType();
8432     }
8433
8437     Dims getStrides() const noexcept
8438     {

```

```

8439         return mImpl->getStrides();
8440     }
8441
8442 protected:
8443     virtual ~IAlgorithmIOInfo() noexcept = default;
8444     apiv::VAlgorithmIOInfo* mImpl;
8445 };
8446
8458 class IAlgorithmVariant : public INoCopy
8459 {
8460 public:
8461     int64_t getImplementation() const noexcept
8462     {
8463         return mImpl->getImplementation();
8464     }
8465
8466     int64_t getTactic() const noexcept
8467     {
8468         return mImpl->getTactic();
8469     }
8470
8471 protected:
8472     virtual ~IAlgorithmVariant() noexcept = default;
8473     apiv::VAlgorithmVariant* mImpl;
8474 };
8475
8488 class IAlgorithmContext : public INoCopy
8489 {
8490 public:
8491     char const* getName() const noexcept
8492     {
8493         return mImpl->getName();
8494     }
8495
8496     Dims getDimensions(int32_t index, OptProfileSelector select) const noexcept
8497     {
8498         return mImpl->getDimensions(index, select);
8499     }
8500
8501     int32_t getNbInputs() const noexcept
8502     {
8503         return mImpl->getNbInputs();
8504     }
8505
8506     int32_t getNbOutputs() const noexcept
8507     {
8508         return mImpl->getNbOutputs();
8509     }
8510
8511 protected:
8512     virtual ~IAlgorithmContext() noexcept = default;
8513     apiv::VAlgorithmContext* mImpl;
8514 };
8515
8528 class IAlgorithm : public INoCopy
8529 {
8530 public:
8531     TRT_DEPRECATED IAlgorithmIOInfo const& getAlgorithmIOInfo(int32_t index) const noexcept
8532     {
8533         return mImpl->getAlgorithmIOInfo(index);
8534     }
8535
8536     IAlgorithmVariant const& getAlgorithmVariant() const noexcept
8537     {
8538         return mImpl->getAlgorithmVariant();
8539     }
8540
8541     float getTimingMSec() const noexcept
8542     {
8543         return mImpl->getTimingMSec();
8544     }
8545
8546     std::size_t getWorkspaceSize() const noexcept
8547     {
8548         return mImpl->getWorkspaceSize();
8549     }
8550
8551     IAlgorithmIOInfo const* getAlgorithmIOInfoByIndex(int32_t index) const noexcept
8552     {
8553         return mImpl->getAlgorithmIOInfoByIndex(index);
8554     }
8555 }

```

```

8597
8598 protected:
8599     virtual ~IAlgorithm() noexcept = default;
8600     apiv::VAlgorithm* mImpl;
8601 }; // IAlgorithm
8602
8603 class IAlgorithmSelector
8604 {
8605 public:
8606     virtual int32_t selectAlgorithms(IAlgorithmContext const& context, IAlgorithm const* const* choices,
8607                                     int32_t nbChoices, int32_t* selection) noexcept = 0;
8608     virtual void reportAlgorithms(IAlgorithmContext const* const* algoContexts, IAlgorithm const* const*
8609                                  algoChoices,
8610                                  int32_t nbAlgorithms) noexcept = 0;
8611     virtual ~IAlgorithmSelector() noexcept = default;
8612 };
8613
8614 using QuantizationFlags = uint32_t;
8615
8616 enum class QuantizationFlag : int32_t
8617 {
8618     kCALIBRATE_BEFORE_FUSION = 0
8619 };
8620
8621 template <>
8622 constexpr inline int32_t EnumMax<QuantizationFlag>() noexcept
8623 {
8624     return 1;
8625 }
8626
8627 using BuilderFlags = uint32_t;
8628
8629 enum class BuilderFlag : int32_t
8630 {
8631     kFP16 = 0,
8632     kINT8 = 1,
8633     kDEBUG = 2,
8634     kGPU_FALLBACK = 3,
8635
8636     kSTRICT_TYPES_TRT_DEPRECATED_ENUM = 4,
8637
8638     kREFIT = 5,
8639     kDISABLE_TIMING_CACHE = 6,
8640
8641     kTF32 = 7,
8642
8643     kSPARSE_WEIGHTS = 8,
8644
8645     kSAFETY_SCOPE = 9,
8646
8647     kOBEY_PRECISION_CONSTRAINTS = 10,
8648
8649     kPREFER_PRECISION_CONSTRAINTS = 11,
8650
8651     kDIRECT_IO = 12,
8652
8653     kREJECT_EMPTY_ALGORITHMS = 13,
8654
8655     kENABLE_TACTIC_HEURISTIC = 14,
8656
8657     kVERSION_COMPATIBLE = 15,
8658
8659     kEXCLUDE_LEAN_RUNTIME = 16,
8660
8661     kFP8 = 17
8662 };
8663
8664 template <>
8665 constexpr inline int32_t EnumMax<BuilderFlag>() noexcept
8666 {
8667     return 18;
8668 }
8669
8670 class ITimingCache : public INoCopy
8671 {
8672 public:
8673     virtual ~ITimingCache() noexcept = default;
8674
8675     nvinfer1::IHostMemory* serialize() const noexcept
8676     {

```

```

8814         return mImpl->serialize();
8815     }
8816
8836     bool combine(ITimingCache const& inputCache, bool ignoreMismatch) noexcept
8837     {
8838         return mImpl->combine(inputCache, ignoreMismatch);
8839     }
8840
8846     bool reset() noexcept
8847     {
8848         return mImpl->reset();
8849     }
8850
8851 protected:
8852     apiv::VTimingCache* mImpl;
8853 };
8854
8862 enum class MemoryPoolType : int32_t
8863 {
8871     kWORKSPACE = 0,
8872
8879     kDLA_MANAGED_SRAM = 1,
8880
8886     kDLA_LOCAL_DRAM = 2,
8887
8893     kDLA_GLOBAL_DRAM = 3,
8894
8902     kTACTIC_DRAM = 4,
8903 };
8904
8910 template <>
8911 constexpr inline int32_t EnumMax<MemoryPoolType>() noexcept
8912 {
8913     return 5;
8914 }
8915
8924 enum class PreviewFeature : int32_t
8925 {
8932     kFASTER_DYNAMIC_SHAPES_0805 = 0,
8933
8948     kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805 = 1,
8949
8955     kPROFILE_SHARING_0806 = 2,
8956
8964     kENABLE_MULTI_STREAM_ENGINE_0806 = 3,
8965 };
8966 namespace impl
8967 {
8973 template <>
8974 struct EnumMaxImpl<PreviewFeature>
8975 {
8976     static constexpr int32_t kVALUE = 4;
8977 };
8978 } // namespace impl
8979
8983 enum class HardwareCompatibilityLevel : int32_t
8984 {
8987     kNONE = 0,
8988
8992     kAMPERE_PLUS = 1,
8993 };
8994
8995 namespace impl
8996 {
9002 template <>
9003 struct EnumMaxImpl<HardwareCompatibilityLevel>
9004 {
9005     static constexpr int32_t kVALUE = 2;
9006 };
9007 } // namespace impl
9008
9016 class IBuilderConfig : public INoCopy
9017 {
9018 public:
9019     virtual ~IBuilderConfig() noexcept = default;
9020
9033     TRT_DEPRECATED virtual void setMinTimingIterations(int32_t minTiming) noexcept
9034     {
9035         mImpl->setMinTimingIterations(minTiming);
9036     }
9037

```

```

9047     TRT_DEPRECATED virtual int32_t getMinTimingIterations() const noexcept
9048     {
9049         return mImpl->getMinTimingIterations();
9050     }
9051
9060     virtual void setAvgTimingIterations(int32_t avgTiming) noexcept
9061     {
9062         mImpl->setAvgTimingIterations(avgTiming);
9063     }
9064
9072     int32_t getAvgTimingIterations() const noexcept
9073     {
9074         return mImpl->getAvgTimingIterations();
9075     }
9076
9085     void setEngineCapability(EngineCapability capability) noexcept
9086     {
9087         mImpl->setEngineCapability(capability);
9088     }
9089
9097     EngineCapability getEngineCapability() const noexcept
9098     {
9099         return mImpl->getEngineCapability();
9100     }
9101
9107     void setInt8Calibrator(IInt8Calibrator* calibrator) noexcept
9108     {
9109         mImpl->setInt8Calibrator(calibrator);
9110     }
9111
9115     IInt8Calibrator* getInt8Calibrator() const noexcept
9116     {
9117         return mImpl->getInt8Calibrator();
9118     }
9119
9130     TRT_DEPRECATED void setMaxWorkspaceSize(std::size_t workspaceSize) noexcept
9131     {
9132         mImpl->setMaxWorkspaceSize(workspaceSize);
9133     }
9134
9147     TRT_DEPRECATED std::size_t getMaxWorkspaceSize() const noexcept
9148     {
9149         return mImpl->getMaxWorkspaceSize();
9150     }
9151
9164     void setFlags(BuilderFlags builderFlags) noexcept
9165     {
9166         mImpl->setFlags(builderFlags);
9167     }
9168
9176     BuilderFlags getFlags() const noexcept
9177     {
9178         return mImpl->getFlags();
9179     }
9180
9188     void clearFlag(BuilderFlag builderFlag) noexcept
9189     {
9190         mImpl->clearFlag(builderFlag);
9191     }
9192
9200     void setFlag(BuilderFlag builderFlag) noexcept
9201     {
9202         mImpl->setFlag(builderFlag);
9203     }
9204
9212     bool getFlag(BuilderFlag builderFlag) const noexcept
9213     {
9214         return mImpl->getFlag(builderFlag);
9215     }
9216
9227     void setDeviceType(ILayer const* layer, DeviceType deviceType) noexcept
9228     {
9229         mImpl->setDeviceType(layer, deviceType);
9230     }
9231
9236     DeviceType getDeviceType(ILayer const* layer) const noexcept
9237     {
9238         return mImpl->getDeviceType(layer);
9239     }
9240
9246     bool isDeviceTypeSet(ILayer const* layer) const noexcept

```

```

9247     {
9248         return mImpl->isDeviceTypeSet(layer);
9249     }
9250
9251     void resetDeviceType(ILayer const* layer) noexcept
9252     {
9253         mImpl->resetDeviceType(layer);
9254     }
9255
9256     bool canRunOnDLA(ILayer const* layer) const noexcept
9257     {
9258         return mImpl->canRunOnDLA(layer);
9259     }
9260
9261     void setDLACore(int32_t dlaCore) noexcept
9262     {
9263         mImpl->setDLACore(dlaCore);
9264     }
9265
9266     int32_t getDLACore() const noexcept
9267     {
9268         return mImpl->getDLACore();
9269     }
9270
9271     void setDefaultDeviceType(DeviceType deviceType) noexcept
9272     {
9273         mImpl->setDefaultDeviceType(deviceType);
9274     }
9275
9276     DeviceType getDefaultDeviceType() const noexcept
9277     {
9278         return mImpl->getDefaultDeviceType();
9279     }
9280
9281     void reset() noexcept
9282     {
9283         mImpl->reset();
9284     }
9285
9286     TRT_DEPRECATED void destroy() noexcept
9287     {
9288         delete this;
9289     }
9290
9291     void setProfileStream(const cudaStream_t stream) noexcept
9292     {
9293         return mImpl->setProfileStream(stream);
9294     }
9295
9296     cudaStream_t getProfileStream() const noexcept
9297     {
9298         return mImpl->getProfileStream();
9299     }
9300
9301     int32_t addOptimizationProfile(IOptimizationProfile const* profile) noexcept
9302     {
9303         return mImpl->addOptimizationProfile(profile);
9304     }
9305
9306     int32_t getNbOptimizationProfiles() const noexcept
9307     {
9308         return mImpl->getNbOptimizationProfiles();
9309     }
9310
9311     void setProfilingVerbosity(ProfilingVerbosity verbosity) noexcept
9312     {
9313         mImpl->setProfilingVerbosity(verbosity);
9314     }
9315
9316     ProfilingVerbosity getProfilingVerbosity() const noexcept
9317     {
9318         return mImpl->getProfilingVerbosity();
9319     }
9320
9321     void setAlgorithmSelector(IAgorithmSelector* selector) noexcept
9322     {
9323         mImpl->setAlgorithmSelector(selector);
9324     }
9325
9326     IAgorithmSelector* getAlgorithmSelector() const noexcept
9327     {

```

```

9430         return mImpl->getAlgorithmSelector();
9431     }
9432
9443     bool setCalibrationProfile(IOptimizationProfile const* profile) noexcept
9444     {
9445         return mImpl->setCalibrationProfile(profile);
9446     }
9447
9453     IOptimizationProfile const* getCalibrationProfile() noexcept
9454     {
9455         return mImpl->getCalibrationProfile();
9456     }
9457
9470     void setQuantizationFlags(QuantizationFlags flags) noexcept
9471     {
9472         mImpl->setQuantizationFlags(flags);
9473     }
9474
9482     QuantizationFlags getQuantizationFlags() const noexcept
9483     {
9484         return mImpl->getQuantizationFlags();
9485     }
9486
9494     void clearQuantizationFlag(QuantizationFlag flag) noexcept
9495     {
9496         mImpl->clearQuantizationFlag(flag);
9497     }
9498
9506     void setQuantizationFlag(QuantizationFlag flag) noexcept
9507     {
9508         mImpl->setQuantizationFlag(flag);
9509     }
9510
9518     bool getQuantizationFlag(QuantizationFlag flag) const noexcept
9519     {
9520         return mImpl->getQuantizationFlag(flag);
9521     }
9522
9540     bool setTacticSources(TacticSources tacticSources) noexcept
9541     {
9542         return mImpl->setTacticSources(tacticSources);
9543     }
9544
9555     TacticSources getTacticSources() const noexcept
9556     {
9557         return mImpl->getTacticSources();
9558     }
9559
9574     nvinfer1::ITimingCache* createTimingCache(void const* blob, std::size_t size) const noexcept
9575     {
9576         return mImpl->createTimingCache(blob, size);
9577     }
9578
9597     bool setTimingCache(ITimingCache const& cache, bool ignoreMismatch) noexcept
9598     {
9599         return mImpl->setTimingCache(cache, ignoreMismatch);
9600     }
9601
9607     nvinfer1::ITimingCache const* getTimingCache() const noexcept
9608     {
9609         return mImpl->getTimingCache();
9610     }
9611
9639     void setMemoryPoolLimit(MemoryPoolType pool, std::size_t poolSize) noexcept
9640     {
9641         mImpl->setMemoryPoolLimit(pool, poolSize);
9642     }
9643
9658     std::size_t getMemoryPoolLimit(MemoryPoolType pool) const noexcept
9659     {
9660         return mImpl->getMemoryPoolLimit(pool);
9661     }
9662
9676     void setPreviewFeature(PreviewFeature feature, bool enable) noexcept
9677     {
9678         mImpl->setPreviewFeature(feature, enable);
9679     }
9680
9690     bool getPreviewFeature(PreviewFeature feature) const noexcept
9691     {
9692         return mImpl->getPreviewFeature(feature);

```

```

9693     }
9694
9710 void setBuilderOptimizationLevel(int32_t level) noexcept
9711 {
9712     mImpl->setBuilderOptimizationLevel(level);
9713 }
9714
9722 int32_t getBuilderOptimizationLevel() noexcept
9723 {
9724     return mImpl->getBuilderOptimizationLevel();
9725 }
9726
9738 void setHardwareCompatibilityLevel(HardwareCompatibilityLevel hardwareCompatibilityLevel) noexcept
9739 {
9740     mImpl->setHardwareCompatibilityLevel(hardwareCompatibilityLevel);
9741 }
9750 HardwareCompatibilityLevel getHardwareCompatibilityLevel() const noexcept
9751 {
9752     return mImpl->getHardwareCompatibilityLevel();
9753 }
9754
9761 void setPluginsToSerialize(char const* const* paths, int32_t nbPaths) noexcept
9762 {
9763     mImpl->setPluginsToSerialize(paths, nbPaths);
9764 }
9765
9771 char const* getPluginToSerialize(int32_t index) const noexcept
9772 {
9773     return mImpl->getPluginToSerialize(index);
9774 }
9775
9781 int32_t getNbPluginsToSerialize() const noexcept
9782 {
9783     return mImpl->getNbPluginsToSerialize();
9784 }
9785
9786 protected:
9787     apiv::VBuilderConfig* mImpl;
9788 };
9789
9796 using NetworkDefinitionCreationFlags = uint32_t;
9797
9806 enum class NetworkDefinitionCreationFlag : int32_t
9807 {
9813     kEXPLICIT_BATCH = 0,
9814
9817     kEXPLICIT_PRECISION TRT_DEPRECATED_ENUM = 1,
9818 };
9819
9825 template <>
9826 constexpr inline int32_t EnumMax<NetworkDefinitionCreationFlag>() noexcept
9827 {
9828     return 2;
9829 }
9830
9838 class IBuilder : public INoCopy
9839 {
9840 public:
9841     virtual ~IBuilder() noexcept = default;
9842
9853     TRT_DEPRECATED void setMaxBatchSize(int32_t batchSize) noexcept
9854     {
9855         mImpl->setMaxBatchSize(batchSize);
9856     }
9857
9868     TRT_DEPRECATED int32_t getMaxBatchSize() const noexcept
9869     {
9870         return mImpl->getMaxBatchSize();
9871     }
9872
9876     bool platformHasFastFp16() const noexcept
9877     {
9878         return mImpl->platformHasFastFp16();
9879     }
9880
9884     bool platformHasFastInt8() const noexcept
9885     {
9886         return mImpl->platformHasFastInt8();
9887     }
9888
9896     TRT_DEPRECATED void destroy() noexcept

```



```

9897     {
9898         delete this;
9899     }
9900
9901     int32_t getMaxDLABatchSize() const noexcept
9902     {
9903         return mImpl->getMaxDLABatchSize();
9904     }
9905
9906     int32_t getNbDLACores() const noexcept
9907     {
9908         return mImpl->getNbDLACores();
9909     }
9910
9911     void setGpuAllocator(IGpuAllocator* allocator) noexcept
9912     {
9913         mImpl->setGpuAllocator(allocator);
9914     }
9915
9916     nvinfer1::IBuilderConfig* createBuilderConfig() noexcept
9917     {
9918         return mImpl->createBuilderConfig();
9919     }
9920
9921     TRT_DEPRECATED nvinfer1::ICudaEngine* buildEngineWithConfig(
9922         INetworkDefinition& network, IBuilderConfig& config) noexcept
9923     {
9924         return mImpl->buildEngineWithConfig(network, config);
9925     }
9926
9927     nvinfer1::INetworkDefinition* createNetworkV2(NetworkDefinitionCreationFlags flags) noexcept
9928     {
9929         return mImpl->createNetworkV2(flags);
9930     }
9931
9932     nvinfer1::IOptimizationProfile* createOptimizationProfile() noexcept
9933     {
9934         return mImpl->createOptimizationProfile();
9935     }
9936
9937     //
9938     void setErrorRecorder(IErrrorRecorder* recorder) noexcept
9939     {
9940         mImpl->setErrorRecorder(recorder);
9941     }
9942
9943     IErrorRecorder* getErrorRecorder() const noexcept
9944     {
9945         return mImpl->getErrorRecorder();
9946     }
9947
9948     void reset() noexcept
9949     {
9950         mImpl->reset();
9951     }
9952
9953     bool platformHasTf32() const noexcept
9954     {
9955         return mImpl->platformHasTf32();
9956     }
9957
9958     nvinfer1::IHostMemory* buildSerializedNetwork(INetworkDefinition& network, IBuilderConfig& config)
9959     noexcept
9960     {
9961         return mImpl->buildSerializedNetwork(network, config);
9962     }
9963
9964     bool isNetworkSupported(INetworkDefinition const& network, IBuilderConfig const& config) const
9965     noexcept
9966     {
9967         return mImpl->isNetworkSupported(network, config);
9968     }
9969
9970     ILogger* getLogger() const noexcept
9971     {
9972         return mImpl->getLogger();
9973     }
9974
9975     bool setMaxThreads(int32_t maxThreads) noexcept
9976     {
9977         return mImpl->setMaxThreads(maxThreads);
9978     }

```

```

10109     }
10110
10120     int32_t getMaxThreads() const noexcept
10121     {
10122         return mImpl->getMaxThreads();
10123     }
10124
10130     IPluginRegistry& getPluginRegistry() noexcept
10131     {
10132         return mImpl->getPluginRegistry();
10133     }
10134
10135 protected:
10136     apiv::VBuilder* mImpl;
10137 };
10138
10139 } // namespace nvinfer1
10140
10145 extern "C" TENSORRTAPI void* createInferBuilder_INTERNAL(void* logger, int32_t version) noexcept;
10146
10147 namespace nvinfer1
10148 {
10149     namespace
10150     {
10151
10159         inline IBuilder* createInferBuilder(ILogger& logger) noexcept
10160         {
10161             return static_cast<IBuilder*>(createInferBuilder_INTERNAL(&logger, NV_TENSORRT_VERSION));
10162         }
10163     }
10164 } // namespace
10165
10176 extern "C" TENSORRTAPI nvinfer1::IPluginRegistry* getBuilderPluginRegistry(
10177     nvinfer1::EngineCapability capability) noexcept;
10178
10179 } // namespace nvinfer1
10180
10181 #endif // NV_INFER_H

```

10.5 NvInferConsistency.h File Reference

```

#include "NvInferConsistencyImpl.h"
#include "NvInferRuntimeCommon.h"

```

Classes

- class [nvinfer1::consistency::IConsistencyChecker](#)
Validates a serialized engine blob.
- class [nvinfer1::consistency::IPluginChecker](#)
Consistency Checker plugin class for user implemented Plugins.

Namespaces

- namespace [nvinfer1](#)
The TensorRT API version 1 namespace.
- namespace [nvinfer1::consistency](#)

Functions

- void * [createConsistencyChecker_INTERNAL](#) (void *logger, void const *blob, size_t size, int32_t version)
Internal C entry point for creating IConsistencyChecker.

10.5.1 Function Documentation

10.5.1.1 createConsistencyChecker_INTERNAL()

```
void * createConsistencyChecker_INTERNAL (
    void * logger,
    void const * blob,
    size_t size,
    int32_t version )
```

Internal C entry point for creating IConsistencyChecker.

10.6 NvInferConsistency.h

[Go to the documentation of this file.](#)

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_INFER_CONSISTENCY_H
14 #define NV_INFER_CONSISTENCY_H
15
16 #include "NvInferConsistencyImpl.h"
17 #include "NvInferRuntimeCommon.h"
18
19
20 namespace nvinfer1
21 {
22
23     namespace consistency
24     {
25
26         class IConsistencyChecker
27         {
28         public:
29             //
30             bool validate() const noexcept
31             {
32                 return mImpl->validate();
33             }
34
35             virtual ~IConsistencyChecker() = default;
36
37         protected:
38             apiv::VConsistencyChecker* mImpl;
39             IConsistencyChecker() = default;
40         };
41     }
42 }
```

```

58     IConsistencyChecker(IConsistencyChecker const& other) = delete;
59     IConsistencyChecker& operator=(IConsistencyChecker const& other) = delete;
60     IConsistencyChecker(IConsistencyChecker&& other) = delete;
61     IConsistencyChecker& operator=(IConsistencyChecker&& other) = delete;
62 };
63
64
65 class IPluginChecker : public IPluginCreator
66 {
67 public:
68     virtual bool validate(char const* name, void const* serialData, size_t serialLength, PluginTensorDesc
69         const* in,
70         size_t nbInputs, PluginTensorDesc const* out, size_t nbOutputs, int64_t workspaceSize) const noexcept
71         = 0;
72
73     IPluginChecker() = default;
74     virtual ~IPluginChecker() override = default;
75
76 protected:
77     IPluginChecker(IPluginChecker const&) = default;
78     IPluginChecker(IPluginChecker&&) = default;
79     IPluginChecker& operator=(IPluginChecker const&) &= default;
80     IPluginChecker& operator=(IPluginChecker&&) &= default;
81 };
82
83 } // namespace consistency
84
85 } // namespace nvinfer1
86
87 extern "C" TENSORRTAPI void* createConsistencyChecker_INTERNAL(void* logger, void const* blob, size_t size,
88     int32_t version);
89
90 namespace nvinfer1
91 {
92     namespace consistency
93     {
94         namespace // anonymous
95         {
96             inline IConsistencyChecker* createConsistencyChecker(ILogger& logger, void const* blob, size_t size)
97             {
98                 return static_cast<IConsistencyChecker*>(
99                     createConsistencyChecker_INTERNAL(&logger, blob, size, NV_TENSORRT_VERSION));
100             }
101         }
102     } // namespace
103
104 } // namespace consistency
105
106 } // namespace nvinfer1
107
108 #endif // NV_INFER_CONSISTENCY_H

```

10.7 NvInferLegacyDims.h File Reference

```
#include "NvInferRuntimeCommon.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.

10.7.1 Detailed Description

This file contains declarations of legacy dimensions types which use channel semantics in their names, and declarations on which those types rely.

10.8 NvInferLegacyDims.h

[Go to the documentation of this file.](#)

```

1  /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_INFER_LEGACY_DIMS_H
14 #define NV_INFER_LEGACY_DIMS_H
15
16 #include "NvInferRuntimeCommon.h"
17
18
19
20
21
22
23
24
25
26
27
28
29 namespace nvinfer1
30 {
31 {
32
33 class Dims2 : public Dims
34 {
35 {
36 public:
37     Dims2()
38         : Dims2(0, 0)
39     {
40     }
41
42     Dims2(int32_t d0, int32_t d1)
43     {
44         nbDims = 2;
45         d[0] = d0;
46         d[1] = d1;
47         for (int32_t i{nbDims}; i < Dims::MAX_DIMS; ++i)
48         {
49             d[i] = 0;
50         }
51     }
52 };
53
54 class DimsHW : public Dims2
55 {
56 public:
57     DimsHW()
58         : Dims2()
59     {
60     }
61
62     DimsHW(int32_t height, int32_t width)
63         : Dims2(height, width)
64     {
65     }
66
67     int32_t& h()
68     {

```

```

98     return d[0];
99 }
100
106 int32_t h() const
107 {
108     return d[0];
109 }
110
116 int32_t& w()
117 {
118     return d[1];
119 }
120
126 int32_t w() const
127 {
128     return d[1];
129 }
130 };
131
136 class Dims3 : public Dims2
137 {
138 public:
142     Dims3()
143         : Dims3(0, 0, 0)
144     {
145     }
146
154     Dims3(int32_t d0, int32_t d1, int32_t d2)
155         : Dims2(d0, d1)
156     {
157         nbDims = 3;
158         d[2] = d2;
159     }
160 };
161
166 class Dims4 : public Dims3
167 {
168 public:
172     Dims4()
173         : Dims4(0, 0, 0, 0)
174     {
175     }
176
185     Dims4(int32_t d0, int32_t d1, int32_t d2, int32_t d3)
186         : Dims3(d0, d1, d2)
187     {
188         nbDims = 4;
189         d[3] = d3;
190     }
191 };
192
193 } // namespace nvinfer1
194
195 #endif // NV_INFER_LEGACY_DIMS_H

```

10.9 NvInferPlugin.h File Reference

```

#include "NvInfer.h"
#include "NvInferPluginUtils.h"

```

Functions

- [TRT_DEPRECATED_API nvinfer1::IPluginV2 * createRPNROIPlugin](#) (int32_t featureStride, int32_t preNms↔ Top, int32_t nmsMaxOut, float iouThreshold, float minBoxSize, float spatialScale, [nvinfer1::DimsHW](#) pooling, [nvinfer1::Weights](#) anchorRatios, [nvinfer1::Weights](#) anchorScales)

Create a plugin layer that fuses the RPN and ROI pooling using user-defined parameters. Registered plugin type "↔ RPROI-TRT". Registered plugin version "1".

- **TRT_DEPRECATED_API** `nvinfer1::IPluginV2 * createNormalizePlugin (nvinfer1::Weights const *scales, bool acrossSpatial, bool channelShared, float eps)`
The Normalize plugin layer normalizes the input to have L2 norm of 1 with scale learnable. Registered plugin type "Normalize-TRT". Registered plugin version "1".
- **TRT_DEPRECATED_API** `nvinfer1::IPluginV2 * createPriorBoxPlugin (nvinfer1::plugin::PriorBoxParameters param)`
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. Registered plugin type "PriorBox-TRT". Registered plugin version "1".
- **TRT_DEPRECATED_API** `nvinfer1::IPluginV2 * createAnchorGeneratorPlugin (nvinfer1::plugin::GridAnchorParameters *param, int32_t numLayers)`
The Grid Anchor Generator plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions (H x W) for all feature maps. GridAnchorParameters defines a set of parameters for creating the GridAnchorGenerator plugin layer. Registered plugin type "GridAnchor-TRT". Registered plugin version "1".
- **TRT_DEPRECATED_API** `nvinfer1::IPluginV2 * createNMSPlugin (nvinfer1::plugin::DetectionOutputParameters param)`
The DetectionOutput plugin layer generates the detection output based on location and confidence predictions by doing non maximum suppression. DetectionOutputParameters defines a set of parameters for creating the DetectionOutput plugin layer. Registered plugin type "NMS-TRT". Registered plugin version "1".
- **TRT_DEPRECATED_API** `nvinfer1::IPluginV2 * createReorgPlugin (int32_t stride)`
*The Reorg plugin reshapes input of shape CxHxW into a (C*stride*stride)x(H/stride)x(W/stride) shape, used in YOLOv2. It does that by taking 1 x stride x stride slices from tensor and flattening them into (stride x stride) x 1 x 1 shape. Registered plugin type "Reorg-TRT". Registered plugin version "1".*
- **TRT_DEPRECATED_API** `nvinfer1::IPluginV2 * createRegionPlugin (nvinfer1::plugin::RegionParameters params)`
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 9416 pre-defined classifications, and these 9416 items are organized as work-tree structure). RegionParameters defines a set of parameters for creating the Region plugin layer. Registered plugin type "Region-TRT". Registered plugin version "1".
- **TRT_DEPRECATED_API** `nvinfer1::IPluginV2 * createBatchedNMSPlugin (nvinfer1::plugin::NMSParameters param)`
The BatchedNMS Plugin performs non-max-suppression on the input boxes, per batch, across all classes. It greedily selects a subset of bounding boxes in descending order of score. Prunes away boxes that have a high intersection-over-union (IOU) overlap with previously selected boxes. Bounding boxes are supplied as [y1, x1, y2, x2], where (y1, x1) and (y2, x2) are the coordinates of any diagonal pair of box corners and the coordinates can be provided as normalized (i.e., lying in the interval [0, 1]) or absolute. The plugin expects two inputs. Input0 is expected to be 4-D float boxes tensor of shape [batch_size, num_boxes, q, 4], where q can be either 1 (if shareLocation is true) or num_classes. Input1 is expected to be a 3-D float scores tensor of shape [batch_size, num_boxes, num_classes] representing a single score corresponding to each box. The plugin returns four outputs. num_detections : A [batch_size] int32 tensor indicating the number of valid detections per batch item. Can be less than keepTopK. Only the top num_detections[i] entries in nmsed_boxes[i], nmsed_scores[i] and nmsed_classes[i] are valid. nmsed_boxes : A [batch_size, max_detections, 4] float32 tensor containing the co-ordinates of non-max suppressed boxes. nmsed_scores : A [batch_size, max_detections] float32 tensor containing the scores for the boxes. nmsed_classes : A [batch_size, max_detections] float32 tensor containing the classes for the boxes.
- **TRT_DEPRECATED_API** `nvinfer1::IPluginV2 * createSplitPlugin (int32_t axis, int32_t *output_lengths, int32_t noutput)`
The Split Plugin performs a split operation on the input tensor. It splits the input tensor into several output tensors, each of a length corresponding to output_lengths. The split occurs along the axis specified by axis.
- **TRT_DEPRECATED_API** `nvinfer1::IPluginV2 * createInstanceNormalizationPlugin (float epsilon, nvinfer1::Weights scale_weights, nvinfer1::Weights bias_weights)`
*The Instance Normalization Plugin computes the instance normalization of an input tensor. The instance normalization is calculated as found in the paper <https://arxiv.org/abs/1607.08022>. The calculation is $y = \text{scale} * (x - \text{mean}) / \sqrt{\text{variance} + \text{epsilon}} + \text{bias}$ where mean and variance are computed per instance per channel.*
- `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.

10.9.1 Detailed Description

This is the API for the Nvidia provided TensorRT plugins.

10.9.2 Function Documentation

10.9.2.1 createAnchorGeneratorPlugin()

```
TRT_DEPRECATED_API nvinfer1::IPluginV2 * createAnchorGeneratorPlugin (
    nvinfer1::plugin::GridAnchorParameters * param,
    int32_t numLayers )
```

The Grid Anchor Generator plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions (H x W) for all feature maps. GridAnchorParameters defines a set of parameters for creating the GridAnchorGenerator plugin layer. Registered plugin type "GridAnchor-TRT". Registered plugin version "1".

Deprecated Deprecated in TensorRT 8.5. Use GridAnchorPluginCreator::createPlugin() to create an instance of "GridAnchor-TRT" version 1 plugin.

10.9.2.2 createBatchedNMSPlugin()

```
TRT_DEPRECATED_API nvinfer1::IPluginV2 * createBatchedNMSPlugin (
    nvinfer1::plugin::NMSParameters param )
```

The BatchedNMS Plugin performs non-max-suppression on the input boxes, per batch, across all classes. It greedily selects a subset of bounding boxes in descending order of score. Prunes away boxes that have a high intersection-over-union (IOU) overlap with previously selected boxes. Bounding boxes are supplied as [y1, x1, y2, x2], where (y1, x1) and (y2, x2) are the coordinates of any diagonal pair of box corners and the coordinates can be provided as normalized (i.e., lying in the interval [0, 1]) or absolute. The plugin expects two inputs. Input0 is expected to be 4-D float boxes tensor of shape [batch_size, num_boxes, q, 4], where q can be either 1 (if shareLocation is true) or num_classes. Input1 is expected to be a 3-D float scores tensor of shape [batch_size, num_boxes, num_classes] representing a single score corresponding to each box. The plugin returns four outputs. num_detections : A [batch_size] int32 tensor indicating the number of valid detections per batch item. Can be less than keepTopK. Only the top num_detections[i] entries in nmsed_boxes[i], nmsed_scores[i] and nmsed_classes[i] are valid. nmsed_boxes : A [batch_size, max_detections, 4] float32 tensor containing the co-ordinates of non-max suppressed boxes. nmsed_scores : A [batch_size, max_detections] float32 tensor containing the scores for the boxes. nmsed_classes : A [batch_size, max_detections] float32 tensor containing the classes for the boxes.

Registered plugin type "BatchedNMS-TRT". Registered plugin version "1".

The batched NMS plugin can require a lot of workspace due to intermediate buffer usage. To get the estimated workspace size for the plugin for a batch size, use the API plugin->getWorkspaceSize(batchSize).

Deprecated Deprecated in TensorRT 8.5. Use BatchedNMSPluginCreator::createPlugin() to create an instance of "BatchedNMS-TRT" version 1 plugin.

10.9.2.3 createInstanceNormalizationPlugin()

```
TRT_DEPRECATED_API nvinfer1::IPluginV2 * createInstanceNormalizationPlugin (
    float epsilon,
    nvinfer1::Weights scale_weights,
    nvinfer1::Weights bias_weights )
```

The Instance Normalization Plugin computes the instance normalization of an input tensor. The instance normalization is calculated as found in the paper <https://arxiv.org/abs/1607.08022>. The calculation is $y = \text{scale} * (x - \text{mean}) / \sqrt{\text{variance} + \text{epsilon}} + \text{bias}$ where mean and variance are computed per instance per channel.

Parameters

<i>epsilon</i>	The epsilon value to use to avoid division by zero.
<i>scale_weights</i>	The input 1-dimensional scale weights of size C to scale.
<i>bias_weights</i>	The input 1-dimensional bias weights of size C to offset.

Deprecated Deprecated in TensorRT 8.5. Use `InstanceNormalizationPluginCreator::createPlugin()` to create an instance of "InstanceNormalization-TRT" version 1 plugin.

10.9.2.4 createNMSPlugin()

```
TRT_DEPRECATED_API nvinfer1::IPluginV2 * createNMSPlugin (
    nvinfer1::plugin::DetectionOutputParameters param )
```

The DetectionOutput plugin layer generates the detection output based on location and confidence predictions by doing non maximum suppression. `DetectionOutputParameters` defines a set of parameters for creating the DetectionOutput plugin layer. Registered plugin type "NMS-TRT". Registered plugin version "1".

Deprecated Deprecated in TensorRT 8.5. Use `NMSPluginCreator::createPlugin()` to create an instance of "NMS-TRT" version 1 plugin.

10.9.2.5 createNormalizePlugin()

```
TRT_DEPRECATED_API nvinfer1::IPluginV2 * createNormalizePlugin (
    nvinfer1::Weights const * scales,
    bool acrossSpatial,
    bool channelShared,
    float eps )
```

The Normalize plugin layer normalizes the input to have L2 norm of 1 with scale learnable. Registered plugin type "Normalize-TRT". Registered plugin version "1".

Parameters

<i>scales</i>	Scale weights that are applied to the output tensor.
<i>acrossSpatial</i>	Whether to compute the norm over adjacent channels (<i>acrossSpatial</i> is true) or nearby spatial locations (within channel in which case <i>acrossSpatial</i> is false).
<i>channelShared</i>	Whether the scale weight(s) is shared across channels.
<i>eps</i>	Epsilon for not dividing by zero.

Deprecated Deprecated in TensorRT 8.5. Use `NormalizePluginCreator::createPlugin()` to create an instance of "←
Normalize_TRT" version 1 plugin.

10.9.2.6 createPriorBoxPlugin()

```
TRT_DEPRECATED_API nvinfer1::IPluginV2 * createPriorBoxPlugin (
    nvinfer1::plugin::PriorBoxParameters param )
```

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. Registered plugin type "PriorBox_TRT". Registered plugin version "1".

Deprecated Deprecated in TensorRT 8.5. Use `PriorBoxPluginCreator::createPlugin()` to create an instance of "Prior←
Box_TRT" version 1 plugin.

10.9.2.7 createRegionPlugin()

```
TRT_DEPRECATED_API nvinfer1::IPluginV2 * createRegionPlugin (
    nvinfer1::plugin::RegionParameters params )
```

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 9416 pre-defined classifications, and these 9416 items are organized as work-tree structure). `RegionParameters` defines a set of parameters for creating the Region plugin layer. Registered plugin type "Region_TRT". Registered plugin version "1".

Deprecated Deprecated in TensorRT 8.5. Use `RegionPluginCreator::createPlugin()` to create an instance of "Region←
_TRT" version 1 plugin.

10.9.2.8 createReorgPlugin()

```
TRT_DEPRECATED_API nvinfer1::IPluginV2 * createReorgPlugin (
    int32_t stride )
```

The Reorg plugin reshapes input of shape CxHxW into a (C*stride*stride)x(H/stride)x(W/stride) shape, used in YOLOv2. It does that by taking 1 x stride x stride slices from tensor and flattening them into (stride x stride) x 1 x 1 shape. Registered plugin type "Reorg_TRT". Registered plugin version "1".

Parameters

<i>stride</i>	Strides in H and W, it should divide both H and W. Also stride * stride should be less than or equal to C.
---------------	--

Deprecated Deprecated in TensorRT 8.5. Use `ReorgPluginCreator::createPlugin()` to create an instance of "Reorg_↵_TRT" version 1 plugin.

10.9.2.9 createRPNROIPlugin()

```
TRT_DEPRECATED_API nvinfer1::IPluginV2 * createRPNROIPlugin (
    int32_t featureStride,
    int32_t preNmsTop,
    int32_t nmsMaxOut,
    float iouThreshold,
    float minBoxSize,
    float spatialScale,
    nvinfer1::DimsHW pooling,
    nvinfer1::Weights anchorRatios,
    nvinfer1::Weights anchorScales )
```

Create a plugin layer that fuses the RPN and ROI pooling using user-defined parameters. Registered plugin type "RPROI_TRT". Registered plugin version "1".

Parameters

<i>featureStride</i>	Feature stride.
<i>preNmsTop</i>	Number of proposals to keep before applying NMS.
<i>nmsMaxOut</i>	Number of remaining proposals after applying NMS.
<i>iouThreshold</i>	IoU threshold.
<i>minBoxSize</i>	Minimum allowed bounding box size before scaling.
<i>spatialScale</i>	Spatial scale between the input image and the last feature map.
<i>pooling</i>	Spatial dimensions of pooled ROIs.
<i>anchorRatios</i>	Aspect ratios for generating anchor windows.
<i>anchorScales</i>	Scales for generating anchor windows.

Returns

Returns a FasterRCNN fused RPN+ROI pooling plugin. Returns nullptr on invalid inputs.

Deprecated Deprecated in TensorRT 8.5. Use `RPROIPluginCreator::createPlugin()` to create an instance of "RPROI_↵_TRT" version 1 plugin.

10.9.2.10 createSplitPlugin()

```
TRT_DEPRECATED_API nvinfer1::IPluginV2 * createSplitPlugin (
    int32_t axis,
    int32_t * output_lengths,
    int32_t noutput )
```

The Split Plugin performs a split operation on the input tensor. It splits the input tensor into several output tensors, each of a length corresponding to output_lengths. The split occurs along the axis specified by axis.

Parameters

<i>axis</i>	The axis to split on.
<i>output_lengths</i>	The lengths of the output tensors.
<i>noutput</i>	The number of output tensors.

Deprecated Deprecated in TensorRT 8.5 along with the "Split" plugin. Use `INetworkDefinition::addSlice()` to add slice layer(s) as necessary to accomplish the required effect.

10.9.2.11 initLibNvInferPlugins()

```
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

<i>logger</i>	Logger object to print plugin registration information
<i>libNamespace</i>	Namespace used to register all the plugins in this library

10.10 NvInferPlugin.h

[Go to the documentation of this file.](#)

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993–2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
```

```

8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_INFER_PLUGIN_H
14 #define NV_INFER_PLUGIN_H
15
16 #include "NvInfer.h"
17 #include "NvInferPluginUtils.h"
18
19 extern "C"
20 {
21     TRT_DEPRECATED_API nvinfer1::IPluginV2* createRPNROIPlugin(int32_t featureStride, int32_t preNmsTop,
22         int32_t nmsMaxOut,
23         float iouThreshold, float minBoxSize, float spatialScale, nvinfer1::DimsHW pooling,
24         nvinfer1::Weights anchorRatios, nvinfer1::Weights anchorScales);
25
26     TRT_DEPRECATED_API nvinfer1::IPluginV2* createNormalizePlugin(
27         nvinfer1::Weights const* scales, bool acrossSpatial, bool channelShared, float eps);
28
29     TRT_DEPRECATED_API nvinfer1::IPluginV2* createPriorBoxPlugin(nvinfer1::plugin::PriorBoxParameters param);
30
31     TRT_DEPRECATED_API nvinfer1::IPluginV2* createAnchorGeneratorPlugin(
32         nvinfer1::plugin::GridAnchorParameters* param, int32_t numLayers);
33
34     TRT_DEPRECATED_API nvinfer1::IPluginV2* createNMSPlugin(nvinfer1::plugin::DetectionOutputParameters
35         param);
36
37     TRT_DEPRECATED_API nvinfer1::IPluginV2* createReorgPlugin(int32_t stride);
38
39     TRT_DEPRECATED_API nvinfer1::IPluginV2* createRegionPlugin(nvinfer1::plugin::RegionParameters params);
40
41     TRT_DEPRECATED_API nvinfer1::IPluginV2* createBatchedNMSPlugin(nvinfer1::plugin::NMSParameters param);
42
43     TRT_DEPRECATED_API nvinfer1::IPluginV2* createSplitPlugin(int32_t axis, int32_t* output_lengths, int32_t
44         noutput);
45
46     TRT_DEPRECATED_API nvinfer1::IPluginV2* createInstanceNormalizationPlugin(
47         float epsilon, nvinfer1::Weights scale_weights, nvinfer1::Weights bias_weights);
48
49     TENSORRT_API bool initLibNvInferPlugins(void* logger, char const* libNamespace);
50 } // extern "C"
51
52 #endif // NV_INFER_PLUGIN_H

```

10.11 NvInferPluginUtils.h File Reference

```
#include "NvInferRuntimeCommon.h"
```

Classes

- struct [nvinfer1::plugin::Quadruple](#)

The *Permute* plugin layer permutes the input tensor by changing the memory order of the data. [Quadruple](#) defines a structure that contains an array of 4 integers. They can represent the permute orders or the strides in each dimension.

- struct [nvinfer1::plugin::PriorBoxParameters](#)

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. It contains:

- struct [nvinfer1::plugin::RPROIParams](#)

[RPROIParams](#) is used to create the *RPROIPlugin* instance. It contains:

- struct [nvinfer1::plugin::GridAnchorParameters](#)

The Anchor Generator plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions ($H \times W$). [GridAnchorParameters](#) defines a set of parameters for creating the plugin layer for all feature maps. It contains:

- struct [nvinfer1::plugin::DetectionOutputParameters](#)

The *DetectionOutput* plugin layer generates the detection output based on location and confidence predictions by doing non maximum suppression. This plugin first decodes the bounding boxes based on the anchors generated. It then performs non_max_suppression on the decoded bounding boxes. [DetectionOutputParameters](#) defines a set of parameters for creating the *DetectionOutput* plugin layer. It contains:

- 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: 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.

- struct [nvinfer1::plugin::NMSParameters](#)

The [NMSParameters](#) are used by the *BatchedNMSPlugin* for performing the non_max_suppression operation over boxes for object detection networks.

Namespaces

- namespace [nvinfer1](#)

The *TensorRT API version 1* namespace.

- namespace [nvinfer1::plugin](#)

Enumerations

- enum class [nvinfer1::plugin::CodeTypeSSD](#) : int32_t { [nvinfer1::plugin::CORNER](#) = 0 , [nvinfer1::plugin::CENTER_SIZE](#) = 1 , [nvinfer1::plugin::CORNER_SIZE](#) = 2 , [nvinfer1::plugin::TF_CENTER](#) = 3 }

The type of encoding used for decoding the bounding boxes and loc.data.

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) 1993-2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
```

```

10  * its affiliates is strictly prohibited.
11  */
12
13  #ifndef NV_INFER_PLUGIN_UTILS_H
14  #define NV_INFER_PLUGIN_UTILS_H
15
16  #include "NvInferRuntimeCommon.h"
17
18
19
20
21
22
23
24
25  namespace nvinfer1
26  {
27      namespace plugin
28      {
29
30
31
32
33
34
35      typedef struct
36      {
37          int32_t data[4];
38      } Quadruple;
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60  struct PriorBoxParameters
61  {
62      float *minSize, *maxSize, *aspectRatios;
63      int32_t numMinSize, numMaxSize, numAspectRatios;
64      bool flip;
65      bool clip;
66      float variance[4];
67      int32_t imgH, imgW;
68      float stepH, stepW;
69      float offset;
70  };
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87  struct RPROIParams
88  {
89      int32_t poolingH;
90      int32_t poolingW;
91      int32_t featureStride;
92      int32_t preNmsTop;
93      int32_t nmsMaxOut;
94      int32_t anchorsRatioCount;
95      int32_t anchorsScaleCount;
96      float iouThreshold;
97      float minBoxSize;
98      float spatialScale;
99  };
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114  struct GridAnchorParameters
115  {
116      float minSize, maxSize;
117      float* aspectRatios;
118      int32_t numAspectRatios, H, W;
119      float variance[4];
120  };
121
122
123
124
125
126  enum class CodeTypeSSD : int32_t
127  {
128      CORNER = 0,
129      CENTER_SIZE = 1,
130      CORNER_SIZE = 2,
131      TF_CENTER = 3
132  };
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153  struct DetectionOutputParameters
154  {
155      bool shareLocation, varianceEncodedInTarget;
156      int32_t backgroundLabelId, numClasses, topK, keepTopK;
157      float confidenceThreshold, nmsThreshold;
158      CodeTypeSSD codeType;
159      int32_t inputOrder[3];
160      bool confSigmoid;
161      bool isNormalized;
162      bool isBatchAgnostic{true};
163  };
164
165
166
167
168  struct softmaxTree
169  {
170      int32_t* leaf;
171      int32_t n;
172      int32_t* parent;
173      int32_t* child;
174      int32_t* group;

```

```

175     char** name;
176
177     int32_t groups;
178     int32_t* groupSize;
179     int32_t* groupOffset;
180 };
181
182 struct RegionParameters
183 {
184     int32_t num;
185     int32_t coords;
186     int32_t classes;
187     softmaxTree* smTree;
188 };
189
216
217 struct NMSPParameters
218 {
219     bool shareLocation;
220     int32_t backgroundLabelId, numClasses, topK, keepTopK;
221     float scoreThreshold, iouThreshold;
222     bool isNormalized;
223 };
224
225 } // namespace plugin
226 } // namespace nvinfer1
227
228 #endif // NV_INFER_PLUGIN_UTILS_H

```

10.13 NvInferRuntime.h File Reference

```

#include "NvInferImpl.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< TensorLocation >](#)
Maximum number of elements in TensorLocation enum.
- class [nvinfer1::IDimensionExpr](#)
- class [nvinfer1::IExprBuilder](#)
- class [nvinfer1::DimsExprs](#)
- class [nvinfer1::DynamicPluginTensorDesc](#)
- class [nvinfer1::IPluginV2DynamicExt](#)
- class [nvinfer1::IProfiler](#)
Application-implemented interface for profiling.
- class [nvinfer1::IRuntime](#)
Allows a serialized functionally unsafe engine to be deserialized.
- class [nvinfer1::IRefitter](#)

- *Updates weights in an engine.*
- class `nvinfer1::IOptimizationProfile`
Optimization profile for dynamic input dimensions and shape tensors.
- class `nvinfer1::ICudaEngine`
An engine for executing inference on a built network, with functionally unsafe features.
- class `nvinfer1::IOutputAllocator`
Callback from ExecutionContext::enqueueV3()
- class `nvinfer1::IExecutionContext`
Context for executing inference using an engine, with functionally unsafe features.
- 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.

Namespaces

- namespace `nvinfer1`
The TensorRT API version 1 namespace.
- namespace `nvinfer1::impl`

Macros

- `#define REGISTER_TENSORRT_PLUGIN(name) static nvinfer1::PluginRegistrar<name> plugin↵ Registrar##name {}`

Typedefs

- using `nvinfer1::TempfileControlFlags` = `uint32_t`
Represents a collection of one or more TempfileControlFlag values combined using bitwise-OR operations.
- using `nvinfer1::TacticSources` = `uint32_t`
Represents a collection of one or more TacticSource values combine using bitwise-OR operations.

Enumerations

- enum class `nvinfer1::EngineCapability` : `int32_t` {
`nvinfer1::kSTANDARD` = 0 , `nvinfer1::kDEFAULT` = `kSTANDARD` , `nvinfer1::kSAFETY` = 1 ,
`nvinfer1::kSAFE_GPU` = `kSAFETY` ,
`nvinfer1::kDLA_STANDALONE` = 2 , `nvinfer1::kSAFE_DLA` = `kDLA_STANDALONE` }
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::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` , `nvinfer1::kDLA` }

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::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::kCUBLAS` = 0 , `nvinfer1::kCUBLAS_LT` = 1 , `nvinfer1::kCUDNN` = 2 , `nvinfer1::kEDGE_MASK_CONVOLUTIONS` = 3 ,
`nvinfer1::kJIT_CONVOLUTIONS` = 4 }

List of tactic sources for TensorRT.

- enum class `nvinfer1::ProfilingVerbosity` : `int32_t` {
`nvinfer1::kLAYER_NAMES_ONLY` = 0 , `nvinfer1::kNONE` = 1 , `nvinfer1::kDETAILED` = 2 , `nvinfer1::kDEFAULT` = `kLAYER_NAMES_ONLY` ,
`nvinfer1::kVERBOSE` = `kDETAILED` }

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

- enum class `nvinfer1::LayerInformationFormat` : `int32_t` { `nvinfer1::kONELINE` = 0 , `nvinfer1::kJSON` = 1 }

The format in which the IEngineInspector prints the layer information.

Functions

- `template<> constexpr int32_t nvinfer1::EnumMax< DimensionOperation > () noexcept`
Maximum number of elements in DimensionOperation enum.
- `template<> constexpr int32_t nvinfer1::EnumMax< WeightsRole > () noexcept`
Maximum number of elements in WeightsRole enum.
- `template<> constexpr int32_t nvinfer1::EnumMax< DeviceType > () noexcept`
Maximum number of elements in DeviceType enum.
- `template<> constexpr int32_t nvinfer1::EnumMax< TempfileControlFlag > () noexcept`
Maximum number of elements in TempfileControlFlag enum.
- `template<> constexpr int32_t nvinfer1::EnumMax< OptProfileSelector > () noexcept`
Number of different values of OptProfileSelector enum.
- `template<> constexpr int32_t nvinfer1::EnumMax< TacticSource > () noexcept`
Maximum number of tactic sources in TacticSource enum.
- `template<> constexpr int32_t nvinfer1::EnumMax< ProfilingVerbosity > () noexcept`
Maximum number of profile verbosity levels in ProfilingVerbosity enum.
- `template<> constexpr int32_t nvinfer1::EnumMax< LayerInformationFormat > () noexcept`
- `nvinfer1::IPluginRegistry * getPluginRegistry () noexcept`
Return the plugin registry.
- `nvinfer1::ILogger * getLogger () noexcept`
Return the logger object.

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 REGISTER_TENSORRT_PLUGIN

```
#define REGISTER_TENSORRT_PLUGIN(  
    name )    static nvinfer1::PluginRegistrar<name> pluginRegistrar##name {}
```

10.13.3 Function Documentation

10.13.3.1 getLogger()

```
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.

10.13.3.2 getPluginRegistry()

```
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) 1993-2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_INFER_RUNTIME_H
14 #define NV_INFER_RUNTIME_H
15
16
17
18
19
20
21
22 #include "NvInferImpl.h"
23 #include "NvInferRuntimeCommon.h"
24
25 namespace nvinfer1
26 {
27
28     class IExecutionContext;
29     class ICudaEngine;
30     class IPluginFactory;
31     class IEngineInspector;
32
33
34
35
36
37
38
39
40
41
42     class INoCopy
43     {
44     protected:
45         INoCopy() = default;
46         virtual ~INoCopy() = default;
47         INoCopy(INoCopy const& other) = delete;
48         INoCopy& operator=(INoCopy const& other) = delete;
49         INoCopy(INoCopy&& other) = delete;
50         INoCopy& operator=(INoCopy&& other) = delete;
51     };
52
53
54
55
56
57     enum class EngineCapability : int32_t
58     {
59     {
60         kSTANDARD = 0,
61
62         kDEFAULT TRT_DEPRECATED_ENUM = kSTANDARD,
63
64         kSAFETY = 1,
65
66         kSAFE_GPU TRT_DEPRECATED_ENUM = kSAFETY,
67
68         kDLA_STANDALONE = 2,
69
70         kSAFE_DLA TRT_DEPRECATED_ENUM = kDLA_STANDALONE,
71     };
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101     namespace impl
102     {
103     {
104         template <>
105         struct EnumMaxImpl<EngineCapability>
106         {
107             static constexpr int32_t kVALUE = 3;
108         };
109     } // namespace impl
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125     class Weights
126     {
127     public:
128         DataType type;
129         void const* values;
130         int64_t count;
131     };
132
133
134
135
136
137
138
139
140
141
142
143     class IHostMemory : public INoCopy
144     {
145     public:
146         virtual ~IHostMemory() noexcept = default;

```

```

147
149     void* data() const noexcept
150     {
151         return mImpl->data();
152     }
153
155     std::size_t size() const noexcept
156     {
157         return mImpl->size();
158     }
159
161     DataType type() const noexcept
162     {
163         return mImpl->type();
164     }
172     TRT_DEPRECATED void destroy() noexcept
173     {
174         delete this;
175     }
176
177 protected:
178     apiv::VHostMemory* mImpl;
179 };
180
191 enum class DimensionOperation : int32_t
192 {
193     kSUM = 0,
194     kPROD = 1,
195     kMAX = 2,
196     kMIN = 3,
197     kSUB = 4,
198     kEQUAL = 5,
199     kLESS = 6,
200     kFLOOR_DIV = 7,
201     kCEIL_DIV = 8
202 };
203
205 template <>
206 constexpr inline int32_t EnumMax<DimensionOperation>() noexcept
207 {
208     return 9;
209 }
210
215 enum class TensorLocation : int32_t
216 {
217     kDEVICE = 0,
218     kHOST = 1,
219 };
220
221 namespace impl
222 {
224 template <>
225 struct EnumMaxImpl<TensorLocation>
226 {
227     static constexpr int32_t kVALUE = 2;
228 };
229 } // namespace impl
230
243 class IDimensionExpr : public INoCopy
244 {
245 public:
247     bool isConstant() const noexcept
248     {
249         return mImpl->isConstant();
250     }
251
254     int32_t getConstantValue() const noexcept
255     {
256         return mImpl->getConstantValue();
257     }
258
259 protected:
260     apiv::VDimensionExpr* mImpl;
261     virtual ~IDimensionExpr() noexcept = default;
262 };
263
281 class IExprBuilder : public INoCopy
282 {
283 public:
285     IDimensionExpr const* constant(int32_t value) noexcept
286     {

```

```

287         return mImpl->constant(value);
288     }
289
292     IDimensionExpr const* operation(
293         DimensionOperation op, IDimensionExpr const& first, IDimensionExpr const& second) noexcept
294     {
295         return mImpl->operation(op, first, second);
296     }
297
298 protected:
299     api::VExprBuilder* mImpl;
300     virtual ~IExprBuilder() noexcept = default;
301 };
302
308 class DimsExprs
309 {
310 public:
311     int32_t nbDims;
312     IDimensionExpr const* d[Dims::MAX_DIMS];
313 };
314
320 struct DynamicPluginTensorDesc
321 {
322     PluginTensorDesc desc;
323
324     Dims min;
325
326     Dims max;
327
328     Dims max;
329 };
330
331
350 class IPluginV2DynamicExt : public nvinfer1::IPluginV2Ext
351 {
352 public:
353     IPluginV2DynamicExt* clone() const noexcept override = 0;
354
379     virtual DimsExprs getOutputDimensions(
380         int32_t outputIndex, DimsExprs const* inputs, int32_t nbInputs, IExprBuilder& exprBuilder) noexcept =
381         0;
382
385     static constexpr int32_t kFORMAT_COMBINATION_LIMIT = 100;
386
419     virtual bool supportsFormatCombination(
420         int32_t pos, PluginTensorDesc const* inOut, int32_t nbInputs, int32_t nbOutputs) noexcept = 0;
421
460     virtual void configurePlugin(DynamicPluginTensorDesc const* in, int32_t nbInputs,
461         DynamicPluginTensorDesc const* out, int32_t nbOutputs) noexcept = 0;
462
472     virtual size_t getWorkspaceSize(PluginTensorDesc const* inputs, int32_t nbInputs, PluginTensorDesc const*
473         outputs,
474         int32_t nbOutputs) const noexcept = 0;
475
487     virtual int32_t enqueue(PluginTensorDesc const* inputDesc, PluginTensorDesc const* outputDesc,
488         void const* const* inputs, void* const* outputs, void* workspace, cudaStream_t stream) noexcept = 0;
489
490 protected:
491     int32_t getTensorRTVersion() const noexcept override
492     {
493         return (static_cast<int32_t>(PluginVersion::kV2_DYNAMICEXT) << 24 | (NV_TENSORRT_VERSION & 0xFFFF));
494     }
495
502     virtual ~IPluginV2DynamicExt() noexcept {}
503
504 private:
505     // Following are obsolete base class methods, and must not be implemented or used.
506
507     void configurePlugin(Dims const*, int32_t, Dims const*, int32_t, DataType const*, DataType const*, bool
508         const*,
509         bool const*, PluginFormat, int32_t) noexcept override final
510     {
511     }
512
513     bool supportsFormat(DataType, PluginFormat) const noexcept override final
514     {
515         return false;
516     }
517
518     Dims getOutputDimensions(int32_t, Dims const*, int32_t) noexcept override final
519     {
520         return Dims{-1, {}};
521     }
522

```

```

523     bool isOutputBroadcastAcrossBatch(int32_t, bool const*, int32_t) const noexcept override final
524     {
525         return false;
526     }
527
528     bool canBroadcastInputAcrossBatch(int32_t) const noexcept override final
529     {
530         return true;
531     }
532
533     size_t getWorkspaceSize(int32_t) const noexcept override final
534     {
535         return 0;
536     }
537
538     int32_t enqueue(int32_t, void const* const*, void* const*, void*, cudaStream_t) noexcept override final
539     {
540         return 1;
541     }
542 };
543
544 class IProfiler
545 {
546 public:
547     virtual void reportLayerTime(char const* layerName, float ms) noexcept = 0;
548     virtual ~IProfiler() noexcept {}
549 };
550
551 enum class WeightsRole : int32_t
552 {
553     kKERNEL = 0,
554     kBIAS = 1,
555     kSHIFT = 2,
556     kSCALE = 3,
557     kCONSTANT = 4,
558     kANY = 5,
559 };
560
561 template <>
562 constexpr inline int32_t EnumMax<WeightsRole>() noexcept
563 {
564     return 6;
565 }
566
567 enum class DeviceType : int32_t
568 {
569     kGPU,
570     kDLA,
571 };
572
573 template <>
574 constexpr inline int32_t EnumMax<DeviceType>() noexcept
575 {
576     return 2;
577 }
578
579 enum class TempfileControlFlag : int32_t
580 {
581     kALLOW_IN_MEMORY_FILES = 0,
582     kALLOW_TEMPORARY_FILES = 1,
583 };
584
585 template <>
586 constexpr inline int32_t EnumMax<TempfileControlFlag>() noexcept
587 {
588     return 2;
589 }
590
591 using TempfileControlFlags = uint32_t;
592
593 class IRuntime : public INoCopy
594 {
595 public:
596     virtual ~IRuntime() noexcept = default;
597
598     TRT_DEPRECATED nvinfer1::ICudaEngine* deserializeCudaEngine(
599         void const* blob, std::size_t size, IPluginFactory* pluginFactory) noexcept
600     {
601         return mImpl->deserializeCudaEngine(blob, size, nullptr);
602     }
603 };

```

```

676     }
677
688     void setDLACore(int32_t dlaCore) noexcept
689     {
690         mImpl->setDLACore(dlaCore);
691     }
692
693     int32_t getDLACore() const noexcept
694     {
695         return mImpl->getDLACore();
696     }
697
698     int32_t getNbDLACores() const noexcept
699     {
700         return mImpl->getNbDLACores();
701     }
702
703     TRT_DEPRECATED void destroy() noexcept
704     {
705         delete this;
706     }
707
708     void setGpuAllocator(IGpuAllocator* allocator) noexcept
709     {
710         mImpl->setGpuAllocator(allocator);
711     }
712
713     //
714     void setErrorRecorder(IErrorRecorder* recorder) noexcept
715     {
716         mImpl->setErrorRecorder(recorder);
717     }
718
719     IErrorRecorder* getErrorRecorder() const noexcept
720     {
721         return mImpl->getErrorRecorder();
722     }
723
724     ICudaEngine* deserializeCudaEngine(void const* blob, std::size_t size) noexcept
725     {
726         return mImpl->deserializeCudaEngine(blob, size, nullptr);
727     }
728
729     ILogger* getLogger() const noexcept
730     {
731         return mImpl->getLogger();
732     }
733
734     bool setMaxThreads(int32_t maxThreads) noexcept
735     {
736         return mImpl->setMaxThreads(maxThreads);
737     }
738
739     int32_t getMaxThreads() const noexcept
740     {
741         return mImpl->getMaxThreads();
742     }
743
744     void setTemporaryDirectory(char const* path) noexcept
745     {
746         return mImpl->setTemporaryDirectory(path);
747     }
748
749     char const* getTemporaryDirectory() const noexcept
750     {
751         return mImpl->getTemporaryDirectory();
752     }
753
754     void setTempfileControlFlags(TempfileControlFlags flags) noexcept
755     {
756         return mImpl->setTempfileControlFlags(flags);
757     }
758
759     TempfileControlFlags getTempfileControlFlags() const noexcept
760     {
761         return mImpl->getTempfileControlFlags();
762     }
763
764     IPluginRegistry& getPluginRegistry() noexcept
765     {
766         return mImpl->getPluginRegistry();
767     }

```



```

905     }
906
921     IRuntime* loadRuntime(char const* path) noexcept
922     {
923         return mImpl->loadRuntime(path);
924     }
925
933     void setEngineHostCodeAllowed(bool allowed) noexcept
934     {
935         return mImpl->setEngineHostCodeAllowed(allowed);
936     }
937
943     bool getEngineHostCodeAllowed() const noexcept
944     {
945         return mImpl->getEngineHostCodeAllowed();
946     }
947
948 protected:
949     apiv::VRuntime* mImpl;
950 };
951
959 class IRefitter : public INoCopy
960 {
961 public:
962     virtual ~IRefitter() noexcept = default;
963
977     bool setWeights(char const* layerName, WeightsRole role, Weights weights) noexcept
978     {
979         return mImpl->setWeights(layerName, role, weights);
980     }
981
992     bool refitCudaEngine() noexcept
993     {
994         return mImpl->refitCudaEngine();
995     }
996
1013     int32_t getMissing(int32_t size, char const** layerNames, WeightsRole* roles) noexcept
1014     {
1015         return mImpl->getMissing(size, layerNames, roles);
1016     }
1017
1030     int32_t getAll(int32_t size, char const** layerNames, WeightsRole* roles) noexcept
1031     {
1032         return mImpl->getAll(size, layerNames, roles);
1033     }
1034
1040     TRT_DEPRECATED void destroy() noexcept
1041     {
1042         delete this;
1043     }
1044
1060     bool setDynamicRange(char const* tensorName, float min, float max) noexcept
1061     {
1062         return mImpl->setDynamicRange(tensorName, min, max);
1063     }
1064
1074     float getDynamicRangeMin(char const* tensorName) const noexcept
1075     {
1076         return mImpl->getDynamicRangeMin(tensorName);
1077     }
1078
1088     float getDynamicRangeMax(char const* tensorName) const noexcept
1089     {
1090         return mImpl->getDynamicRangeMax(tensorName);
1091     }
1092
1104     int32_t getTensorsWithDynamicRange(int32_t size, char const** tensorNames) const noexcept
1105     {
1106         return mImpl->getTensorsWithDynamicRange(size, tensorNames);
1107     }
1108
1120     //
1123     void setErrorRecorder(IErrorRecorder* recorder) noexcept
1124     {
1125         mImpl->setErrorRecorder(recorder);
1126     }
1127
1138     IErrorRecorder* getErrorRecorder() const noexcept
1139     {
1140         return mImpl->getErrorRecorder();
1141     }

```

```

1142
1159     bool setNamedWeights(char const* name, Weights weights) noexcept
1160     {
1161         return mImpl->setNamedWeights(name, weights);
1162     }
1163
1179     int32_t getMissingWeights(int32_t size, char const** weightsNames) noexcept
1180     {
1181         return mImpl->getMissingWeights(size, weightsNames);
1182     }
1183
1195     int32_t getAllWeights(int32_t size, char const** weightsNames) noexcept
1196     {
1197         return mImpl->getAllWeights(size, weightsNames);
1198     }
1199
1205     ILogger* getLogger() const noexcept
1206     {
1207         return mImpl->getLogger();
1208     }
1209
1219     bool setMaxThreads(int32_t maxThreads) noexcept
1220     {
1221         return mImpl->setMaxThreads(maxThreads);
1222     }
1223
1233     int32_t getMaxThreads() const noexcept
1234     {
1235         return mImpl->getMaxThreads();
1236     }
1237
1238 protected:
1239     apiv::VRefitter* mImpl;
1240 };
1241
1252 enum class OptProfileSelector : int32_t
1253 {
1254     kMIN = 0,
1255     kOPT = 1,
1256     kMAX = 2
1257 };
1258
1264 template <>
1265 constexpr inline int32_t EnumMax<OptProfileSelector>() noexcept
1266 {
1267     return 3;
1268 }
1269
1292 class IOptimizationProfile : public INoCopy
1293 {
1294 public:
1322     bool setDimensions(char const* inputName, OptProfileSelector select, Dims dims) noexcept
1323     {
1324         return mImpl->setDimensions(inputName, select, dims);
1325     }
1326
1334     Dims getDimensions(char const* inputName, OptProfileSelector select) const noexcept
1335     {
1336         return mImpl->getDimensions(inputName, select);
1337     }
1338
1380     bool setShapeValues(
1381         char const* inputName, OptProfileSelector select, int32_t const* values, int32_t nbValues) noexcept
1382     {
1383         return mImpl->setShapeValues(inputName, select, values, nbValues);
1384     }
1385
1394     int32_t getNbShapeValues(char const* inputName) const noexcept
1395     {
1396         return mImpl->getNbShapeValues(inputName);
1397     }
1398
1406     int32_t const* getShapeValues(char const* inputName, OptProfileSelector select) const noexcept
1407     {
1408         return mImpl->getShapeValues(inputName, select);
1409     }
1410
1424     bool setExtraMemoryTarget(float target) noexcept
1425     {
1426         return mImpl->setExtraMemoryTarget(target);
1427     }

```

```

1428
1436     float getExtraMemoryTarget() const noexcept
1437     {
1438         return mImpl->getExtraMemoryTarget();
1439     }
1440
1453     bool isValid() const noexcept
1454     {
1455         return mImpl->isValid();
1456     }
1457
1458 protected:
1459     apiv::VOptimizationProfile* mImpl;
1460     virtual ~IOptimizationProfile() noexcept = default;
1461 };
1462
1471 enum class TacticSource : int32_t
1472 {
1473     kCUBLAS = 0,
1474     kCUBLAS_LT = 1,
1475     kCUDNN = 2,
1476
1477     kEDGE_MASK_CONVOLUTIONS = 3,
1478
1479     kJIT_CONVOLUTIONS = 4,
1480 };
1481
1492 template <>
1493 constexpr inline int32_t EnumMax<TacticSource>() noexcept
1494 {
1495     return 5;
1496 }
1497
1498 using TacticSources = uint32_t;
1499
1506 enum class ProfilingVerbosity : int32_t
1507 {
1508     kLAYER_NAMES_ONLY = 0,
1509     kNONE = 1,
1510     kDETAILED = 2,
1511
1512     kDEFAULT_TRT_DEPRECATED_ENUM = kLAYER_NAMES_ONLY,
1513     kVERBOSE_TRT_DEPRECATED_ENUM = kDETAILED
1514 };
1515
1526 template <>
1527 constexpr inline int32_t EnumMax<ProfilingVerbosity>() noexcept
1528 {
1529     return 3;
1530 }
1531
1532 class ICudaEngine : public INoCopy
1533 {
1534 public:
1535     virtual ~ICudaEngine() noexcept = default;
1536
1537     TRT_DEPRECATED int32_t getNbBindings() const noexcept
1538     {
1539         return mImpl->getNbBindings();
1540     }
1541
1542     TRT_DEPRECATED int32_t getBindingIndex(char const* name) const noexcept
1543     {
1544         return mImpl->getBindingIndex(name);
1545     }
1546
1547     TRT_DEPRECATED char const* getBindingName(int32_t bindingIndex) const noexcept
1548     {
1549         return mImpl->getBindingName(bindingIndex);
1550     }
1551
1552     TRT_DEPRECATED bool bindingIsInput(int32_t bindingIndex) const noexcept
1553     {
1554         return mImpl->bindingIsInput(bindingIndex);
1555     }
1556
1557     TRT_DEPRECATED Dims getBindingDimensions(int32_t bindingIndex) const noexcept
1558     {
1559         return mImpl->getBindingDimensions(bindingIndex);
1560     }
1561 }

```

```

1667     Dims getTensorShape(char const* tensorName) const noexcept
1668     {
1669         return mImpl->getTensorShape(tensorName);
1670     }
1671
1682     TRT_DEPRECATED DataType getBindingDataType(int32_t bindingIndex) const noexcept
1683     {
1684         return mImpl->getBindingDataType(bindingIndex);
1685     }
1686
1697     DataType getTensorDataType(char const* tensorName) const noexcept
1698     {
1699         return mImpl->getTensorDataType(tensorName);
1700     }
1701
1713     TRT_DEPRECATED int32_t getMaxBatchSize() const noexcept
1714     {
1715         return mImpl->getMaxBatchSize();
1716     }
1717
1727     int32_t getNbLayers() const noexcept
1728     {
1729         return mImpl->getNbLayers();
1730     }
1731
1741     IHostMemory* serialize() const noexcept
1742     {
1743         return mImpl->serialize();
1744     }
1745
1758     IExecutionContext* createExecutionContext() noexcept
1759     {
1760         return mImpl->createExecutionContext();
1761     }
1762
1770     TRT_DEPRECATED void destroy() noexcept
1771     {
1772         delete this;
1773     }
1774
1788     TRT_DEPRECATED TensorLocation getLocation(int32_t bindingIndex) const noexcept
1789     {
1790         return mImpl->getLocation(bindingIndex);
1791     }
1792
1805     TensorLocation getTensorLocation(char const* tensorName) const noexcept
1806     {
1807         return mImpl->getTensorLocation(tensorName);
1808     }
1809
1825     bool isShapeInferenceIO(char const* tensorName) const noexcept
1826     {
1827         return mImpl->isShapeInferenceIO(tensorName);
1828     }
1829
1839     TensorIOMode getTensorIOMode(char const* tensorName) const noexcept
1840     {
1841         return mImpl->getTensorIOMode(tensorName);
1842     }
1843
1848     IExecutionContext* createExecutionContextWithoutDeviceMemory() noexcept
1849     {
1850         return mImpl->createExecutionContextWithoutDeviceMemory();
1851     }
1852
1858     size_t getDeviceMemorySize() const noexcept
1859     {
1860         return mImpl->getDeviceMemorySize();
1861     }
1862
1868     bool isRefittable() const noexcept
1869     {
1870         return mImpl->isRefittable();
1871     }
1872
1885     TRT_DEPRECATED int32_t getBindingBytesPerComponent(int32_t bindingIndex) const noexcept
1886     {
1887         return mImpl->getBindingBytesPerComponent(bindingIndex);
1888     }
1889
1902     int32_t getTensorBytesPerComponent(char const* tensorName) const noexcept

```

```

1903     {
1904         return mImpl->getTensorBytesPerComponent(tensorName);
1905     }
1906
1918     TRT_DEPRECATED int32_t getBindingComponentsPerElement(int32_t bindingIndex) const noexcept
1919     {
1920         return mImpl->getBindingComponentsPerElement(bindingIndex);
1921     }
1922
1935     int32_t getTensorComponentsPerElement(char const* tensorName) const noexcept
1936     {
1937         return mImpl->getTensorComponentsPerElement(tensorName);
1938     }
1939
1949     TRT_DEPRECATED TensorFormat getBindingFormat(int32_t bindingIndex) const noexcept
1950     {
1951         return mImpl->getBindingFormat(bindingIndex);
1952     }
1953
1962     TensorFormat getTensorFormat(char const* tensorName) const noexcept
1963     {
1964         return mImpl->getTensorFormat(tensorName);
1965     }
1966
1986     TRT_DEPRECATED char const* getBindingFormatDesc(int32_t bindingIndex) const noexcept
1987     {
1988         return mImpl->getBindingFormatDesc(bindingIndex);
1989     }
1990
2008     char const* getTensorFormatDesc(char const* tensorName) const noexcept
2009     {
2010         return mImpl->getTensorFormatDesc(tensorName);
2011     }
2012
2024     TRT_DEPRECATED int32_t getBindingVectorizedDim(int32_t bindingIndex) const noexcept
2025     {
2026         return mImpl->getBindingVectorizedDim(bindingIndex);
2027     }
2028
2039     int32_t getTensorVectorizedDim(char const* tensorName) const noexcept
2040     {
2041         return mImpl->getTensorVectorizedDim(tensorName);
2042     }
2043
2054     char const* getName() const noexcept
2055     {
2056         return mImpl->getName();
2057     }
2058
2065     int32_t getNbOptimizationProfiles() const noexcept
2066     {
2067         return mImpl->getNbOptimizationProfiles();
2068     }
2069
2097     TRT_DEPRECATED Dims getProfileDimensions(
2098         int32_t bindingIndex, int32_t profileIndex, OptProfileSelector select) const noexcept
2099     {
2100         return mImpl->getProfileDimensions(bindingIndex, profileIndex, select);
2101     }
2102
2118     Dims getProfileShape(char const* tensorName, int32_t profileIndex, OptProfileSelector select) const
2119     noexcept
2120     {
2121         return mImpl->getProfileShape(tensorName, profileIndex, select);
2122     }
2147     TRT_DEPRECATED int32_t const* getProfileShapeValues(
2148         int32_t profileIndex, int32_t inputIndex, OptProfileSelector select) const noexcept
2149     {
2150         return mImpl->getProfileShapeValues(profileIndex, inputIndex, select);
2151     }
2152
2186     TRT_DEPRECATED bool isShapeBinding(int32_t bindingIndex) const noexcept
2187     {
2188         return mImpl->isShapeBinding(bindingIndex);
2189     }
2190
2203     TRT_DEPRECATED bool isExecutionBinding(int32_t bindingIndex) const noexcept
2204     {
2205         return mImpl->isExecutionBinding(bindingIndex);
2206     }

```

```

2207
2218 EngineCapability getEngineCapability() const noexcept
2219 {
2220     return mImpl->getEngineCapability();
2221 }
2222
2233 //
2236 void setErrorRecorder(IErrorRecorder* recorder) noexcept
2237 {
2238     return mImpl->setErrorRecorder(recorder);
2239 }
2240
2251 IErrorRecorder* getErrorRecorder() const noexcept
2252 {
2253     return mImpl->getErrorRecorder();
2254 }
2255
2270 bool hasImplicitBatchDimension() const noexcept
2271 {
2272     return mImpl->hasImplicitBatchDimension();
2273 }
2274
2285 TacticSources getTacticSources() const noexcept
2286 {
2287     return mImpl->getTacticSources();
2288 }
2289
2296 ProfilingVerbosity getProfilingVerbosity() const noexcept
2297 {
2298     return mImpl->getProfilingVerbosity();
2299 }
2300
2306 IEngineInspector* createEngineInspector() const noexcept
2307 {
2308     return mImpl->createEngineInspector();
2309 }
2310
2319 int32_t getNbIOTensors() const noexcept
2320 {
2321     return mImpl->getNbIOTensors();
2322 }
2323
2331 char const* getIOTensorName(int32_t index) const noexcept
2332 {
2333     return mImpl->getIOTensorName(index);
2334 }
2335
2343 HardwareCompatibilityLevel getHardwareCompatibilityLevel() const noexcept
2344 {
2345     return mImpl->getHardwareCompatibilityLevel();
2346 }
2347
2348 protected:
2349     apiv::VCudaEngine* mImpl;
2350 };
2351
2361 class IOutputAllocator
2362 {
2363 public:
2371     virtual int32_t getInterfaceVersion() const noexcept
2372     {
2373         return 1;
2374     }
2375
2392     virtual void* reallocateOutput(char const* tensorName, void* currentMemory, uint64_t size, uint64_t
alignment) noexcept = 0;
2393
2402     virtual void notifyShape(char const* tensorName, Dims const& dims) noexcept = 0;
2403
2404     virtual ~IOutputAllocator() = default;
2405 };
2406
2417 class IExecutionContext : public INoCopy
2418 {
2419 public:
2420     virtual ~IExecutionContext() noexcept = default;
2421
2444     TRT_DEPRECATED bool execute(int32_t batchSize, void* const* bindings) noexcept
2445     {
2446         return mImpl->execute(batchSize, bindings);
2447     }

```

```

2448
2478 TRT_DEPRECATED bool enqueue(
2479     int32_t batchSize, void* const* bindings, cudaStream_t stream, cudaEvent_t* inputConsumed) noexcept
2480 {
2481     return mImpl->enqueue(batchSize, bindings, stream, inputConsumed);
2482 }
2483
2492 void setDebugSync(bool sync) noexcept
2493 {
2494     mImpl->setDebugSync(sync);
2495 }
2496
2502 bool getDebugSync() const noexcept
2503 {
2504     return mImpl->getDebugSync();
2505 }
2506
2512 void setProfiler(IProfiler* profiler) noexcept
2513 {
2514     mImpl->setProfiler(profiler);
2515 }
2516
2522 IProfiler* getProfiler() const noexcept
2523 {
2524     return mImpl->getProfiler();
2525 }
2526
2532 ICudaEngine const& getEngine() const noexcept
2533 {
2534     return mImpl->getEngine();
2535 }
2536
2544 TRT_DEPRECATED void destroy() noexcept
2545 {
2546     delete this;
2547 }
2548
2558 void setName(char const* name) noexcept
2559 {
2560     mImpl->setName(name);
2561 }
2562
2568 char const* getName() const noexcept
2569 {
2570     return mImpl->getName();
2571 }
2572
2585 void setDeviceMemory(void* memory) noexcept
2586 {
2587     mImpl->setDeviceMemory(memory);
2588 }
2589
2608 TRT_DEPRECATED Dims getStrides(int32_t bindingIndex) const noexcept
2609 {
2610     return mImpl->getStrides(bindingIndex);
2611 }
2612
2629 Dims getTensorStrides(char const* tensorName) const noexcept
2630 {
2631     return mImpl->getTensorStrides(tensorName);
2632 }
2633
2634 public:
2666 TRT_DEPRECATED
2667 bool setOptimizationProfile(int32_t profileIndex) noexcept
2668 {
2669     return mImpl->setOptimizationProfile(profileIndex);
2670 }
2671
2681 int32_t getOptimizationProfile() const noexcept
2682 {
2683     return mImpl->getOptimizationProfile();
2684 }
2685
2720 TRT_DEPRECATED bool setBindingDimensions(int32_t bindingIndex, Dims dimensions) noexcept
2721 {
2722     return mImpl->setBindingDimensions(bindingIndex, dimensions);
2723 }
2724
2738 bool setInputShape(char const* tensorName, Dims const& dims) noexcept
2739 {

```

```

2740         return mImpl->setInputShape(tensorName, dims);
2741     }
2742
2771     TRT_DEPRECATED Dims getBindingDimensions(int32_t bindingIndex) const noexcept
2772     {
2773         return mImpl->getBindingDimensions(bindingIndex);
2774     }
2775
2808     Dims getTensorShape(char const* tensorName) const noexcept
2809     {
2810         return mImpl->getTensorShape(tensorName);
2811     }
2812
2843     TRT_DEPRECATED bool setInputShapeBinding(int32_t bindingIndex, int32_t const* data) noexcept
2844     {
2845         return mImpl->setInputShapeBinding(bindingIndex, data);
2846     }
2847
2867     TRT_DEPRECATED bool getShapeBinding(int32_t bindingIndex, int32_t* data) const noexcept
2868     {
2869         return mImpl->getShapeBinding(bindingIndex, data);
2870     }
2871
2885     bool allInputDimensionsSpecified() const noexcept
2886     {
2887         return mImpl->allInputDimensionsSpecified();
2888     }
2889
2902     bool allInputShapesSpecified() const noexcept
2903     {
2904         return mImpl->allInputShapesSpecified();
2905     }
2906
2918     //
2921     void setErrorRecorder(IErrorRecorder* recorder) noexcept
2922     {
2923         mImpl->setErrorRecorder(recorder);
2924     }
2925
2936     IErrorRecorder* getErrorRecorder() const noexcept
2937     {
2938         return mImpl->getErrorRecorder();
2939     }
2940
2953     bool executeV2(void* const* bindings) noexcept
2954     {
2955         return mImpl->executeV2(bindings);
2956     }
2957
2983     TRT_DEPRECATED bool enqueueV2(void* const* bindings, cudaStream_t stream, cudaEvent_t* inputConsumed)
2984     noexcept
2985     {
2986         return mImpl->enqueueV2(bindings, stream, inputConsumed);
2987     }
2988
3030     bool setOptimizationProfileAsync(int32_t profileIndex, cudaStream_t stream) noexcept
3031     {
3032         return mImpl->setOptimizationProfileAsync(profileIndex, stream);
3033     }
3034
3045     void setEnqueueEmitsProfile(bool enqueueEmitsProfile) noexcept
3046     {
3047         mImpl->setEnqueueEmitsProfile(enqueueEmitsProfile);
3048     }
3049
3056     bool getEnqueueEmitsProfile() const noexcept
3057     {
3058         return mImpl->getEnqueueEmitsProfile();
3059     }
3060
3085     bool reportToProfiler() const noexcept
3086     {
3087         return mImpl->reportToProfiler();
3088     }
3089
3127     bool setTensorAddress(char const* tensorName, void* data) noexcept
3128     {
3129         return mImpl->setTensorAddress(tensorName, data);
3130     }
3131
3144     void const* getTensorAddress(char const* tensorName) const noexcept

```



```

3145     {
3146         return mImpl->getTensorAddress(tensorName);
3147     }
3148
3166     bool setInputTensorAddress(char const* tensorName, void const* data) noexcept
3167     {
3168         return mImpl->setInputTensorAddress(tensorName, data);
3169     }
3170
3185     void* getOutputTensorAddress(char const* tensorName) const noexcept
3186     {
3187         return mImpl->getOutputTensorAddress(tensorName);
3188     }
3189
3218     int32_t inferShapes(int32_t nbMaxNames, char const** tensorNames) noexcept
3219     {
3220         return mImpl->inferShapes(nbMaxNames, tensorNames);
3221     }
3222
3234     bool setInputConsumedEvent(cudaEvent_t event) noexcept
3235     {
3236         return mImpl->setInputConsumedEvent(event);
3237     }
3238
3244     cudaEvent_t getInputConsumedEvent() const noexcept
3245     {
3246         return mImpl->getInputConsumedEvent();
3247     }
3248
3263     bool setOutputAllocator(char const* tensorName, IOutputAllocator* outputAllocator) noexcept
3264     {
3265         return mImpl->setOutputAllocator(tensorName, outputAllocator);
3266     }
3267
3276     IOutputAllocator* getOutputAllocator(char const* tensorName) const noexcept
3277     {
3278         return mImpl->getOutputAllocator(tensorName);
3279     }
3280
3294     int64_t getMaxOutputSize(char const* tensorName) const noexcept
3295     {
3296         return mImpl->getMaxOutputSize(tensorName);
3297     }
3298
3314     bool setTemporaryStorageAllocator(IGpuAllocator* allocator) noexcept
3315     {
3316         return mImpl->setTemporaryStorageAllocator(allocator);
3317     }
3318
3324     IGPUAllocator* getTemporaryStorageAllocator() const noexcept
3325     {
3326         return mImpl->getTemporaryStorageAllocator();
3327     }
3328
3342     bool enqueueV3(cudaStream_t stream) noexcept
3343     {
3344         return mImpl->enqueueV3(stream);
3345     }
3346
3357     void setPersistentCacheLimit(size_t size) noexcept
3358     {
3359         mImpl->setPersistentCacheLimit(size);
3360     }
3361
3368     size_t getPersistentCacheLimit() const noexcept
3369     {
3370         return mImpl->getPersistentCacheLimit();
3371     }
3372
3392     bool setNvtxVerbosity(ProfilingVerbosity verbosity) noexcept
3393     {
3394         return mImpl->setNvtxVerbosity(verbosity);
3395     }
3396
3404     ProfilingVerbosity getNvtxVerbosity() const noexcept
3405     {
3406         return mImpl->getNvtxVerbosity();
3407     }
3408
3409 protected:
3410     apiv::VExecutionContext* mImpl;

```

```

3411 }; // class IExecutionContext
3412
3420 enum class LayerInformationFormat : int32_t
3421 {
3422     kONELINE = 0,
3423     kJSON = 1,
3424 };
3425
3428 template <>
3429 constexpr inline int32_t EnumMax<LayerInformationFormat>() noexcept
3430 {
3431     return 2;
3432 }
3433
3449 class IEngineInspector : public INoCopy
3450 {
3451 public:
3452     virtual ~IEngineInspector() noexcept = default;
3453
3454     bool setExecutionContext(IExecutionContext const* context) noexcept
3455     {
3456         return mImpl->setExecutionContext(context);
3457     }
3458
3459     IExecutionContext const* getExecutionContext() const noexcept
3460     {
3461         return mImpl->getExecutionContext();
3462     }
3463
3464     char const* getLayerInformation(int32_t layerIndex, LayerInformationFormat format) const noexcept
3465     {
3466         return mImpl->getLayerInformation(layerIndex, format);
3467     }
3468
3469     char const* getEngineInformation(LayerInformationFormat format) const noexcept
3470     {
3471         return mImpl->getEngineInformation(format);
3472     }
3473
3474     //
3475     void setErrorRecorder(IErrorRecorder* recorder) noexcept
3476     {
3477         mImpl->setErrorRecorder(recorder);
3478     }
3479
3480     IErrorRecorder* getErrorRecorder() const noexcept
3481     {
3482         return mImpl->getErrorRecorder();
3483     }
3484
3485 protected:
3486     apiv::VEngineInspector* mImpl;
3487 }; // class IEngineInspector
3488
3491 } // namespace nvinfer1
3492
3493 extern "C" TENSORRTAPI void* createInferRuntime.INTERNAL(void* logger, int32_t version) noexcept;
3494
3495 extern "C" TENSORRTAPI void* createInferRefitter.INTERNAL(void* engine, void* logger, int32_t version)
3496     noexcept;
3497
3498 extern "C" TENSORRTAPI nvinfer1::IPluginRegistry* getPluginRegistry() noexcept;
3499
3500 extern "C" TENSORRTAPI nvinfer1::ILogger* getLogger() noexcept;
3501
3502 namespace nvinfer1
3503 {
3504     namespace // unnamed namespace avoids linkage surprises when linking objects built with different versions
3505         of this
3506         // header.
3507     {
3508         inline IRuntime* createInferRuntime(ILogger& logger) noexcept
3509         {
3510             return static_cast<IRuntime*>(createInferRuntime.INTERNAL(&logger, NV_TENSORRT_VERSION));
3511         }
3512
3513         inline IRefitter* createInferRefitter(ICudaEngine& engine, ILogger& logger) noexcept
3514         {
3515             return static_cast<IRefitter*>(createInferRefitter.INTERNAL(&engine, &logger, NV_TENSORRT_VERSION));
3516         }
3517     }
3518 }
3519

```

```

3622 } // namespace
3623
3635 template <typename T>
3636 class PluginRegistrar
3637 {
3638 public:
3639     PluginRegistrar()
3640     {
3641         getPluginRegistry()->registerCreator(instance, "");
3642     }
3643
3644 private:
3645     T instance{};
3646 };
3647 };
3648
3649 } // namespace nvinfer1
3650
3651 #define REGISTER_TENSORRT_PLUGIN(name)
3652     \
3653     static nvinfer1::PluginRegistrar<name> pluginRegistrar##name {}
3654 #endif // NV_INFER_RUNTIME_H

```

10.15 NvInferRuntimeCommon.h File Reference

```

#include "NvInferVersion.h"
#include <cstddef>
#include <stdint>
#include <cuda_runtime_api.h>

```

Classes

- struct [nvinfer1::impl::EnumMaxImpl< DataType >](#)
Maximum number of elements in DataType enum.
- class [nvinfer1::Dims32](#)
- struct [nvinfer1::impl::EnumMaxImpl< TensorFormat >](#)
Maximum number of elements in TensorFormat enum.
- 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::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::IPluginCreator](#)
Plugin creator class for user implemented layers.
- 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.

- struct `nvinfer1::impl::EnumMaxImpl< AllocatorFlag >`
Maximum number of elements in `AllocatorFlag` enum.
- class `nvinfer1::IGpuAllocator`
Application-implemented class for controlling allocation on the GPU.
- class `nvinfer1::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.
- struct `nvinfer1::impl::EnumMaxImpl< ErrorCode >`
Maximum number of elements in `ErrorCode` enum.
- class `nvinfer1::IErrorRecorder`
Reference counted application-implemented error reporting interface for TensorRT objects.
- struct `nvinfer1::impl::EnumMaxImpl< TensorIOMode >`
Maximum number of elements in `TensorIOMode` enum.

Namespaces

- namespace `nvinfer1`
The TensorRT API version 1 namespace.
- namespace `nvinfer1::impl`

Macros

- `#define TRT_DEPRECATED __attribute__((deprecated))`
- `#define TRT_DEPRECATED_ENUM`
- `#define TRT_DEPRECATED_API __attribute__((deprecated, visibility("default")))`
- `#define TENSORRTAPI`
- `#define TRTNOEXCEPT`
- `#define NV_TENSORRT_VERSION nvinfer1::kNV_TENSORRT_VERSION_IMPL`

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::Dims` = `Dims32`
- using `nvinfer1::PluginFormat` = `TensorFormat`
PluginFormat is reserved for backward compatibility.
- using `nvinfer1::AllocatorFlags` = `uint32_t`

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 }

The type of weights and tensors.

- 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::PluginVersion` : `uint8_t` { `nvinfer1::kV2` = 0 , `nvinfer1::kV2_EXT` = 1 , `nvinfer1::kV2_IOEXT` = 2 , `nvinfer1::kV2_DYNAMICEXT` = 3 }
- 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 }
- enum class `nvinfer1::AllocatorFlag` : `int32_t` { `nvinfer1::kRESIZABLE` = 0 }
- 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
Maximum number of elements in an enumeration type.
- `int32_t` `getInferLibVersion` () noexcept
Return the library version number.

10.15.1 Detailed Description

This is the top-level API file for TensorRT core runtime library.

10.15.2 Macro Definition Documentation

10.15.2.1 NV_TENSORRT_VERSION

```
#define NV_TENSORRT_VERSION nvinfer1::kNV_TENSORRT_VERSION_IMPL
```

10.15.2.2 TENSORRTAPI

```
#define TENSORRTAPI
```

10.15.2.3 TRT_DEPRECATED

```
#define TRT_DEPRECATED __attribute__((deprecated))
```

10.15.2.4 TRT_DEPRECATED_API

```
#define TRT_DEPRECATED_API __attribute__((deprecated, visibility("default")))
```

10.15.2.5 TRT_DEPRECATED_ENUM

```
#define TRT_DEPRECATED_ENUM
```

10.15.2.6 TRTNOEXCEPT

```
#define TRTNOEXCEPT
```

10.15.3 Function Documentation

10.15.3.1 `getInferLibVersion()`

```
int32_t getInferLibVersion ( ) [noexcept]
```

Return the library version number.

The format is as for TENSORRT_VERSION: (TENSORRT_MAJOR * 1000) + (TENSORRT_MINOR * 100) + TENSOR_PATCH.

10.16 `NvInferRuntimeCommon.h`

[Go to the documentation of this file.](#)

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_INFER_RUNTIME_COMMON_H
14 #define NV_INFER_RUNTIME_COMMON_H
15
16 #include "NvInferVersion.h"
17 #include <stddef>
18 #include <stdint>
19 #include <cuda.runtime.api.h>
20
21 // Items that are marked as deprecated will be removed in a future release.
22 #if __cplusplus >= 201402L
23 #define TRT_DEPRECATED [[deprecated]]
24 #if __GNUC__ < 6
25 #define TRT_DEPRECATED_ENUM
26 #else
27 #define TRT_DEPRECATED_ENUM TRT_DEPRECATED
28 #endif
29 #ifdef _MSC_VER
30 #define TRT_DEPRECATED_API __declspec(dllexport)
31 #else
32 #define TRT_DEPRECATED_API [[deprecated]] __attribute__((visibility("default")))
33 #endif
34 #else
35 #ifdef _MSC_VER
36 #define TRT_DEPRECATED
37 #define TRT_DEPRECATED_ENUM
38 #define TRT_DEPRECATED_API __declspec(dllexport)
39 #else
40 #define TRT_DEPRECATED __attribute__((deprecated))
41 #define TRT_DEPRECATED_ENUM
42 #define TRT_DEPRECATED_API __attribute__((deprecated, visibility("default")))
43 #endif
44 #endif
45
46 // Defines which symbols are exported
47 #ifdef TENSORRT_BUILD_LIB
48 #ifdef _MSC_VER
49 #define TENSORRTAPI __declspec(dllexport)
50 #else
51 #define TENSORRTAPI __attribute__((visibility("default")))
52 #endif
53 #else
54 #define TENSORRTAPI
55 #endif
56 #define TRTNOEXCEPT
57
58 // forward declare some CUDA types to avoid an include dependency
59
```

```

65 extern "C"
66 {
67     struct cublasContext;
68     struct cudnnContext;
69 }
70
71 #define NV_TENSORRT_VERSION nvinfer1::kNV_TENSORRT_VERSION_IMPL
72 namespace nvinfer1
73 {
74     static constexpr int32_t kNV_TENSORRT_VERSION_IMPL
75     = (NV_TENSORRT_MAJOR * 1000) + (NV_TENSORRT_MINOR * 100) + NV_TENSORRT_PATCH; // major, minor, patch
76 using char_t = char;
77
78 using AsciiChar = char_t;
79
80 class IErrorRecorder;
81 class IGpuAllocator;
82
83 namespace impl
84 {
85     template <typename T>
86     struct EnumMaxImpl;
87 } // namespace impl
88
89 template <typename T>
90 constexpr int32_t EnumMax() noexcept
91 {
92     return impl::EnumMaxImpl<T>::kVALUE;
93 }
94
95 enum class DataType : int32_t
96 {
97     kFLOAT = 0,
98     kHALF = 1,
99     kINT8 = 2,
100     kINT32 = 3,
101     kBOOL = 4,
102     kUINT8 = 5,
103     kFP8 = 6
104 };
105
106 namespace impl
107 {
108     template <>
109     struct EnumMaxImpl<DataType>
110     {
111         // Declaration of kVALUE that represents maximum number of elements in DataType enum
112         static constexpr int32_t kVALUE = 7;
113     };
114 } // namespace impl
115
116 class Dims32
117 {
118 public:
119     static constexpr int32_t MAX_DIMS{8};
120     int32_t nbDims;
121     int32_t d[MAX_DIMS];
122 };
123
124 using Dims = Dims32;
125
126 enum class TensorFormat : int32_t
127 {
128     kLINEAR = 0,
129     kCHW2 = 1,
130     kHWC8 = 2,
131     kCHW4 = 3,
132     kCHW16 = 4,

```



```

259
269     kCHW32 = 5,
270
277     kDHW8 = 6,
278
285     kCDHW32 = 7,
286
289     kHWC = 8,
290
299     kDLA_LINEAR = 9,
300
314     kDLA_HWC4 = 10,
315
322     kHWC16 = 11,
323
326     kDHW8 = 12
327 };
328
334 using PluginFormat = TensorFormat;
335
336 namespace impl
337 {
338     template <>
339     struct EnumMaxImpl<TensorFormat>
340     {
341     {
342         static constexpr int32_t kVALUE = 13;
343     };
344 } // namespace impl
345
346 struct PluginTensorDesc
347 {
348     Dims dims;
349     DataType type;
350     TensorFormat format;
351     float scale;
352 };
353
354 enum class PluginVersion : uint8_t
355 {
356     kV2 = 0,
357     kV2_EXT = 1,
358     kV2_IOEXT = 2,
359     kV2_DYNAMICEXT = 3,
360 };
361
362 class TRT_DEPRECATED IPluginV2
363 {
364 public:
365     virtual int32_t getTensorRTVersion() const noexcept
366     {
367         return NV_TENSORRT_VERSION;
368     }
369
370     virtual AsciiChar const* getPluginType() const noexcept = 0;
371
372     virtual AsciiChar const* getPluginVersion() const noexcept = 0;
373
374     virtual int32_t getNbOutputs() const noexcept = 0;
375
376     virtual Dims getOutputDimensions(int32_t index, Dims const* inputs, int32_t nbInputDims) noexcept = 0;
377
378     virtual bool supportsFormat(DataType type, PluginFormat format) const noexcept = 0;
379
380     virtual void configureWithFormat(Dims const* inputDims, int32_t nbInputs, Dims const* outputDims, int32_t
381 nbOutputs,
382     DataType type, PluginFormat format, int32_t maxBatchSize) noexcept
383     = 0;
384
385     virtual int32_t initialize() noexcept = 0;
386
387     virtual void terminate() noexcept = 0;
388
389     virtual size_t getWorkspaceSize(int32_t maxBatchSize) const noexcept = 0;
390
391     virtual int32_t enqueue(int32_t batchSize, void const* const* inputs, void* const* outputs, void*
392 workspace,
393     cudaStream_t stream) noexcept
394     = 0;
395
396     virtual size_t getSerializationSize() const noexcept = 0;
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616

```

```

630     virtual void serialize(void* buffer) const noexcept = 0;
631
640     virtual void destroy() noexcept = 0;
641
656     virtual IPluginV2* clone() const noexcept = 0;
657
672     virtual void setPluginNamespace(AsciiChar const* pluginNamespace) noexcept = 0;
673
682     virtual AsciiChar const* getPluginNamespace() const noexcept = 0;
683
684     // @cond SuppressDoxyWarnings
685     IPluginV2() = default;
686     virtual ~IPluginV2() noexcept = default;
687 // @endcond
688
689 protected:
690 // @cond SuppressDoxyWarnings
691     IPluginV2(IPluginV2 const&) = default;
692     IPluginV2(IPluginV2&&) = default;
693     IPluginV2& operator=(IPluginV2 const&) &= default;
694     IPluginV2& operator=(IPluginV2&&) &= default;
695 // @endcond
696 };
697
711 class TRT_DEPRECATED IPluginV2Ext : public IPluginV2
712 {
713 public:
729     virtual nvinfer1::DataType getOutputDataType(
730         int32_t index, nvinfer1::DataType const* inputTypes, int32_t nbInputs) const noexcept
731         = 0;
732
748     virtual bool isOutputBroadcastAcrossBatch(
749         int32_t outputIndex, bool const* inputIsBroadcasted, int32_t nbInputs) const noexcept
750         = 0;
751
770     virtual bool canBroadcastInputAcrossBatch(int32_t inputIndex) const noexcept = 0;
771
806     virtual void configurePlugin(Dims const* inputDims, int32_t nbInputs, Dims const* outputDims, int32_t
nbOutputs,
807         DataType const* inputTypes, DataType const* outputTypes, bool const* inputIsBroadcast,
808         bool const* outputIsBroadcast, PluginFormat floatFormat, int32_t maxBatchSize) noexcept
809         = 0;
810
811     IPluginV2Ext() = default;
812     ~IPluginV2Ext() override = default;
813
837     virtual void attachToContext(
838         cudnnContext* /*cudnn*/, cublasContext* /*cublas*/, IGpuAllocator* /*allocator*/) noexcept
839     {
840     }
841
855     virtual void detachFromContext() noexcept {}
856
869     IPluginV2Ext* clone() const noexcept override = 0;
870
871 protected:
872     // @cond SuppressDoxyWarnings
873     IPluginV2Ext(IPluginV2Ext const&) = default;
874     IPluginV2Ext(IPluginV2Ext&&) = default;
875     IPluginV2Ext& operator=(IPluginV2Ext const&) &= default;
876     IPluginV2Ext& operator=(IPluginV2Ext&&) &= default;
877 // @endcond
878
890     int32_t getTensorRTVersion() const noexcept override
891     {
892         return static_cast<int32_t>((static_cast<uint32_t>(PluginVersion::kV2_EXT) << 24U)
893             | (static_cast<uint32_t>(NV_TENSORRT_VERSION) & 0xFFFFFU));
894     }
895
899     void configureWithFormat(Dims const* /*inputDims*/, int32_t /*nbInputs*/, Dims const* /*outputDims*/,
900         int32_t /*nbOutputs*/, DataType /*type*/, PluginFormat /*format*/, int32_t /*maxBatchSize*/) noexcept
override
901     {
902     }
903 };
904
914 class IPluginV2IOExt : public IPluginV2Ext
915 {
916 public:
934     virtual void configurePlugin(
935         PluginTensorDesc const* in, int32_t nbInput, PluginTensorDesc const* out, int32_t nbOutput) noexcept

```

```

936         = 0;
937
975     virtual bool supportsFormatCombination(
976         int32_t pos, PluginTensorDesc const* inOut, int32_t nbInputs, int32_t nbOutputs) const noexcept
977         = 0;
978
979     // @cond SuppressDoxyWarnings
980     IPluginV2IOExt() = default;
981     ~IPluginV2IOExt() override = default;
982 // @endcond
983
984 protected:
985 // @cond SuppressDoxyWarnings
986     IPluginV2IOExt(IPluginV2IOExt const&) = default;
987     IPluginV2IOExt(IPluginV2IOExt&&) = default;
988     IPluginV2IOExt& operator=(IPluginV2IOExt const&) &= default;
989     IPluginV2IOExt& operator=(IPluginV2IOExt&&) &= default;
990 // @endcond
991
1003     int32_t getTensorRTVersion() const noexcept override
1004     {
1005         return static_cast<int32_t>((static_cast<uint32_t>(PluginVersion::kV2_IOEXT) << 24U)
1006             | (static_cast<uint32_t>(NV_TENSORRT_VERSION) & 0xFFFFFU));
1007     }
1008
1009 private:
1010     // Following are obsolete base class methods, and must not be implemented or used.
1011
1012     void configurePlugin(Dims const*, int32_t, Dims const*, int32_t, DataType const*, DataType const*, bool
1013         const*,
1014         bool const*, PluginFormat, int32_t) noexcept final
1015     {
1016     }
1017
1018     bool supportsFormat(DataType, PluginFormat) const noexcept final
1019     {
1020         return false;
1021     };
1022
1023
1027
1028     enum class PluginFieldType : int32_t
1029     {
1030     {
1031         kFLOAT16 = 0,
1032         kFLOAT32 = 1,
1033         kFLOAT64 = 2,
1034         kINT8 = 3,
1035         kINT16 = 4,
1036         kINT32 = 5,
1037         kCHAR = 6,
1038         kDIMS = 7,
1039         kUNKNOWN = 8
1040     };
1041
1042
1043
1044
1045
1046
1047
1048 };
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106

```

```

1107 public:
1115     virtual int32_t getTensorRTVersion() const noexcept
1116     {
1117         return NV_TENSORRT_VERSION;
1118     }
1119
1132     virtual AsciiChar const* getPluginName() const noexcept = 0;
1133
1146     virtual AsciiChar const* getPluginVersion() const noexcept = 0;
1147
1158     virtual PluginFieldCollection const* getFieldNames() noexcept = 0;
1159
1169     virtual IPluginV2* createPlugin(AsciiChar const* name, PluginFieldCollection const* fc) noexcept = 0;
1170
1180     virtual IPluginV2* deserializePlugin(AsciiChar const* name, void const* serialData, size_t
serialLength) noexcept
1181         = 0;
1182
1195     virtual void setPluginNamespace(AsciiChar const* pluginNamespace) noexcept = 0;
1196
1209     virtual AsciiChar const* getPluginNamespace() const noexcept = 0;
1210
1211     IPluginCreator() = default;
1212     virtual ~IPluginCreator() = default;
1213
1214 protected:
1215     // @cond SuppressDoxyWarnings
1216     IPluginCreator(IPluginCreator const&) = default;
1217     IPluginCreator(IPluginCreator&&) = default;
1218     IPluginCreator& operator=(IPluginCreator const&) &= default;
1219     IPluginCreator& operator=(IPluginCreator&&) &= default;
1220     // @endcond
1221 };
1222
1240
1241 class IPluginRegistry
1242 {
1243 public:
1244     using PluginLibraryHandle = void*;
1257     virtual bool registerCreator(IPluginCreator& creator, AsciiChar const* const pluginNamespace) noexcept
= 0;
1258
1267     virtual IPluginCreator* const* getPluginCreatorList(int32_t* const numCreators) const noexcept = 0;
1268
1280     virtual IPluginCreator* getPluginCreator(AsciiChar const* const pluginName, AsciiChar const* const
pluginVersion,
1281         AsciiChar const* const pluginNamespace = "") noexcept
1282         = 0;
1283
1292     virtual bool isParentSearchEnabled() const = 0;
1293
1301     virtual void setParentSearchEnabled(bool const enabled) = 0;
1302
1311     virtual PluginLibraryHandle loadLibrary(AsciiChar const* pluginPath) noexcept = 0;
1312
1322     virtual PluginLibraryHandle deserializeLibrary(void const* pluginBuffer, size_t size) noexcept = 0;
1323
1330     virtual void deregisterLibrary(PluginLibraryHandle handle) noexcept = 0;
1331
1332     // @cond SuppressDoxyWarnings
1333     IPluginRegistry() = default;
1334     IPluginRegistry(IPluginRegistry const&) = delete;
1335     IPluginRegistry(IPluginRegistry&&) = delete;
1336     IPluginRegistry& operator=(IPluginRegistry const&) &= delete;
1337     IPluginRegistry& operator=(IPluginRegistry&&) &= delete;
1338     // @endcond
1339
1340 protected:
1341     virtual ~IPluginRegistry() noexcept = default;
1342
1343 public:
1353     //
1360     virtual void setErrorRecorder(IErrorRecorder* const recorder) noexcept = 0;
1361
1377     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
1378
1394     virtual bool deregisterCreator(IPluginCreator const& creator) noexcept = 0;
1395 };
1396
1397 enum class AllocatorFlag : int32_t
1398 {

```

```

1399     kRESIZABLE = 0,
1400 };
1401
1402 namespace impl
1403 {
1404     template <>
1405     struct EnumMaxImpl<AllocatorFlag>
1406     {
1407         static constexpr int32_t kVALUE = 1;
1408     };
1409 } // namespace impl
1410
1411 using AllocatorFlags = uint32_t;
1412
1413 class IGPUAllocator
1414 {
1415 public:
1416     virtual void* allocate(uint64_t const size, uint64_t const alignment, AllocatorFlags const flags)
1417     noexcept = 0;
1418
1419     TRT_DEPRECATED virtual void free(void* const memory) noexcept = 0;
1420
1421     virtual ~IGPUAllocator() = default;
1422     IGPUAllocator() = default;
1423
1424     virtual void* reallocate(void* /*baseAddr*/, uint64_t /*alignment*/, uint64_t /*newSize*/) noexcept
1425     {
1426         return nullptr;
1427     }
1428
1429     virtual bool deallocate(void* const memory) noexcept
1430     {
1431         this->free(memory);
1432         return true;
1433     }
1434
1435 protected:
1436     // @cond SuppressDoxyWarnings
1437     IGPUAllocator(IGPUAllocator const&) = default;
1438     IGPUAllocator(IGPUAllocator&&) = default;
1439     IGPUAllocator& operator=(IGPUAllocator const&) &= default;
1440     IGPUAllocator& operator=(IGPUAllocator&&) &= default;
1441     // @endcond
1442 };
1443
1444 class ILogger
1445 {
1446 public:
1447     enum class Severity : int32_t
1448     {
1449         kINTERNAL_ERROR = 0,
1450         kERROR = 1,
1451         kWARNING = 2,
1452         kINFO = 3,
1453         kVERBOSE = 4,
1454     };
1455
1456     virtual void log(Severity severity, AsciiChar const* msg) noexcept = 0;
1457
1458     ILogger() = default;
1459     virtual ~ILogger() = default;
1460
1461 protected:
1462     // @cond SuppressDoxyWarnings
1463     ILogger(ILogger const&) = default;
1464     ILogger(ILogger&&) = default;
1465     ILogger& operator=(ILogger const&) &= default;
1466     ILogger& operator=(ILogger&&) &= default;
1467     // @endcond
1468 };
1469
1470 namespace impl
1471 {
1472     template <>
1473     struct EnumMaxImpl<ILogger::Severity>
1474     {
1475         static constexpr int32_t kVALUE = 5;
1476     };
1477 } // namespace impl
1478
1479 enum class ErrorCode : int32_t

```

```

1622 {
1626     kSUCCESS = 0,
1627
1631     kUNSPECIFIED_ERROR = 1,
1632
1637     kINTERNAL_ERROR = 2,
1638
1643     kINVALID_ARGUMENT = 3,
1644
1652     kINVALID_CONFIG = 4,
1653
1659     kFAILED_ALLOCATION = 5,
1660
1665     kFAILED_INITIALIZATION = 6,
1666
1673     kFAILED_EXECUTION = 7,
1674
1682     kFAILED_COMPUTATION = 8,
1683
1696     kINVALID_STATE = 9,
1697
1708     kUNSUPPORTED_STATE = 10,
1709
1710 };
1711
1712 namespace impl
1713 {
1715     template <>
1716     struct EnumMaxImpl<ErrorCode>
1717     {
1719         static constexpr int32_t kVALUE = 11;
1720     };
1721 } // namespace impl
1722
1746 class IErrorRecorder
1747 {
1748 public:
1752     using ErrorDesc = char const*;
1753
1757     static constexpr size_t kMAX_DESC_LENGTH{127U};
1758
1762     using RefCount = int32_t;
1763
1764     IErrorRecorder() = default;
1765     virtual ~IErrorRecorder() noexcept = default;
1766
1767     // Public API used to retrieve information from the error recorder.
1768
1787     virtual int32_t getNbErrors() const noexcept = 0;
1788
1806     virtual ErrorCode getErrorCode(int32_t errorIdx) const noexcept = 0;
1807
1827     virtual ErrorDesc getErrorDesc(int32_t errorIdx) const noexcept = 0;
1828
1843     virtual bool hasOverflowed() const noexcept = 0;
1844
1859     virtual void clear() noexcept = 0;
1860
1861     // API used by TensorRT to report Error information to the application.
1862
1883     virtual bool reportError(ErrorCode val, ErrorDesc desc) noexcept = 0;
1884
1901     virtual RefCount incRefCount() noexcept = 0;
1902
1919     virtual RefCount decRefCount() noexcept = 0;
1920
1921 protected:
1922     // @cond SuppressDoxyWarnings
1923     IErrorRecorder(IErrorRecorder const&) = default;
1924     IErrorRecorder(IErrorRecorder&&) = default;
1925     IErrorRecorder& operator=(IErrorRecorder const&) &= default;
1926     IErrorRecorder& operator=(IErrorRecorder&&) &= default;
1927     // @endcond
1928 }; // class IErrorRecorder
1929
1935 enum class TensorIOMode : int32_t
1936 {
1938     kNONE = 0,
1939
1941     kINPUT = 1,
1942

```

```

1944     kOUTPUT = 2
1945 };
1946
1947 namespace impl
1948 {
1949     template <>
1950     struct EnumMaxImpl<TensorIOMode>
1951     {
1952         // Declaration of kVALUE that represents maximum number of elements in TensorIOMode enum
1953         static constexpr int32_t kVALUE = 3;
1954     };
1955 } // namespace impl
1956 } // namespace nvinfer1
1957 } // namespace nvinfer1
1958
1959 extern "C" TENSORRTAPI int32_t getInferLibVersion() noexcept;
1960
1961 #endif // NV_INFER_RUNTIME_COMMON_H

```

10.17 NvInferSafeRuntime.h File Reference

```

#include "NvInferRuntimeCommon.h"
#include <stddef>
#include <stdint>

```

Classes

- class [nvinfer1::safe::IRuntime](#)
Allows a serialized functionally safe engine to be deserialized.
- class [nvinfer1::safe::ICudaEngine](#)
A functionally safe engine for executing inference on a built network.
- struct [nvinfer1::safe::FloatingPointErrorInformation](#)
Space to record information about floating point runtime errors.
- class [nvinfer1::safe::IExecutionContext](#)
Functionally safe context for executing inference using an engine.
- class [nvinfer1::safe::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.

Namespaces

- namespace [nvinfer1](#)
The TensorRT API version 1 namespace.
- namespace [nvinfer1::safe](#)
The safety subset of TensorRT's API version 1 namespace.

Macros

- #define [REGISTER_SAFE_TENSORRT_PLUGIN](#)(name) static [nvinfer1::safe::PluginRegistrar<name>](#)
 pluginRegistrar##name {}

Functions

- `IRuntime * nvinfer1::safe::createInferRuntime (ILogger &logger) noexcept`
Create an instance of an `safe::IRuntime` class.
- `nvinfer1::IPluginRegistry * nvinfer1::safe::getSafePluginRegistry () noexcept`
Return the safe plugin registry.

10.17.1 Detailed Description

The functionality in this file is only supported in NVIDIA Drive(R) products.

10.17.2 Macro Definition Documentation

10.17.2.1 REGISTER_SAFE_TENSORRT_PLUGIN

```
#define REGISTER_SAFE_TENSORRT_PLUGIN(  
    name )    static nvinfer1::safe::PluginRegistrar<name> pluginRegistrar##name {}
```

10.18 NvInferSafeRuntime.h

[Go to the documentation of this file.](#)

```
1 /*  
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.  
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary  
4  *  
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual  
6  * property and proprietary rights in and to this material, related  
7  * documentation and any modifications thereto. Any use, reproduction,  
8  * disclosure or distribution of this material and related documentation  
9  * without an express license agreement from NVIDIA CORPORATION or  
10 * its affiliates is strictly prohibited.  
11 */  
12  
13 #ifndef NV_INFER_SAFE_RUNTIME_H  
14 #define NV_INFER_SAFE_RUNTIME_H  
15  
16 #include "NvInferRuntimeCommon.h"  
17 #include <stddef>  
18 #include <stdint>  
19  
20  
21  
22  
23  
24 namespace nvinfer1  
25 {  
26     namespace safe  
27     {  
28  
29         class ICudaEngine;  
30         class IExecutionContext;  
31  
32         class IRuntime  
33         {  
34         public:  
35             virtual ICudaEngine* deserializeCudaEngine(void const* const blob, std::size_t const size) noexcept = 0;  
36  
37             virtual void setGpuAllocator(IGpuAllocator* const allocator) noexcept = 0;  
38  
39         };  
40  
41     };  
42  
43 }  
44  
45
```



```

102 //
109 virtual void setErrorRecorder(IErrorRecorder* const recorder) noexcept = 0;
110
125 virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
126
127 IRuntime() = default;
128 virtual ~IRuntime() noexcept = default;
129 IRuntime(IRuntime const&) = delete;
130 IRuntime(IRuntime&&) = delete;
131 IRuntime& operator=(IRuntime const&) &= delete;
132 IRuntime& operator=(IRuntime&&) &= delete;
133 };
134
142 class ICudaEngine
143 {
144 public:
158 TRT_DEPRECATED virtual std::int32_t getNbBindings() const noexcept = 0;
159
181 TRT_DEPRECATED virtual std::int32_t getBindingIndex(AsciiChar const* const name) const noexcept = 0;
182
200 TRT_DEPRECATED virtual AsciiChar const* getBindingName(std::int32_t const bindingIndex) const noexcept =
0;
201
216 TRT_DEPRECATED virtual bool bindingIsInput(std::int32_t const bindingIndex) const noexcept = 0;
217
232 TRT_DEPRECATED virtual Dims getBindingDimensions(std::int32_t const bindingIndex) const noexcept = 0;
233
248 TRT_DEPRECATED virtual DataType getBindingDataType(std::int32_t const bindingIndex) const noexcept = 0;
249
261 virtual IExecutionContext* createExecutionContext() noexcept = 0;
262
276 virtual IExecutionContext* createExecutionContextWithoutDeviceMemory() noexcept = 0;
277
287 virtual size_t getDeviceMemorySize() const noexcept = 0;
288
304 TRT_DEPRECATED virtual std::int32_t getBindingBytesPerComponent(std::int32_t const bindingIndex) const
noexcept = 0;
305
321 TRT_DEPRECATED virtual std::int32_t getBindingComponentsPerElement(std::int32_t const bindingIndex) const
noexcept = 0;
322
336 TRT_DEPRECATED virtual TensorFormat getBindingFormat(std::int32_t const bindingIndex) const noexcept = 0;
337
353 TRT_DEPRECATED virtual std::int32_t getBindingVectorizedDim(std::int32_t const bindingIndex) const
noexcept = 0;
354
369 virtual AsciiChar const* getName() const noexcept = 0;
370
380 //
387 virtual void setErrorRecorder(IErrorRecorder* const recorder) noexcept = 0;
388
404 virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
405
406 ICudaEngine() = default;
407 virtual ~ICudaEngine() noexcept = default;
408 ICudaEngine(ICudaEngine const&) = delete;
409 ICudaEngine(ICudaEngine&&) = delete;
410 ICudaEngine& operator=(ICudaEngine const&) &= delete;
411 ICudaEngine& operator=(ICudaEngine&&) &= delete;
412
429 virtual Dims getTensorShape(AsciiChar const* tensorName) const noexcept = 0;
430
448 virtual DataType getTensorDataType(AsciiChar const* tensorName) const noexcept = 0;
449
464 virtual TensorIOMode getTensorIOMode(AsciiChar const* tensorName) const noexcept = 0;
465
486 virtual std::int32_t getTensorBytesPerComponent(AsciiChar const* tensorName) const noexcept = 0;
487
508 virtual std::int32_t getTensorComponentsPerElement(AsciiChar const* tensorName) const noexcept = 0;
509
527 virtual TensorFormat getTensorFormat(AsciiChar const* tensorName) const noexcept = 0;
528
549 virtual std::int32_t getTensorVectorizedDim(AsciiChar const* tensorName) const noexcept = 0;
550
562 virtual std::int32_t getNbIOTensors() const noexcept = 0;
563
581 virtual AsciiChar const* getIOTensorName(std::int32_t const index) const noexcept = 0;
582 };
583
590 struct FloatingPointErrorInformation
591 {

```

```

593     int32_t nbNanErrors;
594     int32_t nbInfErrors;
595 };
596
597 class IExecutionContext
598 {
599 public:
600     virtual ICudaEngine const& getEngine() const noexcept = 0;
601
602     virtual void setName(AsciiChar const* const name) noexcept = 0;
603
604     virtual AsciiChar const* getName() const noexcept = 0;
605
606     virtual void setDeviceMemory(void* const memory) noexcept = 0;
607
608     TRT_DEPRECATED virtual Dims getStrides(std::int32_t const bindingIndex) const noexcept = 0;
609
610     //
611     virtual void setErrorRecorder(IErrorRecorder* const recorder) noexcept = 0;
612
613     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
614
615     TRT_DEPRECATED virtual bool enqueueV2(
616         void* const* bindings, cudaStream_t const stream, cudaEvent_t const* const inputConsumed)
617         noexcept = 0;
618
619     IExecutionContext() = default;
620     virtual ~IExecutionContext() noexcept = default;
621     IExecutionContext(IExecutionContext const&) = delete;
622     IExecutionContext(IExecutionContext&&) = delete;
623     IExecutionContext& operator=(IExecutionContext const&) &= delete;
624     IExecutionContext& operator=(IExecutionContext&&) &= delete;
625
626     virtual void setErrorBuffer(FloatingPointErrorInformation* const buffer) noexcept = 0;
627
628     virtual FloatingPointErrorInformation* getErrorBuffer() const noexcept = 0;
629
630     virtual Dims getTensorStrides(AsciiChar const* tensorName) const noexcept = 0;
631
632     virtual bool setInputTensorAddress(AsciiChar const* tensorName, void const* data) noexcept = 0;
633
634     virtual bool setOutputTensorAddress(AsciiChar const* tensorName, void* data) noexcept = 0;
635
636     virtual bool setInputConsumedEvent(cudaEvent_t event) noexcept = 0;
637
638     virtual cudaEvent_t getInputConsumedEvent() const noexcept = 0;
639
640     virtual void const* getInputTensorAddress(AsciiChar const* tensorName) const noexcept = 0;
641
642     virtual void* getOutputTensorAddress(AsciiChar const* tensorName) const noexcept = 0;
643
644     virtual bool enqueueV3(cudaStream_t stream) noexcept = 0;
645 };
646
647 IRuntime* createInferRuntime(ILogger& logger) noexcept;
648
649 extern "C" TENSORRTAPI nvinfer1::IPluginRegistry* getSafePluginRegistry() noexcept;
650
651 template <typename T>
652 class PluginRegistrar
653 {
654 public:
655     PluginRegistrar()
656     {
657         getSafePluginRegistry()->registerCreator(instance, "");
658     }
659
660 private:
661     T instance{};
662 };
663
664 } // namespace safe
665 } // namespace nvinfer1
666
667 #define REGISTER_SAFE_TENSORRT_PLUGIN(name)
668 \
669     static nvinfer1::safe::PluginRegistrar<name> pluginRegistrar##name {}
670 #endif // NV_INFER_SAFE_RUNTIME_H

```

10.19 NvInferVersion.h File Reference

Macros

- #define `NV_TENSORRT_MAJOR` 8
TensorRT major version.
- #define `NV_TENSORRT_MINOR` 6
TensorRT minor version.
- #define `NV_TENSORRT_PATCH` 0
TensorRT patch version.
- #define `NV_TENSORRT_BUILD` 0
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_SONAME_MAJOR` 8
Shared object library major version number.
- #define `NV_TENSORRT_SONAME_MINOR` 6
Shared object library minor version number.
- #define `NV_TENSORRT_SONAME_PATCH` 0
Shared object library patch version number.

10.19.1 Detailed Description

Defines the TensorRT version

10.19.2 Macro Definition Documentation

10.19.2.1 NV_TENSORRT_BUILD

```
#define NV_TENSORRT_BUILD 0
```

TensorRT build number.

10.19.2.2 NV_TENSORRT_LWS_MAJOR

```
#define NV_TENSORRT_LWS_MAJOR 0
```

TensorRT LWS major version.

10.19.2.3 NV_TENSORRT_LWS_MINOR

```
#define NV_TENSORRT_LWS_MINOR 0
```

TensorRT LWS minor version.

10.19.2.4 NV_TENSORRT_LWS_PATCH

```
#define NV_TENSORRT_LWS_PATCH 0
```

TensorRT LWS patch version.

10.19.2.5 NV_TENSORRT_MAJOR

```
#define NV_TENSORRT_MAJOR 8
```

TensorRT major version.

10.19.2.6 NV_TENSORRT_MINOR

```
#define NV_TENSORRT_MINOR 6
```

TensorRT minor version.

10.19.2.7 NV_TENSORRT_PATCH

```
#define NV_TENSORRT_PATCH 0
```

TensorRT patch version.

10.19.2.8 NV_TENSORRT_SONAME_MAJOR

```
#define NV_TENSORRT_SONAME_MAJOR 8
```

Shared object library major version number.

10.19.2.9 NV_TENSORRT_SONAME_MINOR

```
#define NV_TENSORRT_SONAME_MINOR 6
```

Shared object library minor version number.

10.19.2.10 NV_TENSORRT_SONAME_PATCH

```
#define NV_TENSORRT_SONAME_PATCH 0
```

Shared object library patch version number.

10.20 NvInferVersion.h

[Go to the documentation of this file.](#)

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13
14
15
16
17
18
19 #ifndef NV_INFER_VERSION_H
20 #define NV_INFER_VERSION_H
21
22 #define NV_TENSORRT_MAJOR 8
23 #define NV_TENSORRT_MINOR 6
24 #define NV_TENSORRT_PATCH 0
25 #define NV_TENSORRT_BUILD 0
26
27 #define NV_TENSORRT_LWS_MAJOR 0
28 #define NV_TENSORRT_LWS_MINOR 0
29 #define NV_TENSORRT_LWS_PATCH 0
30
31 #define NV_TENSORRT_SONAME_MAJOR 8
32 #define NV_TENSORRT_SONAME_MINOR 6
33 #define NV_TENSORRT_SONAME_PATCH 0
34
35 #endif // NV_INFER_VERSION_H
```

10.21 NvOnnxConfig.h File Reference

```
#include "NvInfer.h"
```

Classes

- class [nvonnxparser::IOnnxConfig](#)
Configuration Manager Class.

Namespaces

- namespace [nvonnxparser](#)
The TensorRT ONNX parser API namespace.

Functions

- IOnnxConfig * [nvonnxparser::createONNXConfig](#) ()

10.21.1 Detailed Description

This is the API file for the Configuration Manager for ONNX Parser for Nvidia TensorRT.

10.22 NvOnnxConfig.h

[Go to the documentation of this file.](#)

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993–2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_OnnxConfig_H
14 #define NV_OnnxConfig_H
15
16 #include "NvInfer.h"
17
18 namespace nvonnxparser
19 {
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41 class IOnnxConfig
42 {
43 public:
44     virtual ~IOnnxConfig() noexcept = default;
45     typedef int32_t Verbosity;
46
47
48
49
50 }
```

```

59     virtual void setModelDtype(const nvinfer1::DataType) noexcept = 0;
60
61     virtual nvinfer1::DataType getModelDtype() const noexcept = 0;
62
63     virtual char const* getModelFileName() const noexcept = 0;
64
65     virtual void setModelFileName(char const* onnxFilename) noexcept = 0;
66
67     virtual Verbosity getVerbosityLevel() const noexcept = 0;
68
69     virtual void addVerbosity() noexcept = 0;
70
71     virtual void reduceVerbosity() noexcept = 0;
72
73     virtual void setVerbosityLevel(Verbosity) noexcept = 0;
74
75     virtual char const* getTextFileName() const noexcept = 0;
76
77     virtual void setTextFileName(char const* textFileName) noexcept = 0;
78
79     virtual char const* getFullTextFileName() const noexcept = 0;
80
81     virtual void setFullTextFileName(char const* fullTextFileName) noexcept = 0;
82
83     virtual bool getPrintLayerInfo() const noexcept = 0;
84
85     virtual void setPrintLayerInfo(bool) noexcept = 0;
86
87     TRT_DEPRECATED virtual void destroy() noexcept = 0;
88
89 }; // class IOnnxConfig
90
91 TENSORRTAPI IOnnxConfig* createONNXConfig();
92
93 } // namespace nvonnxparser
94
95 #endif

```

10.23 NvUffParser.h File Reference

```
#include "NvInfer.h"
```

Classes

- class [nvuffparser::FieldMap](#)
An array of field params used as a layer parameter for plugin layers.
- struct [nvuffparser::FieldCollection](#)
- class [nvuffparser::IUFFParser](#)
Class used for parsing models described using the UFF format.

Namespaces

- namespace [nvuffparser](#)
The TensorRT UFF parser API namespace.

Macros

- `#define` [UFF_REQUIRED_VERSION_MAJOR](#) 0
- `#define` [UFF_REQUIRED_VERSION_MINOR](#) 6
- `#define` [UFF_REQUIRED_VERSION_PATCH](#) 9

Enumerations

- enum class `nvuffparser::UffInputOrder` : `int32_t` { `nvuffparser::kNCHW` = 0 , `nvuffparser::kNHWC` = 1 , `nvuffparser::kNC` = 2 }

The different possible supported input order.

- enum class `nvuffparser::FieldType` : `int32_t` { `nvuffparser::kFLOAT` = 0 , `nvuffparser::kINT32` = 1 , `nvuffparser::kCHAR` = 2 , `nvuffparser::kDIMS` = 4 , `nvuffparser::kDATATYPE` = 5 , `nvuffparser::kUNKNOWN` = 6 }

The possible field types for custom layer.

Functions

- `IUffParser *` `nvuffparser::createUffParser` () noexcept
Creates a `IUffParser` object.
- void `nvuffparser::shutdownProtobufLibrary` (void) noexcept
Shuts down protocol buffers library.

10.23.1 Detailed Description

This is the API for the UFF Parser

10.23.2 Macro Definition Documentation

10.23.2.1 UFF_REQUIRED_VERSION_MAJOR

```
#define UFF_REQUIRED_VERSION_MAJOR 0
```

10.23.2.2 UFF_REQUIRED_VERSION_MINOR

```
#define UFF_REQUIRED_VERSION_MINOR 6
```

10.23.2.3 UFF_REQUIRED_VERSION_PATCH

```
#define UFF_REQUIRED_VERSION_PATCH 9
```


10.24 NvUffParser.h

[Go to the documentation of this file.](#)

```

1  /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_UFF_PARSER_H
14 #define NV_UFF_PARSER_H
15
16 #include "NvInfer.h"
17
18
19
20
21
22
23
24 // Current supported Universal Framework Format (UFF) version for the parser.
25 #define UFF_REQUIRED_VERSION_MAJOR 0
26 #define UFF_REQUIRED_VERSION_MINOR 6
27 #define UFF_REQUIRED_VERSION_PATCH 9
28
29 namespace nvuffparser
30 {
31
32
33
34 enum class UffInputOrder : int32_t
35 {
36     kNCHW = 0,
37     kNHWC = 1,
38     kNC = 2
39 };
40
41 enum class FieldType : int32_t
42 {
43     kFLOAT = 0,
44     kINT32 = 1,
45     kCHAR = 2,
46     kDIMS = 4,
47     kDATATYPE = 5,
48     kUNKNOWN = 6
49 };
50
51 class TENSORRTAPI FieldMap
52 {
53 public:
54     char const* name;
55     void const* data;
56     FieldType type = FieldType::kUNKNOWN;
57     int32_t length = 1;
58
59     FieldMap(char const* name, void const* data, const FieldType type, int32_t length = 1);
60 };
61
62 struct FieldCollection
63 {
64     int32_t nbFields;
65     FieldMap const* fields;
66 };
67
68 class IUffParser
69 {
70 public:
71     virtual bool registerInput(char const* inputName, nvinfer1::Dims inputDims, UffInputOrder inputOrder)
72     noexcept = 0;
73
74     virtual bool registerOutput(char const* outputName) noexcept = 0;
75
76     virtual bool parse(char const* file, nvinfer1::INetworkDefinition& network,
77         nvinfer1::DataType weightsType = nvinfer1::DataType::kFLOAT) noexcept = 0;
78
79     virtual bool parseBuffer(char const* buffer, std::size_t size, nvinfer1::INetworkDefinition& network,
80         nvinfer1::DataType weightsType = nvinfer1::DataType::kFLOAT) noexcept = 0;
81
82     TRT_DEPRECATED virtual void destroy() noexcept = 0;
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139

```

```

140
144     virtual int32_t getUffRequiredVersionMajor() noexcept = 0;
145
149     virtual int32_t getUffRequiredVersionMinor() noexcept = 0;
150
154     virtual int32_t getUffRequiredVersionPatch() noexcept = 0;
155
159     virtual void setPluginNamespace(char const* libNamespace) noexcept = 0;
160
161     virtual ~IUffParser() noexcept = default;
162
163 public:
164     //
165     virtual void setErrorRecorder(nvinfer1::IErrorRecorder* recorder) noexcept = 0;
166
167     virtual nvinfer1::IErrorRecorder* getErrorRecorder() const noexcept = 0;
168 };
169
170 TENSORRTAPI IUffParser* createUffParser() noexcept;
171
172 TENSORRTAPI void shutdownProtobufLibrary(void) noexcept;
173
174 } // namespace nvuffparser
175
176 extern "C" TENSORRTAPI void* createNvUffParser_INTERNAL() noexcept;
177
178 #endif /* !NV_UFF_PARSER_H */

```

10.25 NvUtils.h File Reference

```
#include "NvInfer.h"
```

Namespaces

- namespace `nvinfer1`
The TensorRT API version 1 namespace.
- namespace `nvinfer1::utils`

Functions

- `TRT_DEPRECATED` bool `nvinfer1::utils::reshapeWeights` (Weights const &input, int32_t const *shape, int32_t const *shapeOrder, void *data, int32_t nbDims) noexcept
Reformat the input weights of the given shape based on the new order of dimensions.
- `TRT_DEPRECATED` bool `nvinfer1::utils::reorderSubBuffers` (void *input, int32_t const *order, int32_t num, int32_t size) noexcept
Takes an input stream and re-orders num chunks of the data given the size and order.
- `TRT_DEPRECATED` bool `nvinfer1::utils::transposeSubBuffers` (void *input, DataType type, int32_t num, int32_t height, int32_t width) noexcept
*Transpose num sub-buffers of height * width.*

10.25.1 Detailed Description

This file includes various utility functions

10.26 NvUtils.h

[Go to the documentation of this file.](#)

```

1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_UTILS_H
14 #define NV_UTILS_H
15
16 #include "NvInfer.h"
17
18
19
20
21 namespace nvinfer1
22 {
23     namespace utils
24     {
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

10.27 NvOnnxParser.h File Reference

```

#include "NvInfer.h"
#include <stddef.h>
#include <vector>

```

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

Namespaces

- namespace [nvonnxparser](#)
The TensorRT ONNX parser API namespace.

Macros

- `#define NV_ONNX_PARSER_MAJOR 0`
- `#define NV_ONNX_PARSER_MINOR 1`
- `#define NV_ONNX_PARSER_PATCH 0`

Typedefs

- `typedef std::pair< std::vector< size_t >, bool > SubGraph_t`
The data structure containing the parsing capability of a set of nodes in an ONNX graph.
- `typedef std::vector< SubGraph_t > SubGraphCollection_t`
The data structure containing all SubGraph_t partitioned out of an ONNX graph.

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 }`
the type of parser error

Functions

- `template<typename T >`
`int32_t nvonnxparser::EnumMax ()`
- `template<> int32_t nvonnxparser::EnumMax< ErrorCode > ()`
- `TENSORRTAPI void * createNvOnnxParser_INTERNAL (void *network, void *logger, int version)`
- `TENSORRTAPI int getNvOnnxParserVersion ()`

10.27.1 Detailed Description

This is the API for the ONNX Parser

10.27.2 Macro Definition Documentation

10.27.2.1 NV_ONNX_PARSER_MAJOR

```
#define NV_ONNX_PARSER_MAJOR 0
```

10.27.2.2 NV_ONNX_PARSER_MINOR

```
#define NV_ONNX_PARSER_MINOR 1
```

10.27.2.3 NV_ONNX_PARSER_PATCH

```
#define NV_ONNX_PARSER_PATCH 0
```

10.27.3 Typedef Documentation

10.27.3.1 SubGraph_t

`SubGraph_t`

The data structure containing the parsing capability of a set of nodes in an ONNX graph.

10.27.3.2 SubGraphCollection_t

`SubGraphCollection_t`

The data structure containing all `SubGraph_t` partitioned out of an ONNX graph.

10.27.4 Function Documentation

10.27.4.1 createNvOnnxParser_INTERNAL()

```
TENSORRTAPI void * createNvOnnxParser_INTERNAL (
    void * network,
    void * logger,
    int version )
```

10.27.4.2 getNvOnnxParserVersion()

```
TENSORRTAPI int getNvOnnxParserVersion ( )
```

10.28 NvOnnxParser.h

[Go to the documentation of this file.](#)

```
1 /*
2  * Copyright (c) 1993-2022, NVIDIA CORPORATION. All rights reserved.
3  *
4  * Permission is hereby granted, free of charge, to any person obtaining a
5  * copy of this software and associated documentation files (the "Software"),
6  * to deal in the Software without restriction, including without limitation
7  * the rights to use, copy, modify, merge, publish, distribute, sublicense,
8  * and/or sell copies of the Software, and to permit persons to whom the
9  * Software is furnished to do so, subject to the following conditions:
10 *
11 * The above copyright notice and this permission notice shall be included in
12 * all copies or substantial portions of the Software.
13 *
14 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
17 * THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
18 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
19 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
20 * DEALINGS IN THE SOFTWARE.
21 */
22
23 #ifndef NV_ONNX_PARSER_H
24 #define NV_ONNX_PARSER_H
25
26 #include "NvInfer.h"
27 #include <stddef.h>
28 #include <vector>
29
30
31
32
33 #define NV_ONNX_PARSER_MAJOR 0
34 #define NV_ONNX_PARSER_MINOR 1
35 #define NV_ONNX_PARSER_PATCH 0
36
37 static const int NV_ONNX_PARSER_VERSION = ((NV_ONNX_PARSER_MAJOR * 10000) + (NV_ONNX_PARSER_MINOR * 100) +
38 NV_ONNX_PARSER_PATCH);
39
40
41 typedef std::pair<std::vector<size_t>, bool> SubGraph_t;
42
43
44 typedef std::vector<SubGraph_t> SubGraphCollection_t;
45
46
47 namespace nvonnxparser
48 {
49
50 template <typename T>
51 inline int32_t EnumMax();
52
53 enum class ErrorCode : int
54 {
55     kSUCCESS = 0,
56     kINTERNAL_ERROR = 1,
57     kMEM_ALLOC_FAILED = 2,
58     kMODEL_DESERIALIZE_FAILED = 3,
59     kINVALID_VALUE = 4,
60     kINVALID_GRAPH = 5,
61     kINVALID_NODE = 6,
62     kUNSUPPORTED_GRAPH = 7,
63     kUNSUPPORTED_NODE = 8
64 };
65
66 template <>
67 inline int32_t EnumMax<ErrorCode>()
68 {
69     return 9;
70 }
71
72 }
```

```

94 class IParserError
95 {
96 public:
97     virtual ErrorCode code() const = 0;
98     virtual const char* desc() const = 0;
99     virtual const char* file() const = 0;
100     virtual int line() const = 0;
101     virtual const char* func() const = 0;
102     virtual int node() const = 0;
103
104 protected:
105     virtual ~IParserError() {}
106 };
107
108 class IParser
109 {
110 public:
111     virtual bool parse(void const* serialized_onnx_model,
112                        size_t serialized_onnx_model_size,
113                        const char* model_path = nullptr)
114         = 0;
115
116     virtual bool parseFromFile(const char* onnx_model_file, int verbosity) = 0;
117
118     virtual bool supportsModel(void const* serialized_onnx_model,
119                                size_t serialized_onnx_model_size,
120                                SubGraphCollection_t& sub_graph_collection,
121                                const char* model_path = nullptr)
122         = 0;
123
124     virtual bool parseWithWeightDescriptors(
125         void const* serialized_onnx_model, size_t serialized_onnx_model_size)
126         = 0;
127
128     virtual bool supportsOperator(const char* op_name) const = 0;
129     TRT_DEPRECATED virtual void destroy() = 0;
130     virtual int getNbErrors() const = 0;
131     virtual IParserError const* getError(int index) const = 0;
132     virtual void clearErrors() = 0;
133
134     virtual ~IParser() noexcept = default;
135 };
136
137 } // namespace nvonnxparser
138
139 extern "C" TENSORRTAPI void* createNvOnnxParser_INTERNAL(void* network, void* logger, int version);
140 extern "C" TENSORRTAPI int getNvOnnxParserVersion();
141
142 namespace nvonnxparser
143 {
144     namespace
145     {
146         inline IParser* createParser(nvinfer1::INetworkDefinition& network, nvinfer1::ILogger& logger)
147         {
148             return static_cast<IParser*>(createNvOnnxParser_INTERNAL(&network, &logger, NV_ONNX_PARSER_VERSION));
149         }
150     } // namespace
151 } // namespace nvonnxparser
152
153 #endif // NV_ONNX_PARSER_H

```

Index

- ~IActivationLayer
 - nvinfer1::IActivationLayer, [126](#)
- ~IAlgorithm
 - nvinfer1::IAlgorithm, [130](#)
- ~IAlgorithmContext
 - nvinfer1::IAlgorithmContext, [133](#)
- ~IAlgorithmIOInfo
 - nvinfer1::IAlgorithmIOInfo, [135](#)
- ~IAlgorithmSelector
 - nvinfer1::IAlgorithmSelector, [137](#)
- ~IAlgorithmVariant
 - nvinfer1::IAlgorithmVariant, [139](#)
- ~IAssertionLayer
 - nvinfer1::IAssertionLayer, [141](#)
- ~IBinaryProtoBlob
 - nvcaffeparser1::IBinaryProtoBlob, [143](#)
- ~IBlobNameToTensor
 - nvcaffeparser1::IBlobNameToTensor, [145](#)
- ~IBuilder
 - nvinfer1::IBuilder, [147](#)
- ~IBuilderConfig
 - nvinfer1::IBuilderConfig, [158](#)
- ~ICaffeParser
 - nvcaffeparser1::ICaffeParser, [180](#)
- ~ICastLayer
 - nvinfer1::ICastLayer, [186](#)
- ~IConcatenationLayer
 - nvinfer1::IConcatenationLayer, [187](#)
- ~IConditionLayer
 - nvinfer1::IConditionLayer, [189](#)
- ~IConsistencyChecker
 - nvinfer1::consistency::IConsistencyChecker, [191](#)
- ~IConstantLayer
 - nvinfer1::IConstantLayer, [193](#)
- ~IConvolutionLayer
 - nvinfer1::IConvolutionLayer, [197](#)
- ~ICudaEngine
 - nvinfer1::ICudaEngine, [211](#)
 - nvinfer1::safe::ICudaEngine, [235](#)
- ~IDeconvolutionLayer
 - nvinfer1::IDeconvolutionLayer, [253](#)
- ~IDequantizeLayer
 - nvinfer1::IDequantizeLayer, [265](#)
- ~IDimensionExpr
 - nvinfer1::IDimensionExpr, [267](#)
- ~IEinsumLayer
 - nvinfer1::IEinsumLayer, [269](#)
- ~IElementWiseLayer
 - nvinfer1::IElementWiseLayer, [271](#)
- ~IEngineInspector
 - nvinfer1::IEngineInspector, [273](#)
- ~IErrorRecorder
 - nvinfer1::IErrorRecorder, [279](#)
- ~IExecutionContext
 - nvinfer1::IExecutionContext, [287](#)
 - nvinfer1::safe::IExecutionContext, [315](#)
- ~IExprBuilder
 - nvinfer1::IExprBuilder, [327](#)
- ~IFillLayer
 - nvinfer1::IFillLayer, [330](#)
- ~IFullyConnectedLayer
 - nvinfer1::IFullyConnectedLayer, [336](#)
- ~IGatherLayer
 - nvinfer1::IGatherLayer, [341](#)
- ~IGpuAllocator
 - nvinfer1::IGpuAllocator, [344](#)
- ~IGridSampleLayer
 - nvinfer1::IGridSampleLayer, [349](#)
- ~IHostMemory
 - nvinfer1::IHostMemory, [352](#)
- ~IIdentityLayer
 - nvinfer1::IIdentityLayer, [355](#)
- ~IIIfConditional
 - nvinfer1::IIIfConditional, [357](#)
- ~IIIfConditionalBoundaryLayer
 - nvinfer1::IIIfConditionalBoundaryLayer, [360](#)
- ~IIIfConditionalInputLayer
 - nvinfer1::IIIfConditionalInputLayer, [361](#)
- ~IIIfConditionalOutputLayer
 - nvinfer1::IIIfConditionalOutputLayer, [362](#)
- ~IInt8Calibrator
 - nvinfer1::IInt8Calibrator, [364](#)
- ~IInt8EntropyCalibrator
 - nvinfer1::IInt8EntropyCalibrator, [367](#)
- ~IInt8EntropyCalibrator2
 - nvinfer1::IInt8EntropyCalibrator2, [368](#)
- ~IInt8LegacyCalibrator
 - nvinfer1::IInt8LegacyCalibrator, [369](#)
- ~IInt8MinMaxCalibrator
 - nvinfer1::IInt8MinMaxCalibrator, [371](#)

- ~IteratorLayer
 - nvinfer1::IteratorLayer, [373](#)
- ~ILRNLayer
 - nvinfer1::ILRNLayer, [393](#)
- ~ILayer
 - nvinfer1::ILayer, [376](#)
- ~ILogger
 - nvinfer1::ILogger, [384](#)
- ~ILoop
 - nvinfer1::ILoop, [385](#)
- ~ILoopBoundaryLayer
 - nvinfer1::ILoopBoundaryLayer, [388](#)
- ~ILoopOutputLayer
 - nvinfer1::ILoopOutputLayer, [390](#)
- ~IMatrixMultiplyLayer
 - nvinfer1::IMatrixMultiplyLayer, [398](#)
- ~INMSLayer
 - nvinfer1::INMSLayer, [446](#)
- ~INetworkDefinition
 - nvinfer1::INetworkDefinition, [403](#)
- ~INoCopy
 - nvinfer1::INoCopy, [450](#)
- ~INonZeroLayer
 - nvinfer1::INonZeroLayer, [452](#)
- ~INormalizationLayer
 - nvinfer1::INormalizationLayer, [453](#)
- ~IOnnxConfig
 - nvonnxparser::IOnnxConfig, [460](#)
- ~IOptimizationProfile
 - nvinfer1::IOptimizationProfile, [467](#)
- ~IOutputAllocator
 - nvinfer1::IOutputAllocator, [472](#)
- ~IPaddingLayer
 - nvinfer1::IPaddingLayer, [475](#)
- ~IParametricReLULayer
 - nvinfer1::IParametricReLULayer, [479](#)
- ~IParser
 - nvonnxparser::IParser, [480](#)
- ~IParserError
 - nvonnxparser::IParserError, [485](#)
- ~IPluginChecker
 - nvinfer1::consistency::IPluginChecker, [487](#)
- ~IPluginCreator
 - nvinfer1::IPluginCreator, [490](#)
- ~IPluginFactoryV2
 - nvcaffeparser1::IPluginFactoryV2, [494](#)
- ~IPluginRegistry
 - nvinfer1::IPluginRegistry, [497](#)
- ~IPluginV2DynamicExt
 - nvinfer1::IPluginV2DynamicExt, [514](#)
- ~IPluginV2Ext
 - nvinfer1::IPluginV2Ext, [520](#)
- ~IPluginV2Layer
 - nvinfer1::IPluginV2Layer, [530](#)
- ~IPoolingLayer
 - nvinfer1::IPoolingLayer, [533](#)
- ~IProfiler
 - nvinfer1::IProfiler, [541](#)
- ~IQuantizeLayer
 - nvinfer1::IQuantizeLayer, [544](#)
- ~IRNNv2Layer
 - nvinfer1::IRNNv2Layer, [575](#)
- ~IRaggedSoftMaxLayer
 - nvinfer1::IRaggedSoftMaxLayer, [546](#)
- ~IRecurrenceLayer
 - nvinfer1::IRecurrenceLayer, [548](#)
- ~IReduceLayer
 - nvinfer1::IReduceLayer, [550](#)
- ~IRefitter
 - nvinfer1::IRefitter, [553](#)
- ~IResizeLayer
 - nvinfer1::IResizeLayer, [563](#)
- ~IRReverseSequenceLayer
 - nvinfer1::IRReverseSequenceLayer, [572](#)
- ~IRuntime
 - nvinfer1::IRuntime, [584](#)
 - nvinfer1::safe::IRuntime, [593](#)
- ~IScaleLayer
 - nvinfer1::IScaleLayer, [598](#)
- ~IScatterLayer
 - nvinfer1::IScatterLayer, [604](#)
- ~ISelectLayer
 - nvinfer1::ISelectLayer, [606](#)
- ~IShapeLayer
 - nvinfer1::IShapeLayer, [607](#)
- ~IShuffleLayer
 - nvinfer1::IShuffleLayer, [609](#)
- ~ISliceLayer
 - nvinfer1::ISliceLayer, [615](#)
- ~ISoftMaxLayer
 - nvinfer1::ISoftMaxLayer, [620](#)
- ~ITensor
 - nvinfer1::ITensor, [623](#)
- ~ITimingCache
 - nvinfer1::ITimingCache, [634](#)
- ~ITopKLayer
 - nvinfer1::ITopKLayer, [637](#)
- ~ITripLimitLayer
 - nvinfer1::ITripLimitLayer, [640](#)
- ~IUffParser
 - nvuffparser::IUffParser, [641](#)
- ~IUnaryLayer
 - nvinfer1::IUnaryLayer, [646](#)
- ActivationType
 - nvinfer1, [41](#)
- addActivation
 - nvinfer1::INetworkDefinition, [404](#)

- addAssertion
 - [nvinfer1::INetworkDefinition, 404](#)
- addCast
 - [nvinfer1::INetworkDefinition, 405](#)
- addConcatenation
 - [nvinfer1::INetworkDefinition, 405](#)
- addConstant
 - [nvinfer1::INetworkDefinition, 406](#)
- addConvolution
 - [nvinfer1::INetworkDefinition, 406](#)
- addConvolutionNd
 - [nvinfer1::INetworkDefinition, 407](#)
- addDeconvolution
 - [nvinfer1::INetworkDefinition, 408](#)
- addDeconvolutionNd
 - [nvinfer1::INetworkDefinition, 409](#)
- addDequantize
 - [nvinfer1::INetworkDefinition, 410](#)
- addEinsum
 - [nvinfer1::INetworkDefinition, 410](#)
- addElementWise
 - [nvinfer1::INetworkDefinition, 411](#)
- addFill
 - [nvinfer1::INetworkDefinition, 411](#)
- addFullyConnected
 - [nvinfer1::INetworkDefinition, 412](#)
- addGather
 - [nvinfer1::INetworkDefinition, 413](#)
- addGatherV2
 - [nvinfer1::INetworkDefinition, 413](#)
- addGridSample
 - [nvinfer1::INetworkDefinition, 414](#)
- addIdentity
 - [nvinfer1::INetworkDefinition, 414](#)
- addIfConditional
 - [nvinfer1::INetworkDefinition, 415](#)
- addInput
 - [nvinfer1::IIfConditional, 357](#)
 - [nvinfer1::INetworkDefinition, 415](#)
- addIterator
 - [nvinfer1::ILoop, 386](#)
- addLoop
 - [nvinfer1::INetworkDefinition, 416](#)
- addLoopOutput
 - [nvinfer1::ILoop, 386](#)
- addLRN
 - [nvinfer1::INetworkDefinition, 416](#)
- addMatrixMultiply
 - [nvinfer1::INetworkDefinition, 417](#)
- addNMS
 - [nvinfer1::INetworkDefinition, 418](#)
- addNonZero
 - [nvinfer1::INetworkDefinition, 418](#)
- addNormalization
 - [nvinfer1::INetworkDefinition, 419](#)
- addOneHot
 - [nvinfer1::INetworkDefinition, 419](#)
- addOptimizationProfile
 - [nvinfer1::IBuilderConfig, 159](#)
- addOutput
 - [nvinfer1::IIfConditional, 357](#)
- addPadding
 - [nvinfer1::INetworkDefinition, 420](#)
- addPaddingNd
 - [nvinfer1::INetworkDefinition, 421](#)
- addParametricReLU
 - [nvinfer1::INetworkDefinition, 421](#)
- addPluginV2
 - [nvinfer1::INetworkDefinition, 422](#)
- addPooling
 - [nvinfer1::INetworkDefinition, 422](#)
- addPoolingNd
 - [nvinfer1::INetworkDefinition, 423](#)
- addQuantize
 - [nvinfer1::INetworkDefinition, 424](#)
- addRaggedSoftMax
 - [nvinfer1::INetworkDefinition, 424](#)
- addRecurrence
 - [nvinfer1::ILoop, 386](#)
- addReduce
 - [nvinfer1::INetworkDefinition, 425](#)
- addResize
 - [nvinfer1::INetworkDefinition, 426](#)
- addReverseSequence
 - [nvinfer1::INetworkDefinition, 426](#)
- addRNNv2
 - [nvinfer1::INetworkDefinition, 427](#)
- addScale
 - [nvinfer1::INetworkDefinition, 428](#)
- addScaleNd
 - [nvinfer1::INetworkDefinition, 429](#)
- addScatter
 - [nvinfer1::INetworkDefinition, 430](#)
- addSelect
 - [nvinfer1::INetworkDefinition, 431](#)
- addShape
 - [nvinfer1::INetworkDefinition, 431](#)
- addShuffle
 - [nvinfer1::INetworkDefinition, 433](#)
- addSlice
 - [nvinfer1::INetworkDefinition, 433](#)
- addSoftMax
 - [nvinfer1::INetworkDefinition, 434](#)
- addTopK
 - [nvinfer1::INetworkDefinition, 434](#)
- addTripLimit
 - [nvinfer1::ILoop, 386](#)
- addUnary

- nvinfer1::INetworkDefinition, 435
- addVerbosity
 - nvonnxparser::IOnnxConfig, 460
- allInputDimensionsSpecified
 - nvinfer1::IExecutionContext, 287
- allInputShapesSpecified
 - nvinfer1::IExecutionContext, 288
- allocate
 - nvinfer1::IGpuAllocator, 345
- AllocatorFlag
 - nvinfer1, 41
- AllocatorFlags
 - nvinfer1, 38
- anchorsRatioCount
 - nvinfer1::plugin::RPROIParams, 662
- anchorsScaleCount
 - nvinfer1::plugin::RPROIParams, 662
- AsciiChar
 - nvinfer1, 38
- aspectRatios
 - nvinfer1::plugin::GridAnchorParameters, 124
 - nvinfer1::plugin::PriorBoxParameters, 657
- attachToContext
 - nvinfer1::IPluginV2Ext, 521
- backgroundLabelId
 - nvinfer1::plugin::DetectionOutputParameters, 94
 - nvinfer1::plugin::NMSParameters, 648
- bindingIsInput
 - nvinfer1::ICudaEngine, 211
 - nvinfer1::safe::ICudaEngine, 235
- BoundingBoxFormat
 - nvinfer1, 42
- buildEngineWithConfig
 - nvinfer1::IBuilder, 147
- BuilderFlag
 - nvinfer1, 42
- BuilderFlags
 - nvinfer1, 38
- buildSerializedNetwork
 - nvinfer1::IBuilder, 148
- CalibrationAlgoType
 - nvinfer1, 44
- canBroadcastInputAcrossBatch
 - nvinfer1::IPluginV2Ext, 521
- canRunOnDLA
 - nvinfer1::IBuilderConfig, 159
- CENTER_SIZE
 - nvinfer1::plugin, 84
- char_t
 - nvinfer1, 38
- child
 - nvinfer1::plugin::softmaxTree, 664
- classes
 - nvinfer1::plugin::RegionParameters, 661
- clear
 - nvinfer1::IErrorRecorder, 279
- clearErrors
 - nvonnxparser::IParser, 481
- clearFlag
 - nvinfer1::IBuilderConfig, 159
- clearQuantizationFlag
 - nvinfer1::IBuilderConfig, 160
- clip
 - nvinfer1::plugin::PriorBoxParameters, 657
- clone
 - nvinfer1::IPluginV2, 504
 - nvinfer1::IPluginV2DynamicExt, 515
 - nvinfer1::IPluginV2Ext, 522
- code
 - nvonnxparser::IParserError, 485
- codeType
 - nvinfer1::plugin::DetectionOutputParameters, 94
- CodeTypeSSD
 - nvinfer1::plugin, 83
- combine
 - nvinfer1::ITimingCache, 634
- confidenceThreshold
 - nvinfer1::plugin::DetectionOutputParameters, 94
- configurePlugin
 - nvinfer1::IPluginV2DynamicExt, 515
 - nvinfer1::IPluginV2Ext, 522
 - nvinfer1::IPluginV2IOExt, 527
- configureWithFormat
 - nvinfer1::IPluginV2, 504
 - nvinfer1::IPluginV2Ext, 524
- confSigmoid
 - nvinfer1::plugin::DetectionOutputParameters, 94
- constant
 - nvinfer1::IExprBuilder, 327
- coords
 - nvinfer1::plugin::RegionParameters, 661
- CORNER
 - nvinfer1::plugin, 84
- CORNER_SIZE
 - nvinfer1::plugin, 84
- count
 - nvinfer1::Weights, 666
- createAnchorGeneratorPlugin
 - NvInferPlugin.h, 727
- createBatchedNMSPlugin
 - NvInferPlugin.h, 727
- createBuilderConfig
 - nvinfer1::IBuilder, 148
- createCaffeParser
 - nvcaffeparser1, 27
- createConsistencyChecker_INTERNAL
 - NvInferConsistency.h, 722

- createEngineInspector
 - nvinfer1::ICudaEngine, 212
- createExecutionContext
 - nvinfer1::ICudaEngine, 212
 - nvinfer1::safe::ICudaEngine, 236
- createExecutionContextWithoutDeviceMemory
 - nvinfer1::ICudaEngine, 212
 - nvinfer1::safe::ICudaEngine, 236
- createInferRuntime
 - nvinfer1::safe, 84
- createInstanceNormalizationPlugin
 - NvInferPlugin.h, 727
- createNetworkV2
 - nvinfer1::IBuilder, 149
- createNMSPlugin
 - NvInferPlugin.h, 728
- createNormalizePlugin
 - NvInferPlugin.h, 728
- createNvOnnxParser_INTERNAL
 - NvOnnxParser.h, 780
- createONNXConfig
 - nvonnxparser, 90
- createOptimizationProfile
 - nvinfer1::IBuilder, 149
- createPlugin
 - nvcaffeparser1::IPluginFactoryV2, 494
 - nvinfer1::IPluginCreator, 490
- createPriorBoxPlugin
 - NvInferPlugin.h, 729
- createRegionPlugin
 - NvInferPlugin.h, 729
- createReorgPlugin
 - NvInferPlugin.h, 729
- createRPNROIPlugin
 - NvInferPlugin.h, 730
- createSplitPlugin
 - NvInferPlugin.h, 730
- createTimingCache
 - nvinfer1::IBuilderConfig, 160
- createUffParser
 - nvuffparser, 92
- d
 - nvinfer1::Dims32, 100
 - nvinfer1::DimsExprs, 102
- data
 - nvinfer1::IHostMemory, 353
 - nvinfer1::plugin::Quadruple, 660
 - nvinfer1::PluginField, 651
 - nvuffparser::FieldMap, 122
- DataType
 - nvinfer1, 44
- deallocate
 - nvinfer1::IGpuAllocator, 345
- decRefCount
 - nvinfer1::IErrorRecorder, 279
- deregisterCreator
 - nvinfer1::IPluginRegistry, 497
- deregisterLibrary
 - nvinfer1::IPluginRegistry, 497
- desc
 - nvinfer1::DynamicPluginTensorDesc, 106
 - nvonnxparser::IParserError, 485
- deserializeCudaEngine
 - nvinfer1::IRuntime, 584
 - nvinfer1::safe::IRuntime, 594
- deserializeLibrary
 - nvinfer1::IPluginRegistry, 498
- deserializePlugin
 - nvinfer1::IPluginCreator, 490
- destroy
 - nvcaffeparser1::IBinaryProtoBlob, 143
 - nvcaffeparser1::ICaffeParser, 181
 - nvinfer1::IBuilder, 149
 - nvinfer1::IBuilderConfig, 161
 - nvinfer1::ICudaEngine, 213
 - nvinfer1::IExecutionContext, 288
 - nvinfer1::IHostMemory, 353
 - nvinfer1::INetworkDefinition, 436
 - nvinfer1::IPluginV2, 505
 - nvinfer1::IRefitter, 553
 - nvinfer1::IRuntime, 585
 - nvonnxparser::IOnnxConfig, 460
 - nvonnxparser::IParser, 481
 - nvuffparser::IUffParser, 642
- detachFromContext
 - nvinfer1::IPluginV2Ext, 524
- DeviceType
 - nvinfer1, 45
- DimensionOperation
 - nvinfer1, 45
- Dims, 96
 - nvinfer1, 38
- dims
 - nvinfer1::PluginTensorDesc, 655
- Dims2
 - nvinfer1::Dims2, 97
- Dims3
 - nvinfer1::Dims3, 98, 99
- Dims4
 - nvinfer1::Dims4, 101
- DimsHW
 - nvinfer1::DimsHW, 104
- dynamicRangelsSet
 - nvinfer1::ITensor, 624
- ElementWiseOperation
 - nvinfer1, 46

EngineCapability
 nvinfer1, 46
 enqueue
 nvinfer1::IExecutionContext, 289
 nvinfer1::IPluginV2, 505
 nvinfer1::IPluginV2DynamicExt, 516
 enqueueV2
 nvinfer1::IExecutionContext, 289
 nvinfer1::safe::IExecutionContext, 315
 enqueueV3
 nvinfer1::IExecutionContext, 290
 nvinfer1::safe::IExecutionContext, 316
 EnumMax
 nvinfer1, 74
 nvonnxparser, 90
 EnumMax< BoundingBoxFormat >
 nvinfer1, 74
 EnumMax< BuilderFlag >
 nvinfer1, 74
 EnumMax< CalibrationAlgoType >
 nvinfer1, 74
 EnumMax< DeviceType >
 nvinfer1, 74
 EnumMax< DimensionOperation >
 nvinfer1, 75
 EnumMax< ErrorCode >
 nvonnxparser, 90
 EnumMax< FillOperation >
 nvinfer1, 75
 EnumMax< GatherMode >
 nvinfer1, 75
 EnumMax< LayerInformationFormat >
 nvinfer1, 75
 EnumMax< LayerType >
 nvinfer1, 76
 EnumMax< LoopOutput >
 nvinfer1, 76
 EnumMax< MatrixOperation >
 nvinfer1, 76
 EnumMax< MemoryPoolType >
 nvinfer1, 76
 EnumMax< NetworkDefinitionCreationFlag >
 nvinfer1, 77
 EnumMax< OptProfileSelector >
 nvinfer1, 77
 EnumMax< ProfilingVerbosity >
 nvinfer1, 77
 EnumMax< QuantizationFlag >
 nvinfer1, 77
 EnumMax< ReduceOperation >
 nvinfer1, 78
 EnumMax< RNNDirection >
 nvinfer1, 78
 EnumMax< RNNGateType >
 nvinfer1, 78
 EnumMax< RNNInputMode >
 nvinfer1, 78
 EnumMax< RNNOperation >
 nvinfer1, 79
 EnumMax< SampleMode >
 nvinfer1, 79
 EnumMax< ScaleMode >
 nvinfer1, 79
 EnumMax< ScatterMode >
 nvinfer1, 79
 EnumMax< TacticSource >
 nvinfer1, 80
 EnumMax< TempfileControlFlag >
 nvinfer1, 80
 EnumMax< TopKOperation >
 nvinfer1, 80
 EnumMax< TripLimit >
 nvinfer1, 80
 EnumMax< UnaryOperation >
 nvinfer1, 81
 EnumMax< WeightsRole >
 nvinfer1, 81
 ErrorCode
 nvinfer1, 47
 nvonnxparser, 89
 ErrorDesc
 nvinfer1::IErrorRecorder, 278
 execute
 nvinfer1::IExecutionContext, 291
 executeV2
 nvinfer1::IExecutionContext, 291
 featureStride
 nvinfer1::plugin::RPROIParams, 662
 FieldMap
 nvuffparser::FieldMap, 121
 fields
 nvinfer1::PluginFieldCollection, 652
 nvuffparser::FieldCollection, 120
 FieldType
 nvuffparser, 91
 file
 nvonnxparser::IParserError, 485
 FillOperation
 nvinfer1, 49
 find
 nvcaffeparser1::IBlobNameToTensor, 145
 flip
 nvinfer1::plugin::PriorBoxParameters, 657
 format
 nvinfer1::PluginTensorDesc, 655
 free
 nvinfer1::IGpuAllocator, 346

- func
 - nvonnxparser::IParserError, [486](#)
- GatherMode
 - nvinfer1, [49](#)
- getActivationType
 - nvinfer1::IActivationLayer, [126](#)
- getAlgorithm
 - nvinfer1::IInt8Calibrator, [364](#)
 - nvinfer1::IInt8EntropyCalibrator, [367](#)
 - nvinfer1::IInt8EntropyCalibrator2, [368](#)
 - nvinfer1::IInt8LegacyCalibrator, [369](#)
 - nvinfer1::IInt8MinMaxCalibrator, [372](#)
- getAlgorithmIOInfo
 - nvinfer1::IAlgorithm, [130](#)
- getAlgorithmIOInfoByIndex
 - nvinfer1::IAlgorithm, [130](#)
- getAlgorithmSelector
 - nvinfer1::IBuilderConfig, [161](#)
- getAlgorithmVariant
 - nvinfer1::IAlgorithm, [131](#)
- getAlignCorners
 - nvinfer1::IGridSampleLayer, [349](#)
 - nvinfer1::IResizeLayer, [564](#)
- getAll
 - nvinfer1::IRefitter, [554](#)
- getAllowedFormats
 - nvinfer1::ITensor, [624](#)
- getAllWeights
 - nvinfer1::IRefitter, [554](#)
- getAlpha
 - nvinfer1::IActivationLayer, [126](#)
 - nvinfer1::IFillLayer, [330](#)
 - nvinfer1::ILRNLayer, [394](#)
- getAverageCountExcludesPadding
 - nvinfer1::IPoolingLayer, [533](#)
- getAvgTimingIterations
 - nvinfer1::IBuilderConfig, [161](#)
- getAxes
 - nvinfer1::INormalizationLayer, [454](#)
 - nvinfer1::ISoftMaxLayer, [620](#)
- getAxis
 - nvinfer1::IConcatenationLayer, [188](#)
 - nvinfer1::IDequantizeLayer, [265](#)
 - nvinfer1::IIteratorLayer, [373](#)
 - nvinfer1::ILoopOutputLayer, [391](#)
 - nvinfer1::IOneHotLayer, [458](#)
 - nvinfer1::IQuantizeLayer, [545](#)
 - nvinfer1::IScatterLayer, [604](#)
- getBatch
 - nvinfer1::IInt8Calibrator, [364](#)
- getBatchAxis
 - nvinfer1::IReverseSequenceLayer, [572](#)
- getBatchSize
 - nvinfer1::IInt8Calibrator, [365](#)
- getBeta
 - nvinfer1::IActivationLayer, [127](#)
 - nvinfer1::IFillLayer, [330](#)
 - nvinfer1::ILRNLayer, [394](#)
- getBiasForGate
 - nvinfer1::IRNNv2Layer, [575](#)
- getBiasWeights
 - nvinfer1::IConvolutionLayer, [198](#)
 - nvinfer1::IDeconvolutionLayer, [253](#)
 - nvinfer1::IFullyConnectedLayer, [336](#)
- getBindingBytesPerComponent
 - nvinfer1::ICudaEngine, [213](#)
 - nvinfer1::safe::ICudaEngine, [237](#)
- getBindingComponentsPerElement
 - nvinfer1::ICudaEngine, [213](#)
 - nvinfer1::safe::ICudaEngine, [237](#)
- getBindingDataType
 - nvinfer1::ICudaEngine, [214](#)
 - nvinfer1::safe::ICudaEngine, [238](#)
- getBindingDimensions
 - nvinfer1::ICudaEngine, [214](#)
 - nvinfer1::IExecutionContext, [292](#)
 - nvinfer1::safe::ICudaEngine, [239](#)
- getBindingFormat
 - nvinfer1::ICudaEngine, [215](#)
 - nvinfer1::safe::ICudaEngine, [239](#)
- getBindingFormatDesc
 - nvinfer1::ICudaEngine, [216](#)
- getBindingIndex
 - nvinfer1::ICudaEngine, [216](#)
 - nvinfer1::safe::ICudaEngine, [240](#)
- getBindingName
 - nvinfer1::ICudaEngine, [217](#)
 - nvinfer1::safe::ICudaEngine, [241](#)
- getBindingVectorizedDim
 - nvinfer1::ICudaEngine, [218](#)
 - nvinfer1::safe::ICudaEngine, [241](#)
- getBlendFactor
 - nvinfer1::IPoolingLayer, [533](#)
- getBoundingBoxFormat
 - nvinfer1::INMSLayer, [446](#)
- getBroadcastAcrossBatch
 - nvinfer1::ITensor, [624](#)
- getBuilderOptimizationLevel
 - nvinfer1::IBuilderConfig, [161](#)
- getBuilderPluginRegistry
 - nvinfer1, [81](#)
- getCalibrationProfile
 - nvinfer1::IBuilderConfig, [162](#)
- getCellState
 - nvinfer1::IRNNv2Layer, [576](#)
- getChannelAxis
 - nvinfer1::IScaleLayer, [599](#)

getComputePrecision
 nvinfer1::INormalizationLayer, 454
 getConditional
 nvinfer1::IIfConditionalBoundaryLayer, 360
 getConstantValue
 nvinfer1::IDimensionExpr, 267
 getCoordinateTransformation
 nvinfer1::IResizeLayer, 564
 getCubicCoeff
 nvinfer1::IResizeLayer, 564
 getData
 nvcaffeparser1::IBinaryProtoBlob, 143
 getDataLength
 nvinfer1::IRNNv2Layer, 576
 getDataType
 nvcaffeparser1::IBinaryProtoBlob, 144
 nvinfer1::IAlgorithmIOInfo, 135
 getDebugSync
 nvinfer1::IExecutionContext, 293
 getDefaultDeviceType
 nvinfer1::IBuilderConfig, 162
 getDeviceMemorySize
 nvinfer1::ICudaEngine, 218
 nvinfer1::safe::ICudaEngine, 242
 getDeviceType
 nvinfer1::IBuilderConfig, 162
 getDilation
 nvinfer1::IConvolutionLayer, 198
 getDilationNd
 nvinfer1::IConvolutionLayer, 198
 nvinfer1::IDeconvolutionLayer, 253
 getDimensionName
 nvinfer1::ITensor, 624
 getDimensions
 nvcaffeparser1::IBinaryProtoBlob, 144
 nvinfer1::IAlgorithmContext, 133
 nvinfer1::IConstantLayer, 193
 nvinfer1::IFillLayer, 331
 nvinfer1::IOptimizationProfile, 467
 nvinfer1::ITensor, 625
 getDirection
 nvinfer1::IRNNv2Layer, 576
 getDLACore
 nvinfer1::IBuilderConfig, 162
 nvinfer1::IRuntime, 585
 getDynamicRangeMax
 nvinfer1::IRefitter, 555
 nvinfer1::ITensor, 625
 getDynamicRangeMin
 nvinfer1::IRefitter, 555
 nvinfer1::ITensor, 625
 getEngine
 nvinfer1::IExecutionContext, 293
 nvinfer1::safe::IExecutionContext, 316
 getEngineCapability
 nvinfer1::IBuilderConfig, 163
 nvinfer1::ICudaEngine, 218
 getEngineHostCodeAllowed
 nvinfer1::IRuntime, 586
 getEngineInformation
 nvinfer1::IEngineInspector, 274
 getEnqueueEmitsProfile
 nvinfer1::IExecutionContext, 293
 getEpsilon
 nvinfer1::INormalizationLayer, 454
 getEquation
 nvinfer1::IEinsumLayer, 269
 getError
 nvonnxparser::IParser, 481
 getErrorBuffer
 nvinfer1::safe::IExecutionContext, 317
 getErrorCode
 nvinfer1::IErrorRecorder, 280
 getErrorDesc
 nvinfer1::IErrorRecorder, 281
 getErrorRecorder
 nvcaffeparser1::ICaffeParser, 181
 nvinfer1::IBuilder, 150
 nvinfer1::ICudaEngine, 219
 nvinfer1::IEngineInspector, 274
 nvinfer1::IExecutionContext, 294
 nvinfer1::INetworkDefinition, 436
 nvinfer1::IPluginRegistry, 498
 nvinfer1::IRefitter, 555
 nvinfer1::IRuntime, 586
 nvinfer1::safe::ICudaEngine, 242
 nvinfer1::safe::IExecutionContext, 317
 nvinfer1::safe::IRuntime, 595
 nvuffparser::IUffParser, 642
 getExcludeOutside
 nvinfer1::IResizeLayer, 564
 getExecutionContext
 nvinfer1::IEngineInspector, 275
 getExtraMemoryTarget
 nvinfer1::IOptimizationProfile, 468
 getFieldNames
 nvinfer1::IPluginCreator, 491
 getFirstTranspose
 nvinfer1::IShuffleLayer, 609
 getFlag
 nvinfer1::IBuilderConfig, 163
 getFlags
 nvinfer1::IBuilderConfig, 163
 getFullTextFileName
 nvonnxparser::IOnnxConfig, 461
 getGatherAxis
 nvinfer1::IGatherLayer, 342
 getHardwareCompatibilityLevel

- nvinfer1::IBuilderConfig, [164](#)
 - nvinfer1::ICudaEngine, [219](#)
- getHiddenSize
 - nvinfer1::IRNNv2Layer, [576](#)
- getHiddenState
 - nvinfer1::IRNNv2Layer, [577](#)
- getImplementation
 - nvinfer1::IAlgorithmVariant, [139](#)
- getInferLibVersion
 - NvInferRuntimeCommon.h, [757](#)
- getInput
 - nvinfer1::ILayer, [376](#)
 - nvinfer1::INetworkDefinition, [437](#)
- getInputConsumedEvent
 - nvinfer1::IExecutionContext, [294](#)
 - nvinfer1::safe::IExecutionContext, [318](#)
- getInputMode
 - nvinfer1::IRNNv2Layer, [577](#)
- getInputTensorAddress
 - nvinfer1::safe::IExecutionContext, [318](#)
- getInt8Calibrator
 - nvinfer1::IBuilderConfig, [164](#)
- getInterfaceVersion
 - nvinfer1::IOutputAllocator, [472](#)
- getInterpolationMode
 - nvinfer1::IGridSampleLayer, [349](#)
- getIOTensorName
 - nvinfer1::ICudaEngine, [219](#)
 - nvinfer1::safe::ICudaEngine, [243](#)
- getK
 - nvinfer1::ILRNLayer, [394](#)
 - nvinfer1::ITopKLayer, [637](#)
- getKeepDimensions
 - nvinfer1::IReduceLayer, [550](#)
- getKernelSize
 - nvinfer1::IConvolutionLayer, [198](#)
 - nvinfer1::IDeconvolutionLayer, [254](#)
- getKernelSizeNd
 - nvinfer1::IConvolutionLayer, [199](#)
 - nvinfer1::IDeconvolutionLayer, [254](#)
- getKernelWeights
 - nvinfer1::IConvolutionLayer, [199](#)
 - nvinfer1::IDeconvolutionLayer, [254](#)
 - nvinfer1::IFullyConnectedLayer, [336](#)
- getLayer
 - nvinfer1::INetworkDefinition, [437](#)
- getLayerCount
 - nvinfer1::IRNNv2Layer, [577](#)
- getLayerInformation
 - nvinfer1::IEngineInspector, [275](#)
- getLocation
 - nvinfer1::ICudaEngine, [220](#)
 - nvinfer1::ITensor, [626](#)
- getLogger
 - nvinfer1::IBuilder, [150](#)
 - nvinfer1::IRefitter, [556](#)
 - nvinfer1::IRuntime, [586](#)
 - NvInferRuntime.h, [738](#)
- getLoop
 - nvinfer1::ILoopBoundaryLayer, [389](#)
- getLoopOutput
 - nvinfer1::ILoopOutputLayer, [391](#)
- getMaxBatchSize
 - nvinfer1::IBuilder, [150](#)
 - nvinfer1::ICudaEngine, [220](#)
- getMaxDLABatchSize
 - nvinfer1::IBuilder, [151](#)
- getMaxOutputSize
 - nvinfer1::IExecutionContext, [294](#)
- getMaxSeqLength
 - nvinfer1::IRNNv2Layer, [577](#)
- getMaxThreads
 - nvinfer1::IBuilder, [151](#)
 - nvinfer1::IRefitter, [556](#)
 - nvinfer1::IRuntime, [587](#)
- getMaxWorkspaceSize
 - nvinfer1::IBuilderConfig, [164](#)
- getMemoryPoolLimit
 - nvinfer1::IBuilderConfig, [165](#)
- getMessage
 - nvinfer1::IAssertionLayer, [141](#)
- getMinTimingIterations
 - nvinfer1::IBuilderConfig, [165](#)
- getMissing
 - nvinfer1::IRefitter, [556](#)
- getMissingWeights
 - nvinfer1::IRefitter, [557](#)
- getMode
 - nvinfer1::IGatherLayer, [342](#)
 - nvinfer1::IScaleLayer, [599](#)
 - nvinfer1::IScatterLayer, [604](#)
 - nvinfer1::ISliceLayer, [615](#)
- getModelDtype
 - nvonnxparser::IOnnxConfig, [461](#)
- getModelFileName
 - nvonnxparser::IOnnxConfig, [461](#)
- getName
 - nvinfer1::IAlgorithmContext, [133](#)
 - nvinfer1::ICudaEngine, [221](#)
 - nvinfer1::IExecutionContext, [295](#)
 - nvinfer1::IIfConditional, [358](#)
 - nvinfer1::ILayer, [377](#)
 - nvinfer1::ILoop, [387](#)
 - nvinfer1::INetworkDefinition, [438](#)
 - nvinfer1::ITensor, [626](#)
 - nvinfer1::safe::ICudaEngine, [244](#)
 - nvinfer1::safe::IExecutionContext, [319](#)
- getNbBindings

- nvinfer1::ICudaEngine, 221
 - nvinfer1::safe::ICudaEngine, 244
- getNbDLACores
 - nvinfer1::IBuilder, 151
 - nvinfer1::IRuntime, 587
- getNbElementWiseDims
 - nvinfer1::IGatherLayer, 342
- getNbErrors
 - nvinfer1::IErrorRecorder, 281
 - nvonnxparser::IParser, 481
- getNbGroups
 - nvinfer1::IConvolutionLayer, 199
 - nvinfer1::IDeconvolutionLayer, 254
 - nvinfer1::INormalizationLayer, 454
- getNbInputs
 - nvinfer1::IAlgorithmContext, 133
 - nvinfer1::ILayer, 377
 - nvinfer1::INetworkDefinition, 438
- getNbIOTensors
 - nvinfer1::ICudaEngine, 221
 - nvinfer1::safe::ICudaEngine, 245
- getNbLayers
 - nvinfer1::ICudaEngine, 222
 - nvinfer1::INetworkDefinition, 438
- getNbOptimizationProfiles
 - nvinfer1::IBuilderConfig, 166
 - nvinfer1::ICudaEngine, 222
- getNbOutputChannels
 - nvinfer1::IFullyConnectedLayer, 336
- getNbOutputMaps
 - nvinfer1::IConvolutionLayer, 199
 - nvinfer1::IDeconvolutionLayer, 255
- getNbOutputs
 - nvinfer1::IAlgorithmContext, 133
 - nvinfer1::ILayer, 377
 - nvinfer1::INetworkDefinition, 439
 - nvinfer1::IPluginV2, 506
- getNbPluginsToSerialize
 - nvinfer1::IBuilderConfig, 166
- getNbShapeValues
 - nvinfer1::IOptimizationProfile, 468
- getNearestRounding
 - nvinfer1::IResizeLayer, 565
- getNvOnnxParserVersion
 - NvOnnxParser.h, 780
- getNvtxVerbosity
 - nvinfer1::IExecutionContext, 295
- getOperation
 - nvinfer1::IElementWiseLayer, 271
 - nvinfer1::IFillLayer, 331
 - nvinfer1::IMatrixMultiplyLayer, 398
 - nvinfer1::IReduceLayer, 550
 - nvinfer1::IRNNv2Layer, 577
 - nvinfer1::ITopKLayer, 637
 - nvinfer1::IUnaryLayer, 646
- getOptimizationProfile
 - nvinfer1::IExecutionContext, 295
- getOutput
 - nvinfer1::ILayer, 377
 - nvinfer1::INetworkDefinition, 439
- getOutputAllocator
 - nvinfer1::IExecutionContext, 296
- getOutputDataType
 - nvinfer1::IPluginV2Ext, 524
- getOutputDimensions
 - nvinfer1::IPluginV2, 506
 - nvinfer1::IPluginV2DynamicExt, 517
 - nvinfer1::IResizeLayer, 565
- getOutputTensorAddress
 - nvinfer1::IExecutionContext, 296
 - nvinfer1::safe::IExecutionContext, 319
- getOutputType
 - nvinfer1::ILayer, 377
- getPadding
 - nvinfer1::IConvolutionLayer, 200
 - nvinfer1::IDeconvolutionLayer, 255
 - nvinfer1::IPoolingLayer, 533
- getPaddingMode
 - nvinfer1::IConvolutionLayer, 200
 - nvinfer1::IDeconvolutionLayer, 255
 - nvinfer1::IPoolingLayer, 534
- getPaddingNd
 - nvinfer1::IConvolutionLayer, 200
 - nvinfer1::IDeconvolutionLayer, 256
 - nvinfer1::IPoolingLayer, 534
- getPersistentCacheLimit
 - nvinfer1::IExecutionContext, 297
- getPlugin
 - nvinfer1::IPluginV2Layer, 530
- getPluginCreator
 - nvinfer1::IPluginRegistry, 498
- getPluginCreatorList
 - nvinfer1::IPluginRegistry, 499
- getPluginName
 - nvinfer1::IPluginCreator, 491
- getPluginNamespace
 - nvinfer1::IPluginCreator, 492
 - nvinfer1::IPluginV2, 507
- getPluginRegistry
 - nvinfer1::IBuilder, 152
 - nvinfer1::IRuntime, 587
 - NvInferRuntime.h, 738
- getPluginToSerialize
 - nvinfer1::IBuilderConfig, 166
- getPluginType
 - nvinfer1::IPluginV2, 507
- getPluginVersion
 - nvinfer1::IPluginCreator, 492

- nvinfer1::IPluginV2, [508](#)
- getPoolingType
 - nvinfer1::IPoolingLayer, [534](#)
- getPostPadding
 - nvinfer1::IConvolutionLayer, [201](#)
 - nvinfer1::IDeconvolutionLayer, [256](#)
 - nvinfer1::IPaddingLayer, [475](#)
 - nvinfer1::IPoolingLayer, [535](#)
- getPostPaddingNd
 - nvinfer1::IPaddingLayer, [475](#)
- getPower
 - nvinfer1::IScaleLayer, [599](#)
- getPrecision
 - nvinfer1::ILayer, [378](#)
- getPrePadding
 - nvinfer1::IConvolutionLayer, [201](#)
 - nvinfer1::IDeconvolutionLayer, [256](#)
 - nvinfer1::IPaddingLayer, [476](#)
 - nvinfer1::IPoolingLayer, [535](#)
- getPrePaddingNd
 - nvinfer1::IPaddingLayer, [476](#)
- getPreviewFeature
 - nvinfer1::IBuilderConfig, [167](#)
- getPrintLayerInfo
 - nvonnxparser::IOnnxConfig, [462](#)
- getProfileDimensions
 - nvinfer1::ICudaEngine, [222](#)
- getProfiler
 - nvinfer1::IExecutionContext, [297](#)
- getProfileShape
 - nvinfer1::ICudaEngine, [223](#)
- getProfileShapeValues
 - nvinfer1::ICudaEngine, [224](#)
- getProfileStream
 - nvinfer1::IBuilderConfig, [167](#)
- getProfilingVerbosity
 - nvinfer1::IBuilderConfig, [167](#)
 - nvinfer1::ICudaEngine, [224](#)
- getQuantile
 - nvinfer1::IInt8LegacyCalibrator, [369](#)
- getQuantizationFlag
 - nvinfer1::IBuilderConfig, [168](#)
- getQuantizationFlags
 - nvinfer1::IBuilderConfig, [168](#)
- getReduceAxes
 - nvinfer1::IReduceLayer, [550](#)
 - nvinfer1::ITopKLayer, [637](#)
- getRegressionCutoff
 - nvinfer1::IInt8LegacyCalibrator, [370](#)
- getReshapeDimensions
 - nvinfer1::IShuffleLayer, [609](#)
- getResizeMode
 - nvinfer1::IResizeLayer, [565](#)
- getReverse
 - nvinfer1::IIteratorLayer, [373](#)
- getSafePluginRegistry
 - nvinfer1::safe, [85](#)
- getSampleMode
 - nvinfer1::IGridSampleLayer, [350](#)
- getScale
 - nvinfer1::IScaleLayer, [599](#)
- getScales
 - nvinfer1::IResizeLayer, [565](#)
- getSecondTranspose
 - nvinfer1::IShuffleLayer, [610](#)
- getSelectorForSinglePixel
 - nvinfer1::IResizeLayer, [566](#)
- getSequenceAxis
 - nvinfer1::IReverseSequenceLayer, [572](#)
- getSequenceLengths
 - nvinfer1::IRNNv2Layer, [578](#)
- getSerializationSize
 - nvinfer1::IPluginV2, [508](#)
- getShapeBinding
 - nvinfer1::IExecutionContext, [297](#)
- getShapeValues
 - nvinfer1::IOptimizationProfile, [468](#)
- getShift
 - nvinfer1::IScaleLayer, [600](#)
- getSize
 - nvinfer1::ISliceLayer, [615](#)
- getStart
 - nvinfer1::ISliceLayer, [616](#)
- getStride
 - nvinfer1::IConvolutionLayer, [201](#)
 - nvinfer1::IDeconvolutionLayer, [256](#)
 - nvinfer1::IPoolingLayer, [535](#)
 - nvinfer1::ISliceLayer, [616](#)
- getStrideNd
 - nvinfer1::IConvolutionLayer, [201](#)
 - nvinfer1::IDeconvolutionLayer, [257](#)
 - nvinfer1::IPoolingLayer, [535](#)
- getStrides
 - nvinfer1::IAlgorithmIOInfo, [135](#)
 - nvinfer1::IExecutionContext, [298](#)
 - nvinfer1::safe::IExecutionContext, [320](#)
- getTactic
 - nvinfer1::IAlgorithmVariant, [140](#)
- getTacticSources
 - nvinfer1::IBuilderConfig, [168](#)
 - nvinfer1::ICudaEngine, [225](#)
- getTempfileControlFlags
 - nvinfer1::IRuntime, [587](#)
- getTemporaryDirectory
 - nvinfer1::IRuntime, [588](#)
- getTemporaryStorageAllocator
 - nvinfer1::IExecutionContext, [298](#)
- getTensorAddress

- nvinfer1::IExecutionContext, 299
- getTensorBytesPerComponent
 - nvinfer1::ICudaEngine, 225
 - nvinfer1::safe::ICudaEngine, 245
- getTensorComponentsPerElement
 - nvinfer1::ICudaEngine, 226
 - nvinfer1::safe::ICudaEngine, 246
- getTensorDataType
 - nvinfer1::ICudaEngine, 226
 - nvinfer1::safe::ICudaEngine, 247
- getTensorFormat
 - nvinfer1::IAlgorithmIOInfo, 136
 - nvinfer1::ICudaEngine, 227
 - nvinfer1::safe::ICudaEngine, 247
- getTensorFormatDesc
 - nvinfer1::ICudaEngine, 227
- getTensorIOMode
 - nvinfer1::ICudaEngine, 228
 - nvinfer1::safe::ICudaEngine, 248
- getTensorLocation
 - nvinfer1::ICudaEngine, 228
- getTensorRTVersion
 - nvinfer1::IPluginCreator, 493
 - nvinfer1::IPluginV2, 509
 - nvinfer1::IPluginV2DynamicExt, 517
 - nvinfer1::IPluginV2Ext, 525
 - nvinfer1::IPluginV2IOExt, 528
- getTensorShape
 - nvinfer1::ICudaEngine, 229
 - nvinfer1::IExecutionContext, 299
 - nvinfer1::safe::ICudaEngine, 249
- getTensorStrides
 - nvinfer1::IExecutionContext, 300
 - nvinfer1::safe::IExecutionContext, 320
- getTensorsWithDynamicRange
 - nvinfer1::IRefitter, 557
- getTensorVectorizedDim
 - nvinfer1::ICudaEngine, 229
 - nvinfer1::safe::ICudaEngine, 249
- getTextFileName
 - nvonnxparser::IOnnxConfig, 462
- getTimingCache
 - nvinfer1::IBuilderConfig, 169
- getTimingMSec
 - nvinfer1::IAlgorithm, 131
- getTopKBoxLimit
 - nvinfer1::INMSLayer, 446
- getToType
 - nvinfer1::ICastLayer, 186
- getTripLimit
 - nvinfer1::ITripLimitLayer, 640
- getType
 - nvinfer1::ILayer, 378
 - nvinfer1::ITensor, 626
- getUffRequiredVersionMajor
 - nvuffparser::IUffParser, 642
 - getUffRequiredVersionMinor
 - nvuffparser::IUffParser, 642
 - getUffRequiredVersionPatch
 - nvuffparser::IUffParser, 643
 - getVerbosityLevel
 - nvonnxparser::IOnnxConfig, 462
 - getWeights
 - nvinfer1::IConstantLayer, 194
 - getWeightsForGate
 - nvinfer1::IRNNv2Layer, 578
 - getWindowSize
 - nvinfer1::ILRNLayer, 394
 - nvinfer1::IPoolingLayer, 536
 - getWindowSizeNd
 - nvinfer1::IPoolingLayer, 536
 - getWorkspaceSize
 - nvinfer1::IAlgorithm, 131
 - nvinfer1::IPluginV2, 509
 - nvinfer1::IPluginV2DynamicExt, 517
 - getZeroIsPlaceholder
 - nvinfer1::IShuffleLayer, 610
- group
 - nvinfer1::plugin::softmaxTree, 664
- groupOffset
 - nvinfer1::plugin::softmaxTree, 664
- groups
 - nvinfer1::plugin::softmaxTree, 664
- groupSize
 - nvinfer1::plugin::softmaxTree, 664
- H
 - nvinfer1::plugin::GridAnchorParameters, 124
- h
 - nvinfer1::DimsHW, 104
- HardwareCompatibilityLevel
 - nvinfer1, 49
- hasExplicitPrecision
 - nvinfer1::INetworkDefinition, 440
- hasImplicitBatchDimension
 - nvinfer1::ICudaEngine, 230
 - nvinfer1::INetworkDefinition, 440
- hasOverflowed
 - nvinfer1::IErrorRecorder, 282
- IConsistencyChecker
 - nvinfer1::consistency::IConsistencyChecker, 191
- ICudaEngine
 - nvinfer1::safe::ICudaEngine, 235
- IErrorRecorder
 - nvinfer1::IErrorRecorder, 278
- IExecutionContext
 - nvinfer1::safe::IExecutionContext, 314, 315
- IGpuAllocator

- nvinfer1::IGpuAllocator, [344](#)
- ILogger
 - nvinfer1::ILogger, [384](#)
- imgH
 - nvinfer1::plugin::PriorBoxParameters, [657](#)
- imgW
 - nvinfer1::plugin::PriorBoxParameters, [658](#)
- incRefCount
 - nvinfer1::IErrorRecorder, [282](#)
- inferShapes
 - nvinfer1::IExecutionContext, [301](#)
- initialize
 - nvinfer1::IPluginV2, [510](#)
- initLibNvInferPlugins
 - NvInferPlugin.h, [731](#)
- INoCopy
 - nvinfer1::INoCopy, [450](#)
- INonZero, [451](#)
- inputOrder
 - nvinfer1::plugin::DetectionOutputParameters, [95](#)
- InterpolationMode
 - nvinfer1, [50](#)
- iouThreshold
 - nvinfer1::plugin::NMSPParameters, [648](#)
 - nvinfer1::plugin::RPROIParams, [662](#)
- IPluginChecker
 - nvinfer1::consistency::IPluginChecker, [487](#), [488](#)
- IPluginCreator
 - nvinfer1::IPluginCreator, [490](#)
- IPluginV2Ext
 - nvinfer1::IPluginV2Ext, [520](#)
- IRuntime
 - nvinfer1::safe::IRuntime, [593](#), [594](#)
- isBatchAgnostic
 - nvinfer1::plugin::DetectionOutputParameters, [95](#)
- isConstant
 - nvinfer1::IDimensionExpr, [267](#)
- isDeviceTypeSet
 - nvinfer1::IBuilderConfig, [169](#)
- isExecutionBinding
 - nvinfer1::ICudaEngine, [230](#)
- isExecutionTensor
 - nvinfer1::ITensor, [627](#)
- isNetworkInput
 - nvinfer1::ITensor, [627](#)
- isNetworkOutput
 - nvinfer1::ITensor, [627](#)
- isNetworkSupported
 - nvinfer1::IBuilder, [152](#)
- isNormalized
 - nvinfer1::plugin::DetectionOutputParameters, [95](#)
 - nvinfer1::plugin::NMSPParameters, [648](#)
- isOutputBroadcastAcrossBatch
 - nvinfer1::IPluginV2Ext, [525](#)
- isParentSearchEnabled
 - nvinfer1::IPluginRegistry, [499](#)
- isPluginV2
 - nvcaffeparser1::IPluginFactoryV2, [495](#)
- isRefittable
 - nvinfer1::ICudaEngine, [230](#)
- isShapeBinding
 - nvinfer1::ICudaEngine, [231](#)
- isShapeInferenceIO
 - nvinfer1::ICudaEngine, [231](#)
- isShapeTensor
 - nvinfer1::ITensor, [628](#)
- isValid
 - nvinfer1::IOptimizationProfile, [469](#)
- kABS
 - nvinfer1, [73](#)
- kACOS
 - nvinfer1, [73](#)
- kACOSH
 - nvinfer1, [73](#)
- kACTIVATION
 - nvinfer1, [51](#)
- kALIGN_CORNERS
 - nvinfer1, [61](#)
- kALLOW_IN_MEMORY_FILES
 - nvinfer1, [68](#)
- kALLOW_TEMPORARY_FILES
 - nvinfer1, [68](#)
- kAMPERE_PLUS
 - nvinfer1, [50](#)
- kAND
 - nvinfer1, [46](#)
- kANY
 - nvinfer1, [73](#)
- kASIN
 - nvinfer1, [73](#)
- kASINH
 - nvinfer1, [73](#)
- kASSERTION
 - nvinfer1, [52](#)
- kASYMMETRIC
 - nvinfer1, [61](#)
- kATAN
 - nvinfer1, [73](#)
- kATANH
 - nvinfer1, [73](#)
- kAVERAGE
 - nvinfer1, [58](#)
- kAVG
 - nvinfer1, [61](#)
- kBIAS
 - nvinfer1, [73](#)
- kBIDIRECTION

- nvinfer1, [63](#)
- kBOOL
 - nvinfer1, [44](#)
- kCAFFE_ROUND_DOWN
 - nvinfer1, [57](#)
- kCAFFE_ROUND_UP
 - nvinfer1, [57](#)
- kCALIBRATE_BEFORE_FUSION
 - nvinfer1, [60](#)
- kCAST
 - nvinfer1, [52](#)
- kCDHW32
 - nvinfer1, [70](#)
- kCEIL
 - nvinfer1, [62](#), [73](#)
- kCEIL_DIV
 - nvinfer1, [46](#)
- kCELL
 - nvinfer1, [63](#)
- kCENTER_SIZES
 - nvinfer1, [42](#)
- kCHANNEL
 - nvinfer1, [67](#)
- kCHAR
 - nvinfer1, [58](#)
 - nvuffparser, [91](#)
- kCHW16
 - nvinfer1, [70](#)
- kCHW2
 - nvinfer1, [70](#)
- kCHW32
 - nvinfer1, [70](#)
- kCHW4
 - nvinfer1, [70](#)
- kCLAMP
 - nvinfer1, [66](#)
- kCLIP
 - nvinfer1, [41](#)
- kCONCATENATE
 - nvinfer1, [52](#)
- kCONCATENATION
 - nvinfer1, [51](#)
- kCONDITION
 - nvinfer1, [52](#)
- kCONDITIONAL_INPUT
 - nvinfer1, [52](#)
- kCONDITIONAL_OUTPUT
 - nvinfer1, [52](#)
- kCONSTANT
 - nvinfer1, [51](#), [73](#)
- kCONVOLUTION
 - nvinfer1, [51](#)
- kCORNER_PAIRS
 - nvinfer1, [42](#)
- kCOS
 - nvinfer1, [73](#)
- kCOSH
 - nvinfer1, [73](#)
- kCOUNT
 - nvinfer1, [72](#)
- kCUBIC
 - nvinfer1, [50](#)
- kCUBLAS
 - nvinfer1, [67](#)
- kCUBLAS_LT
 - nvinfer1, [67](#)
- kCUDNN
 - nvinfer1, [68](#)
- kDATATYPE
 - nvuffparser, [91](#)
- kDEBUG
 - nvinfer1, [42](#)
- kDECONVOLUTION
 - nvinfer1, [51](#)
- kDEFAULT
 - nvinfer1, [47](#), [49](#), [60](#), [66](#)
- kDEQUANTIZE
 - nvinfer1, [52](#)
- kDETAILED
 - nvinfer1, [60](#)
- kDEVICE
 - nvinfer1, [71](#)
- kDHW_C
 - nvinfer1, [71](#)
- kDHW_C8
 - nvinfer1, [70](#)
- kDIMS
 - nvinfer1, [58](#)
 - nvuffparser, [91](#)
- kDIRECT_IO
 - nvinfer1, [43](#)
- kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805
 - nvinfer1, [59](#)
- kDISABLE_TIMING_CACHE
 - nvinfer1, [43](#)
- kDIV
 - nvinfer1, [46](#)
- kDLA
 - nvinfer1, [45](#)
- kDLA_GLOBAL_DRAM
 - nvinfer1, [53](#)
- kDLA_HWC4
 - nvinfer1, [71](#)
- kDLA_LINEAR
 - nvinfer1, [70](#)
- kDLA_LOCAL_DRAM
 - nvinfer1, [53](#)
- kDLA_MANAGED_SRAM

- nvinfer1, [53](#)
- kDLA_STANDALONE
 - nvinfer1, [47](#)
- kEDGE_MASK_CONVOLUTIONS
 - nvinfer1, [68](#)
- keepTopK
 - nvinfer1::plugin::DetectionOutputParameters, [95](#)
 - nvinfer1::plugin::NMSParameters, [648](#)
- KEINSUM
 - nvinfer1, [52](#)
- KELEMENT
 - nvinfer1, [49](#), [67](#)
- KELEMENTWISE
 - nvinfer1, [51](#), [67](#)
- KELU
 - nvinfer1, [41](#)
- KENABLE_MULTI_STREAM_ENGINE_0806
 - nvinfer1, [59](#)
- KENABLE_TACTIC_HEURISTIC
 - nvinfer1, [43](#)
- KENTROPY_CALIBRATION
 - nvinfer1, [44](#)
- KENTROPY_CALIBRATION_2
 - nvinfer1, [44](#)
- KEQUAL
 - nvinfer1, [46](#)
- KERF
 - nvinfer1, [73](#)
- kERROR
 - nvinfer1::ILogger, [383](#)
- kEXCLUDE_LEAN_RUNTIME
 - nvinfer1, [44](#)
- kEXP
 - nvinfer1, [73](#)
- kEXPLICIT_BATCH
 - nvinfer1, [54](#)
- kEXPLICIT_PRECISION
 - nvinfer1, [54](#)
- kEXPLICIT_ROUND_DOWN
 - nvinfer1, [57](#)
- kEXPLICIT_ROUND_UP
 - nvinfer1, [57](#)
- kFAILED_ALLOCATION
 - nvinfer1, [48](#)
- kFAILED_COMPUTATION
 - nvinfer1, [48](#)
- kFAILED_EXECUTION
 - nvinfer1, [48](#)
- kFAILED_INITIALIZATION
 - nvinfer1, [48](#)
- kFASTER_DYNAMIC_SHAPES_0805
 - nvinfer1, [59](#)
- kFILL
 - nvinfer1, [51](#), [66](#)
- kFLOAT
 - nvinfer1, [44](#)
 - nvuffparser, [91](#)
- kFLOAT16
 - nvinfer1, [58](#)
- kFLOAT32
 - nvinfer1, [58](#)
- kFLOAT64
 - nvinfer1, [58](#)
- kFLOOR
 - nvinfer1, [62](#), [73](#)
- kFLOOR_DIV
 - nvinfer1, [46](#)
- kFORGET
 - nvinfer1, [63](#)
- kFORMAT_COMBINATION_LIMIT
 - nvinfer1::IPluginV2DynamicExt, [519](#)
- kFORMULA
 - nvinfer1, [62](#)
- kFP16
 - nvinfer1, [42](#)
- kFP8
 - nvinfer1, [44](#), [45](#)
- kFULLY_CONNECTED
 - nvinfer1, [51](#)
- kGATHER
 - nvinfer1, [51](#)
- kGPU
 - nvinfer1, [45](#)
- kGPU_FALLBACK
 - nvinfer1, [42](#)
- kGREATER
 - nvinfer1, [46](#)
- kGRID_SAMPLE
 - nvinfer1, [52](#)
- kGRU
 - nvinfer1, [66](#)
- kHALF
 - nvinfer1, [44](#)
- kHALF_DOWN
 - nvinfer1, [62](#)
- kHALF_PIXEL
 - nvinfer1, [61](#)
- kHALF_UP
 - nvinfer1, [62](#)
- kHARD_SIGMOID
 - nvinfer1, [41](#)
- kHIDDEN
 - nvinfer1, [63](#)
- kHOST
 - nvinfer1, [71](#)
- kHWC
 - nvinfer1, [70](#)
- kHWC16

- nvinfer1, [71](#)
- kHWC8
 - nvinfer1, [70](#)
- kIDENTITY
 - nvinfer1, [51](#)
- kINFO
 - nvinfer1::ILogger, [383](#)
- kINPUT
 - nvinfer1, [63](#), [71](#)
- kINT16
 - nvinfer1, [58](#)
- kINT32
 - nvinfer1, [44](#), [58](#)
 - nvuffparser, [91](#)
- kINT8
 - nvinfer1, [42](#), [44](#), [58](#)
- kINTERNAL_ERROR
 - nvinfer1, [48](#)
 - nvinfer1::ILogger, [383](#)
 - nvonnxparser, [89](#)
- kINVALID_ARGUMENT
 - nvinfer1, [48](#)
- kINVALID_CONFIG
 - nvinfer1, [48](#)
- kINVALID_GRAPH
 - nvonnxparser, [89](#)
- kINVALID_NODE
 - nvonnxparser, [89](#)
- kINVALID_STATE
 - nvinfer1, [48](#)
- kINVALID_VALUE
 - nvonnxparser, [89](#)
- kISINF
 - nvinfer1, [73](#)
- kITERATOR
 - nvinfer1, [51](#)
- kJIT_CONVOLUTIONS
 - nvinfer1, [68](#)
- kJSON
 - nvinfer1, [50](#)
- kKERNEL
 - nvinfer1, [73](#)
- kLAST_VALUE
 - nvinfer1, [52](#)
- kLAYER_NAMES_ONLY
 - nvinfer1, [60](#)
- kLEAKY_RELU
 - nvinfer1, [41](#)
- kLEGACY_CALIBRATION
 - nvinfer1, [44](#)
- kLESS
 - nvinfer1, [46](#)
- kLINEAR
 - nvinfer1, [50](#), [64](#), [70](#)
- kLinspace
 - nvinfer1, [49](#)
- kLOG
 - nvinfer1, [73](#)
- kLOOP_OUTPUT
 - nvinfer1, [51](#)
- kLRN
 - nvinfer1, [51](#)
- kLSTM
 - nvinfer1, [66](#)
- kMATRIX_MULTIPLY
 - nvinfer1, [51](#)
- kMAX
 - nvinfer1, [45](#), [46](#), [54](#), [58](#), [61](#), [72](#)
- kMAX_AVERAGE_BLEND
 - nvinfer1, [58](#)
- kMAX_DESC_LENGTH
 - nvinfer1::IErrorRecorder, [284](#)
- kMEM_ALLOC_FAILED
 - nvonnxparser, [89](#)
- kMIN
 - nvinfer1, [46](#), [54](#), [61](#), [72](#)
- kMINMAX_CALIBRATION
 - nvinfer1, [44](#)
- kMODEL_DESERIALIZE_FAILED
 - nvonnxparser, [89](#)
- kNC
 - nvuffparser, [91](#)
- kNCHW
 - nvuffparser, [91](#)
- kND
 - nvinfer1, [49](#), [67](#)
- kNEAREST
 - nvinfer1, [50](#)
- kNEG
 - nvinfer1, [73](#)
- kNHWC
 - nvuffparser, [91](#)
- kNMS
 - nvinfer1, [52](#)
- kNON_ZERO
 - nvinfer1, [52](#)
- kNONE
 - nvinfer1, [50](#), [53](#), [60](#), [71](#)
- kNORMALIZATION
 - nvinfer1, [52](#)
- kNOT
 - nvinfer1, [73](#)
- kOBEY_PRECISION_CONSTRAINTS
 - nvinfer1, [43](#)
- kONE_HOT
 - nvinfer1, [52](#)
- kONELINE
 - nvinfer1, [50](#)

- kOPT
 - nvinfer1, [54](#)
- kOR
 - nvinfer1, [46](#)
- kOUTPUT
 - nvinfer1, [63](#), [71](#)
- kPADDING
 - nvinfer1, [51](#)
- kPARAMETRIC_RELU
 - nvinfer1, [51](#)
- kPLUGIN
 - nvinfer1, [51](#)
- kPLUGIN_V2
 - nvinfer1, [51](#)
- kPOOLING
 - nvinfer1, [51](#)
- kPOW
 - nvinfer1, [46](#)
- kPREFER_PRECISION_CONSTRAINTS
 - nvinfer1, [43](#)
- kPROD
 - nvinfer1, [45](#), [46](#), [61](#)
- kPROFILE_SHARING_0806
 - nvinfer1, [59](#)
- kQUANTIZE
 - nvinfer1, [52](#)
- kRAGGED_SOFTMAX
 - nvinfer1, [51](#)
- kRANDOM_NORMAL
 - nvinfer1, [49](#)
- kRANDOM_UNIFORM
 - nvinfer1, [49](#)
- kRECIP
 - nvinfer1, [73](#)
- kRECURRENCE
 - nvinfer1, [51](#)
- kREDUCE
 - nvinfer1, [51](#)
- kREFIT
 - nvinfer1, [43](#)
- kREFLECT
 - nvinfer1, [66](#)
- kREJECT_EMPTY_ALGORITHMS
 - nvinfer1, [43](#)
- kRELU
 - nvinfer1, [41](#), [66](#)
- kRESET
 - nvinfer1, [63](#)
- kRESIZABLE
 - nvinfer1, [42](#)
- kRESIZE
 - nvinfer1, [51](#)
- kREVERSE
 - nvinfer1, [52](#)
- kREVERSE_SEQUENCE
 - nvinfer1, [52](#)
- kRNN_V2
 - nvinfer1, [51](#)
- kROUND
 - nvinfer1, [73](#)
- kSAFE_DLA
 - nvinfer1, [47](#)
- kSAFE_GPU
 - nvinfer1, [47](#)
- kSAFETY
 - nvinfer1, [47](#)
- kSAFETY_SCOPE
 - nvinfer1, [43](#)
- kSAME_LOWER
 - nvinfer1, [57](#)
- kSAME_UPPER
 - nvinfer1, [57](#)
- kSCALE
 - nvinfer1, [51](#), [73](#)
- kSCALED_TANH
 - nvinfer1, [41](#)
- kSCATTER
 - nvinfer1, [52](#)
- kSELECT
 - nvinfer1, [51](#)
- kSELU
 - nvinfer1, [41](#)
- kSHAPE
 - nvinfer1, [51](#)
- kSHIFT
 - nvinfer1, [73](#)
- kSHUFFLE
 - nvinfer1, [51](#)
- kSIGMOID
 - nvinfer1, [41](#)
- kSIGN
 - nvinfer1, [73](#)
- kSIN
 - nvinfer1, [73](#)
- kSINH
 - nvinfer1, [73](#)
- kSKIP
 - nvinfer1, [64](#)
- kSLICE
 - nvinfer1, [51](#)
- kSOFTMAX
 - nvinfer1, [51](#)
- kSOFTPLUS
 - nvinfer1, [41](#)
- kSOFTSIGN
 - nvinfer1, [41](#)
- kSPARSE_WEIGHTS
 - nvinfer1, [43](#)

- kSQRT
 - nvinfer1, [73](#)
- kSTANDARD
 - nvinfer1, [47](#)
- kSTRICT_BOUNDS
 - nvinfer1, [66](#)
- kSTRICT_TYPES
 - nvinfer1, [43](#)
- kSUB
 - nvinfer1, [46](#)
- kSUCCESS
 - nvinfer1, [47](#)
 - nvonnxparser, [89](#)
- kSUM
 - nvinfer1, [45](#), [46](#), [61](#)
- kTACTIC_DRAM
 - nvinfer1, [53](#)
- kTAN
 - nvinfer1, [73](#)
- kTANH
 - nvinfer1, [41](#), [66](#)
- kTF32
 - nvinfer1, [43](#)
- kTHRESHOLDED_RELU
 - nvinfer1, [41](#)
- kTOPK
 - nvinfer1, [51](#)
- kTRANPOSE
 - nvinfer1, [53](#)
- kTRIP_LIMIT
 - nvinfer1, [51](#)
- kUINT8
 - nvinfer1, [45](#)
- kUNARY
 - nvinfer1, [51](#)
- kUNIDIRECTION
 - nvinfer1, [63](#)
- kUNIFORM
 - nvinfer1, [67](#)
- kUNKNOWN
 - nvinfer1, [58](#)
 - nvuffparser, [91](#)
- kUNSPECIFIED_ERROR
 - nvinfer1, [47](#)
- kUNSUPPORTED_GRAPH
 - nvonnxparser, [89](#)
- kUNSUPPORTED_NODE
 - nvonnxparser, [89](#)
- kUNSUPPORTED_STATE
 - nvinfer1, [48](#)
- kUPDATE
 - nvinfer1, [63](#)
- kUPPER
 - nvinfer1, [62](#)
- kV2
 - nvinfer1, [58](#)
- kV2_DYNAMICEXT
 - nvinfer1, [58](#)
- kV2_EXT
 - nvinfer1, [58](#)
- kV2_IOEXT
 - nvinfer1, [58](#)
- kVALUE
 - nvinfer1::impl::EnumMaxImpl< ActivationType >, [107](#)
 - nvinfer1::impl::EnumMaxImpl< AllocatorFlag >, [108](#)
 - nvinfer1::impl::EnumMaxImpl< DataType >, [109](#)
 - nvinfer1::impl::EnumMaxImpl< ElementWiseOperation >, [109](#)
 - nvinfer1::impl::EnumMaxImpl< EngineCapability >, [110](#)
 - nvinfer1::impl::EnumMaxImpl< ErrorCode >, [111](#)
 - nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel >, [112](#)
 - nvinfer1::impl::EnumMaxImpl< ILogger::Severity >, [113](#)
 - nvinfer1::impl::EnumMaxImpl< InterpolationMode >, [113](#)
 - nvinfer1::impl::EnumMaxImpl< PaddingMode >, [114](#)
 - nvinfer1::impl::EnumMaxImpl< PoolingType >, [115](#)
 - nvinfer1::impl::EnumMaxImpl< PreviewFeature >, [115](#)
 - nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation >, [116](#)
 - nvinfer1::impl::EnumMaxImpl< ResizeRoundMode >, [117](#)
 - nvinfer1::impl::EnumMaxImpl< ResizeSelector >, [117](#)
 - nvinfer1::impl::EnumMaxImpl< TensorFormat >, [118](#)
 - nvinfer1::impl::EnumMaxImpl< TensorIOMode >, [119](#)
 - nvinfer1::impl::EnumMaxImpl< TensorLocation >, [120](#)
- kVECTOR
 - nvinfer1, [53](#)
- kVERBOSE
 - nvinfer1, [60](#)
 - nvinfer1::ILogger, [383](#)
- kVERSION_COMPATIBLE
 - nvinfer1, [43](#)
- kWARNING
 - nvinfer1::ILogger, [383](#)
- kWHILE
 - nvinfer1, [72](#)

- kWORKSPACE
 - nvinfer1, [53](#)
- kWRAP
 - nvinfer1, [66](#)
- kXOR
 - nvinfer1, [46](#)
- LayerInformationFormat
 - nvinfer1, [50](#)
- LayerType
 - nvinfer1, [50](#)
- leaf
 - nvinfer1::plugin::softmaxTree, [665](#)
- length
 - nvinfer1::PluginField, [651](#)
 - nvuffparser::FieldMap, [122](#)
- line
 - nvonnxparser::IParserError, [486](#)
- loadLibrary
 - nvinfer1::IPluginRegistry, [499](#)
- loadRuntime
 - nvinfer1::IRuntime, [588](#)
- log
 - nvinfer1::ILogger, [384](#)
- LoopOutput
 - nvinfer1, [52](#)
- markOutput
 - nvinfer1::INetworkDefinition, [440](#)
- markOutputForShapes
 - nvinfer1::INetworkDefinition, [441](#)
- MatrixOperation
 - nvinfer1, [52](#)
- max
 - nvinfer1::DynamicPluginTensorDesc, [106](#)
- MAX_DIMS
 - nvinfer1::Dims32, [100](#)
- maxSize
 - nvinfer1::plugin::GridAnchorParameters, [124](#)
 - nvinfer1::plugin::PriorBoxParameters, [658](#)
- mBoundary
 - nvinfer1::IIfConditionalBoundaryLayer, [360](#)
 - nvinfer1::ILoopBoundaryLayer, [389](#)
- MemoryPoolType
 - nvinfer1, [53](#)
- mImpl
 - nvinfer1::consistency::IConsistencyChecker, [192](#)
 - nvinfer1::IActivationLayer, [128](#)
 - nvinfer1::IAlgorithm, [131](#)
 - nvinfer1::IAlgorithmContext, [134](#)
 - nvinfer1::IAlgorithmIOInfo, [136](#)
 - nvinfer1::IAlgorithmVariant, [140](#)
 - nvinfer1::IAssertionLayer, [142](#)
 - nvinfer1::IBuilder, [155](#)
 - nvinfer1::IBuilderConfig, [179](#)
 - nvinfer1::ICastLayer, [186](#)
 - nvinfer1::IConcatenationLayer, [188](#)
 - nvinfer1::IConditionLayer, [190](#)
 - nvinfer1::IConstantLayer, [195](#)
 - nvinfer1::IConvolutionLayer, [208](#)
 - nvinfer1::ICudaEngine, [233](#)
 - nvinfer1::IDeconvolutionLayer, [263](#)
 - nvinfer1::IDequantizeLayer, [265](#)
 - nvinfer1::IDimensionExpr, [267](#)
 - nvinfer1::IEinsumLayer, [270](#)
 - nvinfer1::IElementWiseLayer, [272](#)
 - nvinfer1::IEngineInspector, [277](#)
 - nvinfer1::IExecutionContext, [313](#)
 - nvinfer1::IExprBuilder, [328](#)
 - nvinfer1::IFillLayer, [334](#)
 - nvinfer1::IFullyConnectedLayer, [338](#)
 - nvinfer1::IGatherLayer, [344](#)
 - nvinfer1::IGridSampleLayer, [351](#)
 - nvinfer1::IHostMemory, [354](#)
 - nvinfer1::IIdentityLayer, [355](#)
 - nvinfer1::IIfConditional, [359](#)
 - nvinfer1::IIfConditionalInputLayer, [361](#)
 - nvinfer1::IIfConditionalOutputLayer, [363](#)
 - nvinfer1::IIteratorLayer, [374](#)
 - nvinfer1::ILoop, [388](#)
 - nvinfer1::ILoopOutputLayer, [392](#)
 - nvinfer1::ILRNLayer, [396](#)
 - nvinfer1::IMatrixMultiplyLayer, [399](#)
 - nvinfer1::INetworkDefinition, [444](#)
 - nvinfer1::INMSLayer, [448](#)
 - nvinfer1::INonZeroLayer, [452](#)
 - nvinfer1::INormalizationLayer, [456](#)
 - nvinfer1::IOneHotLayer, [458](#)
 - nvinfer1::IOptimizationProfile, [471](#)
 - nvinfer1::IPaddingLayer, [478](#)
 - nvinfer1::IParametricReLULayer, [479](#)
 - nvinfer1::IPluginV2Layer, [531](#)
 - nvinfer1::IPoolingLayer, [541](#)
 - nvinfer1::IQuantizeLayer, [545](#)
 - nvinfer1::IRaggedSoftMaxLayer, [547](#)
 - nvinfer1::IRecurrenceLayer, [548](#)
 - nvinfer1::IReduceLayer, [552](#)
 - nvinfer1::IRefitter, [561](#)
 - nvinfer1::IResizeLayer, [570](#)
 - nvinfer1::IReverseSequenceLayer, [573](#)
 - nvinfer1::IRNNv2Layer, [582](#)
 - nvinfer1::IRuntime, [592](#)
 - nvinfer1::IScaleLayer, [602](#)
 - nvinfer1::IScatterLayer, [605](#)
 - nvinfer1::ISelectLayer, [606](#)
 - nvinfer1::IShapeLayer, [607](#)
 - nvinfer1::IShuffleLayer, [613](#)
 - nvinfer1::ISliceLayer, [619](#)
 - nvinfer1::ISoftMaxLayer, [621](#)

- `nvinfer1::ITensor`, 632
 - `nvinfer1::ITimingCache`, 635
 - `nvinfer1::ITopKLayer`, 639
 - `nvinfer1::ITripLimitLayer`, 640
 - `nvinfer1::IUnaryLayer`, 647
- `min`
 - `nvinfer1::DynamicPluginTensorDesc`, 106
- `minBoxSize`
 - `nvinfer1::plugin::RPROIParams`, 662
- `minSize`
 - `nvinfer1::plugin::GridAnchorParameters`, 124
 - `nvinfer1::plugin::PriorBoxParameters`, 658
- `mLayer`
 - `nvinfer1::ILayer`, 382
- `n`
 - `nvinfer1::plugin::softmaxTree`, 665
- `name`
 - `nvinfer1::plugin::softmaxTree`, 665
 - `nvinfer1::PluginField`, 651
 - `nvuffparser::FieldMap`, 122
- `nbDims`
 - `nvinfer1::Dims32`, 100
 - `nvinfer1::DimsExprs`, 102
- `nbFields`
 - `nvinfer1::PluginFieldCollection`, 652
 - `nvuffparser::FieldCollection`, 120
- `nbInfErrors`
 - `nvinfer1::safe::FloatingPointErrorInformation`, 123
- `nbNanErrors`
 - `nvinfer1::safe::FloatingPointErrorInformation`, 123
- `NetworkDefinitionCreationFlag`
 - `nvinfer1`, 54
- `NetworkDefinitionCreationFlags`
 - `nvinfer1`, 39
- `nmsMaxOut`
 - `nvinfer1::plugin::RPROIParams`, 663
- `nmsThreshold`
 - `nvinfer1::plugin::DetectionOutputParameters`, 95
- `node`
 - `nvonnxparser::IParserError`, 486
- `notifyShape`
 - `nvinfer1::IOutputAllocator`, 473
- `num`
 - `nvinfer1::plugin::RegionParameters`, 661
- `numAspectRatios`
 - `nvinfer1::plugin::GridAnchorParameters`, 124
 - `nvinfer1::plugin::PriorBoxParameters`, 658
- `numClasses`
 - `nvinfer1::plugin::DetectionOutputParameters`, 95
 - `nvinfer1::plugin::NMSParameters`, 648
- `numMaxSize`
 - `nvinfer1::plugin::PriorBoxParameters`, 658
- `numMinSize`
 - `nvinfer1::plugin::PriorBoxParameters`, 658
- `NV_ONNX_PARSER_MAJOR`
 - `NvOnnxParser.h`, 779
- `NV_ONNX_PARSER_MINOR`
 - `NvOnnxParser.h`, 779
- `NV_ONNX_PARSER_PATCH`
 - `NvOnnxParser.h`, 780
- `NV_TENSORRT_BUILD`
 - `NvInferVersion.h`, 770
- `NV_TENSORRT_LWS_MAJOR`
 - `NvInferVersion.h`, 770
- `NV_TENSORRT_LWS_MINOR`
 - `NvInferVersion.h`, 771
- `NV_TENSORRT_LWS_PATCH`
 - `NvInferVersion.h`, 771
- `NV_TENSORRT_MAJOR`
 - `NvInferVersion.h`, 771
- `NV_TENSORRT_MINOR`
 - `NvInferVersion.h`, 771
- `NV_TENSORRT_PATCH`
 - `NvInferVersion.h`, 771
- `NV_TENSORRT_SONAME_MAJOR`
 - `NvInferVersion.h`, 771
- `NV_TENSORRT_SONAME_MINOR`
 - `NvInferVersion.h`, 772
- `NV_TENSORRT_SONAME_PATCH`
 - `NvInferVersion.h`, 772
- `NV_TENSORRT_VERSION`
 - `NvInferRuntimeCommon.h`, 756
- `NvCaffeParser.h`, 667, 668
- `nvcaffeparser1`, 27
 - `createCaffeParser`, 27
 - `shutdownProtobufLibrary`, 28
- `nvcaffeparser1::IBinaryProtoBlob`, 142
 - `~IBinaryProtoBlob`, 143
 - `destroy`, 143
 - `getData`, 143
 - `getDataType`, 144
 - `getDimensions`, 144
- `nvcaffeparser1::IBlobNameToTensor`, 144
 - `~IBlobNameToTensor`, 145
 - `find`, 145
- `nvcaffeparser1::ICaffeParser`, 180
 - `~ICaffeParser`, 180
 - `destroy`, 181
 - `getErrorRecorder`, 181
 - `parse`, 181
 - `parseBinaryProto`, 182
 - `parseBuffers`, 182
 - `setErrorRecorder`, 183
 - `setPluginFactoryV2`, 184
 - `setPluginNamespace`, 184
 - `setProtobufBufferSize`, 184
- `nvcaffeparser1::IPluginFactoryV2`, 494

- ~IPluginFactoryV2, 494
- createPlugin, 494
- isPluginV2, 495
- NvInfer.h, 669, 676
- nvinfer1, 28
 - ActivationType, 41
 - AllocatorFlag, 41
 - AllocatorFlags, 38
 - AsciiChar, 38
 - BoundingBoxFormat, 42
 - BuilderFlag, 42
 - BuilderFlags, 38
 - CalibrationAlgoType, 44
 - char_t, 38
 - DataType, 44
 - DeviceType, 45
 - DimensionOperation, 45
 - Dims, 38
 - ElementWiseOperation, 46
 - EngineCapability, 46
 - EnumMax, 74
 - EnumMax< BoundingBoxFormat >, 74
 - EnumMax< BuilderFlag >, 74
 - EnumMax< CalibrationAlgoType >, 74
 - EnumMax< DeviceType >, 74
 - EnumMax< DimensionOperation >, 75
 - EnumMax< FillOperation >, 75
 - EnumMax< GatherMode >, 75
 - EnumMax< LayerInformationFormat >, 75
 - EnumMax< LayerType >, 76
 - EnumMax< LoopOutput >, 76
 - EnumMax< MatrixOperation >, 76
 - EnumMax< MemoryPoolType >, 76
 - EnumMax< NetworkDefinitionCreationFlag >, 77
 - EnumMax< OptProfileSelector >, 77
 - EnumMax< ProfilingVerbosity >, 77
 - EnumMax< QuantizationFlag >, 77
 - EnumMax< ReduceOperation >, 78
 - EnumMax< RNNDirection >, 78
 - EnumMax< RNNGateType >, 78
 - EnumMax< RNNInputMode >, 78
 - EnumMax< RNNOperation >, 79
 - EnumMax< SampleMode >, 79
 - EnumMax< ScaleMode >, 79
 - EnumMax< ScatterMode >, 79
 - EnumMax< TacticSource >, 80
 - EnumMax< TempfileControlFlag >, 80
 - EnumMax< TopKOperation >, 80
 - EnumMax< TripLimit >, 80
 - EnumMax< UnaryOperation >, 81
 - EnumMax< WeightsRole >, 81
 - ErrorCode, 47
 - FillOperation, 49
 - GatherMode, 49
 - getBuilderPluginRegistry, 81
 - HardwareCompatibilityLevel, 49
 - InterpolationMode, 50
 - kABS, 73
 - kACOS, 73
 - kACOSH, 73
 - kACTIVATION, 51
 - kALIGN_CORNERS, 61
 - kALLOW_IN_MEMORY_FILES, 68
 - kALLOW_TEMPORARY_FILES, 68
 - kAMPERE_PLUS, 50
 - kAND, 46
 - kANY, 73
 - kASIN, 73
 - kASINH, 73
 - kASSERTION, 52
 - kASYMMETRIC, 61
 - kATAN, 73
 - kATANH, 73
 - kAVERAGE, 58
 - kAVG, 61
 - kBIAS, 73
 - kBIDIRECTION, 63
 - kBOOL, 44
 - kCAFFE_ROUND_DOWN, 57
 - kCAFFE_ROUND_UP, 57
 - kCALIBRATE_BEFORE_FUSION, 60
 - kCAST, 52
 - kCDHW32, 70
 - kCEIL, 62, 73
 - kCEIL_DIV, 46
 - kCELL, 63
 - kCENTER_SIZES, 42
 - kCHANNEL, 67
 - kCHAR, 58
 - kCHW16, 70
 - kCHW2, 70
 - kCHW32, 70
 - kCHW4, 70
 - kCLAMP, 66
 - kCLIP, 41
 - kCONCATENATE, 52
 - kCONCATENATION, 51
 - kCONDITION, 52
 - kCONDITIONAL_INPUT, 52
 - kCONDITIONAL_OUTPUT, 52
 - kCONSTANT, 51, 73
 - kCONVOLUTION, 51
 - kCORNER_PAIRS, 42
 - kCOS, 73
 - kCOSH, 73
 - kCOUNT, 72
 - kCUBIC, 50
 - kCUBLAS, 67

kCUBLAS_LT, 67
 kCUDNN, 68
 kDEBUG, 42
 kDECONVOLUTION, 51
 kDEFAULT, 47, 49, 60, 66
 kDEQUANTIZE, 52
 kDETAILED, 60
 kDEVICE, 71
 kDHWC, 71
 kDHWC8, 70
 kDIMS, 58
 kDIRECT_IO, 43
 kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805, 59
 kDISABLE_TIMING_CACHE, 43
 kDIV, 46
 kDLA, 45
 kDLA_GLOBAL_DRAM, 53
 kDLA_HWC4, 71
 kDLA_LINEAR, 70
 kDLA_LOCAL_DRAM, 53
 kDLA_MANAGED_SRAM, 53
 kDLA_STANDALONE, 47
 kEDGE_MASK_CONVOLUTIONS, 68
 kEINSUM, 52
 kELEMENT, 49, 67
 kELEMENTWISE, 51, 67
 kELU, 41
 kENABLE_MULTI_STREAM_ENGINE_0806, 59
 kENABLE_TACTIC_HEURISTIC, 43
 kENTROPY_CALIBRATION, 44
 kENTROPY_CALIBRATION_2, 44
 kEQUAL, 46
 kERF, 73
 kEXCLUDE_LEAN_RUNTIME, 44
 kEXP, 73
 kEXPLICIT_BATCH, 54
 kEXPLICIT_PRECISION, 54
 kEXPLICIT_ROUND_DOWN, 57
 kEXPLICIT_ROUND_UP, 57
 kFAILED_ALLOCATION, 48
 kFAILED_COMPUTATION, 48
 kFAILED_EXECUTION, 48
 kFAILED_INITIALIZATION, 48
 kFASTER_DYNAMIC_SHAPES_0805, 59
 kFILL, 51, 66
 kFLOAT, 44
 kFLOAT16, 58
 kFLOAT32, 58
 kFLOAT64, 58
 kFLOOR, 62, 73
 kFLOOR_DIV, 46
 kFORGET, 63
 kFORMULA, 62
 kFP16, 42
 kFP8, 44, 45
 kFULLY_CONNECTED, 51
 kGATHER, 51
 kGPU, 45
 kGPU_FALLBACK, 42
 kGREATER, 46
 kGRID_SAMPLE, 52
 kGRU, 66
 kHALF, 44
 kHALF_DOWN, 62
 kHALF_PIXEL, 61
 kHALF_UP, 62
 kHARD_SIGMOID, 41
 kHIDDEN, 63
 kHOST, 71
 kHWC, 70
 kHWC16, 71
 kHWC8, 70
 kIDENTITY, 51
 kINPUT, 63, 71
 kINT16, 58
 kINT32, 44, 58
 kINT8, 42, 44, 58
 kINTERNAL_ERROR, 48
 kINVALID_ARGUMENT, 48
 kINVALID_CONFIG, 48
 kINVALID_STATE, 48
 kISINF, 73
 kITERATOR, 51
 kJIT_CONVOLUTIONS, 68
 kJSON, 50
 kKERNEL, 73
 kLAST_VALUE, 52
 kLAYER_NAMES_ONLY, 60
 kLEAKY_RELU, 41
 kLEGACY_CALIBRATION, 44
 kLESS, 46
 kLINEAR, 50, 64, 70
 kLinspace, 49
 kLOG, 73
 kLOOP_OUTPUT, 51
 kLRN, 51
 kLSTM, 66
 kMATRIX_MULTIPLY, 51
 kMAX, 45, 46, 54, 58, 61, 72
 kMAX_AVERAGE_BLEND, 58
 kMIN, 46, 54, 61, 72
 kMINMAX_CALIBRATION, 44
 kND, 49, 67
 kNEAREST, 50
 kNEG, 73
 kNMS, 52
 kNON_ZERO, 52

kNONE, [50](#), [53](#), [60](#), [71](#)
kNORMALIZATION, [52](#)
kNOT, [73](#)
kOBEY_PRECISION_CONSTRAINTS, [43](#)
kONE_HOT, [52](#)
kONELINE, [50](#)
kOPT, [54](#)
kOR, [46](#)
kOUTPUT, [63](#), [71](#)
kPADDING, [51](#)
kPARAMETRIC_RELU, [51](#)
kPLUGIN, [51](#)
kPLUGIN_V2, [51](#)
kPOOLING, [51](#)
kPOW, [46](#)
kPREFER_PRECISION_CONSTRAINTS, [43](#)
kPROD, [45](#), [46](#), [61](#)
kPROFILE_SHARING_0806, [59](#)
kQUANTIZE, [52](#)
kRAGGED_SOFTMAX, [51](#)
kRANDOM_NORMAL, [49](#)
kRANDOM_UNIFORM, [49](#)
kRECIP, [73](#)
kRECURRENCE, [51](#)
kREDUCE, [51](#)
kREFIT, [43](#)
kREFLECT, [66](#)
kREJECT_EMPTY_ALGORITHMS, [43](#)
kRELU, [41](#), [66](#)
kRESET, [63](#)
kRESIZABLE, [42](#)
kRESIZE, [51](#)
kREVERSE, [52](#)
kREVERSE_SEQUENCE, [52](#)
kRNN_V2, [51](#)
kROUND, [73](#)
kSAFE_DLA, [47](#)
kSAFE_GPU, [47](#)
kSAFETY, [47](#)
kSAFETY_SCOPE, [43](#)
kSAME_LOWER, [57](#)
kSAME_UPPER, [57](#)
kSCALE, [51](#), [73](#)
kSCALED_TANH, [41](#)
kSCATTER, [52](#)
kSELECT, [51](#)
kSELU, [41](#)
kSHAPE, [51](#)
kSHIFT, [73](#)
kSHUFFLE, [51](#)
kSIGMOID, [41](#)
kSIGN, [73](#)
kSIN, [73](#)
kSINH, [73](#)
kSKIP, [64](#)
kSLICE, [51](#)
kSOFTMAX, [51](#)
kSOFTPLUS, [41](#)
kSOFTSIGN, [41](#)
kSPARSE_WEIGHTS, [43](#)
kSQRT, [73](#)
kSTANDARD, [47](#)
kSTRICT_BOUNDS, [66](#)
kSTRICT_TYPES, [43](#)
kSUB, [46](#)
kSUCCESS, [47](#)
kSUM, [45](#), [46](#), [61](#)
kTACTIC_DRAM, [53](#)
kTAN, [73](#)
kTANH, [41](#), [66](#)
kTF32, [43](#)
kTHRESHOLDED_RELU, [41](#)
kTOPK, [51](#)
kTRANPOSE, [53](#)
kTRIP_LIMIT, [51](#)
kUINT8, [45](#)
kUNARY, [51](#)
kUNIDIRECTION, [63](#)
kUNIFORM, [67](#)
kUNKNOWN, [58](#)
kUNSPECIFIED_ERROR, [47](#)
kUNSUPPORTED_STATE, [48](#)
kUPDATE, [63](#)
kUPPER, [62](#)
kV2, [58](#)
kV2_DYNAMICEXT, [58](#)
kV2_EXT, [58](#)
kV2_IOEXT, [58](#)
kVECTOR, [53](#)
kVERBOSE, [60](#)
kVERSION_COMPATIBLE, [43](#)
kWHILE, [72](#)
kWORKSPACE, [53](#)
kWRAP, [66](#)
kXOR, [46](#)
LayerInformationFormat, [50](#)
LayerType, [50](#)
LoopOutput, [52](#)
MatrixOperation, [52](#)
MemoryPoolType, [53](#)
NetworkDefinitionCreationFlag, [54](#)
NetworkDefinitionCreationFlags, [39](#)
OptProfileSelector, [54](#)
PaddingMode, [54](#)
PluginFieldType, [58](#)
PluginFormat, [39](#)
PluginVersion, [58](#)
PoolingType, [58](#)

- PreviewFeature, [59](#)
- ProfilingVerbosity, [59](#)
- QuantizationFlag, [60](#)
- QuantizationFlags, [39](#)
- ReduceOperation, [60](#)
- ResizeCoordinateTransformation, [61](#)
- ResizeMode, [39](#)
- ResizeRoundMode, [62](#)
- ResizeSelector, [62](#)
- RNNDirection, [62](#)
- RNNGateType, [63](#)
- RNNInputMode, [63](#)
- RNNOperation, [64](#)
- SampleMode, [66](#)
- ScaleMode, [66](#)
- ScatterMode, [67](#)
- SliceMode, [40](#)
- TacticSource, [67](#)
- TacticSources, [40](#)
- TempfileControlFlag, [68](#)
- TempfileControlFlags, [40](#)
- TensorFormat, [68](#)
- TensorFormats, [40](#)
- TensorIOMode, [71](#)
- TensorLocation, [71](#)
- TopKOperation, [71](#)
- TripLimit, [72](#)
- UnaryOperation, [72](#)
- WeightsRole, [73](#)
- nvinfer1::consistency, [82](#)
- nvinfer1::consistency::IConsistencyChecker, [190](#)
 - ~IConsistencyChecker, [191](#)
 - IConsistencyChecker, [191](#)
 - mImpl, [192](#)
 - operator=, [191](#)
 - validate, [192](#)
- nvinfer1::consistency::IPluginChecker, [486](#)
 - ~IPluginChecker, [487](#)
 - IPluginChecker, [487, 488](#)
 - operator=, [488](#)
 - validate, [488](#)
- nvinfer1::Dims2, [97](#)
 - Dims2, [97](#)
- nvinfer1::Dims3, [98](#)
 - Dims3, [98, 99](#)
- nvinfer1::Dims32, [99](#)
 - d, [100](#)
 - MAX_DIMS, [100](#)
 - nbDims, [100](#)
- nvinfer1::Dims4, [101](#)
 - Dims4, [101](#)
- nvinfer1::DimsExprs, [102](#)
 - d, [102](#)
 - nbDims, [102](#)
- nvinfer1::DimsHW, [103](#)
 - DimsHW, [104](#)
 - h, [104](#)
 - w, [105](#)
- nvinfer1::DynamicPluginTensorDesc, [105](#)
 - desc, [106](#)
 - max, [106](#)
 - min, [106](#)
- nvinfer1::IActivationLayer, [125](#)
 - ~IActivationLayer, [126](#)
 - getActivationType, [126](#)
 - getAlpha, [126](#)
 - getBeta, [127](#)
 - mImpl, [128](#)
 - setActivationType, [127](#)
 - setAlpha, [127](#)
 - setBeta, [128](#)
- nvinfer1::IAlgorithm, [129](#)
 - ~IAlgorithm, [130](#)
 - getAlgorithmIOInfo, [130](#)
 - getAlgorithmIOInfoByIndex, [130](#)
 - getAlgorithmVariant, [131](#)
 - getTimingMSec, [131](#)
 - getWorkspaceSize, [131](#)
 - mImpl, [131](#)
- nvinfer1::IAlgorithmContext, [132](#)
 - ~IAlgorithmContext, [133](#)
 - getDimensions, [133](#)
 - getName, [133](#)
 - getNbInputs, [133](#)
 - getNbOutputs, [133](#)
 - mImpl, [134](#)
- nvinfer1::IAlgorithmIOInfo, [134](#)
 - ~IAlgorithmIOInfo, [135](#)
 - getDataType, [135](#)
 - getStrides, [135](#)
 - getTensorFormat, [136](#)
 - mImpl, [136](#)
- nvinfer1::IAlgorithmSelector, [136](#)
 - ~IAlgorithmSelector, [137](#)
 - reportAlgorithms, [137](#)
 - selectAlgorithms, [137](#)
- nvinfer1::IAlgorithmVariant, [138](#)
 - ~IAlgorithmVariant, [139](#)
 - getImplementation, [139](#)
 - getTactic, [140](#)
 - mImpl, [140](#)
- nvinfer1::IAssertionLayer, [140](#)
 - ~IAssertionLayer, [141](#)
 - getMessage, [141](#)
 - mImpl, [142](#)
 - setMessage, [142](#)
- nvinfer1::IBuilder, [145](#)
 - ~IBuilder, [147](#)

- buildEngineWithConfig, 147
- buildSerializedNetwork, 148
- createBuilderConfig, 148
- createNetworkV2, 149
- createOptimizationProfile, 149
- destroy, 149
- getErrorRecorder, 150
- getLogger, 150
- getMaxBatchSize, 150
- getMaxDLABatchSize, 151
- getMaxThreads, 151
- getNbDLACores, 151
- getPluginRegistry, 152
- isNetworkSupported, 152
- mImpl, 155
- platformHasFastFp16, 152
- platformHasFastInt8, 153
- platformHasTf32, 153
- reset, 153
- setErrorRecorder, 153
- setGpuAllocator, 154
- setMaxBatchSize, 154
- setMaxThreads, 155
- nvinfer1::IBuilderConfig, 155
 - ~IBuilderConfig, 158
 - addOptimizationProfile, 159
 - canRunOnDLA, 159
 - clearFlag, 159
 - clearQuantizationFlag, 160
 - createTimingCache, 160
 - destroy, 161
 - getAlgorithmSelector, 161
 - getAvgTimingIterations, 161
 - getBuilderOptimizationLevel, 161
 - getCalibrationProfile, 162
 - getDefaultDeviceType, 162
 - getDeviceType, 162
 - getDLACore, 162
 - getEngineCapability, 163
 - getFlag, 163
 - getFlags, 163
 - getHardwareCompatibilityLevel, 164
 - getInt8Calibrator, 164
 - getMaxWorkspaceSize, 164
 - getMemoryPoolLimit, 165
 - getMinTimingIterations, 165
 - getNbOptimizationProfiles, 166
 - getNbPluginsToSerialize, 166
 - getPluginToSerialize, 166
 - getPreviewFeature, 167
 - getProfileStream, 167
 - getProfilingVerbosity, 167
 - getQuantizationFlag, 168
 - getQuantizationFlags, 168
 - getTacticSources, 168
 - getTimingCache, 169
 - isDeviceTypeSet, 169
 - mImpl, 179
 - reset, 169
 - resetDeviceType, 170
 - setAlgorithmSelector, 170
 - setAvgTimingIterations, 170
 - setBuilderOptimizationLevel, 170
 - setCalibrationProfile, 171
 - setDefaultDeviceType, 171
 - setDeviceType, 172
 - setDLACore, 172
 - setEngineCapability, 173
 - setFlag, 173
 - setFlags, 173
 - setHardwareCompatibilityLevel, 174
 - setInt8Calibrator, 174
 - setMaxWorkspaceSize, 174
 - setMemoryPoolLimit, 175
 - setMinTimingIterations, 175
 - setPluginsToSerialize, 176
 - setPreviewFeature, 176
 - setProfileStream, 177
 - setProfilingVerbosity, 177
 - setQuantizationFlag, 177
 - setQuantizationFlags, 178
 - setTacticSources, 178
 - setTimingCache, 179
- nvinfer1::ICastLayer, 185
 - ~ICastLayer, 186
 - getToType, 186
 - mImpl, 186
 - setToType, 186
- nvinfer1::IConcatenationLayer, 187
 - ~IConcatenationLayer, 187
 - getAxis, 188
 - mImpl, 188
 - setAxis, 188
- nvinfer1::IConditionLayer, 189
 - ~IConditionLayer, 189
 - mImpl, 190
- nvinfer1::IConstantLayer, 192
 - ~IConstantLayer, 193
 - getDimensions, 193
 - getWeights, 194
 - mImpl, 195
 - setDimensions, 194
 - setWeights, 194
- nvinfer1::IConvolutionLayer, 195
 - ~IConvolutionLayer, 197
 - getBiasWeights, 198
 - getDilation, 198
 - getDilationNd, 198

- getKernelSize, 198
- getKernelSizeNd, 199
- getKernelWeights, 199
- getNbGroups, 199
- getNbOutputMaps, 199
- getPadding, 200
- getPaddingMode, 200
- getPaddingNd, 200
- getPostPadding, 201
- getPrePadding, 201
- getStride, 201
- getStrideNd, 201
- mImpl, 208
- setBiasWeights, 202
- setDilation, 202
- setDilationNd, 202
- setInput, 203
- setKernelSize, 203
- setKernelSizeNd, 204
- setKernelWeights, 204
- setNbGroups, 204
- setNbOutputMaps, 205
- setPadding, 205
- setPaddingMode, 206
- setPaddingNd, 206
- setPostPadding, 206
- setPrePadding, 207
- setStride, 207
- setStrideNd, 207
- nvInfer1::ICudaEngine, 208
 - ~ICudaEngine, 211
 - bindingIsInput, 211
 - createEngineInspector, 212
 - createExecutionContext, 212
 - createExecutionContextWithoutDeviceMemory, 212
 - destroy, 213
 - getBindingBytesPerComponent, 213
 - getBindingComponentsPerElement, 213
 - getBindingDataType, 214
 - getBindingDimensions, 214
 - getBindingFormat, 215
 - getBindingFormatDesc, 216
 - getBindingIndex, 216
 - getBindingName, 217
 - getBindingVectorizedDim, 218
 - getDeviceMemorySize, 218
 - getEngineCapability, 218
 - getErrorRecorder, 219
 - getHardwareCompatibilityLevel, 219
 - getIOTensorName, 219
 - getLocation, 220
 - getMaxBatchSize, 220
 - getName, 221
 - getNbBindings, 221
 - getNbIOTensors, 221
 - getNbLayers, 222
 - getNbOptimizationProfiles, 222
 - getProfileDimensions, 222
 - getProfileShape, 223
 - getProfileShapeValues, 224
 - getProfilingVerbosity, 224
 - getTacticSources, 225
 - getTensorBytesPerComponent, 225
 - getTensorComponentsPerElement, 226
 - getTensorDataType, 226
 - getTensorFormat, 227
 - getTensorFormatDesc, 227
 - getTensorIOMode, 228
 - getTensorLocation, 228
 - getTensorShape, 229
 - getTensorVectorizedDim, 229
 - hasImplicitBatchDimension, 230
 - isExecutionBinding, 230
 - isRefittable, 230
 - isShapeBinding, 231
 - isShapeInferenceIO, 231
 - mImpl, 233
 - serialize, 232
 - setErrorRecorder, 232
- nvInfer1::IDeconvolutionLayer, 251
 - ~IDeconvolutionLayer, 253
 - getBiasWeights, 253
 - getDilationNd, 253
 - getKernelSize, 254
 - getKernelSizeNd, 254
 - getKernelWeights, 254
 - getNbGroups, 254
 - getNbOutputMaps, 255
 - getPadding, 255
 - getPaddingMode, 255
 - getPaddingNd, 256
 - getPostPadding, 256
 - getPrePadding, 256
 - getStride, 256
 - getStrideNd, 257
 - mImpl, 263
 - setBiasWeights, 257
 - setDilationNd, 257
 - setInput, 258
 - setKernelSize, 258
 - setKernelSizeNd, 259
 - setKernelWeights, 259
 - setNbGroups, 259
 - setNbOutputMaps, 260
 - setPadding, 260
 - setPaddingMode, 261
 - setPaddingNd, 261
 - setPostPadding, 261

- setPrePadding, [262](#)
 - setStride, [262](#)
 - setStrideNd, [262](#)
- [nvinfer1::IDequantizeLayer](#), [263](#)
 - ~IDequantizeLayer, [265](#)
 - getAxis, [265](#)
 - mImpl, [265](#)
 - setAxis, [265](#)
- [nvinfer1::IDimensionExpr](#), [266](#)
 - ~IDimensionExpr, [267](#)
 - getConstantValue, [267](#)
 - isConstant, [267](#)
 - mImpl, [267](#)
- [nvinfer1::IEinsumLayer](#), [268](#)
 - ~IEinsumLayer, [269](#)
 - getEquation, [269](#)
 - mImpl, [270](#)
 - setEquation, [269](#)
- [nvinfer1::IElementWiseLayer](#), [270](#)
 - ~IElementWiseLayer, [271](#)
 - getOperation, [271](#)
 - mImpl, [272](#)
 - setOperation, [271](#)
- [nvinfer1::IEngineInspector](#), [272](#)
 - ~IEngineInspector, [273](#)
 - getEngineInformation, [274](#)
 - getErrorRecorder, [274](#)
 - getExecutionContext, [275](#)
 - getLayerInformation, [275](#)
 - mImpl, [277](#)
 - setErrorRecorder, [276](#)
 - setExecutionContext, [276](#)
- [nvinfer1::IErrorRecorder](#), [277](#)
 - ~IErrorRecorder, [279](#)
 - clear, [279](#)
 - decRefCount, [279](#)
 - ErrorDesc, [278](#)
 - getErrorCode, [280](#)
 - getErrorDesc, [281](#)
 - getNbErrors, [281](#)
 - hasOverflowed, [282](#)
 - IErrorRecorder, [278](#)
 - incRefCount, [282](#)
 - kMAX_DESC_LENGTH, [284](#)
 - RefCount, [278](#)
 - reportError, [283](#)
- [nvinfer1::IExecutionContext](#), [284](#)
 - ~IExecutionContext, [287](#)
 - allInputDimensionsSpecified, [287](#)
 - allInputShapesSpecified, [288](#)
 - destroy, [288](#)
 - enqueue, [289](#)
 - enqueueV2, [289](#)
 - enqueueV3, [290](#)
 - execute, [291](#)
 - executeV2, [291](#)
 - getBindingDimensions, [292](#)
 - getDebugSync, [293](#)
 - getEngine, [293](#)
 - getEnqueueEmitsProfile, [293](#)
 - getErrorRecorder, [294](#)
 - getInputConsumedEvent, [294](#)
 - getMaxOutputSize, [294](#)
 - getName, [295](#)
 - getNvtxVerbosity, [295](#)
 - getOptimizationProfile, [295](#)
 - getOutputAllocator, [296](#)
 - getOutputTensorAddress, [296](#)
 - getPersistentCacheLimit, [297](#)
 - getProfiler, [297](#)
 - getShapeBinding, [297](#)
 - getStrides, [298](#)
 - getTemporaryStorageAllocator, [298](#)
 - getTensorAddress, [299](#)
 - getTensorShape, [299](#)
 - getTensorStrides, [300](#)
 - inferShapes, [301](#)
 - mImpl, [313](#)
 - reportToProfiler, [301](#)
 - setBindingDimensions, [302](#)
 - setDebugSync, [303](#)
 - setDeviceMemory, [303](#)
 - setEnqueueEmitsProfile, [304](#)
 - setErrorRecorder, [304](#)
 - setInputConsumedEvent, [305](#)
 - setInputShape, [305](#)
 - setInputShapeBinding, [306](#)
 - setInputTensorAddress, [306](#)
 - setName, [307](#)
 - setNvtxVerbosity, [307](#)
 - setOptimizationProfile, [308](#)
 - setOptimizationProfileAsync, [309](#)
 - setOutputAllocator, [310](#)
 - setPersistentCacheLimit, [311](#)
 - setProfiler, [311](#)
 - setTemporaryStorageAllocator, [311](#)
 - setTensorAddress, [312](#)
- [nvinfer1::IExprBuilder](#), [326](#)
 - ~IExprBuilder, [327](#)
 - constant, [327](#)
 - mImpl, [328](#)
 - operation, [327](#)
- [nvinfer1::IFillLayer](#), [328](#)
 - ~IFillLayer, [330](#)
 - getAlpha, [330](#)
 - getBeta, [330](#)
 - getDimensions, [331](#)
 - getOperation, [331](#)

- mImpl, 334
- setAlpha, 331
- setBeta, 332
- setDimensions, 332
- setInput, 333
- setOperation, 334
- nvinfer1::IFullyConnectedLayer, 334
 - ~IFullyConnectedLayer, 336
 - getBiasWeights, 336
 - getKernelWeights, 336
 - getNbOutputChannels, 336
 - mImpl, 338
 - setBiasWeights, 337
 - setInput, 337
 - setKernelWeights, 338
 - setNbOutputChannels, 338
- nvinfer1::IGatherLayer, 339
 - ~IGatherLayer, 341
 - getGatherAxis, 342
 - getMode, 342
 - getNbElementWiseDims, 342
 - mImpl, 344
 - setGatherAxis, 342
 - setMode, 343
 - setNbElementWiseDims, 343
- nvinfer1::IGpuAllocator, 344
 - ~IGpuAllocator, 344
 - allocate, 345
 - deallocate, 345
 - free, 346
 - IGpuAllocator, 344
 - reallocate, 347
- nvinfer1::IGridSampleLayer, 348
 - ~IGridSampleLayer, 349
 - getAlignCorners, 349
 - getInterpolationMode, 349
 - getSampleMode, 350
 - mImpl, 351
 - setAlignCorners, 350
 - setInterpolationMode, 350
 - setSampleMode, 351
- nvinfer1::IHostMemory, 352
 - ~IHostMemory, 352
 - data, 353
 - destroy, 353
 - mImpl, 354
 - size, 353
 - type, 353
- nvinfer1::IIdentityLayer, 354
 - ~IIdentityLayer, 355
 - mImpl, 355
- nvinfer1::IIfConditional, 356
 - ~IIfConditional, 357
 - addInput, 357
 - addOutput, 357
 - getName, 358
 - mImpl, 359
 - setCondition, 358
 - setName, 358
- nvinfer1::IIfConditionalBoundaryLayer, 359
 - ~IIfConditionalBoundaryLayer, 360
 - getConditional, 360
 - mBoundary, 360
- nvinfer1::IIfConditionalInputLayer, 361
 - ~IIfConditionalInputLayer, 361
 - mImpl, 361
- nvinfer1::IIfConditionalOutputLayer, 362
 - ~IIfConditionalOutputLayer, 362
 - mImpl, 363
- nvinfer1::IInt8Calibrator, 363
 - ~IInt8Calibrator, 364
 - getAlgorithm, 364
 - getBatch, 364
 - getBatchSize, 365
 - readCalibrationCache, 365
 - writeCalibrationCache, 366
- nvinfer1::IInt8EntropyCalibrator, 366
 - ~IInt8EntropyCalibrator, 367
 - getAlgorithm, 367
- nvinfer1::IInt8EntropyCalibrator2, 367
 - ~IInt8EntropyCalibrator2, 368
 - getAlgorithm, 368
- nvinfer1::IInt8LegacyCalibrator, 368
 - ~IInt8LegacyCalibrator, 369
 - getAlgorithm, 369
 - getQuantile, 369
 - getRegressionCutoff, 370
 - readHistogramCache, 370
 - writeHistogramCache, 370
- nvinfer1::IInt8MinMaxCalibrator, 371
 - ~IInt8MinMaxCalibrator, 371
 - getAlgorithm, 372
- nvinfer1::IIteratorLayer, 372
 - ~IIteratorLayer, 373
 - getAxis, 373
 - getReverse, 373
 - mImpl, 374
 - setAxis, 373
 - setReverse, 373
- nvinfer1::ILayer, 374
 - ~ILayer, 376
 - getInput, 376
 - getName, 377
 - getNbInputs, 377
 - getNbOutputs, 377
 - getOutput, 377
 - getOutputType, 377
 - getPrecision, 378

- getType, 378
- mLayer, 382
- outputTypeIsSet, 378
- precisionIsSet, 379
- resetOutputType, 379
- resetPrecision, 380
- setInput, 380
- setName, 380
- setOutputType, 381
- setPrecision, 382
- nvinfer1::ILogger, 383
 - ~ILogger, 384
 - ILogger, 384
 - kERROR, 383
 - kINFO, 383
 - kINTERNAL_ERROR, 383
 - kVERBOSE, 383
 - kWARNING, 383
 - log, 384
 - Severity, 383
- nvinfer1::ILoop, 385
 - ~ILoop, 385
 - addIterator, 386
 - addLoopOutput, 386
 - addRecurrence, 386
 - addTripLimit, 386
 - getName, 387
 - mImpl, 388
 - setName, 387
- nvinfer1::ILoopBoundaryLayer, 388
 - ~ILoopBoundaryLayer, 388
 - getLoop, 389
 - mBoundary, 389
- nvinfer1::ILoopOutputLayer, 389
 - ~ILoopOutputLayer, 390
 - getAxis, 391
 - getLoopOutput, 391
 - mImpl, 392
 - setAxis, 391
 - setInput, 391
- nvinfer1::ILRNLayer, 392
 - ~ILRNLayer, 393
 - getAlpha, 394
 - getBeta, 394
 - getK, 394
 - getWindowSize, 394
 - mImpl, 396
 - setAlpha, 395
 - setBeta, 395
 - setK, 395
 - setWindowSize, 396
- nvinfer1::IMatrixMultiplyLayer, 397
 - ~IMatrixMultiplyLayer, 398
 - getOperation, 398
 - mImpl, 399
 - setOperation, 398
- nvinfer1::impl, 82
- nvinfer1::impl::EnumMaxImpl< ActivationType >, 107
 - kVALUE, 107
- nvinfer1::impl::EnumMaxImpl< AllocatorFlag >, 108
 - kVALUE, 108
- nvinfer1::impl::EnumMaxImpl< DataType >, 108
 - kVALUE, 109
- nvinfer1::impl::EnumMaxImpl< ElementWiseOperation >, 109
 - kVALUE, 109
- nvinfer1::impl::EnumMaxImpl< EngineCapability >, 110
 - kVALUE, 110
- nvinfer1::impl::EnumMaxImpl< ErrorCode >, 111
 - kVALUE, 111
- nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel >, 111
 - kVALUE, 112
- nvinfer1::impl::EnumMaxImpl< ILogger::Severity >, 112
 - kVALUE, 113
- nvinfer1::impl::EnumMaxImpl< InterpolationMode >, 113
 - kVALUE, 113
- nvinfer1::impl::EnumMaxImpl< PaddingMode >, 114
 - kVALUE, 114
- nvinfer1::impl::EnumMaxImpl< PoolingType >, 114
 - kVALUE, 115
- nvinfer1::impl::EnumMaxImpl< PreviewFeature >, 115
 - kVALUE, 115
- nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation >, 116
 - kVALUE, 116
- nvinfer1::impl::EnumMaxImpl< ResizeRoundMode >, 116
 - kVALUE, 117
- nvinfer1::impl::EnumMaxImpl< ResizeSelector >, 117
 - kVALUE, 117
- nvinfer1::impl::EnumMaxImpl< T >, 107
- nvinfer1::impl::EnumMaxImpl< TensorFormat >, 118
 - kVALUE, 118
- nvinfer1::impl::EnumMaxImpl< TensorIOMode >, 119
 - kVALUE, 119
- nvinfer1::impl::EnumMaxImpl< TensorLocation >, 119
 - kVALUE, 120
- nvinfer1::INetworkDefinition, 399
 - ~INetworkDefinition, 403
 - addActivation, 404
 - addAssertion, 404
 - addCast, 405
 - addConcatenation, 405
 - addConstant, 406

- addConvolution, [406](#)
- addConvolutionNd, [407](#)
- addDeconvolution, [408](#)
- addDeconvolutionNd, [409](#)
- addDequantize, [410](#)
- addEinsum, [410](#)
- addElementWise, [411](#)
- addFill, [411](#)
- addFullyConnected, [412](#)
- addGather, [413](#)
- addGatherV2, [413](#)
- addGridSample, [414](#)
- addIdentity, [414](#)
- addIfConditional, [415](#)
- addInput, [415](#)
- addLoop, [416](#)
- addLRN, [416](#)
- addMatrixMultiply, [417](#)
- addNMS, [418](#)
- addNonZero, [418](#)
- addNormalization, [419](#)
- addOneHot, [419](#)
- addPadding, [420](#)
- addPaddingNd, [421](#)
- addParametricReLU, [421](#)
- addPluginV2, [422](#)
- addPooling, [422](#)
- addPoolingNd, [423](#)
- addQuantize, [424](#)
- addRaggedSoftMax, [424](#)
- addReduce, [425](#)
- addResize, [426](#)
- addReverseSequence, [426](#)
- addRNNv2, [427](#)
- addScale, [428](#)
- addScaleNd, [429](#)
- addScatter, [430](#)
- addSelect, [431](#)
- addShape, [431](#)
- addShuffle, [433](#)
- addSlice, [433](#)
- addSoftMax, [434](#)
- addTopK, [434](#)
- addUnary, [435](#)
- destroy, [436](#)
- getErrorRecorder, [436](#)
- getInput, [437](#)
- getLayer, [437](#)
- getName, [438](#)
- getNbInputs, [438](#)
- getNbLayers, [438](#)
- getNbOutputs, [439](#)
- getOutput, [439](#)
- hasExplicitPrecision, [440](#)
- hasImplicitBatchDimension, [440](#)
- markOutput, [440](#)
- markOutputForShapes, [441](#)
- mImpl, [444](#)
- removeTensor, [441](#)
- setErrorRecorder, [442](#)
- setName, [442](#)
- setWeightsName, [443](#)
- unmarkOutput, [443](#)
- unmarkOutputForShapes, [444](#)
- nvInfer1::INMSLayer, [444](#)
 - ~INMSLayer, [446](#)
 - getBoundingBoxFormat, [446](#)
 - getTopKBoxLimit, [446](#)
 - mImpl, [448](#)
 - setBoundingBoxFormat, [447](#)
 - setInput, [447](#)
 - setTopKBoxLimit, [448](#)
- nvInfer1::INoCopy, [449](#)
 - ~INoCopy, [450](#)
 - INoCopy, [450](#)
 - operator=, [450](#)
- nvInfer1::INonZeroLayer, [451](#)
 - ~INonZeroLayer, [452](#)
 - mImpl, [452](#)
- nvInfer1::INormalizationLayer, [452](#)
 - ~INormalizationLayer, [453](#)
 - getAxes, [454](#)
 - getComputePrecision, [454](#)
 - getEpsilon, [454](#)
 - getNbGroups, [454](#)
 - mImpl, [456](#)
 - setAxes, [455](#)
 - setComputePrecision, [455](#)
 - setEpsilon, [455](#)
 - setNbGroups, [456](#)
- nvInfer1::IOneHotLayer, [457](#)
 - getAxis, [458](#)
 - mImpl, [458](#)
 - setAxis, [458](#)
- nvInfer1::IOptimizationProfile, [466](#)
 - ~IOptimizationProfile, [467](#)
 - getDimensions, [467](#)
 - getExtraMemoryTarget, [468](#)
 - getNbShapeValues, [468](#)
 - getShapeValues, [468](#)
 - isValid, [469](#)
 - mImpl, [471](#)
 - setDimensions, [469](#)
 - setExtraMemoryTarget, [470](#)
 - setShapeValues, [470](#)
- nvInfer1::IOutputAllocator, [472](#)
 - ~IOutputAllocator, [472](#)
 - getInterfaceVersion, [472](#)

- notifyShape, 473
 - reallocateOutput, 473
- nvinfer1::IPaddingLayer, 474
 - ~IPaddingLayer, 475
 - getPostPadding, 475
 - getPostPaddingNd, 475
 - getPrePadding, 476
 - getPrePaddingNd, 476
 - mImpl, 478
 - setPostPadding, 476
 - setPostPaddingNd, 477
 - setPrePadding, 477
 - setPrePaddingNd, 477
- nvinfer1::IParametricReLULayer, 478
 - ~IParametricReLULayer, 479
 - mImpl, 479
- nvinfer1::IPluginCreator, 489
 - ~IPluginCreator, 490
 - createPlugin, 490
 - deserializePlugin, 490
 - getFieldNames, 491
 - getPluginName, 491
 - getPluginNamespace, 492
 - getPluginVersion, 492
 - getTensorRTVersion, 493
 - IPluginCreator, 490
 - setPluginNamespace, 493
- nvinfer1::IPluginRegistry, 495
 - ~IPluginRegistry, 497
 - deregisterCreator, 497
 - deregisterLibrary, 497
 - deserializeLibrary, 498
 - getErrorRecorder, 498
 - getPluginCreator, 498
 - getPluginCreatorList, 499
 - isParentSearchEnabled, 499
 - loadLibrary, 499
 - PluginLibraryHandle, 497
 - registerCreator, 501
 - setErrorRecorder, 501
 - setParentSearchEnabled, 502
- nvinfer1::IPluginV2, 502
 - clone, 504
 - configureWithFormat, 504
 - destroy, 505
 - enqueue, 505
 - getNbOutputs, 506
 - getOutputDimensions, 506
 - getPluginNamespace, 507
 - getPluginType, 507
 - getPluginVersion, 508
 - getSerializationSize, 508
 - getTensorRTVersion, 509
 - getWorkspaceSize, 509
 - initialize, 510
 - serialize, 510
 - setPluginNamespace, 511
 - supportsFormat, 511
 - terminate, 512
- nvinfer1::IPluginV2DynamicExt, 513
 - ~IPluginV2DynamicExt, 514
 - clone, 515
 - configurePlugin, 515
 - enqueue, 516
 - getOutputDimensions, 517
 - getTensorRTVersion, 517
 - getWorkspaceSize, 517
 - kFORMAT_COMBINATION_LIMIT, 519
 - supportsFormatCombination, 518
- nvinfer1::IPluginV2Ext, 519
 - ~IPluginV2Ext, 520
 - attachToContext, 521
 - canBroadcastInputAcrossBatch, 521
 - clone, 522
 - configurePlugin, 522
 - configureWithFormat, 524
 - detachFromContext, 524
 - getOutputDataType, 524
 - getTensorRTVersion, 525
 - IPluginV2Ext, 520
 - isOutputBroadcastAcrossBatch, 525
- nvinfer1::IPluginV2IOExt, 526
 - configurePlugin, 527
 - getTensorRTVersion, 528
 - supportsFormatCombination, 528
- nvinfer1::IPluginV2Layer, 529
 - ~IPluginV2Layer, 530
 - getPlugin, 530
 - mImpl, 531
- nvinfer1::IPoolingLayer, 531
 - ~IPoolingLayer, 533
 - getAverageCountExcludesPadding, 533
 - getBlendFactor, 533
 - getPadding, 533
 - getPaddingMode, 534
 - getPaddingNd, 534
 - getPoolingType, 534
 - getPostPadding, 535
 - getPrePadding, 535
 - getStride, 535
 - getStrideNd, 535
 - getWindowSize, 536
 - getWindowSizeNd, 536
 - mImpl, 541
 - setAverageCountExcludesPadding, 536
 - setBlendFactor, 537
 - setPadding, 537
 - setPaddingMode, 537

- setPaddingNd, 538
- setPoolingType, 538
- setPostPadding, 538
- setPrePadding, 539
- setStride, 539
- setStrideNd, 539
- setWindowSize, 540
- setWindowSizeNd, 540
- nvinfer1::IProfiler, 541
 - ~IProfiler, 541
 - reportLayerTime, 542
- nvinfer1::IQuantizeLayer, 543
 - ~IQuantizeLayer, 544
 - getAxis, 545
 - mImpl, 545
 - setAxis, 545
- nvinfer1::IRaggedSoftMaxLayer, 546
 - ~IRaggedSoftMaxLayer, 546
 - mImpl, 547
- nvinfer1::IRecurrenceLayer, 547
 - ~IRecurrenceLayer, 548
 - mImpl, 548
 - setInput, 548
- nvinfer1::IReduceLayer, 549
 - ~IReduceLayer, 550
 - getKeepDimensions, 550
 - getOperation, 550
 - getReduceAxes, 550
 - mImpl, 552
 - setKeepDimensions, 551
 - setOperation, 551
 - setReduceAxes, 551
- nvinfer1::IRefitter, 552
 - ~IRefitter, 553
 - destroy, 553
 - getAll, 554
 - getAllWeights, 554
 - getDynamicRangeMax, 555
 - getDynamicRangeMin, 555
 - getErrorRecorder, 555
 - getLogger, 556
 - getMaxThreads, 556
 - getMissing, 556
 - getMissingWeights, 557
 - getTensorsWithDynamicRange, 557
 - mImpl, 561
 - refitCudaEngine, 558
 - setDynamicRange, 558
 - setErrorRecorder, 559
 - setMaxThreads, 559
 - setNamedWeights, 559
 - setWeights, 560
- nvinfer1::IResizeLayer, 561
 - ~IResizeLayer, 563
 - getAlignCorners, 564
 - getCoordinateTransformation, 564
 - getCubicCoeff, 564
 - getExcludeOutside, 564
 - getNearestRounding, 565
 - getOutputDimensions, 565
 - getResizeMode, 565
 - getScales, 565
 - getSelectorForSinglePixel, 566
 - mImpl, 570
 - setAlignCorners, 566
 - setCoordinateTransformation, 566
 - setCubicCoeff, 567
 - setExcludeOutside, 567
 - setInput, 567
 - setNearestRounding, 568
 - setOutputDimensions, 568
 - setResizeMode, 569
 - setScales, 569
 - setSelectorForSinglePixel, 570
- nvinfer1::IReverseSequenceLayer, 571
 - ~IReverseSequenceLayer, 572
 - getBatchAxis, 572
 - getSequenceAxis, 572
 - mImpl, 573
 - setBatchAxis, 572
 - setSequenceAxis, 573
- nvinfer1::IRNNv2Layer, 574
 - ~IRNNv2Layer, 575
 - getBiasForGate, 575
 - getCellState, 576
 - getDataLength, 576
 - getDirection, 576
 - getHiddenSize, 576
 - getHiddenState, 577
 - getInputMode, 577
 - getLayerCount, 577
 - getMaxSeqLength, 577
 - getOperation, 577
 - getSequenceLengths, 578
 - getWeightsForGate, 578
 - mImpl, 582
 - setBiasForGate, 578
 - setCellState, 579
 - setDirection, 579
 - setHiddenState, 579
 - setInputMode, 580
 - setOperation, 580
 - setSequenceLengths, 580
 - setWeightsForGate, 581
- nvinfer1::IRuntime, 582
 - ~IRuntime, 584
 - deserializeCudaEngine, 584
 - destroy, 585

- getDLACore, 585
- getEngineHostCodeAllowed, 586
- getErrorRecorder, 586
- getLogger, 586
- getMaxThreads, 587
- getNbDLACores, 587
- getPluginRegistry, 587
- getTempfileControlFlags, 587
- getTemporaryDirectory, 588
- loadRuntime, 588
- mImpl, 592
- setDLACore, 589
- setEngineHostCodeAllowed, 589
- setErrorRecorder, 589
- setGpuAllocator, 590
- setMaxThreads, 590
- setTempfileControlFlags, 591
- setTemporaryDirectory, 591
- nvinfer1::IScaleLayer, 597
 - ~IScaleLayer, 598
 - getChannelAxis, 599
 - getMode, 599
 - getPower, 599
 - getScale, 599
 - getShift, 600
 - mImpl, 602
 - setChannelAxis, 600
 - setMode, 600
 - setPower, 601
 - setScale, 601
 - setShift, 601
- nvinfer1::IScatterLayer, 602
 - ~IScatterLayer, 604
 - getAxis, 604
 - getMode, 604
 - mImpl, 605
 - setAxis, 604
 - setMode, 604
- nvinfer1::ISelectLayer, 605
 - ~ISelectLayer, 606
 - mImpl, 606
- nvinfer1::IShapeLayer, 606
 - ~IShapeLayer, 607
 - mImpl, 607
- nvinfer1::IShuffleLayer, 608
 - ~IShuffleLayer, 609
 - getFirstTranspose, 609
 - getReshapeDimensions, 609
 - getSecondTranspose, 610
 - getZeroIsPlaceholder, 610
 - mImpl, 613
 - setFirstTranspose, 610
 - setInput, 611
 - setReshapeDimensions, 611
 - setSecondTranspose, 612
 - setZeroIsPlaceholder, 612
- nvinfer1::ISliceLayer, 613
 - ~ISliceLayer, 615
 - getMode, 615
 - getSize, 615
 - getStart, 616
 - getStride, 616
 - mImpl, 619
 - setInput, 616
 - setMode, 617
 - setSize, 617
 - setStart, 618
 - setStride, 618
- nvinfer1::ISoftMaxLayer, 619
 - ~ISoftMaxLayer, 620
 - getAxes, 620
 - mImpl, 621
 - setAxes, 620
- nvinfer1::ITensor, 621
 - ~ITensor, 623
 - dynamicRangeIsSet, 624
 - getAllowedFormats, 624
 - getBroadcastAcrossBatch, 624
 - getDimensionName, 624
 - getDimensions, 625
 - getDynamicRangeMax, 625
 - getDynamicRangeMin, 625
 - getLocation, 626
 - getName, 626
 - getType, 626
 - isExecutionTensor, 627
 - isNetworkInput, 627
 - isNetworkOutput, 627
 - isShapeTensor, 628
 - mImpl, 632
 - resetDynamicRange, 628
 - setAllowedFormats, 628
 - setBroadcastAcrossBatch, 629
 - setDimensionName, 630
 - setDimensions, 630
 - setDynamicRange, 631
 - setLocation, 631
 - setName, 631
 - setType, 632
- nvinfer1::ITimingCache, 633
 - ~ITimingCache, 634
 - combine, 634
 - mImpl, 635
 - reset, 634
 - serialize, 635
- nvinfer1::ITopKLayer, 636
 - ~ITopKLayer, 637
 - getK, 637

- getOperation, [637](#)
 - getReduceAxes, [637](#)
 - mImpl, [639](#)
 - setInput, [638](#)
 - setK, [638](#)
 - setOperation, [638](#)
 - setReduceAxes, [639](#)
- nvinfer1::ITripLimitLayer, [639](#)
 - ~ITripLimitLayer, [640](#)
 - getTripLimit, [640](#)
 - mImpl, [640](#)
- nvinfer1::IUnaryLayer, [645](#)
 - ~IUnaryLayer, [646](#)
 - getOperation, [646](#)
 - mImpl, [647](#)
 - setOperation, [646](#)
- nvinfer1::Permutation, [649](#)
 - order, [649](#)
- nvinfer1::plugin, [83](#)
 - CENTER_SIZE, [84](#)
 - CodeTypeSSD, [83](#)
 - CORNER, [84](#)
 - CORNER_SIZE, [84](#)
 - TF_CENTER, [84](#)
- nvinfer1::plugin::DetectionOutputParameters, [93](#)
 - backgroundLabelId, [94](#)
 - codeType, [94](#)
 - confidenceThreshold, [94](#)
 - confSigmoid, [94](#)
 - inputOrder, [95](#)
 - isBatchAgnostic, [95](#)
 - isNormalized, [95](#)
 - keepTopK, [95](#)
 - nmsThreshold, [95](#)
 - numClasses, [95](#)
 - shareLocation, [95](#)
 - topK, [96](#)
 - varianceEncodedInTarget, [96](#)
- nvinfer1::plugin::GridAnchorParameters, [123](#)
 - aspectRatios, [124](#)
 - H, [124](#)
 - maxSize, [124](#)
 - minSize, [124](#)
 - numAspectRatios, [124](#)
 - variance, [124](#)
 - W, [125](#)
- nvinfer1::plugin::NMSPParameters, [647](#)
 - backgroundLabelId, [648](#)
 - iouThreshold, [648](#)
 - isNormalized, [648](#)
 - keepTopK, [648](#)
 - numClasses, [648](#)
 - scoreThreshold, [649](#)
 - shareLocation, [649](#)
 - topK, [649](#)
- nvinfer1::plugin::PriorBoxParameters, [656](#)
 - aspectRatios, [657](#)
 - clip, [657](#)
 - flip, [657](#)
 - imgH, [657](#)
 - imgW, [658](#)
 - maxSize, [658](#)
 - minSize, [658](#)
 - numAspectRatios, [658](#)
 - numMaxSize, [658](#)
 - numMinSize, [658](#)
 - offset, [658](#)
 - stepH, [659](#)
 - stepW, [659](#)
 - variance, [659](#)
- nvinfer1::plugin::Quadruple, [659](#)
 - data, [660](#)
- nvinfer1::plugin::RegionParameters, [660](#)
 - classes, [661](#)
 - coords, [661](#)
 - num, [661](#)
 - smTree, [661](#)
- nvinfer1::plugin::RPROIParams, [661](#)
 - anchorsRatioCount, [662](#)
 - anchorsScaleCount, [662](#)
 - featureStride, [662](#)
 - iouThreshold, [662](#)
 - minBoxSize, [662](#)
 - nmsMaxOut, [663](#)
 - poolingH, [663](#)
 - poolingW, [663](#)
 - preNmsTop, [663](#)
 - spatialScale, [663](#)
- nvinfer1::plugin::softmaxTree, [663](#)
 - child, [664](#)
 - group, [664](#)
 - groupOffset, [664](#)
 - groups, [664](#)
 - groupSize, [664](#)
 - leaf, [665](#)
 - n, [665](#)
 - name, [665](#)
 - parent, [665](#)
- nvinfer1::PluginField, [650](#)
 - data, [651](#)
 - length, [651](#)
 - name, [651](#)
 - PluginField, [650](#)
 - type, [651](#)
- nvinfer1::PluginFieldCollection, [652](#)
 - fields, [652](#)
 - nbFields, [652](#)
- nvinfer1::PluginRegistrar< T >, [653](#)

- PluginRegistrar, 653
- nvInfer1::PluginTensorDesc, 654
 - dims, 655
 - format, 655
 - scale, 655
 - type, 655
- nvInfer1::safe, 84
 - createInferRuntime, 84
 - getSafePluginRegistry, 85
- nvInfer1::safe::FloatingPointErrorInformation, 122
 - nbInfErrors, 123
 - nbNanErrors, 123
- nvInfer1::safe::ICudaEngine, 233
 - ~ICudaEngine, 235
 - bindingIsInput, 235
 - createExecutionContext, 236
 - createExecutionContextWithoutDeviceMemory, 236
 - getBindingBytesPerComponent, 237
 - getBindingComponentsPerElement, 237
 - getBindingDataType, 238
 - getBindingDimensions, 239
 - getBindingFormat, 239
 - getBindingIndex, 240
 - getBindingName, 241
 - getBindingVectorizedDim, 241
 - getDeviceMemorySize, 242
 - getErrorRecorder, 242
 - getIOTensorName, 243
 - getName, 244
 - getNbBindings, 244
 - getNbIOTensors, 245
 - getTensorBytesPerComponent, 245
 - getTensorComponentsPerElement, 246
 - getTensorDataType, 247
 - getTensorFormat, 247
 - getTensorIOMode, 248
 - getTensorShape, 249
 - getTensorVectorizedDim, 249
 - ICudaEngine, 235
 - operator=, 250
 - setErrorRecorder, 250
- nvInfer1::safe::IExecutionContext, 313
 - ~IExecutionContext, 315
 - enqueueV2, 315
 - enqueueV3, 316
 - getEngine, 316
 - getErrorBuffer, 317
 - getErrorRecorder, 317
 - getInputConsumedEvent, 318
 - getInputTensorAddress, 318
 - getName, 319
 - getOutputTensorAddress, 319
 - getStrides, 320
 - getTensorStrides, 320
 - IExecutionContext, 314, 315
 - operator=, 321
 - setDeviceMemory, 321
 - setErrorBuffer, 322
 - setErrorRecorder, 323
 - setInputConsumedEvent, 323
 - setInputTensorAddress, 324
 - setName, 325
 - setOutputTensorAddress, 325
- nvInfer1::safe::IRuntime, 592
 - ~IRuntime, 593
 - deserializeCudaEngine, 594
 - getErrorRecorder, 595
 - IRuntime, 593, 594
 - operator=, 595
 - setErrorRecorder, 596
 - setGpuAllocator, 596
- nvInfer1::safe::PluginRegistrar< T >, 653
 - PluginRegistrar, 654
- nvInfer1::utils, 85
 - reorderSubBuffers, 86
 - reshapeWeights, 87
 - transposeSubBuffers, 88
- nvInfer1::Weights, 665
 - count, 666
 - type, 666
 - values, 666
- NvInferConsistency.h, 721, 722
 - createConsistencyChecker_INTERNAL, 722
- NvInferLegacyDims.h, 723, 724
- NvInferPlugin.h, 725, 731
 - createAnchorGeneratorPlugin, 727
 - createBatchedNMSPlugin, 727
 - createInstanceNormalizationPlugin, 727
 - createNMSPlugin, 728
 - createNormalizePlugin, 728
 - createPriorBoxPlugin, 729
 - createRegionPlugin, 729
 - createReorgPlugin, 729
 - createRPNROIPlugin, 730
 - createSplitPlugin, 730
 - initLibNvInferPlugins, 731
- NvInferPluginUtils.h, 732, 733
- NvInferRuntime.h, 735, 739
 - getLogger, 738
 - getPluginRegistry, 738
 - REGISTER_TENSORRT_PLUGIN, 738
- NvInferRuntimeCommon.h, 754, 758
 - getInferLibVersion, 757
 - NV_TENSORRT_VERSION, 756
 - TENSORRTAPI, 757
 - TRT_DEPRECATED, 757
 - TRT_DEPRECATED_API, 757
 - TRT_DEPRECATED_ENUM, 757

- TRTNOEXCEPT, 757
- NvInferSafeRuntime.h, 766, 767
 - REGISTER_SAFE_TENSORRT_PLUGIN, 767
- NvInferVersion.h, 770, 772
 - NV_TENSORRT_BUILD, 770
 - NV_TENSORRT_LWS_MAJOR, 770
 - NV_TENSORRT_LWS_MINOR, 771
 - NV_TENSORRT_LWS_PATCH, 771
 - NV_TENSORRT_MAJOR, 771
 - NV_TENSORRT_MINOR, 771
 - NV_TENSORRT_PATCH, 771
 - NV_TENSORRT_SONAME_MAJOR, 771
 - NV_TENSORRT_SONAME_MINOR, 772
 - NV_TENSORRT_SONAME_PATCH, 772
- NvOnnxConfig.h, 773
- nvonnxparser, 88
 - createONNXConfig, 90
 - EnumMax, 90
 - EnumMax< ErrorCode >, 90
 - ErrorCode, 89
 - kINTERNAL_ERROR, 89
 - kINVALID_GRAPH, 89
 - kINVALID_NODE, 89
 - kINVALID_VALUE, 89
 - kMEM_ALLOC_FAILED, 89
 - kMODEL_DESERIALIZE_FAILED, 89
 - kSUCCESS, 89
 - kUNSUPPORTED_GRAPH, 89
 - kUNSUPPORTED_NODE, 89
- NvOnnxParser.h, 778, 781
 - createNvOnnxParser_INTERNAL, 780
 - getNvOnnxParserVersion, 780
 - NV_ONNX_PARSER_MAJOR, 779
 - NV_ONNX_PARSER_MINOR, 779
 - NV_ONNX_PARSER_PATCH, 780
 - SubGraph_t, 780
 - SubGraphCollection_t, 780
- nvonnxparser::IOnnxConfig, 459
 - ~IOnnxConfig, 460
 - addVerbosity, 460
 - destroy, 460
 - getFullTextFileName, 461
 - getModelDtype, 461
 - getModelFileName, 461
 - getPrintLayerInfo, 462
 - getTextFileName, 462
 - getVerbosityLevel, 462
 - reduceVerbosity, 463
 - setFullTextFileName, 463
 - setModelDtype, 464
 - setModelFileName, 464
 - setPrintLayerInfo, 464
 - setTextFileName, 465
 - setVerbosityLevel, 465
 - Verbosity, 460
- nvonnxparser::IParser, 480
 - ~IParser, 480
 - clearErrors, 481
 - destroy, 481
 - getError, 481
 - getNbErrors, 481
 - parse, 482
 - parseFromFile, 482
 - parseWithWeightDescriptors, 483
 - supportsModel, 483
 - supportsOperator, 484
- nvonnxparser::IParserError, 484
 - ~IParserError, 485
 - code, 485
 - desc, 485
 - file, 485
 - func, 486
 - line, 486
 - node, 486
- nvuffparser, 90
 - createUffParser, 92
 - FieldType, 91
 - kCHAR, 91
 - kDATATYPE, 91
 - kDIMS, 91
 - kFLOAT, 91
 - kINT32, 91
 - kNC, 91
 - kNCHW, 91
 - kNHWC, 91
 - kUNKNOWN, 91
 - shutdownProtobufLibrary, 92
 - UffInputOrder, 91
- NvUffParser.h, 774, 776
 - UFF_REQUIRED_VERSION_MAJOR, 775
 - UFF_REQUIRED_VERSION_MINOR, 775
 - UFF_REQUIRED_VERSION_PATCH, 775
- nvuffparser::FieldCollection, 120
 - fields, 120
 - nbFields, 120
- nvuffparser::FieldMap, 121
 - data, 122
 - FieldMap, 121
 - length, 122
 - name, 122
 - type, 122
- nvuffparser::IUffParser, 641
 - ~IUffParser, 641
 - destroy, 642
 - getErrorRecorder, 642
 - getUffRequiredVersionMajor, 642
 - getUffRequiredVersionMinor, 642
 - getUffRequiredVersionPatch, 643

- parse, [643](#)
 - parseBuffer, [643](#)
 - registerInput, [644](#)
 - registerOutput, [644](#)
 - setErrorRecorder, [644](#)
 - setPluginNamespace, [645](#)
- NvUtils.h, [777](#), [778](#)
- offset
 - nvinfer1::plugin::PriorBoxParameters, [658](#)
- operation
 - nvinfer1::IExprBuilder, [327](#)
- operator=
 - nvinfer1::consistency::IConsistencyChecker, [191](#)
 - nvinfer1::consistency::IPluginChecker, [488](#)
 - nvinfer1::INoCopy, [450](#)
 - nvinfer1::safe::ICudaEngine, [250](#)
 - nvinfer1::safe::IExecutionContext, [321](#)
 - nvinfer1::safe::IRuntime, [595](#)
- OptProfileSelector
 - nvinfer1, [54](#)
- order
 - nvinfer1::Permutation, [649](#)
- outputTypeIsSet
 - nvinfer1::ILayer, [378](#)
- PaddingMode
 - nvinfer1, [54](#)
- parent
 - nvinfer1::plugin::softmaxTree, [665](#)
- parse
 - nvcaffeparser1::ICaffeParser, [181](#)
 - nvonnxparser::IParser, [482](#)
 - nvuffparser::IUffParser, [643](#)
- parseBinaryProto
 - nvcaffeparser1::ICaffeParser, [182](#)
- parseBuffer
 - nvuffparser::IUffParser, [643](#)
- parseBuffers
 - nvcaffeparser1::ICaffeParser, [182](#)
- parseFromFile
 - nvonnxparser::IParser, [482](#)
- parseWithWeightDescriptors
 - nvonnxparser::IParser, [483](#)
- platformHasFastFp16
 - nvinfer1::IBuilder, [152](#)
- platformHasFastInt8
 - nvinfer1::IBuilder, [153](#)
- platformHasTf32
 - nvinfer1::IBuilder, [153](#)
- PluginField
 - nvinfer1::PluginField, [650](#)
- PluginFieldType
 - nvinfer1, [58](#)
- PluginFormat
 - nvinfer1, [39](#)
- PluginLibraryHandle
 - nvinfer1::IPluginRegistry, [497](#)
- PluginRegistrar
 - nvinfer1::PluginRegistrar< T >, [653](#)
 - nvinfer1::safe::PluginRegistrar< T >, [654](#)
- PluginVersion, [656](#)
 - nvinfer1, [58](#)
- poolingH
 - nvinfer1::plugin::RPROIParams, [663](#)
- PoolingType
 - nvinfer1, [58](#)
- poolingW
 - nvinfer1::plugin::RPROIParams, [663](#)
- precisionIsSet
 - nvinfer1::ILayer, [379](#)
- preNmsTop
 - nvinfer1::plugin::RPROIParams, [663](#)
- PreviewFeature
 - nvinfer1, [59](#)
- ProfilingVerbosity
 - nvinfer1, [59](#)
- QuantizationFlag
 - nvinfer1, [60](#)
- QuantizationFlags
 - nvinfer1, [39](#)
- readCalibrationCache
 - nvinfer1::IInt8Calibrator, [365](#)
- readHistogramCache
 - nvinfer1::IInt8LegacyCalibrator, [370](#)
- reallocate
 - nvinfer1::IGpuAllocator, [347](#)
- reallocateOutput
 - nvinfer1::IOutputAllocator, [473](#)
- ReduceOperation
 - nvinfer1, [60](#)
- reduceVerbosity
 - nvonnxparser::IOnnxConfig, [463](#)
- RefCount
 - nvinfer1::IErrorRecorder, [278](#)
- refitCudaEngine
 - nvinfer1::IRefitter, [558](#)
- REGISTER_SAFE_TENSORRT_PLUGIN
 - NvInferSafeRuntime.h, [767](#)
- REGISTER_TENSORRT_PLUGIN
 - NvInferRuntime.h, [738](#)
- registerCreator
 - nvinfer1::IPluginRegistry, [501](#)
- registerInput
 - nvuffparser::IUffParser, [644](#)
- registerOutput
 - nvuffparser::IUffParser, [644](#)
- removeTensor

- nvinfer1::INetworkDefinition, [441](#)
- reorderSubBuffers
 - nvinfer1::utils, [86](#)
- reportAlgorithms
 - nvinfer1::IAlgorithmSelector, [137](#)
- reportError
 - nvinfer1::IErrorRecorder, [283](#)
- reportLayerTime
 - nvinfer1::IProfiler, [542](#)
- reportToProfiler
 - nvinfer1::IExecutionContext, [301](#)
- reset
 - nvinfer1::IBuilder, [153](#)
 - nvinfer1::IBuilderConfig, [169](#)
 - nvinfer1::ITimingCache, [634](#)
- resetDeviceType
 - nvinfer1::IBuilderConfig, [170](#)
- resetDynamicRange
 - nvinfer1::ITensor, [628](#)
- resetOutputType
 - nvinfer1::ILayer, [379](#)
- resetPrecision
 - nvinfer1::ILayer, [380](#)
- reshapeWeights
 - nvinfer1::utils, [87](#)
- ResizeCoordinateTransformation
 - nvinfer1, [61](#)
- ResizeMode
 - nvinfer1, [39](#)
- ResizeRoundMode
 - nvinfer1, [62](#)
- ResizeSelector
 - nvinfer1, [62](#)
- RNNDirection
 - nvinfer1, [62](#)
- RNNGateType
 - nvinfer1, [63](#)
- RNNInputMode
 - nvinfer1, [63](#)
- RNNOperation
 - nvinfer1, [64](#)
- SampleMode
 - nvinfer1, [66](#)
- scale
 - nvinfer1::PluginTensorDesc, [655](#)
- ScaleMode
 - nvinfer1, [66](#)
- ScatterMode
 - nvinfer1, [67](#)
- scoreThreshold
 - nvinfer1::plugin::NMSPParameters, [649](#)
- selectAlgorithms
 - nvinfer1::IAlgorithmSelector, [137](#)

- serialize
 - nvinfer1::ICudaEngine, [232](#)
 - nvinfer1::IPluginV2, [510](#)
 - nvinfer1::ITimingCache, [635](#)
- setActivationType
 - nvinfer1::IActivationLayer, [127](#)
- setAlgorithmSelector
 - nvinfer1::IBuilderConfig, [170](#)
- setAlignCorners
 - nvinfer1::IGridSampleLayer, [350](#)
 - nvinfer1::IResizeLayer, [566](#)
- setAllowedFormats
 - nvinfer1::ITensor, [628](#)
- setAlpha
 - nvinfer1::IActivationLayer, [127](#)
 - nvinfer1::IFillLayer, [331](#)
 - nvinfer1::ILRNLayer, [395](#)
- setAverageCountExcludesPadding
 - nvinfer1::IPoolingLayer, [536](#)
- setAvgTimingIterations
 - nvinfer1::IBuilderConfig, [170](#)
- setAxes
 - nvinfer1::INormalizationLayer, [455](#)
 - nvinfer1::ISoftMaxLayer, [620](#)
- setAxis
 - nvinfer1::IConcatenationLayer, [188](#)
 - nvinfer1::IDequantizeLayer, [265](#)
 - nvinfer1::IIteratorLayer, [373](#)
 - nvinfer1::ILoopOutputLayer, [391](#)
 - nvinfer1::IOneHotLayer, [458](#)
 - nvinfer1::IQuantizeLayer, [545](#)
 - nvinfer1::IScatterLayer, [604](#)
- setBatchAxis
 - nvinfer1::IReverseSequenceLayer, [572](#)
- setBeta
 - nvinfer1::IActivationLayer, [128](#)
 - nvinfer1::IFillLayer, [332](#)
 - nvinfer1::ILRNLayer, [395](#)
- setBiasForGate
 - nvinfer1::IRNNv2Layer, [578](#)
- setBiasWeights
 - nvinfer1::IConvolutionLayer, [202](#)
 - nvinfer1::IDeconvolutionLayer, [257](#)
 - nvinfer1::IFullyConnectedLayer, [337](#)
- setBindingDimensions
 - nvinfer1::IExecutionContext, [302](#)
- setBlendFactor
 - nvinfer1::IPoolingLayer, [537](#)
- setBoundingBoxFormat
 - nvinfer1::INMSLayer, [447](#)
- setBroadcastAcrossBatch
 - nvinfer1::ITensor, [629](#)
- setBuilderOptimizationLevel
 - nvinfer1::IBuilderConfig, [170](#)

- setCalibrationProfile
 - nvinfer1::IBuilderConfig, 171
- setCellState
 - nvinfer1::IRNNv2Layer, 579
- setChannelAxis
 - nvinfer1::IScaleLayer, 600
- setComputePrecision
 - nvinfer1::INormalizationLayer, 455
- setCondition
 - nvinfer1::IIfConditional, 358
- setCoordinateTransformation
 - nvinfer1::IResizeLayer, 566
- setCubicCoeff
 - nvinfer1::IResizeLayer, 567
- setDebugSync
 - nvinfer1::IExecutionContext, 303
- setDefaultDeviceType
 - nvinfer1::IBuilderConfig, 171
- setDeviceMemory
 - nvinfer1::IExecutionContext, 303
 - nvinfer1::safe::IExecutionContext, 321
- setDeviceType
 - nvinfer1::IBuilderConfig, 172
- setDilation
 - nvinfer1::IConvolutionLayer, 202
- setDilationNd
 - nvinfer1::IConvolutionLayer, 202
 - nvinfer1::IDeconvolutionLayer, 257
- setDimensionName
 - nvinfer1::ITensor, 630
- setDimensions
 - nvinfer1::IConstantLayer, 194
 - nvinfer1::IFillLayer, 332
 - nvinfer1::IOptimizationProfile, 469
 - nvinfer1::ITensor, 630
- setDirection
 - nvinfer1::IRNNv2Layer, 579
- setDLACore
 - nvinfer1::IBuilderConfig, 172
 - nvinfer1::IRuntime, 589
- setDynamicRange
 - nvinfer1::IRefitter, 558
 - nvinfer1::ITensor, 631
- setEngineCapability
 - nvinfer1::IBuilderConfig, 173
- setEngineHostCodeAllowed
 - nvinfer1::IRuntime, 589
- setEnqueueEmitsProfile
 - nvinfer1::IExecutionContext, 304
- setEpsilon
 - nvinfer1::INormalizationLayer, 455
- setEquation
 - nvinfer1::IEinsumLayer, 269
- setErrorBuffer
 - nvinfer1::safe::IExecutionContext, 322
- setErrorRecorder
 - nvcaffeparser1::ICaffeParser, 183
 - nvinfer1::IBuilder, 153
 - nvinfer1::ICudaEngine, 232
 - nvinfer1::IEngineInspector, 276
 - nvinfer1::IExecutionContext, 304
 - nvinfer1::INetworkDefinition, 442
 - nvinfer1::IPluginRegistry, 501
 - nvinfer1::IRefitter, 559
 - nvinfer1::IRuntime, 589
 - nvinfer1::safe::ICudaEngine, 250
 - nvinfer1::safe::IExecutionContext, 323
 - nvinfer1::safe::IRuntime, 596
 - nvuffparser::IUffParser, 644
- setExcludeOutside
 - nvinfer1::IResizeLayer, 567
- setExecutionContext
 - nvinfer1::IEngineInspector, 276
- setExtraMemoryTarget
 - nvinfer1::IOptimizationProfile, 470
- setFirstTranspose
 - nvinfer1::IShuffleLayer, 610
- setFlag
 - nvinfer1::IBuilderConfig, 173
- setFlags
 - nvinfer1::IBuilderConfig, 173
- setFullTextFileName
 - nvonnxparser::IOnnxConfig, 463
- setGatherAxis
 - nvinfer1::IGatherLayer, 342
- setGpuAllocator
 - nvinfer1::IBuilder, 154
 - nvinfer1::IRuntime, 590
 - nvinfer1::safe::IRuntime, 596
- setHardwareCompatibilityLevel
 - nvinfer1::IBuilderConfig, 174
- setHiddenState
 - nvinfer1::IRNNv2Layer, 579
- setInput
 - nvinfer1::IConvolutionLayer, 203
 - nvinfer1::IDeconvolutionLayer, 258
 - nvinfer1::IFillLayer, 333
 - nvinfer1::IFullyConnectedLayer, 337
 - nvinfer1::ILayer, 380
 - nvinfer1::ILoopOutputLayer, 391
 - nvinfer1::INMSLayer, 447
 - nvinfer1::IRecurrenceLayer, 548
 - nvinfer1::IResizeLayer, 567
 - nvinfer1::IShuffleLayer, 611
 - nvinfer1::ISliceLayer, 616
 - nvinfer1::ITopKLayer, 638
- setInputConsumedEvent
 - nvinfer1::IExecutionContext, 305

- nvinfer1::safe::IExecutionContext, [323](#)
- setInputMode
 - nvinfer1::IRNNv2Layer, [580](#)
- setInputShape
 - nvinfer1::IExecutionContext, [305](#)
- setInputShapeBinding
 - nvinfer1::IExecutionContext, [306](#)
- setInputTensorAddress
 - nvinfer1::IExecutionContext, [306](#)
 - nvinfer1::safe::IExecutionContext, [324](#)
- setInt8Calibrator
 - nvinfer1::IBuilderConfig, [174](#)
- setInterpolationMode
 - nvinfer1::IGridSampleLayer, [350](#)
- setK
 - nvinfer1::ILRNLayer, [395](#)
 - nvinfer1::ITopKLayer, [638](#)
- setKeepDimensions
 - nvinfer1::IReduceLayer, [551](#)
- setKernelSize
 - nvinfer1::IConvolutionLayer, [203](#)
 - nvinfer1::IDeconvolutionLayer, [258](#)
- setKernelSizeNd
 - nvinfer1::IConvolutionLayer, [204](#)
 - nvinfer1::IDeconvolutionLayer, [259](#)
- setKernelWeights
 - nvinfer1::IConvolutionLayer, [204](#)
 - nvinfer1::IDeconvolutionLayer, [259](#)
 - nvinfer1::IFullyConnectedLayer, [338](#)
- setLocation
 - nvinfer1::ITensor, [631](#)
- setMaxBatchSize
 - nvinfer1::IBuilder, [154](#)
- setMaxThreads
 - nvinfer1::IBuilder, [155](#)
 - nvinfer1::IRefitter, [559](#)
 - nvinfer1::IRuntime, [590](#)
- setMaxWorkspaceSize
 - nvinfer1::IBuilderConfig, [174](#)
- setMemoryPoolLimit
 - nvinfer1::IBuilderConfig, [175](#)
- setMessage
 - nvinfer1::IAssertionLayer, [142](#)
- setMinTimingIterations
 - nvinfer1::IBuilderConfig, [175](#)
- setMode
 - nvinfer1::IGatherLayer, [343](#)
 - nvinfer1::IScaleLayer, [600](#)
 - nvinfer1::IScatterLayer, [604](#)
 - nvinfer1::ISliceLayer, [617](#)
- setModelDtype
 - nvonnxparser::IOnnxConfig, [464](#)
- setModelFileName
 - nvonnxparser::IOnnxConfig, [464](#)
- setName
 - nvinfer1::IExecutionContext, [307](#)
 - nvinfer1::IIfConditional, [358](#)
 - nvinfer1::ILayer, [380](#)
 - nvinfer1::ILoop, [387](#)
 - nvinfer1::INetworkDefinition, [442](#)
 - nvinfer1::ITensor, [631](#)
 - nvinfer1::safe::IExecutionContext, [325](#)
- setNamedWeights
 - nvinfer1::IRefitter, [559](#)
- setNbElementWiseDims
 - nvinfer1::IGatherLayer, [343](#)
- setNbGroups
 - nvinfer1::IConvolutionLayer, [204](#)
 - nvinfer1::IDeconvolutionLayer, [259](#)
 - nvinfer1::INormalizationLayer, [456](#)
- setNbOutputChannels
 - nvinfer1::IFullyConnectedLayer, [338](#)
- setNbOutputMaps
 - nvinfer1::IConvolutionLayer, [205](#)
 - nvinfer1::IDeconvolutionLayer, [260](#)
- setNearestRounding
 - nvinfer1::IResizeLayer, [568](#)
- setNvtxVerbosity
 - nvinfer1::IExecutionContext, [307](#)
- setOperation
 - nvinfer1::IElementWiseLayer, [271](#)
 - nvinfer1::IFillLayer, [334](#)
 - nvinfer1::IMatrixMultiplyLayer, [398](#)
 - nvinfer1::IReduceLayer, [551](#)
 - nvinfer1::IRNNv2Layer, [580](#)
 - nvinfer1::ITopKLayer, [638](#)
 - nvinfer1::IUnaryLayer, [646](#)
- setOptimizationProfile
 - nvinfer1::IExecutionContext, [308](#)
- setOptimizationProfileAsync
 - nvinfer1::IExecutionContext, [309](#)
- setOutputAllocator
 - nvinfer1::IExecutionContext, [310](#)
- setOutputDimensions
 - nvinfer1::IResizeLayer, [568](#)
- setOutputTensorAddress
 - nvinfer1::safe::IExecutionContext, [325](#)
- setOutputType
 - nvinfer1::ILayer, [381](#)
- setPadding
 - nvinfer1::IConvolutionLayer, [205](#)
 - nvinfer1::IDeconvolutionLayer, [260](#)
 - nvinfer1::IPoolingLayer, [537](#)
- setPaddingMode
 - nvinfer1::IConvolutionLayer, [206](#)
 - nvinfer1::IDeconvolutionLayer, [261](#)
 - nvinfer1::IPoolingLayer, [537](#)
- setPaddingNd

- [nvinfer1::IConvolutionLayer](#), 206
 - [nvinfer1::IDeconvolutionLayer](#), 261
 - [nvinfer1::IPoolingLayer](#), 538
- [setParentSearchEnabled](#)
 - [nvinfer1::IPluginRegistry](#), 502
- [setPersistentCacheLimit](#)
 - [nvinfer1::IExecutionContext](#), 311
- [setPluginFactoryV2](#)
 - [nvcaffeparser1::ICaffeParser](#), 184
- [setPluginNamespace](#)
 - [nvcaffeparser1::ICaffeParser](#), 184
 - [nvinfer1::IPluginCreator](#), 493
 - [nvinfer1::IPluginV2](#), 511
 - [nvuffparser::IUffParser](#), 645
- [setPluginsToSerialize](#)
 - [nvinfer1::IBuilderConfig](#), 176
- [setPoolingType](#)
 - [nvinfer1::IPoolingLayer](#), 538
- [setPostPadding](#)
 - [nvinfer1::IConvolutionLayer](#), 206
 - [nvinfer1::IDeconvolutionLayer](#), 261
 - [nvinfer1::IPaddingLayer](#), 476
 - [nvinfer1::IPoolingLayer](#), 538
- [setPostPaddingNd](#)
 - [nvinfer1::IPaddingLayer](#), 477
- [setPower](#)
 - [nvinfer1::IScaleLayer](#), 601
- [setPrecision](#)
 - [nvinfer1::ILayer](#), 382
- [setPrePadding](#)
 - [nvinfer1::IConvolutionLayer](#), 207
 - [nvinfer1::IDeconvolutionLayer](#), 262
 - [nvinfer1::IPaddingLayer](#), 477
 - [nvinfer1::IPoolingLayer](#), 539
- [setPrePaddingNd](#)
 - [nvinfer1::IPaddingLayer](#), 477
- [setPreviewFeature](#)
 - [nvinfer1::IBuilderConfig](#), 176
- [setPrintLayerInfo](#)
 - [nvonnxparser::IOnnxConfig](#), 464
- [setProfiler](#)
 - [nvinfer1::IExecutionContext](#), 311
- [setProfileStream](#)
 - [nvinfer1::IBuilderConfig](#), 177
- [setProfilingVerbosity](#)
 - [nvinfer1::IBuilderConfig](#), 177
- [setProtobufBufferSize](#)
 - [nvcaffeparser1::ICaffeParser](#), 184
- [setQuantizationFlag](#)
 - [nvinfer1::IBuilderConfig](#), 177
- [setQuantizationFlags](#)
 - [nvinfer1::IBuilderConfig](#), 178
- [setReduceAxes](#)
 - [nvinfer1::IReduceLayer](#), 551
- [nvinfer1::ITopKLayer](#), 639
- [setReshapeDimensions](#)
 - [nvinfer1::IShuffleLayer](#), 611
- [setResizeMode](#)
 - [nvinfer1::IResizeLayer](#), 569
- [setReverse](#)
 - [nvinfer1::IIteratorLayer](#), 373
- [setSampleMode](#)
 - [nvinfer1::IGridSampleLayer](#), 351
- [setScale](#)
 - [nvinfer1::IScaleLayer](#), 601
- [setScales](#)
 - [nvinfer1::IResizeLayer](#), 569
- [setSecondTranspose](#)
 - [nvinfer1::IShuffleLayer](#), 612
- [setSelectorForSinglePixel](#)
 - [nvinfer1::IResizeLayer](#), 570
- [setSequenceAxis](#)
 - [nvinfer1::IRReverseSequenceLayer](#), 573
- [setSequenceLengths](#)
 - [nvinfer1::IRNNv2Layer](#), 580
- [setShapeValues](#)
 - [nvinfer1::IOptimizationProfile](#), 470
- [setShift](#)
 - [nvinfer1::IScaleLayer](#), 601
- [setSize](#)
 - [nvinfer1::ISliceLayer](#), 617
- [setStart](#)
 - [nvinfer1::ISliceLayer](#), 618
- [setStride](#)
 - [nvinfer1::IConvolutionLayer](#), 207
 - [nvinfer1::IDeconvolutionLayer](#), 262
 - [nvinfer1::IPoolingLayer](#), 539
 - [nvinfer1::ISliceLayer](#), 618
- [setStrideNd](#)
 - [nvinfer1::IConvolutionLayer](#), 207
 - [nvinfer1::IDeconvolutionLayer](#), 262
 - [nvinfer1::IPoolingLayer](#), 539
- [setTacticSources](#)
 - [nvinfer1::IBuilderConfig](#), 178
- [setTempfileControlFlags](#)
 - [nvinfer1::IRuntime](#), 591
- [setTemporaryDirectory](#)
 - [nvinfer1::IRuntime](#), 591
- [setTemporaryStorageAllocator](#)
 - [nvinfer1::IExecutionContext](#), 311
- [setTensorAddress](#)
 - [nvinfer1::IExecutionContext](#), 312
- [setTextFileName](#)
 - [nvonnxparser::IOnnxConfig](#), 465
- [setTimingCache](#)
 - [nvinfer1::IBuilderConfig](#), 179
- [setTopKBoxLimit](#)
 - [nvinfer1::INMSLayer](#), 448

- setToType
 - nvinfer1::ICastLayer, [186](#)
- setType
 - nvinfer1::ITensor, [632](#)
- setVerbosityLevel
 - nvonnxparser::IOnnxConfig, [465](#)
- setWeights
 - nvinfer1::IConstantLayer, [194](#)
 - nvinfer1::IRefitter, [560](#)
- setWeightsForGate
 - nvinfer1::IRNNv2Layer, [581](#)
- setWeightsName
 - nvinfer1::INetworkDefinition, [443](#)
- setWindowSize
 - nvinfer1::ILRNLayer, [396](#)
 - nvinfer1::IPoolingLayer, [540](#)
- setWindowSizeNd
 - nvinfer1::IPoolingLayer, [540](#)
- setZeroIsPlaceholder
 - nvinfer1::IShuffleLayer, [612](#)
- Severity
 - nvinfer1::ILogger, [383](#)
- shareLocation
 - nvinfer1::plugin::DetectionOutputParameters, [95](#)
 - nvinfer1::plugin::NMSParameters, [649](#)
- shutdownProtobufLibrary
 - nvcaffeparser1, [28](#)
 - nvuffparser, [92](#)
- size
 - nvinfer1::IHostMemory, [353](#)
- SliceMode
 - nvinfer1, [40](#)
- smTree
 - nvinfer1::plugin::RegionParameters, [661](#)
- spatialScale
 - nvinfer1::plugin::RPROIParams, [663](#)
- stepH
 - nvinfer1::plugin::PriorBoxParameters, [659](#)
- stepW
 - nvinfer1::plugin::PriorBoxParameters, [659](#)
- SubGraph_t
 - NvOnnxParser.h, [780](#)
- SubGraphCollection_t
 - NvOnnxParser.h, [780](#)
- supportsFormat
 - nvinfer1::IPluginV2, [511](#)
- supportsFormatCombination
 - nvinfer1::IPluginV2DynamicExt, [518](#)
 - nvinfer1::IPluginV2IOExt, [528](#)
- supportsModel
 - nvonnxparser::IParser, [483](#)
- supportsOperator
 - nvonnxparser::IParser, [484](#)
- TacticSource
 - nvinfer1, [67](#)
- TacticSources
 - nvinfer1, [40](#)
- TempfileControlFlag
 - nvinfer1, [68](#)
- TempfileControlFlags
 - nvinfer1, [40](#)
- TensorFormat
 - nvinfer1, [68](#)
- TensorFormats
 - nvinfer1, [40](#)
- TensorIOMode
 - nvinfer1, [71](#)
- TensorLocation
 - nvinfer1, [71](#)
- TENSORRTAPI
 - NvInferRuntimeCommon.h, [757](#)
- terminate
 - nvinfer1::IPluginV2, [512](#)
- TF_CENTER
 - nvinfer1::plugin, [84](#)
- topK
 - nvinfer1::plugin::DetectionOutputParameters, [96](#)
 - nvinfer1::plugin::NMSParameters, [649](#)
- TopKOperation
 - nvinfer1, [71](#)
- transposeSubBuffers
 - nvinfer1::utils, [88](#)
- TripLimit
 - nvinfer1, [72](#)
- TRT_DEPRECATED
 - NvInferRuntimeCommon.h, [757](#)
- TRT_DEPRECATED_API
 - NvInferRuntimeCommon.h, [757](#)
- TRT_DEPRECATED_ENUM
 - NvInferRuntimeCommon.h, [757](#)
- TRTNOEXCEPT
 - NvInferRuntimeCommon.h, [757](#)
- type
 - nvinfer1::IHostMemory, [353](#)
 - nvinfer1::PluginField, [651](#)
 - nvinfer1::PluginTensorDesc, [655](#)
 - nvinfer1::Weights, [666](#)
 - nvuffparser::FieldMap, [122](#)
- UFF_REQUIRED_VERSION_MAJOR
 - NvUffParser.h, [775](#)
- UFF_REQUIRED_VERSION_MINOR
 - NvUffParser.h, [775](#)
- UFF_REQUIRED_VERSION_PATCH
 - NvUffParser.h, [775](#)
- UffInputOrder
 - nvuffparser, [91](#)

- UnaryOperation
 - `nvinfer1`, [72](#)
- unmarkOutput
 - `nvinfer1::INetworkDefinition`, [443](#)
- unmarkOutputForShapes
 - `nvinfer1::INetworkDefinition`, [444](#)
- validate
 - `nvinfer1::consistency::IConsistencyChecker`, [192](#)
 - `nvinfer1::consistency::IPluginChecker`, [488](#)
- values
 - `nvinfer1::Weights`, [666](#)
- variance
 - `nvinfer1::plugin::GridAnchorParameters`, [124](#)
 - `nvinfer1::plugin::PriorBoxParameters`, [659](#)
- varianceEncodedInTarget
 - `nvinfer1::plugin::DetectionOutputParameters`, [96](#)
- Verbosity
 - `nvonnxparser::IOnnxConfig`, [460](#)
- W
 - `nvinfer1::plugin::GridAnchorParameters`, [125](#)
- w
 - `nvinfer1::DimsHW`, [105](#)
- WeightsRole
 - `nvinfer1`, [73](#)
- writeCalibrationCache
 - `nvinfer1::IInt8Calibrator`, [366](#)
- writeHistogramCache
 - `nvinfer1::IInt8LegacyCalibrator`, [370](#)

NVIDIA TensorRT Standard Python API Documentation

Release 8.6.0

NVIDIA

Dec 15, 2022

TENSORRT PYTHON API REFERENCE

1	Getting Started with TensorRT	1
1.1	Installation	1
1.2	Samples	1
1.3	Installing PyCUDA	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	Foundational Types	5
3.1	DataType	5
3.2	Weights	6
3.3	Dims	6
3.3.1	Volume	6
3.3.2	Dims	7
3.3.3	Dims2	7
3.3.4	DimsHW	7
3.3.5	Dims3	8
3.3.6	Dims4	8
3.4	IHostMemory	8
4	Core	9
4.1	Logger	9
4.2	Profiler	10
4.3	IOptimizationProfile	11
4.4	IBuilderConfig	13
4.5	Builder	20
4.5.1	NetworkDefinitionCreationFlag	20
4.5.2	Builder	21
4.6	ICudaEngine	23
4.7	IExecutionContext	29
4.8	Runtime	36
4.9	Refitter	37
4.10	IErrorRecorder	39
4.11	ITimingCache	41

4.12	GPU Allocator	42
4.12.1	AllocatorFlag	42
4.12.2	IGpuAllocator	42
4.13	EngineInspector	43
5	Network	45
5.1	INetworkDefinition	45
5.2	Layer Base Classes	59
5.2.1	ITensor	59
5.2.2	ILayer	62
5.3	Layers	64
5.3.1	PaddingMode	64
5.3.2	IConvolutionLayer	65
5.3.3	IFullyConnectedLayer	66
5.3.4	IGridSampleLayer	66
5.3.5	IActivationLayer	67
5.3.6	IPoolingLayer	68
5.3.7	ILRNLayer	69
5.3.8	IScaleLayer	69
5.3.9	ISoftMaxLayer	70
5.3.10	IConcatenationLayer	71
5.3.11	IDeconvolutionLayer	71
5.3.12	IElementWiseLayer	72
5.3.13	IGatherLayer	72
5.3.14	RNN Layers	73
5.3.14.1	IRNNv2Layer	75
5.3.15	IPluginV2Layer	77
5.3.16	IUnaryLayer	77
5.3.17	IReduceLayer	78
5.3.18	IPaddingLayer	78
5.3.19	IParametricReLULayer	79
5.3.20	ISelectLayer	79
5.3.21	IShuffleLayer	79
5.3.22	ISliceLayer	80
5.3.23	IShapeLayer	81
5.3.24	ITopKLayer	82
5.3.25	IMatrixMultiplyLayer	82
5.3.26	IRaggedSoftMaxLayer	83
5.3.27	IIdentityLayer	83
5.3.28	IConstantLayer	83
5.3.29	IResizeLayer	84
5.3.30	ILoop	85
5.3.30.1	ILoopBoundaryLayer	86
5.3.30.1.1	ITripLimitLayer	86
5.3.30.1.2	IRecurrenceLayer	87
5.3.30.1.3	IIteratorLayer	87
5.3.30.1.4	ILoopOutputLayer	87
5.3.31	IFillLayer	88
5.3.32	IQuantizeLayer	89
5.3.33	IDequantizeLayer	89
5.3.34	IScatterLayer	90
5.3.35	IIfConditional	90
5.3.36	IConditionalLayer	91
5.3.37	IIfConditionalOutputLayer	91

5.3.38	IIfConditionalInputLayer	91
5.3.39	IEinsumLayer	91
5.3.40	IAssertionLayer	92
5.3.41	IOneHotLayer	92
5.3.42	INonZeroLayer	92
5.3.43	INMSLayer	92
5.3.44	IRReverseSequenceLayer	94
5.3.45	INormalizationLayer	94
6	Plugin	95
6.1	IPluginCreator	95
6.2	IPluginRegistry	97
7	Int8	99
7.1	IInt8Calibrator	99
7.2	IInt8LegacyCalibrator	101
7.3	IInt8EntropyCalibrator	102
7.4	IInt8EntropyCalibrator2	104
7.5	IInt8MinMaxCalibrator	105
8	Algorithm Selector	107
9	UFF Parser	111
9.1	Fields	112
10	Caffe Parser	115
10.1	Plugins	116
11	Onnx Parser	117
12	UFF Converter	121
12.1	Conversion Tools	121
12.1.1	Tensorflow Modelstream to UFF	121
12.1.2	Tensorflow Frozen Protobuf Model to UFF	122
13	UFF Operators	123
13.1	Input	123
13.1.1	Supported Datatypes	123
13.2	Identity	123
13.2.1	Inputs	123
13.2.2	Supported Datatypes	123
13.3	Const	123
13.3.1	Supported Datatypes	124
13.4	Conv	124
13.4.1	Inputs	124
13.4.2	Attributes	124
13.4.3	Supported Datatypes	124
13.5	ConvTranspose	124
13.5.1	Inputs	124
13.5.2	Attributes	125
13.5.3	Supported Datatypes	125
13.6	Pool	125
13.6.1	Inputs	125
13.6.2	Attributes	125
13.6.3	Supported Datatypes	125

13.7	FullyConnected	125
13.7.1	Inputs	125
13.7.2	Supported Datatypes	126
13.8	LRN	126
13.8.1	Inputs	126
13.8.2	Attributes	126
13.8.3	Supported Datatypes	126
13.9	Binary	126
13.9.1	Inputs	126
13.9.2	Attributes	126
13.9.3	Supported Datatypes	127
13.10	Unary	127
13.10.1	Inputs	127
13.10.2	Attributes	127
13.10.3	Supported Datatypes	127
13.11	Reshape	127
13.11.1	Inputs	127
13.11.2	Supported Datatypes	128
13.12	ExpandDims	128
13.12.1	Inputs	128
13.12.2	Attributes	128
13.12.3	Supported Datatypes	128
13.13	ArgMax	128
13.13.1	Inputs	128
13.13.2	Attributes	128
13.13.3	Supported Datatypes	129
13.14	ArgMin	129
13.14.1	Inputs	129
13.14.2	Attributes	129
13.14.3	Supported Datatypes	129
13.15	Transpose	129
13.15.1	Inputs	129
13.15.2	Attributes	129
13.15.3	Supported Datatypes	129
13.16	Reduce	130
13.16.1	Inputs	130
13.16.2	Attributes	130
13.16.3	Supported Datatypes	130
13.17	Concat	130
13.17.1	Inputs	130
13.17.2	Attributes	130
13.17.3	Supported Datatypes	131
13.18	MarkOutput	131
13.18.1	Inputs	131
13.18.2	Supported Datatypes	131
13.19	Activation	131
13.19.1	Inputs	131
13.19.2	Attributes	131
13.19.3	Supported Datatypes	131
13.20	Softmax	131
13.20.1	Inputs	132
13.20.2	Attributes	132
13.20.3	Supported Datatypes	132
13.21	BatchNorm	132

13.21.1	Inputs	132
13.21.2	Attributes	132
13.21.3	Supported Datatypes	132
13.22	Shape	132
13.22.1	Inputs	133
13.22.2	Supported Datatypes	133
13.23	StridedSlice	133
13.23.1	Inputs	133
13.23.2	Attributes	133
13.23.3	Supported Datatypes	133
13.24	Stack	133
13.24.1	Inputs	134
13.24.2	Attributes	134
13.24.3	Supported Datatypes	134
13.25	Squeeze	134
13.26	Flatten	134
13.26.1	Inputs	134
13.26.2	Supported Datatypes	134
13.27	Pad	134
13.27.1	Inputs	134
13.27.2	Supported Datatypes	135
13.28	Gather	135
13.28.1	Inputs	135
13.28.2	Supported Datatypes	135
13.29	GatherV2	135
13.29.1	Inputs	135
13.29.2	Attributes	135
13.29.3	Supported Datatypes	135
14	Graph Surgeon	137
14.1	Node Creation	137
14.2	Static Graph	138
14.3	Dynamic Graph (Inherits from StaticGraph)	140
Index		143

GETTING STARTED WITH TENSORRT

1.1 Installation

For installation instructions, please refer to <https://docs.nvidia.com/deeplearning/sdk/tensorrt-install-guide/index.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 Installing PyCUDA

Although not required by the TensorRT Python API, PyCUDA is used in several samples. For installation instructions, please refer to <https://wiki.tiker.net/PyCuda/Installation>

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.

FOUNDATIONAL TYPES

3.1 DataType

`tensorrt.DataType`

Represents data types.

ivar `itemsz` `int` The size in bytes of this *DataType* .

Members:

FLOAT : 32-bit floating point format.

HALF : IEEE 16-bit floating-point format.

INT8 : Signed 8-bit integer representing a quantized floating-point value.

INT32 : Signed 32-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` }.

TensorRT also exposes some short-hand, NumPy-style `DataType` aliases that can be used across the library:

Type	Alias
<code>tensorrt.DataType.FLOAT</code>	<code>tensorrt.float32</code>
<code>tensorrt.DataType.HALF</code>	<code>tensorrt.float16</code>
<code>tensorrt.DataType.INT32</code>	<code>tensorrt.int32</code>
<code>tensorrt.DataType.INT8</code>	<code>tensorrt.int8</code>
<code>tensorrt.DataType.BOOL</code>	<code>tensorrt.bool</code>

`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.

3.2 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` , `IDeconvolutionLayer` , or `IFullyConnectedLayer` .

BIAS : Bias for `IConvolutionLayer` , `IDeconvolutionLayer` , or `IFullyConnectedLayer` .

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, a: numpy.ndarray) -> None`

a A numpy array whose values to use. No deep copies are made.

numpy(self: `tensorrt.tensorrt.Weights`) → `numpy.ndarray`

Create a numpy array using the underlying buffer of this weights object.

Returns A new numpy array that holds a reference to this weight object's buffer - no deep copy is made.

3.3 Dims

3.3.1 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.

3.3.2 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: List[int]) -> None`

property `MAX_DIMS`

The maximum number of dimensions supported by *Dims*.

3.3.3 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: int, dim1: int) -> None`
3. `__init__(self: tensorrt.tensorrt.Dims2, shape: List[int]) -> None`

3.3.4 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: int, w: int) -> None`
3. `__init__(self: tensorrt.tensorrt.DimsHW, shape: List[int]) -> None`

3.3.5 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: int, dim1: int, dim2: int) -> None`
3. `__init__(self: tensorrt.tensorrt.Dims3, shape: List[int]) -> None`

3.3.6 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: int, dim1: int, dim2: int, dim3: int) -> None`
3. `__init__(self: tensorrt.tensorrt.Dims4, shape: List[int]) -> None`

3.4 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.

`__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)`

4.1 Logger

class `tensorrt.ILogger`(*self*: `tensorrt.tensorrt.ILogger`) → None
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*: `int`) → None

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*: `nvinfer1::ILogger::Severity`, *msg*: `str`) → None
Logs a message to `stderr` . This function must be overridden 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>`) → None

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 `stderr` .

Parameters

- **severity** – The severity of the message.
- **msg** – The log message.

4.2 Profiler

class `tensorrt.IProfiler`(*self*: `tensorrt.tensorrt.IProfiler`) → None

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\(\)](#) or [IExecutionContext.execute_async_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*: `float`) → None

Reports time in milliseconds for each layer. This function must be overridden a derived class.

Parameters

- **layer_name** – The name of the layer, set when constructing the [INetworkDefinition](#) .
- **ms** – The time in milliseconds to execute the layer.

class `tensorrt.Profiler`(*self*: `tensorrt.tensorrt.Profiler`) → None

When this class is added to an [IExecutionContext](#), the profiler will be called once per layer for each invocation of [IExecutionContext.execute_v2\(\)](#) or [IExecutionContext.execute_async_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.Profiler](#), *layer_name*: *str*, *ms*: *float*) → 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.

4.3 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 at 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) $\text{len}(\text{min}) == \text{len}(\text{opt}) == \text{len}(\text{max}) == \text{len}(\text{real_shape})$
- (2) $0 \leq \text{min}[i] \leq \text{opt}[i] \leq \text{max}[i]$ for all *i*
- (3) if $\text{real_shape}[i] \neq -1$, then $\text{min}[i] == \text{opt}[i] == \text{max}[i] == \text{real_shape}[i]$

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*: *List[int]*, *opt*: *List[int]*, *max*: *List[int]*) → 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 `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 $\text{min}[i] \leq \text{opt}[i] \leq \text{max}[i]$ 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.

4.4 IBuilderConfig

tensorrt.QuantizationFlag

List of valid flags for quantizing the network to int8.

Members:

CALIBRATE_BEFORE_FUSION : Run int8 calibration pass before layer fusion. Only valid for IInt8LegacyCalibrator and IInt8EntropyCalibrator. We always run int8 calibration pass before layer fusion for IInt8MinMaxCalibrator and IInt8EntropyCalibrator2. Disabled by default.

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.

DEFAULT : [DEPRECATED] Same as **LAYER_NAMES_ONLY**.

VERBOSE : [DEPRECATED] Same as **DETAILED**.

tensorrt.TacticSource

Tactic sources that can provide tactics for TensorRT.

Members:

CUBLAS : Enables cuBLAS tactics. Enabled by default. **NOTE:** Disabling this value will cause the cublas handle passed to plugins in attachToContext to be null.

CUBLAS_LT : Enables cuBLAS LT tactics. Enabled for x86 platforms and only enabled for non-x86 platforms when CUDA \geq 11.0 by default

CUDNN : Enables cuDNN tactics. Enabled by default.

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. When BuilderFlag::kSAFETY_SCOPE is not set (by default), 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 *nvinfer1::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:

DEFAULT : [DEPRECATED] Unrestricted: TensorRT mode without any restrictions using TensorRT `nvInfer1` APIs.

SAFE_GPU : [DEPRECATED] Safety-restricted: TensorRT mode for GPU devices using TensorRT safety APIs. See safety documentation for list of supported layers and formats.

SAFE_DLA : [DEPRECATED] DLA-restricted: TensorRT mode for DLA devices using cuDLA APIs. Only FP16 and Int8 modes are supported.

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:

FP16 : Enable FP16 layer selection

INT8 : Enable Int8 layer selection

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

STRICT_TYPES : [DEPRECATED] Enables strict type constraints. Equivalent to setting `PREFER_PRECISION_CONSTRAINTS`, `DIRECT_IO`, and `REJECT_EMPTY_ALGORITHMS`.

REFIT : Enable building a refittable engine

DISABLE_TIMING_CACHE : Disable reuse of timing information across identical layers.

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.

OBEY_PRECISION_CONSTRAINTS : Require that layers execute in specified precisions. Build fails otherwise.

PREFER_PRECISION_CONSTRAINTS : Prefer that layers execute in specified precisions. Fall back (with warning) to another precision if build would otherwise fail.

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.

REJECT_EMPTY_ALGORITHMS : Fail if `IAgorithmSelector.select_algorithms` returns an empty set of algorithms.

ENABLE_TACTIC_HEURISTIC : Enable heuristic-based tactic selection for shorter engine generation time. The performance of the generated engine may not be as performant as a profiling-based builder.

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.

FP8 : Enable FP8 layer selection

`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.

They are provided as opt-in features for at least one release. For example, to enable faster dynamic shapes, call `set_preview_feature()` with `PreviewFeature.FASTER_DYNAMIC_SHAPES_0805`

Members:

FASTER_DYNAMIC_SHAPES_0805 : Optimize runtime dimensions with TensorRT's DL Compiler. Potentially reduces run time and decreases device memory usage and engine size. Models most likely to benefit from enabling **FASTER_DYNAMIC_SHAPES_0805** are transformer-based models, and models containing dynamic control flows.

DISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805 : Disable usage of cuDNN/cuBLAS/cuBLASLt tactics in the TensorRT core library. When the flag is enabled, TensorRT core will not use these tactics even if they are specified in `set_tactic_sources`, but `cudaContext` and `cublasContext` handles will still be passed to plugins via `IPluginV2::attachToContext()` if the appropriate tactic sources are set. This allows users to experiment with disabling external library tactics without having to modify their application's plugins to support nullptr handles. The default value for this flag is off.

PROFILE_SHARING_0806 : Allows optimization profiles to be shared across execution contexts. This will become the default behavior in TensorRT 9.0 and the flag defaults to false.

ENABLE_MULTI_STREAM_ENGINE_0806 : Build an engine with multi-stream data. Using multi-stream increases layer dispatching overhead and the activation memory consumption. Users shall decide which configuration is better based on experimentation. Using CUDA graphs can help to reduce dispatching overheads.

`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 is equivalent to the deprecated `IBuilderConfig.max_workspace_size` and overrides that value. This defaults to max device memory. Set to a smaller value to restrict tactics that use over the threshold en masse. For more targeted removal of tactics use the `IAgorithmSelector` interface.

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, and Xavier shares 4 MiB across all of its accelerator cores.

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: kTACTIC_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.

`class tensorrt.IBuilderConfig`

Variables

- **min_timing_iterations** – int [DEPRECATED] The number of minimization 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 minimization. By default the minimum number of iterations is 1.
- **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.
- **int8_calibrator** – [*IInt8Calibrator*](#) Int8 Calibration interface. The calibrator is to minimize the information loss during the INT8 quantization process.
- **max_workspace_size** – int [DEPRECATED] The maximum workspace size. The maximum GPU temporary memory which the engine can use at execution time.
- **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.
- **algorithm_selector** – The [*IAlgorithmSelector*](#) to use.
- **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 2. Valid values include integers from 0 to the maximum optimization level, which is currently 4. Setting it to be greater than the maximum level results in identical behavior to the maximum level.

- **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.

__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.

clear_quantization_flag(*self*: [tensorrt.tensorrt.IBuilderConfig](#), *flag*: [tensorrt.tensorrt.QuantizationFlag](#)) → None

Clears the quantization flag from the enabled quantization flags.

Parameters **flag** – The flag to clear.

create_timing_cache(*self*: [tensorrt.tensorrt.IBuilderConfig](#), *serialized_timing_cache*: *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_calibration_profile(*self*: [tensorrt.tensorrt.IBuilderConfig](#)) → [tensorrt.tensorrt.IOptimizationProfile](#)

Get the current calibration profile.

Returns The current calibration profile or nullptr if calibration profile is unset.

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_quantization_flag(*self*: [tensorrt.tensorrt.IBuilderConfig](#), *flag*: [tensorrt.tensorrt.QuantizationFlag](#)) → bool

Check if a quantization flag is set.

Parameters **flag** – The flag to check.

Returns A *bool* indicating whether the flag is set.

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_calibration_profile(*self*: [tensorrt.tensorrt.IBuilderConfig](#), *profile*: [tensorrt.tensorrt.IOptimizationProfile](#)) → bool

Set a calibration profile.

Calibration optimization profile must be set if int8 calibration is used to set scales for a network with runtime dimensions.

Parameters **profile** – The new calibration profile, which must satisfy `bool(profile) == True` or be `nullptr`. MIN and MAX values will be overwritten by `kOPT`.

Returns True if the calibration profile was set correctly.

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 `kSAFE_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*: `int`) → 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_quantization_flag(*self*: [tensorrt.tensorrt.IBuilderConfig](#), *flag*: [tensorrt.tensorrt.QuantizationFlag](#))
→ None

Add the input quantization flag to the already enabled quantization flags.

Parameters **flag** – The flag to set.

set_tactic_sources(*self*: [tensorrt.tensorrt.IBuilderConfig](#), *tactic_sources*: *int*) → 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 cublas and cublasLt as tactic sources, use a value of: `1 << int(trt.TacticSource.CUBLAS) | 1 << int(trt.TacticSource.CUBLAS_LT)`

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.

4.5 Builder

4.5.1 NetworkDefinitionCreationFlag

tensorrt.NetworkDefinitionCreationFlag

List of immutable network properties expressed at network creation time. For example, to enable explicit batch mode, pass a value of `1 << int(NetworkDefinitionCreationFlag.EXPLICIT_BATCH)` to `create_network()`

Members:

EXPLICIT_BATCH : Specify that the network should be created with an explicit batch dimension. Creating a network without this flag has been deprecated.

EXPLICIT_PRECISION : [DEPRECATED] This flag has no effect now.

4.5.2 Builder

class `tensorrt.Builder`(*self*: `tensorrt.tensorrt.Builder`, *logger*: `tensorrt.tensorrt.ILogger`) → None
Builds an *ICudaEngine* from a *INetworkDefinition*.

Variables

- **max_batch_size** – int [DEPRECATED] For networks built with implicit batch, the maximum batch size which can be used at execution time, and also the batch size for which the *ICudaEngine* will be optimized. This no effect for networks created with explicit batch dimension mode.
- **platform_has_tf32** – bool Whether the platform has tf32 support.
- **platform_has_fast_fp16** – bool Whether the platform has fast native fp16.
- **platform_has_fast_int8** – bool Whether the platform has fast native int8.
- **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(*self*: `tensorrt.tensorrt.Builder`, *network*: `tensorrt.tensorrt.INetworkDefinition`, *config*: `tensorrt.tensorrt.IBuilderConfig`) → `tensorrt.tensorrt.ICudaEngine`

Builds an engine for the given *INetworkDefinition* and *IBuilderConfig*.

This enables the builder to build multiple engines based on the same network definition, but with different builder configurations.

Parameters

- **network** – The TensorRT *INetworkDefinition*.
- **config** – The TensorRT *IBuilderConfig*.

Returns A new *ICudaEngine*.

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.

Parameters

- **network** – Network definition.
- **config** – Builder configuration.

Returns A pointer to a [IHostMemory](#) object that contains a serialized network.

create_builder_config(*self*: [tensorrt.tensorrt.Builder](#)) → [tensorrt.tensorrt.IBuilderConfig](#)

Create a builder configuration object.

See [IBuilderConfig](#)

create_network(*self*: [tensorrt.tensorrt.Builder](#), *flags*: *int* = 0) → [tensorrt.tensorrt.INetworkDefinition](#)

Create a [INetworkDefinition](#) object.

Parameters **flags** – [NetworkDefinitionCreationFlag](#)s combined using bitwise OR.
Please enable the [NetworkDefinitionCreationFlag.EXPLICIT_BATCH](#) flag whenever possible.

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: 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.

4.6 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_bindings** – int The number of binding indices.
- **num_io_tensors** – int The number of IO tensors.
- **max_batch_size** – int [DEPRECATED] The maximum batch size which can be used for inference for an engine built from an *INetworkDefinition* with implicit batch dimension. For an engine built from an *INetworkDefinition* with explicit batch dimension, this will always be 1.
- **has_implicit_batch_dimension** – bool Whether the engine was built with an implicit batch dimension. This is an engine-wide property. Either all tensors in the engine have an implicit batch dimension or none of them do. This is True if and only if the *INetworkDefinition* from which this engine was built was created without the `NetworkDefinitionCreationFlag.EXPLICIT_BATCH` flag.
- **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** – int The amount of device memory required by an *IExecutionContext*.
- **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.

__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__(*args, **kwargs)

Overloaded function.

1. **__getitem__**(*self*: [tensorrt.tensorrt.ICudaEngine](#), *arg0*: str) -> int

2. **__getitem__**(*self*: [tensorrt.tensorrt.ICudaEngine](#), *arg0*: int) -> str

__init__(*args, **kwargs)

__len__(*self*: [tensorrt.tensorrt.ICudaEngine](#)) → int

binding_is_input(*args, **kwargs)

Overloaded function.

1. **binding_is_input**(*self*: [tensorrt.tensorrt.ICudaEngine](#), *index*: int) -> bool

Determine whether a binding is an input binding.

index The binding index.

returns True if the index corresponds to an input binding and the index is in range.

2. **binding_is_input**(*self*: [tensorrt.tensorrt.ICudaEngine](#), *name*: str) -> bool

Determine whether a binding is an input binding.

name The name of the tensor corresponding to an engine binding.

returns True if the index corresponds to an input binding and the index is in range.

create_engine_inspector(*self*: [tensorrt.tensorrt.ICudaEngine](#)) → [nvinfer1::IEngineInspector](#)

Create an [IEngineInspector](#) which prints out the layer information of an engine or an execution context.

Returns The [IEngineInspector](#).

create_execution_context(*self*: [tensorrt.tensorrt.ICudaEngine](#)) → [tensorrt.tensorrt.IExecutionContext](#)

Create an [IExecutionContext](#).

Returns The newly created [IExecutionContext](#).

create_execution_context_without_device_memory(*self*: [tensorrt.tensorrt.ICudaEngine](#)) → [tensorrt.tensorrt.IExecutionContext](#)

Create an [IExecutionContext](#) without any device memory allocated. The memory for execution of this device context must be supplied by the application.

Returns An [IExecutionContext](#) without device memory allocated.

get_binding_bytes_per_component(*self*: [tensorrt.tensorrt.ICudaEngine](#), *index*: int) → int

Return the number of bytes per component of an element. The vector component size is returned if [get_binding_vectorized_dim\(\)](#) != -1.

Parameters **index** – The binding index.

get_binding_components_per_element(*self*: [tensorrt.tensorrt.ICudaEngine](#), *index*: *int*) → *int*

Return the number of components included in one element.

The number of elements in the vectors is returned if [get_binding_vectorized_dim\(\)](#) != -1.

Parameters *index* – The binding index.

get_binding_dtype(**args*, ***kwargs*)

Overloaded function.

1. [get_binding_dtype](#)(*self*: [tensorrt.tensorrt.ICudaEngine](#), *index*: *int*) → [tensorrt.tensorrt.DataType](#)

Determine the required data type for a buffer from its binding index.

index The binding index.

Returns The type of data in the buffer.

2. [get_binding_dtype](#)(*self*: [tensorrt.tensorrt.ICudaEngine](#), *name*: *str*) → [tensorrt.tensorrt.DataType](#)

Determine the required data type for a buffer from its binding index.

name The name of the tensor corresponding to an engine binding.

Returns The type of data in the buffer.

get_binding_format(*self*: [tensorrt.tensorrt.ICudaEngine](#), *index*: *int*) → [tensorrt.tensorrt.TensorFormat](#)

Return the binding format.

Parameters *index* – The binding index.

get_binding_format_desc(*self*: [tensorrt.tensorrt.ICudaEngine](#), *index*: *int*) → *str*

Return the human readable description of the tensor format.

The description includes the order, vectorization, data type, strides, etc. For example:

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 *index* – The binding index.

get_binding_index(*self*: [tensorrt.tensorrt.ICudaEngine](#), *name*: *str*) → *int*

Retrieve the binding index for a named tensor.

You can also use engine’s [__getitem__\(\)](#) with engine[*name*]. When invoked with a *str*, this will return the corresponding binding index.

[IExecutionContext.execute_async_v2\(\)](#) and [IExecutionContext.execute_v2\(\)](#) require an array of buffers. Engine bindings map from tensor names to indices in this array. Binding indices are assigned at [ICudaEngine](#) build time, and take values in the range [0 ... n-1] where n is the total number of inputs and outputs.

Parameters *name* – The tensor name.

Returns The binding index for the named tensor, or -1 if the name is not found.

get_binding_name(self: `tensorrt.tensorrt.ICudaEngine`, index: `int`) → `str`

Retrieve the name corresponding to a binding index.

You can also use engine's `__getitem__()` with `engine[index]`. When invoked with an `int`, this will return the corresponding binding name.

This is the reverse mapping to that provided by `get_binding_index()`.

Parameters **index** – The binding index.

Returns The name corresponding to the binding index.

get_binding_shape(*args, **kwargs)

Overloaded function.

1. `get_binding_shape(self: tensorrt.tensorrt.ICudaEngine, index: int) -> tensorrt.tensorrt.Dims`

Get the shape of a binding.

index The binding index.

Returns The shape of the binding if the index is in range, otherwise `Dims()`

2. `get_binding_shape(self: tensorrt.tensorrt.ICudaEngine, name: str) -> tensorrt.tensorrt.Dims`

Get the shape of a binding.

name The name of the tensor corresponding to an engine binding.

Returns The shape of the binding if the tensor is present, otherwise `Dims()`

get_binding_vectorized_dim(self: `tensorrt.tensorrt.ICudaEngine`, index: `int`) → `int`

Return the dimension index that the buffer is vectorized.

Specifically -1 is returned if scalars per vector is 1.

Parameters **index** – The binding index.

get_location(*args, **kwargs)

Overloaded function.

1. `get_location(self: tensorrt.tensorrt.ICudaEngine, index: int) -> tensorrt.tensorrt.TensorLocation`

Get location of binding. This lets you know whether the binding should be a pointer to device or host memory.

index The binding index.

returns The location of the bound tensor with given index.

2. `get_location(self: tensorrt.tensorrt.ICudaEngine, name: str) -> tensorrt.tensorrt.TensorLocation`

Get location of binding. This lets you know whether the binding should be a pointer to device or host memory.

name The name of the tensor corresponding to an engine binding.

returns The location of the bound tensor with given index.

get_profile_shape(*args, **kwargs)

Overloaded function.

1. `get_profile_shape(self: tensorrt.tensorrt.ICudaEngine, profile_index: int, binding: int) -> List[tensorrt.tensorrt.Dims]`

Get the minimum/optimum/maximum dimensions for a particular binding under an optimization profile.

arg profile_index The index of the profile.

arg binding The binding index or name.

returns A List[Dims] of length 3, containing the minimum, optimum, and maximum shapes, in that order.

2. `get_profile_shape(self: tensorrt.tensorrt.ICudaEngine, profile_index: int, binding: str) -> List[tensorrt.tensorrt.Dims]`

Get the minimum/optimum/maximum dimensions for a particular binding under an optimization profile.

arg profile_index The index of the profile.

arg binding The binding index or name.

returns A List[Dims] of length 3, containing the minimum, optimum, and maximum shapes, in that order.

get_profile_shape_input(*args, **kwargs)

Overloaded function.

1. `get_profile_shape_input(self: tensorrt.tensorrt.ICudaEngine, profile_index: int, binding: int) -> 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.

arg profile_index The index of the profile.

arg binding The binding index or name.

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.

2. `get_profile_shape_input(self: tensorrt.tensorrt.ICudaEngine, profile_index: int, binding: 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.

arg profile_index The index of the profile.

arg binding The binding index or name.

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_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.

Parameters name – The tensor name.

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.

Parameters 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(*self*: [tensorrt.tensorrt.ICudaEngine](#), *name*: *str*) → [tensorrt.tensorrt.TensorFormat](#)
Return the tensor format.

Parameters *name* – The tensor name.

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: 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 *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*) → [nvinfer1::TensorIOMode](#)
Determine whether a tensor is an input or output tensor.

Parameters *name* – The tensor name.

get_tensor_name(*self*: [tensorrt.tensorrt.ICudaEngine](#), *index*: *int*) → *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*: *int*) →
[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_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(*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.

Parameters *name* – The tensor name.

is_execution_binding(*self*: [tensorrt.tensorrt.ICudaEngine](#), *binding*: *int*) → *bool*
Returns True if tensor is required for execution phase, false otherwise.

For example, if a network uses an input tensor with binding *i* ONLY as the reshape dimensions for an [IShuffleLayer](#), then `is_execution_binding(i) == False`, and a binding of 0 can be supplied for

it when calling `IEExecutionContext.execute_v2()` or `IEExecutionContext.execute_async_v2()`.

Parameters `binding` – The binding index.

is_shape_binding(*self*: `tensorrt.tensorrt.ICudaEngine`, *binding*: `int`) → `bool`

Returns True if tensor is required as input for shape calculations or output from them.

TensorRT evaluates a network in two phases:

1. Compute shape information required to determine memory allocation requirements and validate that runtime sizes make sense.
2. Process tensors on the device.

Some tensors are required in phase 1. These tensors are called “shape tensors”, and always have type `tensorrt.int32` and no more than one dimension. These tensors are not always shapes themselves, but might be used to calculate tensor shapes for phase 2.

`is_shape_binding()` returns true if the tensor is a required input or an output computed in phase 1. `is_execution_binding()` returns true if the tensor is a required input or an output computed in phase 2.

For example, if a network uses an input tensor with binding `i` as an input to an `IElementWiseLayer` that computes the reshape dimensions for an `IShuffleLayer`, `is_shape_binding(i) == True`

It’s possible to have a tensor be required by both phases. For instance, a tensor can be used as a shape in an `IShuffleLayer` and as the indices for an `IGatherLayer` collecting floating-point data.

It’s also possible to have a tensor required by neither phase that shows up in the engine’s inputs. For example, if an input tensor is used only as an input to an `IShapeLayer`, only its shape matters and its values are irrelevant.

Parameters `binding` – The binding index.

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`.

4.7 IExecutionContext

class `tensorrt.IOutputAllocator`(*self*: `tensorrt.tensorrt.IOutputAllocator`) → `None`

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 notify_shape(self, tensor_name, shape):
        ... # Your implementation here
```

__init__(*self*: [tensorrt.tensorrt.IOutputAllocator](#)) → None

notify_shape(*self*: [tensorrt.tensorrt.IOutputAllocator](#), *tensor_name*: str, *shape*: [tensorrt.tensorrt.Dims](#)) → None

Called by TensorRT when the shape of the output tensor is known.

Parameters

- **tensor_name** – The output tensor name.
- **shape** – The output tensor shape.

reallocate_output(*self*: [tensorrt.tensorrt.IOutputAllocator](#), *tensor_name*: str, *memory*: capsule, *size*: int, *alignment*: int) → capsule

A callback implemented by the application to handle acquisition of output tensor memory.

If an allocation request cannot be satisfied, None should be returned.

Parameters

- **tensor_name** – The output tensor name.
- **memory** – The output tensor memory address.
- **size** – The number of bytes required.
- **alignment** – The required alignment of memory.

Returns The address of the output tensor memory.

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\(\)](#). It has no effect when using [execute_async_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 on a 256-byte boundary, and its size must be at least `engine.device_memory_size`. If using [execute_async_v2\(\)](#) to run the network, The memory is in use from the invocation of [execute_async_v2\(\)](#) 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 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\(\)](#) or [execute_async_v2\(\)](#) . Profile 0 is selected by default. Changing this value will invalidate all dynamic bindings for the current execution context, so that they have to be set again using [set_binding_shape\(\)](#) before calling either [execute_v2\(\)](#) or [execute_async_v2\(\)](#) .
- **all_binding_shapes_specified** – bool Whether all dynamic dimensions of input tensors have been specified by calling [set_binding_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.

- **all_shape_inputs_specified** – bool Whether values for all input shape tensors have been specified by calling `set_shape_input()`. Trivially true if network has no input shape bindings. Does not work with name-base interfaces eg. `set_input_shape()`. Use `infer_shapes()` instead.
- **error_recorder** – *IErrRecorder* 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 `IEExecutionContext::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 kDETAILED verbosity will generally increase latency in `enqueueV2/V3()`. 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(self: tensorrt.tensorrt.IExecutionContext, batch_size: int = 1, bindings: List[int])` → bool

[DEPRECATED] Please use `execute_v2()` instead if the engine is built from a network with explicit batch dimension mode enabled.

Synchronously execute inference on a batch. This method requires a array of input and output buffers. The mapping from tensor names to indices can be queried using `ICudaEngine.get_binding_index()`.

Parameters

- **batch_size** – The batch size. This is at most the value supplied when the *ICudaEngine* was built. This has no effect if the engine is built from a network with explicit batch dimension mode enabled.
- **bindings** – A list of integers representing input and output buffer addresses for the network.

Returns True if execution succeeded.

`execute_async(self: tensorrt.tensorrt.IExecutionContext, batch_size: int = 1, bindings: List[int], stream_handle: int, input_consumed: capsule = None)` → bool

[DEPRECATED] Please use `execute_async_v2()` instead if the engine is built from a network with explicit batch dimension mode enabled.

Asynchronously execute inference on a batch. This method requires a array of input and output buffers. The mapping from tensor names to indices can be queried using `ICudaEngine::get_binding_index()`.

Parameters

- **batch_size** – The batch size. This is at most the value supplied when the *ICudaEngine* was built. This has no effect if the engine is built from a network with explicit batch dimension mode enabled.
- **bindings** – A list of integers representing input and output buffer addresses for the network.
- **stream_handle** – A handle for a CUDA stream on which the inference kernels will be executed.
- **input_consumed** – An optional event which will be signaled when the input buffers can be refilled with new data

Returns True if the kernels were executed successfully.

execute_async_v2(*self*: *tensorrt.tensorrt.IExecutionContext*, *bindings*: *List[int]*, *stream_handle*: *int*, *input_consumed*: *capsule = None*) → bool

Asynchronously execute inference on a batch. This method requires a array of input and output buffers. The mapping from tensor names to indices can be queried using *ICudaEngine::get_binding_index()*. This method only works for execution contexts built from networks with no implicit batch dimension.

Parameters

- **bindings** – A list of integers representing input and output buffer addresses for the network.
- **stream_handle** – A handle for a CUDA stream on which the inference kernels will be executed.
- **input_consumed** – An optional event which will be signaled when the input buffers can be refilled with new data

Returns True if the kernels were executed successfully.

execute_async_v3(*self*: *tensorrt.tensorrt.IExecutionContext*, *stream_handle*: *int*) → 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.

execute_v2(*self*: *tensorrt.tensorrt.IExecutionContext*, *bindings*: *List[int]*) → bool

Synchronously execute inference on a batch. This method requires a array of input and output buffers. The mapping from tensor names to indices can be queried using *ICudaEngine.get_binding_index()*. This method only works for execution contexts built from networks with no implicit batch dimension.

Parameters **bindings** – A list of integers representing input and output buffer addresses for the network.

Returns True if execution succeeded.

get_binding_shape(*self*: *tensorrt.tensorrt.IExecutionContext*, *binding*: *int*) → *tensorrt.tensorrt.Dims*

Get the dynamic shape of a binding.

If *set_binding_shape()* has been called on this binding (or if there are no dynamic dimensions), all dimensions will be positive. Otherwise, it is necessary to call *set_binding_shape()* before *execute_async_v2()* or *execute_v2()* may be called.

If the binding is out of range, an invalid Dims with nbDims == -1 is returned.

If `ICudaEngine.binding_is_input(binding)` is `False`, then both `all_binding_shapes_specified` and `all_shape_inputs_specified` must be `True` before calling this method.

Parameters `binding` – The binding index.

Returns A *Dims* object representing the currently selected shape.

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*) → *nvinfer1::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_shape(*self*: *tensorrt.tensorrt.IExecutionContext*, *binding*: *int*) → *List[int]*

Get values of an input shape tensor required for shape calculations or an output tensor produced by shape calculations.

Parameters `binding` – The binding index of an input tensor for which `ICudaEngine.is_shape_binding(binding)` is true.

If `ICudaEngine.binding_is_input(binding) == False`, then both `all_binding_shapes_specified` and `all_shape_inputs_specified` must be `True` before calling this method.

Returns An iterable containing the values of the shape tensor.

get_strides(*self*: *tensorrt.tensorrt.IExecutionContext*, *binding*: *int*) → *tensorrt.tensorrt.Dims*

Return the strides of the buffer for the given binding.

Note that strides can be different for different execution contexts with dynamic shapes.

Parameters `binding` – The binding index.

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_binding_shape(*self*: `tensorrt.tensorrt.IExecutionContext`, *binding*: int, *shape*: `tensorrt.tensorrt.Dims`) → bool

Set the dynamic shape of a binding.

Requires the engine to be built without an implicit batch dimension. The binding must be an input tensor, and all dimensions must be compatible with the network definition (i.e. only the wildcard dimension -1 can be replaced with a new dimension > 0). Furthermore, the dimensions must be in the valid range for the currently selected optimization profile.

For all dynamic non-output bindings (which have at least one wildcard dimension of -1), this method needs to be called after setting `active_optimization_profile` before either `execute_async_v2()` or `execute_v2()` may be called. When all input shapes have been specified, `all_binding_shapes_specified` is set to True.

Parameters

- **binding** – The binding index.
- **shape** – The shape to set.

Returns False if an error occurs (e.g. specified binding is out of range for the currently selected optimization profile or specified shape is inconsistent with min-max range of the optimization profile), else True.

Note that the network can still be invalid for certain combinations of input shapes that lead to invalid output shapes. To confirm the correctness of the network input shapes, check whether the output binding has valid shape using `get_binding_shape()` on the output binding.

set_input_consumed_event(*self*: `tensorrt.tensorrt.IExecutionContext`, *event*: int) → 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(*self*: [tensorrt.tensorrt.IExecutionContext](#), *name*: *str*, *shape*: [tensorrt.tensorrt.Dims](#)) → bool

Set shape for the given input tensor.

Parameters

- **name** – The input tensor name.
- **shape** – The input tensor shape.

set_optimization_profile_async(*self*: [tensorrt.tensorrt.IExecutionContext](#), *profile_index*: *int*, *stream_handle*: *int*) → 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*: [nvinfer1::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_shape_input(*self*: [tensorrt.tensorrt.IExecutionContext](#), *binding*: *int*, *shape*: *List[int]*) → bool

Set values of an input shape tensor required by shape calculations.

Parameters

- **binding** – The binding index of an input tensor for which `ICudaEngine.is_shape_binding(binding)` and `ICudaEngine.binding_is_input(binding)` are both true.
- **shape** – An iterable containing the values of the input shape tensor. The number of values should be the product of the dimensions returned by `get_binding_shape(binding)`.

If `ICudaEngine.is_shape_binding(binding)` and `ICudaEngine.binding_is_input(binding)` are both true, this method must be called before [execute_async_v2\(\)](#) or [execute_v2\(\)](#) may be called. Additionally, this method must not be called if either `ICudaEngine.is_shape_binding(binding)` or `ICudaEngine.binding_is_input(binding)` are false.

Returns False if an error occurs (e.g. specified binding is out of range for the currently selected optimization profile or specified shape values are inconsistent with min-max range of the optimization profile), else True.

Note that the network can still be invalid for certain combinations of input shapes that lead to invalid output shapes. To confirm the correctness of the network input shapes, check whether the output binding has valid shape using `get_binding_shape()` on the output binding.

set_tensor_address(*self*: `tensorrt.tensorrt.IExecutionContext`, *name*: *str*, *memory*: *int*) → bool
Set memory address for the given input or output tensor.

Parameters

- **name** – The tensor name.
- **memory** – The memory address.

4.8 Runtime

class `tensorrt.Runtime`(*self*: `tensorrt.tensorrt.Runtime`, *logger*: `tensorrt.tensorrt.ILogger`) → None
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`.
- **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(*self*: `tensorrt.tensorrt.Runtime`, *serialized_engine*: *buffer*) → `tensorrt.tensorrt.ICudaEngine`

Deserialize an `ICudaEngine` from a stream.

Parameters **serialized_engine** – The buffer that holds the serialized `ICudaEngine`.

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.

4.9 Refitter

class `tensorrt.Refitter`(*self*: `tensorrt.tensorrt.Refitter`, *engine*: `tensorrt.tensorrt.ICudaEngine`, *logger*: `tensorrt.tensorrt.ILogger`) → None

Updates weights in an *ICudaEngine*.

Variables

- **error_recorder** – *IErrRecorder* 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*.

Parameters

- **engine** – The engine to refit.
- **logger** – The logger to use.

__del__(*self*: `tensorrt.tensorrt.Refitter`) → 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.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_dynamic_range(*self*: `tensorrt.tensorrt.Refitter`, *tensor_name*: str) → tuple

Gets the dynamic range of a tensor. If the dynamic range was never set, returns the range computed during calibration.

Parameters **tensor_name** – The name of the tensor whose dynamic range to retrieve.

Returns Tuple[float, float] A tuple containing the [minimum, maximum] of the dynamic range.

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_tensors_with_dynamic_range(*self*: [tensorrt.tensorrt.Refitter](#)) → List[str]

Get names of all tensors that have refittable dynamic ranges.

Returns The names of tensors with refittable dynamic ranges.

refit_cuda_engine(*self*: [tensorrt.tensorrt.Refitter](#)) → bool

Updates associated engine. Return True if successful.

Failure occurs if [get_missing\(\)](#) != 0 before the call.

set_dynamic_range(*self*: [tensorrt.tensorrt.Refitter](#), *tensor_name*: str, *range*: List[float]) → bool

Update dynamic range for a tensor.

Parameters

- **tensor_name** – The name of the tensor whose dynamic range to update.
- **range** – The new range.

Returns True if successful, False otherwise.

Returns false if there is no Int8 engine tensor derived from a network tensor of that name. If successful, then [get_missing\(\)](#) may report that some weights need to be supplied.

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 number of weights is inconsistent with the original specification.

Modifying the weights before method [refit_cuda_engine\(\)](#) completes will result in undefined behavior.

Parameters

- **name** – The name of the weights to be refitted.
- **weights** – The new weights to associate with the name.

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 number of weights is inconsistent with the layer's original specification.

Modifying the weights before [refit_cuda_engine\(\)](#) completes 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.

4.10 IErrorRecorder

tensorrt.IErrorCodeTRT

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 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`) → None

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*: int) → `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*: int) → 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 occurring, a TensorRT API

can return correct results, but still register errors with the Error Recorder. The value of `getNbErrors` must monotonically increase 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.

4.11 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.

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.

4.12 GPU Allocator

4.12.1 AllocatorFlag

`tensorrt.AllocatorFlag`

Members:

`RESIZABLE` : TensorRT may call `realloc()` on this allocation

4.12.2 IGpuAllocator

class `tensorrt.IGpuAllocator`(*self*: `tensorrt.tensorrt.IGpuAllocator`) → None

Application-implemented class for controlling allocation on the GPU.

__init__(*self*: `tensorrt.tensorrt.IGpuAllocator`) → None

allocate(*self*: `tensorrt.tensorrt.IGpuAllocator`, *size*: int, *alignment*: int, *flags*: int) → capsule

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

deallocate(*self*: `tensorrt.tensorrt.IGpuAllocator`, *memory*: capsule) → bool

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.

free(*self*: `tensorrt.tensorrt.IGpuAllocator`, *memory*: capsule) → None

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.

realloc(*self*: `tensorrt.tensorrt.IGpuAllocator`, *address*: capsule, *alignment*: int, *new_size*: int) → capsule

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 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 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

4.13 EngineInspector

`class tensorrt.EngineInspector`

An engine inspector which prints out the layer information of an engine or an execution context. The engine or the context must be set before `get_layer_information()` or `get_engine_information()` can be called.

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

- **engine** – *ICudaEngine* Set or get the engine currently being inspected.
- **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: int, 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.

NETWORK

5.1 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 Whether the network was created with an implicit batch dimension. This is a network-wide property. Either all tensors in the network have an implicit batch dimension or none of them do. This is True when the `INetworkDefinition` is created with default flags: `create_network()`. To specify explicit batch, set the flag: `create_network(flags=1 << int(tensorrt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))`.
- **has_explicit_precision** – bool True if and only if this `INetworkDefinition` was created with `NetworkDefinitionCreationFlag.EXPLICIT_PRECISION` set: `create_network(flags=(1 << int(NetworkDefinitionCreationFlag.EXPLICIT_PRECISION)))`.
- **error_recorder** – `IErrorRecorder` Application-implemented error reporting interface for TensorRT objects.

`__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*: int) → `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_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*: [List\[tensorrt.tensorrt.ITensor\]](#)) → [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(*self*: [tensorrt.tensorrt.INetworkDefinition](#), *input*: [tensorrt.tensorrt.ITensor](#), *num_output_maps*: [int](#), *kernel_shape*: [tensorrt.tensorrt.DimsHW](#), *kernel*: [tensorrt.tensorrt.Weights](#), *bias*: [tensorrt.tensorrt.Weights](#) = `None`) → [tensorrt.tensorrt.IConvolutionLayer](#)

Add a 2D 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_convolution_nd(self: tensorrt.tensorrt.INetworkDefinition, input: tensorrt.tensorrt.ITensor,
                    num_output_maps: int, 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_deconvolution(self: tensorrt.tensorrt.INetworkDefinition, input: tensorrt.tensorrt.ITensor,
                   num_output_maps: int, kernel_shape: tensorrt.tensorrt.DimsHW, kernel:
                   tensorrt.tensorrt.Weights, bias: tensorrt.tensorrt.Weights = None) →
                   tensorrt.tensorrt.IDeconvolutionLayer
```

Add a 2D 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_deconvolution_nd(self: tensorrt.tensorrt.INetworkDefinition, input: tensorrt.tensorrt.ITensor,
                      num_output_maps: int, 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`) → `tensorrt.tensorrt.IDequantizeLayer`

Add a dequantization layer to the network. See [`IDequantizeLayer`](#) for more information.

Parameters

- **input** – A tensor to quantize.
- **scale** – A tensor with the scale coefficients.

Returns The new dequantization layer, or `None` if it could not be created.

add_einsum(*self*: `tensorrt.tensorrt.INetworkDefinition`, *inputs*: `List[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`) → `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.

Returns The new fill layer, or `None` if it could not be created.

add_fully_connected(*self*: `tensorrt.tensorrt.INetworkDefinition`, *input*: `tensorrt.tensorrt.ITensor`, *num_outputs*: `int`, *kernel*: `tensorrt.tensorrt.Weights`, *bias*: `tensorrt.tensorrt.Weights` or `None`) → `tensorrt.tensorrt.IFullyConnectedLayer`

Add a fully connected layer to the network. See [`IFullyConnectedLayer`](#) for more information.

Parameters

- **input** – The input tensor to the layer.
- **num_outputs** – The number of outputs of the layer.
- **kernel** – The kernel weights for the convolution.
- **bias** – The optional bias weights for the convolution.

Returns The new fully connected layer, or None if it could not be created.

add_gather(*self*: [tensorrt.tensorrt.INetworkDefinition](#), *input*: [tensorrt.tensorrt.ITensor](#), *indices*: [tensorrt.tensorrt.ITensor](#), *axis*: *int*) → [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-conditional, 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.
- **dtype** – The data type of the tensor. Currently, `tensorrt.int8` is not supported for inputs.
- **shape** – The dimensions of the tensor. The total volume must be less than 2^{30} elements.

Returns The newly added Tensor.

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*: *int*, *alpha*: *float*, *beta*: *float*, *k*: *float*) → [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_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.

Parameters

- **boxes** – The input boxes tensor to the layer.
- **scores** – The input scores tensor to the layer.
- **max_output_boxes_per_class** – The maxOutputBoxesPerClass tensor to the layer.

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 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}.

Returns The new NMS layer, or None if it could not be created.

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.

Parameters **input** – The input tensor to the layer.

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*: int) → [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.

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*: int) → [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(*self*: [tensorrt.tensorrt.INetworkDefinition](#), *input*: [tensorrt.tensorrt.ITensor](#), *pre_padding*: [tensorrt.tensorrt.DimsHW](#), *post_padding*: [tensorrt.tensorrt.DimsHW](#)) → [tensorrt.tensorrt.IPaddingLayer](#)

Add a 2D 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_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_v2(*self*: [tensorrt.tensorrt.INetworkDefinition](#), *inputs*: [List\[tensorrt.tensorrt.ITensor\]](#), *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_pooling(*self*: [tensorrt.tensorrt.INetworkDefinition](#), *input*: [tensorrt.tensorrt.ITensor](#), *type*: [tensorrt.tensorrt.PoolingType](#), *window_size*: [tensorrt.tensorrt.DimsHW](#)) → [tensorrt.tensorrt.IPoolingLayer](#)

Add a 2D 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_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](#)) → [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.

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*: *int*, *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_rnn_v2(*self*: [tensorrt.tensorrt.INetworkDefinition](#), *input*: [tensorrt.tensorrt.ITensor](#), *layer_count*: *int*,
hidden_size: *int*, *max_seq_length*: *int*, *op*: [tensorrt.tensorrt.RNNOperation](#)) →
[tensorrt.tensorrt.IRNNv2Layer](#)

Add an RNNv2 layer to the network. See [IRNNv2Layer](#) for more information.

Add an `layer_count` deep RNN layer to the network with `hidden_size` internal states that can take a batch with fixed or variable sequence lengths.

Parameters

- **input** – The input tensor to the layer (see below).
- **layer_count** – The number of layers in the RNN.
- **hidden_size** – Size of the internal hidden state for each layer.
- **max_seq_length** – Maximum sequence length for the input.
- **op** – The type of RNN to execute.

By default, the layer is configured with `RNNDirection.UNIDIRECTION` and `RNNInputMode.LINEAR`. To change these settings, set `IRNNv2Layer.direction` and `IRNNv2Layer.input_mode`.

Weights and biases for the added layer should be set using [IRNNv2Layer.set_weights_for_gate\(\)](#) and [IRNNv2Layer.set_bias_for_gate\(\)](#) prior to building an engine using this network.

The input tensors must be of the type `float32` or `float16`. The layout of the weights is row major and must be the same datatype as the input tensor. `weights` contain 8 matrices and `bias` contains 8 vectors.

See [IRNNv2Layer.set_weights_for_gate\(\)](#) and [IRNNv2Layer.set_bias_for_gate\(\)](#) for details on the required input format for `weights` and `bias`.

The input `ITensor` should contain zero or more index dimensions $\{N1, \dots, Np\}$, followed by two dimensions, defined as follows:

S_{max} is the maximum allowed sequence length (number of RNN iterations)

E specifies the embedding length (unless `RNNInputMode.SKIP` is set, in which case it should match `IRNNv2Layer.hidden_size`).

By default, all sequences in the input are assumed to be size `max_seq_length`. To provide explicit sequence lengths for each input sequence in the batch, set `IRNNv2Layer.seq_lengths`.

The RNN layer outputs up to three tensors.

The first output tensor is the output of the final RNN layer across all timesteps, with dimensions $\{Nl, \dots, Np, S_max, H\}$:

$Nl..Np$ are the index dimensions specified by the input tensor

S_max is the maximum allowed sequence length (number of RNN iterations)

H is an output hidden state (equal to `IRNNv2Layer.hidden_size` or `2x IRNNv2Layer.hidden_size`)

The second tensor is the final hidden state of the RNN across all layers, and if the RNN is an LSTM (i.e. `IRNNv2Layer.op` is `RNNOperation.LSTM`), then the third tensor is the final cell state of the RNN across all layers. Both the second and third output tensors have dimensions $\{Nl, \dots, Np, L, H\}$:

$Nl..Np$ are the index dimensions specified by the input tensor

L is the number of layers in the RNN, equal to `IRNNv2Layer.num_layers`

H is the hidden state for each layer, equal to `IRNNv2Layer.hidden_size` if `getDirection` is `RNNDirection.UNIDIRECTION`, and `2x IRNNv2Layer.hidden_size` otherwise.

Returns The new RNNv2 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:
    int) → 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_topk(*self*: [tensorrt.tensorrt.INetworkDefinition](#), *input*: [tensorrt.tensorrt.ITensor](#), *op*: [tensorrt.tensorrt.TopKOperation](#), *k*: *int*, *axes*: *int*) → [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.

Parameters

- **input** – The input tensor to the layer.
- **op** – Operation to perform.
- **k** – Number of elements to keep.
- **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.

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.

get_input(*self*: [tensorrt.tensorrt.INetworkDefinition](#), *index*: *int*) → [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*: *int*) → [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*: *int*) → [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.

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.

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_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.

5.2 Layer Base Classes

5.2.1 ITensor

tensorrt.**TensorLocation**

The physical location of the data.

Members:

DEVICE : Data is stored on the device.

HOST : Data is stored on the device.

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 the product of 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. 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. 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 bound to 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 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 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. 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].

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.
- **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 Whether to enable broadcast of tensor across the batch. When a tensor is broadcast across a batch, it has the same value for every member in the batch. Memory is only allocated once for the single member. This method is only valid for network input tensors, since the flags of layer output tensors are inferred based on layer inputs and parameters. If this state is modified for a tensor in the network, the states of all dependent tensors will be recomputed.
- **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.
- **dynamic_range** – Tuple[float, float] A tuple containing the [minimum, maximum] of the dynamic range, or None if the range was not set.
- **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: int) → 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.

reset_dynamic_range(self: *tensorrt.tensorrt.ITensor*) → None

Undo the effect of setting the dynamic range.

set_dimension_name(self: *tensorrt.tensorrt.ITensor*, index: int, 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.

set_dynamic_range(self: *tensorrt.tensorrt.ITensor*, min: float, max: float) → bool

Set dynamic range for the tensor. NOTE: It is suggested to use `tensor.dynamic_range = (min, max)` instead.

Parameters

- **min** – Minimum of the dynamic range.
- **max** – Maximum of the dyanmic range.

Returns true if succeed in setting range. Otherwise false.

5.2.2 ILayer

`tensorrt.LayerType`

Type of Layer

Members:

CONVOLUTION : Convolution layer
FULLY_CONNECTED : Fully connected 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
RNN_V2 : RNNv2 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

class `tensorrt.ILayer`

Base class for all layer classes in an [*INetworkDefinition*](#).

Variables

- **name** – `str` The name of the layer.
- **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.
- **precision** – [*DataType*](#) The computation precision.
- **precision_is_set** – `bool` Whether the precision is set or not.

get_input(*self*: [*tensorrt.tensorrt.ILayer*](#), *index*: `int`) → [*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*: `int`) → [*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*: `int`) → `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`.

output_type_is_set(*self*: [tensorrt.tensorrt.ILayer](#), *index*: *int*) → bool

Whether the output type has been set for this layer.

Parameters *index* – The index of the output.

Returns Whether the output type has been explicitly set.

reset_output_type(*self*: [tensorrt.tensorrt.ILayer](#), *index*: *int*) → None

Reset output type of this layer.

Parameters *index* – The index of the output.

reset_precision(*self*: [tensorrt.tensorrt.ILayer](#)) → None

Reset the computation precision of the layer.

set_input(*self*: [tensorrt.tensorrt.ILayer](#), *index*: *int*, *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.

set_output_type(*self*: [tensorrt.tensorrt.ILayer](#), *index*: *int*, *dtype*: [tensorrt.tensorrt.DataType](#)) → None

Constraint layer to generate output data with given type. Note that this method cannot be used to set the data type of the second output tensor of the topK layer. The data type of the second output tensor of the topK layer is always `int32`.

Parameters

- *index* – The index of the output tensor to set the type.
- *dtype* – `DataType` of the output.

5.3 Layers

5.3.1 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`

CAFFE_ROUND_DOWN : Use CAFFE padding, rounding the output size down

CAFFE_ROUND_UP : Use CAFFE padding, rounding the output size up

5.3.2 IConvolutionLayer

class `tensorrt.IConvolutionLayer`

A convolution layer in an *INetworkDefinition*.

This layer performs a correlation operation between 3-dimensional filter with a 4-dimensional tensor to produce another 4-dimensional tensor.

An optional bias argument is supported, which adds a per-channel constant to each value in the output.

Variables

- **kernel_size** – *DimsHW* The HW kernel size of the convolution.
- **num_output_maps** – `int` The number of output maps for the convolution.
- **stride** – *DimsHW* The stride of the convolution. Default: (1, 1)
- **padding** – *DimsHW* The padding of the convolution. The input will be zero-padded by this number of elements in the height and width directions. If the padding is asymmetric, this value corresponds to the pre-padding. Default: (0, 0)
- **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.
- **dilation** – *DimsHW* The dilation for a convolution. Default: (1, 1)
- **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)

5.3.3 IFullyConnectedLayer

class tensorrt.IFullyConnectedLayer

A fully connected layer in an *INetworkDefinition*.

This layer expects an input tensor of three or more non-batch dimensions. The input is automatically reshaped into an $M \times V$ tensor X , where V is a product of the last three dimensions and M is a product of the remaining dimensions (where the product over 0 dimensions is defined as 1). For example:

- If the input tensor has shape $\{C, H, W\}$, then the tensor is reshaped into $\{1, C*H*W\}$.
- If the input tensor has shape $\{P, C, H, W\}$, then the tensor is reshaped into $\{P, C*H*W\}$.

The layer then performs:

$$Y := \text{matmul}(X, W^T) + \text{bias}$$

Where X is the $M \times V$ tensor defined above, W is the $K \times V$ weight tensor of the layer, and bias is a row vector size K that is broadcasted to $M \times K$. K is the number of output channels, and configurable via `IFullyConnectedLayer.num_output_channels`. If bias is not specified, it is implicitly 0.

The $M \times K$ result Y is then reshaped such that the last three dimensions are $\{K, 1, 1\}$ and the remaining dimensions match the dimensions of the input tensor. For example:

- If the input tensor has shape $\{C, H, W\}$, then the output tensor will have shape $\{K, 1, 1\}$.
- If the input tensor has shape $\{P, C, H, W\}$, then the output tensor will have shape $\{P, K, 1, 1\}$.

Variables

- **num_output_channels** – `int` The number of output channels K from the fully connected layer.
- **kernel** – *Weights* The kernel weights, given as a $K \times C$ matrix in row-major order.
- **bias** – *Weights* The bias weights. Bias is optional. To omit bias, set this to an empty *Weights* object.

5.3.4 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.
DEFAULT : [DEPRECATED] Use STRICT_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.

5.3.5 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 $x \geq 0$, $f(x) = \alpha * x$ if $x < 0$

ELU : Elu activation: $f(x) = x$ if $x \geq 0$, $f(x) = \alpha * (\exp(x) - 1)$ if $x < 0$

SELU : Selu activation: $f(x) = \beta * x$ if $x > 0$, $f(x) = \beta * (\alpha * \exp(x) - \alpha)$ if $x \leq 0$

SOFTSIGN : Softsign activation: $f(x) = x / (1 + \text{abs}(x))$

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 $x \leq \alpha$

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, SOFT-PLUS, CLIP, HARD_SIGMOID, SCALED_TANH). Other activations ignore this parameter.

5.3.6 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: $(1 - \text{blendFactor}) * \text{maxPool} + \text{blendFactor} * \text{avgPool}$

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.
- **window_size** – [DimsHW](#) The window size for pooling.
- **stride** – [DimsHW](#) The stride for pooling. Default: (1, 1)
- **padding** – [DimsHW](#) The padding for pooling. Default: (0, 0)
- **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: $\text{max_average_blendPool} = (1 - \text{blendFactor}) * \text{maxPool} + \text{blendFactor} * \text{avgPool}$. `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)

5.3.7 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].

5.3.8 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:

$$\text{output} = (\text{input} * \text{scale} + \text{shift})^{\text{power}}$$

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.

5.3.9 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).

In implicit mode :

Bit 0 corresponds to the C dimension boolean.

Bit 1 corresponds to the H dimension boolean.

Bit 2 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 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.

In explicit mode :

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 2 with implicit batch mode, set bit 3 with explicit batch mode.

On Xavier, this layer is not supported on DLA. Otherwise, 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.

5.3.10 IConcatenationLayer

class `trt.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. For implicit batch mode, the number of tensor dimensions does NOT include the implicit batch dimension.

5.3.11 IDEconvolutionLayer

class `trt.IDeconvolutionLayer`

A deconvolution layer in an *INetworkDefinition*.

Variables

- **kernel_size** – *DimsHW* The HW kernel size of the convolution.
- **num_output_maps** – `int` The number of output feature maps for the deconvolution.
- **stride** – *DimsHW* The stride of the deconvolution. Default: (1, 1)
- **padding** – *DimsHW* The padding of the deconvolution. The input will be zero-padded by this number of elements in the height and width directions. Padding is symmetric. Default: (0, 0)
- **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)

5.3.12 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.

5.3.13 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 must be 0 if there is an implicit batch dimension. It can be 0 or 1 if there is not an implicit batch dimension. 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.

5.3.14 RNN Layers

`tensorrt.RNNOperation`

The RNN operations that may be performed by an RNN layer.

Equation definitions

In the equations below, we use the following naming convention:

t := current time step

i := input gate

o := output gate

f := forget gate

z := update gate

r := reset gate

c := cell gate

h := hidden gate

$g[t]$ denotes the output of gate g at timestep t , e.g. $f[t]$ is the output of the forget gate f .

$X[t]$:= input tensor for timestep t

$C[t]$:= cell state for timestep t

$H[t]$:= hidden state for timestep t

$W[g]$:= W (input) parameter weight matrix for gate g

$R[g]$:= U (recurrent) parameter weight matrix for gate g

$Wb[g]$:= W (input) parameter bias vector for gate g

$Rb[g]$:= U (recurrent) parameter bias vector for gate g

Unless otherwise specified, all operations apply pointwise to elements of each operand tensor.

$ReLU(X) := \max(X, 0)$

$\tanh(X)$:= hyperbolic tangent of X

$\text{sigmoid}(X) := 1 / (1 + \exp(-X))$

$\exp(X) := e^X$

$A.B$ denotes matrix multiplication of A and B .

$A*B$ denotes pointwise multiplication of A and B .

Equations

Depending on the value of `RNNOperation` chosen, each sub-layer of the RNN layer will perform one of the following operations:

RELU

$H[t] := ReLU(W[i].X[t] + R[i].H[t - 1] + Wb[i] + Rb[i])$

TANH

$$H[t] := \tanh(W[i].X[t] + R[i].H[t-1] + Wb[i] + Rb[i])$$

LSTM

$$\begin{aligned} i[t] &:= \text{sigmoid}(W[i].X[t] + R[i].H[t-1] + Wb[i] + Rb[i]) \\ f[t] &:= \text{sigmoid}(W[f].X[t] + R[f].H[t-1] + Wb[f] + Rb[f]) \\ o[t] &:= \text{sigmoid}(W[o].X[t] + R[o].H[t-1] + Wb[o] + Rb[o]) \\ c[t] &:= \tanh(W[c].X[t] + R[c].H[t-1] + Wb[c] + Rb[c]) \end{aligned}$$

$$\begin{aligned} C[t] &:= f[t] * C[t-1] + i[t] * c[t] \\ H[t] &:= o[t] * \tanh(C[t]) \end{aligned}$$

GRU

$$\begin{aligned} z[t] &:= \text{sigmoid}(W[z].X[t] + R[z].H[t-1] + Wb[z] + Rb[z]) \\ r[t] &:= \text{sigmoid}(W[r].X[t] + R[r].H[t-1] + Wb[r] + Rb[r]) \\ h[t] &:= \tanh(W[h].X[t] + r[t] * (R[h].H[t-1] + Rb[h]) + Wb[h]) \\ H[t] &:= (1 - z[t]) * h[t] + z[t] * H[t-1] \end{aligned}$$

Members:

RELU : Single gate RNN w/ ReLU activation
 TANH : Single gate RNN w/ TANH activation
 LSTM : Four-gate LSTM network w/o peephole connections
 GRU : Three-gate network consisting of Gated Recurrent Units

tensorrt.RNNDirection

The RNN direction that may be performed by an RNN layer.

Members:

UNIDIRECTION : Network iterates from first input to last input
 BIDIRECTION : Network iterates from first to last (and vice versa) and outputs concatenated

tensorrt.RNNInputMode

The RNN input modes that may occur with an RNN layer.

If the RNN is configured with `RNNInputMode.LINEAR`, then for each gate g in the first layer of the RNN, the input vector $X[t]$ (length E) is left-multiplied by the gate's corresponding weight matrix $W[g]$ (dimensions $H \times E$) as usual, before being used to compute the gate output as described by [RNNOperation](#).

If the RNN is configured with `RNNInputMode.SKIP`, then this initial matrix multiplication is “skipped” and $W[g]$ is conceptually an identity matrix. In this case, the input vector $X[t]$ must have length H (the size of the hidden state).

Members:

LINEAR : Perform the normal matrix multiplication in the first recurrent layer
 SKIP : No operation is performed on the first recurrent layer

5.3.14.1 IRNNv2Layer

tensorrt.RNNGateType

The RNN input modes that may occur with an RNN layer.

If the RNN is configured with `RNNInputMode.LINEAR`, then for each gate g in the first layer of the RNN, the input vector $X[t]$ (length E) is left-multiplied by the gate's corresponding weight matrix $W[g]$ (dimensions $H \times E$) as usual, before being used to compute the gate output as described by [RNNOperation](#).

If the RNN is configured with `RNNInputMode.SKIP`, then this initial matrix multiplication is “skipped” and $W[g]$ is conceptually an identity matrix. In this case, the input vector $X[t]$ must have length H (the size of the hidden state).

Members:

INPUT : Input Gate

OUTPUT : Output Gate

FORGET : Forget Gate

UPDATE : Update Gate

RESET : Reset Gate

CELL : Cell Gate

HIDDEN : Hidden Gate

class tensorrt.IRNNv2Layer

An RNN layer in an [INetworkDefinition](#), version 2

Variables

- **num_layers** – int The layer count of the RNN.
- **hidden_size** – int The hidden size of the RNN.
- **max_seq_length** – int The maximum sequence length of the RNN.
- **data_length** – int The embedding length of the RNN.
- **seq_lengths** – [ITensor](#) Individual sequence lengths in the batch with the [ITensor](#) provided. The **seq_lengths** [ITensor](#) should be a $\{N1, \dots, Np\}$ tensor, where $N1..Np$ are the index dimensions of the input tensor to the RNN. If **seq_lengths** is not specified, then the RNN layer assumes all sequences are size **max_seq_length**. All sequence lengths in **seq_lengths** should be in the range $[1, \text{max_seq_length}]$. Zero-length sequences are not supported. This tensor must be of type `int32`.
- **op** – [RNNOperation](#) The operation of the RNN layer.
- **input_mode** – int The input mode of the RNN layer.
- **direction** – int The direction of the RNN layer.
- **hidden_state** – [ITensor](#) the initial hidden state of the RNN with the provided **hidden_state** [ITensor](#). The **hidden_state** [ITensor](#) should have the dimensions $\{N1, \dots, Np, L, H\}$, where: $N1..Np$ are the index dimensions specified by the input tensor L is the number of layers in the RNN, equal to **num_layers** H is the hidden state for each layer, equal to **hidden_size** if **direction** is `RNNDirection.UNIDIRECTION`, and $2 \times \text{hidden_size}$ otherwise.

- **cell_state** – *ITensor* The initial cell state of the LSTM with the provided *cell_state ITensor*. The *cell_state ITensor* should have the dimensions $\{N1, \dots, Np, L, H\}$, where: $N1..Np$ are the index dimensions specified by the input tensor L is the number of layers in the RNN, equal to `num_layers` H is the hidden state for each layer, equal to `hidden_size` if `direction` is `RNNDirection.UNIDIRECTION`, and $2 \times \text{hidden_size}$ otherwise. It is an error to set this on an RNN layer that is not configured with `RNNOperation.LSTM`.

get_bias_for_gate(*self*: `tensorrt.tensorrt.IRNNv2Layer`, *layer_index*: `int`, *gate*: `tensorrt.tensorrt.RNNGateType`, *is_w*: `bool`) → `numpy.ndarray`

Get the bias parameters for an individual gate in the RNN.

Parameters

- **layer_index** – The index of the layer that contains this gate.
- **gate** – The name of the gate within the RNN layer.
- **is_w** – True if the bias parameters are for the input bias $Wb[g]$ and false if they are for the recurrent input bias $Rb[g]$.

Returns The bias parameters.

get_weights_for_gate(*self*: `tensorrt.tensorrt.IRNNv2Layer`, *layer_index*: `int`, *gate*: `tensorrt.tensorrt.RNNGateType`, *is_w*: `bool`) → `numpy.ndarray`

Get the weight parameters for an individual gate in the RNN.

Parameters

- **layer_index** – The index of the layer that contains this gate.
- **gate** – The name of the gate within the RNN layer.
- **is_w** – True if the weight parameters are for the input matrix $W[g]$ and false if they are for the recurrent input matrix $R[g]$.

Returns The weight parameters.

set_bias_for_gate(*self*: `tensorrt.tensorrt.IRNNv2Layer`, *layer_index*: `int`, *gate*: `tensorrt.tensorrt.RNNGateType`, *is_w*: `bool`, *bias*: `tensorrt.tensorrt.Weights`) → `None`

Set the bias parameters for an individual gate in the RNN.

Parameters

- **layer_index** – The index of the layer that contains this gate.
- **gate** – The name of the gate within the RNN layer. The gate name must correspond to one of the gates used by this layer's *RNNOperation*.
- **is_w** – True if the bias parameters are for the input bias $Wb[g]$ and false if they are for the recurrent input bias $Rb[g]$. See *RNNOperation* for equations showing how these bias vectors are used in the RNN gate.
- **bias** – The weight structure holding the bias parameters, which should be an array of size `hidden_size`.

set_weights_for_gate(*self*: `tensorrt.tensorrt.IRNNv2Layer`, *layer_index*: `int`, *gate*: `tensorrt.tensorrt.RNNGateType`, *is_w*: `bool`, *weights*: `tensorrt.tensorrt.Weights`) → `None`

Set the weight parameters for an individual gate in the RNN.

Parameters

- **layer_index** – The index of the layer that contains this gate.

- **gate** – The name of the gate within the RNN layer. The gate name must correspond to one of the gates used by this layer’s *RNNOperation*.
- **is_w** – True if the weight parameters are for the input matrix $W[g]$ and false if they are for the recurrent input matrix $R[g]$. See *RNNOperation* for equations showing how these matrices are used in the RNN gate.
- **weights** – The weight structure holding the weight parameters, which are stored as a row-major 2D matrix. For more information, see `IRNNv2Layer::setWeights()`.

5.3.15 IPluginV2Layer

class `tensorrt.IPluginV2Layer`

A plugin layer in an *INetworkDefinition*.

Variables `plugin` – IPluginV2 The plugin for the layer.

5.3.16 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.

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.

5.3.17 IReduceLayer

tensorrt.ReduceOperation

The reduce operations that may be performed by a Reduce layer

Members:

SUM :

PROD :

MAX :

MIN :

AVG :

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.

5.3.18 IPaddingLayer

class tensorrt.IPaddingLayer

A padding layer in an *INetworkDefinition*.

Variables

- **pre_padding** – *DimsHW* The padding that is applied at the start of the tensor. Negative padding results in trimming the edge by the specified amount.
- **post_padding** – *DimsHW* The padding that is applied at the end of the tensor. Negative padding results in trimming the edge by the specified amount
- **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.

5.3.19 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.

5.3.20 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.

5.3.21 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: List[int]) -> 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.IShuffleLayer](#), *index*: *int*, *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 <code>int32</code> 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.

5.3.22 ISliceLayer

Note: [Deprecated] Use `SampleMode` instead.

`tensorrt.SliceMode`

Controls how `ISliceLayer` and `IGridSample` handles out of bounds coordinates

Members:

`STRICT_BOUNDS` : Fail with error when the coordinates are out of bounds.

`DEFAULT` : [DEPRECATED] Use `STRICT_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.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 or stride as `ITensor`s` , by using `:func:`ILayer.set_input` to add a second, third, or fourth 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 input (third 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 `int32` 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}}`

When the sliceMode is `SliceMode.CLAMP` or `SliceMode.REFLECT`, for each input dimension, if its size is 0 then the corresponding output dimension must be 0 too.

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.
- The number of elements in the output tensor does not exceed $2 * \text{Dims.MAX_DIMS}$.

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. * sliceMode is `SliceMode.DEFAULT`. * Strides are 1 for all dimensions. * Slicing is not performed on the first dimension * The input tensor has four dimensions

Variables

- **start** – `Dims` The start offset.
- **shape** – `Dims` The output dimensions.
- **stride** – `Dims` The slicing stride.
- **mode** – `SliceMode` Controls how `ISliceLayer` handles out of bounds coordinates.

set_input(*self*: `tensorrt.tensorrt.ISliceLayer`, *index*: `int`, *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 <code>SliceMode.FILL</code> slice mode. Disallowed for other modes.

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.

5.3.23 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 `int32` tensor of length 4 containing the sequence 2, 3, 5, 7.

5.3.24 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.

set_input(*self*: tensorrt.tensorrt.ITopKLayer, *index*: int, *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.

5.3.25 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.

5.3.26 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.

5.3.27 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)
```

5.3.28 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.

5.3.29 IResizeLayer

tensorrt.ResizeMode

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.

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:

- `ResizeMode.NEAREST` - resizes innermost m dimensions of N-D, where $0 < m \leq \min(3, N)$ and $N > 0$.
- `ResizeMode.LINEAR` - resizes innermost m dimensions of N-D, where $0 < m \leq \min(3, N)$ and $N > 0$.
- `ResizeMode.CUBIC` - resizes innermost 2 dimensions of N-D, $N \geq 2$.

Default resize mode is `ResizeMode.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 $\text{floor}(\text{input dimension} * \text{scale})$.** 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** – *ResizeMode* 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.IResizeLayer*, *index*: int, *tensor*: *tensorrt.tensorrt.ITensor*) → None

Sets the input tensor for the given index.

If index == 1 and num_inputs == 1, and there is no implicit batch dimension, in which case 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.

5.3.30 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*: int = 0, *reverse*: bool = False) → *tensorrt.tensorrt.IteratorLayer*

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*: int = 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.

5.3.30.1 ILoopBoundaryLayer

class [tensorrt.ILoopBoundaryLayer](#)

Variables **loop** – [ILoop](#) associated with this boundary layer.

5.3.30.1.1 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](#)

5.3.30.1.2 IRecurrenceLayer

class `tensorrt.IRecurrenceLayer`

set_input(*self*: `tensorrt.tensorrt.IRecurrenceLayer`, *index*: `int`, *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.

5.3.30.1.3 IteratorLayer

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`.

5.3.30.1.4 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.ILoopOutputLayer](#), *index*: *int*, *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.

5.3.31 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](#).

set_input(*self*: [tensorrt.tensorrt.IFillLayer](#), *index*: *int*, *tensor*: [tensorrt.tensorrt.ITensor](#)) → None
replace an input of this layer with a specific tensor.

In- dex	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.

5.3.32 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. The input type, if constrained, must be constrained to tensorrt.float32 (FP16 input is not supported). The output size is the same as the input size.

IQuantizeLayer only supports tensorrt.float32 precision and will default to this precision during instantiation. IQuantizeLayer only supports tensorrt.int8 output.

Variables `axis` – int The axis along which quantization occurs. The quantization axis is in reference to the input tensor's dimensions.

5.3.33 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. The input type, if constrained, must be constrained to tensorrt.float32 (FP16 input is not supported). The output size is the same as the input size.

IDequantizeLayer only supports tensorrt.int8 precision and will default to this precision during instantiation. IDequantizeLayer only supports tensorrt.float32 output.

Variables `axis` – int The axis along which dequantization occurs. The dequantization axis is in reference to the input tensor's dimensions.

5.3.34 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.

5.3.35 IfConditional

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 true/false subgraphs are the same.

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.

5.3.36 IConditionLayer

class `tensorrt.IConditionLayer`

Describes the boolean condition of an if-conditional.

5.3.37 IIfConditionalOutputLayer

class `tensorrt.IIfConditionalOutputLayer`

Describes kinds of if-conditional outputs.

5.3.38 IIfConditionalInputLayer

class `tensorrt.IIfConditionalInputLayer`

Describes kinds of if-conditional inputs.

5.3.39 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.

5.3.40 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.

5.3.41 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.

5.3.42 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.

5.3.43 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. 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}.

set_input(self: `tensorrt.tensorrt.INMSLayer`, index: `int`, 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.

5.3.44 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.

5.3.45 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

$$Y = (X - \text{Mean}(X, \text{axes})) / \text{Sqrt}(\text{Variance}(X) + \text{epsilon}) * S + B$$

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.
- **compute_precision** – *DataType* The datatype used for the compute precision of this layer. By default TensorRT will run the normalization computation in `DataType.kFLOAT32` even in mixed precision mode regardless of any set builder flags to avoid overflow errors. `ILayer.precision` and `ILayer.set_output_type` can still be set to control input and output types of this layer. Only `DataType.kFLOAT32` and `DataType.kHALF` are valid for this member. Default: `Datatype.FLOAT`.

6.1 IPluginCreator

`tensorrt.PluginFieldType`

The possible field types for custom layer.

Members:

FLOAT16
FLOAT32
FLOAT64
INT8
INT16
INT32
CHAR
DIMS
UNKNOWN

`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: 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: Iterable) -> None`

append(*self: tensorrt.tensorrt.PluginFieldCollection, x: nvinfer1::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: Iterable) -> None`

Extend the list by appending all the items in the given list

insert(*self: tensorrt.tensorrt.PluginFieldCollection, i: int, x: nvinfer1::PluginField*) → None

Insert an item at a given position.

pop(*args, **kwargs)

Overloaded function.

1. `pop(self: tensorrt.tensorrt.PluginFieldCollection) -> nvinfer1::PluginField`

Remove and return the last item

2. `pop(self: tensorrt.tensorrt.PluginFieldCollection, i: int) -> nvinfer1::PluginField`

Remove and return the item at index *i*

class `tensorrt.IPluginCreator`

Plugin creator class for user implemented layers

Variables

- **tensorrt_version** – int Number of *PluginField* entries.
- **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(*self*: [tensorrt.tensorrt.IPluginCreator](#), *name*: *str*, *serialized_plugin*: *buffer*) → [tensorrt.tensorrt.IPluginV2](#)

Creates a plugin object from a serialized plugin.

Parameters

- **name** – Name of the plugin.
- **serialized_plugin** – A buffer containing a serialized plugin.

Returns A new [IPluginV2](#)

6.2 IPluginRegistry

class [tensorrt.IPluginRegistry](#)

Registers plugin creators.

Variables

- **plugin_creator_list** – All the registered plugin creators.
- **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.

deregister_creator(*self*: [tensorrt.tensorrt.IPluginRegistry](#), *creator*: [tensorrt.tensorrt.IPluginCreator](#)) → 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.

Parameters **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.

deregister_library(*self*: [tensorrt.tensorrt.IPluginRegistry](#), *handle*: *capsule*) → 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.

deserialize_library(*self*: [tensorrt.tensorrt.IPluginRegistry](#), *serialized_plugin_library*: *buffer*) → *capsule*

Load and register a shared library of plugins from a memory buffer.

Arg **serialized_plugin_library**: a pointer to a plugin buffer to deserialize.

Returns The loaded plugin library handle. The call will fail and return None if any of the plugins are already registered.

get_plugin_creator(*self*: [tensorrt.tensorrt.IPluginRegistry](#), *type*: *str*, *version*: *str*, *plugin_namespace*: *str* = "") → [tensorrt.tensorrt.IPluginCreator](#)

Return plugin creator based on type and version

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*) → capsule

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(*self*: *tensorrt.tensorrt.IPluginRegistry*, *creator*: *tensorrt.tensorrt.IPluginCreator*, *plugin_namespace*: *str* = "") → bool

Register a plugin creator.

Parameters

- **creator** – The IPluginCreator instance.
- **plugin_namespace** – The namespace of the plugin creator.

Returns False if one with the same type is already registered.

tensorrt.get_plugin_registry() → *tensorrt.tensorrt.IPluginRegistry*

Return the plugin registry for standard runtime

tensorrt.init_libnvinfer_plugins(*logger*: *capsule*, *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*: *nvinfer1::EngineCapability*) → *tensorrt.tensorrt.IPluginRegistry*

Return the plugin registry used for building engines for the specified runtime

7.1 IInt8Calibrator

`trt.tensorrt.CalibrationAlgoType`

Version of calibration algorithm to use.

Members:

`LEGACY_CALIBRATION`

`ENTROPY_CALIBRATION`

`ENTROPY_CALIBRATION_2`

`MINMAX_CALIBRATION`

class `trt.IInt8Calibrator`(*self*: `trt.tensorrt.IInt8Calibrator`) → None

Application-implemented interface for calibration. Calibration is a step performed by the builder when deciding suitable scale factors for 8-bit inference. It must also provide a method for retrieving representative images which the calibration process can use to examine the distribution of activations. It may optionally implement a method for caching the calibration result for reuse on subsequent runs.

To implement a custom calibrator, ensure that you explicitly instantiate the base class in `__init__()` :

```
class MyCalibrator(trt.IInt8Calibrator):
    def __init__(self):
        trt.IInt8Calibrator.__init__(self)
```

Variables

- **batch_size** – int The batch size used for calibration batches.
- **algorithm** – *CalibrationAlgoType* The algorithm used by this calibrator.

get_algorithm(*self*: `trt.tensorrt.IInt8Calibrator`) → `trt.tensorrt.CalibrationAlgoType`

Get the algorithm used by this calibrator.

Returns The algorithm used by this calibrator.

get_batch(*self*: `trt.tensorrt.IInt8Calibrator`, *names*: `List[str]`) → `List[int]`

Get a batch of input for calibration. The batch size of the input must match the batch size returned by `get_batch_size()`.

A possible implementation may look like this:

```
def get_batch(names):
    try:
        # Assume self.batches is a generator that provides batch data.
        data = next(self.batches)
        # Assume that self.device_input is a device buffer allocated by the
        ↪ constructor.
        cuda.memcpy_htod(self.device_input, data)
        return [int(self.device_input)]
    except StopIteration:
        # When we're out of batches, we return either [] or None.
        # This signals to TensorRT that there is no calibration data remaining.
        return None
```

Parameters *names* – The names of the network inputs for each object in the bindings array.

Returns A list of device memory pointers set to the memory containing each network input data, or an empty list if there are no more batches for calibration. You can allocate these device buffers with pycuda, for example, and then cast them to `int` to retrieve the pointer.

get_batch_size(*self*: `tensorrt.tensorrt.Int8Calibrator`) → `int`

Get the batch size used for calibration batches.

Returns The batch size.

read_calibration_cache(*self*: `tensorrt.tensorrt.Int8Calibrator`) → `buffer`

Load a calibration cache.

Calibration is potentially expensive, so it can be useful to generate the calibration data once, then use it on subsequent builds of the network. The cache includes the regression cutoff and quantile values used to generate it, and will not be used if these do not match the settings of the current calibrator. However, the network should also be recalibrated if its structure changes, or the input data set changes, and it is the responsibility of the application to ensure this.

Reading a cache is just like reading any other file in Python. For example, one possible implementation is:

```
def read_calibration_cache(self):
    # If there is a cache, use it instead of calibrating again. Otherwise,
    ↪ implicitly return None.
    if os.path.exists(self.cache_file):
        with open(self.cache_file, "rb") as f:
            return f.read()
```

Returns A cache object or `None` if there is no data.

write_calibration_cache(*self*: `tensorrt.tensorrt.Int8Calibrator`, *cache*: `buffer`) → `None`

Save a calibration cache.

Writing a cache is just like writing any other buffer in Python. For example, one possible implementation is:

```
def write_calibration_cache(self, cache):
    with open(self.cache_file, "wb") as f:
        f.write(cache)
```

Parameters *cache* – The calibration cache to write.

7.2 IInt8LegacyCalibrator

class `tensorrt.IInt8LegacyCalibrator`(*self*: `tensorrt.tensorrt.IInt8LegacyCalibrator`) → None

Extends the `IInt8Calibrator` class. This calibrator requires user parameterization, and is provided as a fallback option if the other calibrators yield poor results.

To implement a custom calibrator, ensure that you explicitly instantiate the base class in `__init__()` :

```
class MyCalibrator(trt.IInt8LegacyCalibrator):
    def __init__(self):
        trt.IInt8LegacyCalibrator.__init__(self)
```

Variables

- **quantile** – float The quantile (between 0 and 1) that will be used to select the region maximum when the quantile method is in use. See the user guide for more details on how the quantile is used.
- **regression_cutoff** – float The fraction (between 0 and 1) of the maximum used to define the regression cutoff when using regression to determine the region maximum. See the user guide for more details on how the regression cutoff is used

get_algorithm(*self*: `tensorrt.tensorrt.IInt8LegacyCalibrator`) → `tensorrt.tensorrt.CalibrationAlgoType`

Signals that this is the legacy calibrator.

Returns `CalibrationAlgoType.LEGACY_CALIBRATION`

get_batch(*self*: `tensorrt.tensorrt.IInt8LegacyCalibrator`, *names*: `List[str]`) → `List[int]`

Get a batch of input for calibration. The batch size of the input must match the batch size returned by `get_batch_size()`.

A possible implementation may look like this:

```
def get_batch(names):
    try:
        # Assume self.batches is a generator that provides batch data.
        data = next(self.batches)
        # Assume that self.device_input is a device buffer allocated by the
        ↪ constructor.
        cuda.memcpy_htod(self.device_input, data)
        return [int(self.device_input)]
    except StopIteration:
        # When we're out of batches, we return either [] or None.
        # This signals to TensorRT that there is no calibration data remaining.
        return None
```

Parameters *names* – The names of the network inputs for each object in the bindings array.

Returns A list of device memory pointers set to the memory containing each network input data, or an empty list if there are no more batches for calibration. You can allocate these device buffers with pycuda, for example, and then cast them to `int` to retrieve the pointer.

get_batch_size(*self*: `tensorrt.tensorrt.IInt8LegacyCalibrator`) → `int`

Get the batch size used for calibration batches.

Returns The batch size.

read_calibration_cache(*self*: `tensorrt.tensorrt.IInt8LegacyCalibrator`) → `buffer`

Load a calibration cache.

Calibration is potentially expensive, so it can be useful to generate the calibration data once, then use it on subsequent builds of the network. The cache includes the regression cutoff and quantile values used to generate it, and will not be used if these do not match the settings of the current calibrator. However, the network should also be recalibrated if its structure changes, or the input data set changes, and it is the responsibility of the application to ensure this.

Reading a cache is just like reading any other file in Python. For example, one possible implementation is:

```
def read_calibration_cache(self):
    # If there is a cache, use it instead of calibrating again. Otherwise,
    # implicitly return None.
    if os.path.exists(self.cache_file):
        with open(self.cache_file, "rb") as f:
            return f.read()
```

Returns A cache object or None if there is no data.

write_calibration_cache(*self*: `tensorrt.tensorrt.IInt8LegacyCalibrator`, *cache*: `buffer`) → `None`

Save a calibration cache.

Writing a cache is just like writing any other buffer in Python. For example, one possible implementation is:

```
def write_calibration_cache(self, cache):
    with open(self.cache_file, "wb") as f:
        f.write(cache)
```

Parameters *cache* – The calibration cache to write.

7.3 IInt8EntropyCalibrator

class `tensorrt.IInt8EntropyCalibrator`(*self*: `tensorrt.tensorrt.IInt8EntropyCalibrator`) → `None`

Extends the `IInt8Calibrator` class.

To implement a custom calibrator, ensure that you explicitly instantiate the base class in `__init__()` :

```
class MyCalibrator(trt.IInt8EntropyCalibrator):
    def __init__(self):
        trt.IInt8EntropyCalibrator.__init__(self)
```

This is the Legacy Entropy calibrator. It is less complicated than the legacy calibrator and produces better results.

get_algorithm(*self*: `tensorrt.tensorrt.IInt8EntropyCalibrator`) → `tensorrt.tensorrt.CalibrationAlgoType`

Signals that this is the entropy calibrator.

Returns `CalibrationAlgoType.ENTROPY_CALIBRATION`

get_batch(*self*: `tensorrt.tensorrt.IInt8EntropyCalibrator`, *names*: `List[str]`) → `List[int]`

Get a batch of input for calibration. The batch size of the input must match the batch size returned by `get_batch_size()`.

A possible implementation may look like this:


```
def get_batch(names):
    try:
        # Assume self.batches is a generator that provides batch data.
        data = next(self.batches)
        # Assume that self.device_input is a device buffer allocated by the
        ↪ constructor.
        cuda.memcpy_htod(self.device_input, data)
        return [int(self.device_input)]
    except StopIteration:
        # When we're out of batches, we return either [] or None.
        # This signals to TensorRT that there is no calibration data remaining.
        return None
```

Parameters `names` – The names of the network inputs for each object in the bindings array.

Returns A list of device memory pointers set to the memory containing each network input data, or an empty list if there are no more batches for calibration. You can allocate these device buffers with pycuda, for example, and then cast them to `int` to retrieve the pointer.

get_batch_size(*self*: `tensorrt.tensorrt.IInt8EntropyCalibrator`) → `int`

Get the batch size used for calibration batches.

Returns The batch size.

read_calibration_cache(*self*: `tensorrt.tensorrt.IInt8EntropyCalibrator`) → `buffer`

Load a calibration cache.

Calibration is potentially expensive, so it can be useful to generate the calibration data once, then use it on subsequent builds of the network. The cache includes the regression cutoff and quantile values used to generate it, and will not be used if these do not match the settings of the current calibrator. However, the network should also be recalibrated if its structure changes, or the input data set changes, and it is the responsibility of the application to ensure this.

Reading a cache is just like reading any other file in Python. For example, one possible implementation is:

```
def read_calibration_cache(self):
    # If there is a cache, use it instead of calibrating again. Otherwise,
    ↪ implicitly return None.
    if os.path.exists(self.cache_file):
        with open(self.cache_file, "rb") as f:
            return f.read()
```

Returns A cache object or `None` if there is no data.

write_calibration_cache(*self*: `tensorrt.tensorrt.IInt8EntropyCalibrator`, *cache*: `buffer`) → `None`

Save a calibration cache.

Writing a cache is just like writing any other buffer in Python. For example, one possible implementation is:

```
def write_calibration_cache(self, cache):
    with open(self.cache_file, "wb") as f:
        f.write(cache)
```

Parameters `cache` – The calibration cache to write.

7.4 IInt8EntropyCalibrator2

class `trt.IInt8EntropyCalibrator2`(*self*: `trt.IInt8EntropyCalibrator2`) → None

Extends the `IInt8Calibrator` class.

To implement a custom calibrator, ensure that you explicitly instantiate the base class in `__init__()` :

```
class MyCalibrator(trt.IInt8EntropyCalibrator2):
    def __init__(self):
        trt.IInt8EntropyCalibrator2.__init__(self)
```

This is the preferred calibrator. This is the required calibrator for DLA, as it supports per activation tensor scaling.

get_algorithm(*self*: `trt.IInt8EntropyCalibrator2`) → `trt.CalibrationAlgoType`

Signals that this is the entropy calibrator 2.

Returns `CalibrationAlgoType.ENTROPY_CALIBRATION_2`

get_batch(*self*: `trt.IInt8EntropyCalibrator2`, *names*: `List[str]`) → `List[int]`

Get a batch of input for calibration. The batch size of the input must match the batch size returned by `get_batch_size()`.

A possible implementation may look like this:

```
def get_batch(names):
    try:
        # Assume self.batches is a generator that provides batch data.
        data = next(self.batches)
        # Assume that self.device_input is a device buffer allocated by the
        # constructor.
        cuda.memcpy_htod(self.device_input, data)
        return [int(self.device_input)]
    except StopIteration:
        # When we're out of batches, we return either [] or None.
        # This signals to TensorRT that there is no calibration data remaining.
        return None
```

Parameters *names* – The names of the network inputs for each object in the bindings array.

Returns A list of device memory pointers set to the memory containing each network input data, or an empty list if there are no more batches for calibration. You can allocate these device buffers with pycuda, for example, and then cast them to `int` to retrieve the pointer.

get_batch_size(*self*: `trt.IInt8EntropyCalibrator2`) → `int`

Get the batch size used for calibration batches.

Returns The batch size.

read_calibration_cache(*self*: `trt.IInt8EntropyCalibrator2`) → `buffer`

Load a calibration cache.

Calibration is potentially expensive, so it can be useful to generate the calibration data once, then use it on subsequent builds of the network. The cache includes the regression cutoff and quantile values used to generate it, and will not be used if these do not match the settings of the current calibrator. However, the network should also be recalibrated if its structure changes, or the input data set changes, and it is the responsibility of the application to ensure this.

Reading a cache is just like reading any other file in Python. For example, one possible implementation is:

```
def read_calibration_cache(self):
    # If there is a cache, use it instead of calibrating again. Otherwise,
    ↪ implicitly return None.
    if os.path.exists(self.cache_file):
        with open(self.cache_file, "rb") as f:
            return f.read()
```

Returns A cache object or None if there is no data.

write_calibration_cache(*self*: `tensorrt.tensorrt.IInt8EntropyCalibrator2`, *cache*: *buffer*) → None

Save a calibration cache.

Writing a cache is just like writing any other buffer in Python. For example, one possible implementation is:

```
def write_calibration_cache(self, cache):
    with open(self.cache_file, "wb") as f:
        f.write(cache)
```

Parameters **cache** – The calibration cache to write.

7.5 IInt8MinMaxCalibrator

class `tensorrt.IInt8MinMaxCalibrator`(*self*: `tensorrt.tensorrt.IInt8MinMaxCalibrator`) → None

Extends the `IInt8Calibrator` class.

To implement a custom calibrator, ensure that you explicitly instantiate the base class in `__init__()` :

```
class MyCalibrator(trt.IInt8MinMaxCalibrator):
    def __init__(self):
        trt.IInt8MinMaxCalibrator.__init__(self)
```

This is the preferred calibrator for NLP tasks for all backends. It supports per activation tensor scaling.

get_algorithm(*self*: `tensorrt.tensorrt.IInt8MinMaxCalibrator`) → `tensorrt.tensorrt.CalibrationAlgoType`

Signals that this is the minmax calibrator.

Returns `CalibrationAlgoType.MINMAX_CALIBRATION`

get_batch(*self*: `tensorrt.tensorrt.IInt8MinMaxCalibrator`, *names*: *List[str]*) → *List[int]*

Get a batch of input for calibration. The batch size of the input must match the batch size returned by `get_batch_size()`.

A possible implementation may look like this:

```
def get_batch(names):
    try:
        # Assume self.batches is a generator that provides batch data.
        data = next(self.batches)
        # Assume that self.device_input is a device buffer allocated by the
        ↪ constructor.
        cuda.memcpy_htod(self.device_input, data)
```

(continues on next page)

(continued from previous page)

```

    return [int(self.device_input)]
except StopIteration:
    # When we're out of batches, we return either [] or None.
    # This signals to TensorRT that there is no calibration data remaining.
    return None

```

Parameters **names** – The names of the network inputs for each object in the bindings array.

Returns A list of device memory pointers set to the memory containing each network input data, or an empty list if there are no more batches for calibration. You can allocate these device buffers with pycuda, for example, and then cast them to `int` to retrieve the pointer.

get_batch_size(*self*: `tensorrt.tensorrt.IInt8MinMaxCalibrator`) → `int`

Get the batch size used for calibration batches.

Returns The batch size.

read_calibration_cache(*self*: `tensorrt.tensorrt.IInt8MinMaxCalibrator`) → `buffer`

Load a calibration cache.

Calibration is potentially expensive, so it can be useful to generate the calibration data once, then use it on subsequent builds of the network. The cache includes the regression cutoff and quantile values used to generate it, and will not be used if these do not match the settings of the current calibrator. However, the network should also be recalibrated if its structure changes, or the input data set changes, and it is the responsibility of the application to ensure this.

Reading a cache is just like reading any other file in Python. For example, one possible implementation is:

```

def read_calibration_cache(self):
    # If there is a cache, use it instead of calibrating again. Otherwise,
    # implicitly return None.
    if os.path.exists(self.cache_file):
        with open(self.cache_file, "rb") as f:
            return f.read()

```

Returns A cache object or `None` if there is no data.

write_calibration_cache(*self*: `tensorrt.tensorrt.IInt8MinMaxCalibrator`, *cache*: `buffer`) → `None`

Save a calibration cache.

Writing a cache is just like writing any other buffer in Python. For example, one possible implementation is:

```

def write_calibration_cache(self, cache):
    with open(self.cache_file, "wb") as f:
        f.write(cache)

```

Parameters **cache** – The calibration cache to write.

ALGORITHM SELECTOR

class tensorrt.IAlgorithmIOInfo

This class carries information about input or output of the algorithm. IAlgorithmIOInfo for all the input and output along with IAlgorithmVariant denotes the variation of algorithm and can be used to select or reproduce an algorithm using IAlgorithmSelector.select_algorithms().

Variables

- **tensor_format** – *TensorFormat* TensorFormat of the input/output of algorithm.
- **dtype** – *DataType* DataType of the input/output of algorithm.
- **strides** – *Dims* strides of the input/output tensor of algorithm.

__init__(*args, **kwargs)

class tensorrt.IAlgorithmVariant

provides a unique 128-bit identifier, which along with the input and output information denotes the variation of algorithm and can be used to select or reproduce an algorithm, using IAlgorithmSelector.select_algorithms() see IAlgorithmIOInfo, IAlgorithm, IAlgorithmSelector.select_algorithms() note A single implementation can have multiple tactics.

Variables

- **implementation** – int implementation of the algorithm.
- **tactic** – int tactic of the algorithm.

__init__(*args, **kwargs)

class tensorrt.IAlgorithmContext

Describes the context and requirements, that could be fulfilled by one or more instances of IAlgorithm. see IAlgorithm

Variables

- **name** – str name of the algorithm node.
- **num_inputs** – int number of inputs of the algorithm.
- **num_outputs** – int number of outputs of the algorithm.

__init__(*args, **kwargs)

get_shape(self: *tensorrt.tensorrt.IAlgorithmContext*, index: int) → List[*tensorrt.tensorrt.Dims*]

Get the minimum / optimum / maximum dimensions for a dynamic input tensor.

Parameters **index** – Index of the input or output of the algorithm. Incremental numbers assigned to indices of inputs and the outputs.

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.`

class `tensorrt.IAlgorithm`

Application-implemented interface for selecting and reporting the tactic selection of a layer. Tactic Selection is a step performed by the builder for deciding best algorithms for a layer.

Variables

- **algorithm_variant** – `IAAlgorithmVariant`& the algorithm variant.
- **timing_msec** – float The time in milliseconds to execute the algorithm.
- **workspace_size** – int The size of the GPU temporary memory in bytes which the algorithm uses at execution time.

`__init__(*args, **kwargs)`

`get_algorithm_io_info(self: tensorrt.tensorrt.IAlgorithm, index: int) → tensorrt.tensorrt.IAlgorithmIOInfo`

A single call for both inputs and outputs. Incremental numbers assigned to indices of inputs and the outputs.

Parameters **index** – Index of the input or output of the algorithm. Incremental numbers assigned to indices of inputs and the outputs.

Returns A `IAAlgorithmIOInfo`&

class `tensorrt.IAlgorithmSelector`(`self: tensorrt.tensorrt.IAlgorithmSelector`) → None

Interface implemented by application for selecting and reporting algorithms of a layer provided by the builder. note A layer in context of algorithm selection may be different from `ILayer` in `INetworkDefinition`. For example, an algorithm might be implementing a conglomeration of multiple `ILayers` in `INetworkDefinition`.

To implement a custom algorithm selector, ensure that you explicitly instantiate the base class in `__init__()` :

```
class MyAlgoSelector(trt.IAlgorithmSelector):
    def __init__(self):
        trt.IAlgorithmSelector.__init__(self)
```

`__init__(self: tensorrt.tensorrt.IAlgorithmSelector) → None`

`report_algorithms(self: tensorrt.tensorrt.IAlgorithmSelector, contexts: List[tensorrt.tensorrt.IAlgorithmContext], choices: List[tensorrt.tensorrt.IAlgorithm]) → None`

Called by TensorRT to report choices it made.

Note: For a given optimization profile, this call comes after all calls to `select_algorithms`. `choices[i]` is the choice that TensorRT made for `algoContexts[i]`, for `i` in `[0, num_algorithms-1]`

For example, a possible implementation may look like this:

```
def report_algorithms(self, contexts, choices):
    # Prints the time of the chosen algorithm by TRT from the
    # selection list passed in by select_algorithms
    for choice in choices:
        print(choice.timing_msec)
```

Parameters

- **contexts** – The list of all algorithm contexts.

- **choices** – The list of algorithm choices made by TensorRT corresponding to each context.

select_algorithms(*self*: `tensorrt.tensorrt.IAlgorithmSelector`, *context*: `tensorrt.tensorrt.IAlgorithmContext`, *choices*: `List[tensorrt.tensorrt.IAlgorithm]`) → `List[int]`

Select Algorithms for a layer from the given list of algorithm choices.

Note: TRT uses its default algorithm selection to choose from the list returned by the user. If the returned list is empty, TRT's default algorithm selection is used unless strict type constraints are set. The list of choices is valid only for this specific algorithm context.

For example, the simplest implementation looks like this:

```
def select_algorithms(self, context, choices):  
    assert len(choices) > 0  
    return list(range(len(choices)))
```

Parameters

- **context** – The context for which the algorithm choices are valid.
- **choices** – The list of algorithm choices to select for implementation of this layer.

Returns A `List[int]` indicating the indices from the choices vector that TensorRT should choose from.

UFF PARSER

tensorrt.UffInputOrder

The different possible supported input orders.

Members:

NCHW

NHWC

NC

class `tensorrt.UffParser`(*self*: `tensorrt.tensorrt.UffParser`) → None

This class is used for parsing models described using the UFF format.

Variables

- **uff_required_version_major** – int Version Major of the UFF.
- **uff_required_version_minor** – int Version Minor of the UFF.
- **uff_required_version_patch** – int Version Patch of the UFF.
- **plugin_namespace** – str The namespace used to lookup and create plugins in the network.
- **error_recorder** – `IErrorRecorder` Application-implemented error reporting interface for TensorRT objects.

__del__(*self*: `tensorrt.tensorrt.UffParser`) → 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.UffParser`) → None

parse(*self*: `tensorrt.tensorrt.UffParser`, *file*: str, *network*: `tensorrt.tensorrt.INetworkDefinition`, *weights_type*: `tensorrt.tensorrt.DataType` = `<DataType.FLOAT: 0>`) → bool

Parse a UFF file.

Parameters

- **file** – File name of the UFF file.
- **network** – Network in which the `UffParser` will fill the layers.
- **weights_type** – The type on which the weights will be transformed in.

Returns True if the UFF file is parsed without error.

parse_buffer(*self: tensorrt.tensorrt.UffParser, buffer: buffer, network: tensorrt.tensorrt.INetworkDefinition, weights_type: tensorrt.tensorrt.DataType = <DataType.FLOAT: 0>*) → bool

Parse a UFF buffer - useful if the file is already live in memory.

Parameters

- **buffer** – The UFF buffer.
- **network** – Network in which the UFFParser will fill the layers.
- **weights_type** – The type on which the weights will be transformed in.

Returns True if the UFF buffer is parsed without error.

register_input(*self: tensorrt.tensorrt.UffParser, name: str, shape: tensorrt.tensorrt.Dims, order: tensorrt.tensorrt.UffInputOrder = <UffInputOrder.NCHW: 0>*) → bool

Register an input name of a UFF network with the associated Dimensions.

Parameters

- **name** – Input name.
- **shape** – Input shape.
- **order** – Input order on which the framework input was originally.

Returns True if the name registers without error.

register_output(*self: tensorrt.tensorrt.UffParser, name: str*) → bool

Register an output name of a UFF network.

Parameters **output_name** – Output name.

Returns True if the name registers without error.

9.1 Fields

tensorrt.FieldType

The possible field types for the custom layer.

Members:

FLOAT

INT32

CHAR

DIMS

DATATYPE

UNKNOWN

class tensorrt.**FieldMap**(*self: tensorrt.tensorrt.FieldMap, name: str, data: capsule, type: tensorrt.tensorrt.FieldType, length: int = 1*) → None

This is a class containing an array of field params used as a layer parameter for plugin layers. The node fields are passed by the parser to the API through the plugin constructor. The implementation of the plugin should parse the contents of the *FieldMap* as part of the plugin constructor.

Variables

- **name** – str field param
- **data** – capsule field param

- **type** – *FieldType* field param
- **length** – int field param

class `tensorrt.FieldCollection`

This class contains an array of *FieldMap* s.

Variables

- **num_fields** – int The number of *FieldMap* s.
- **fields** – capsule The array of *FieldMap* s.

CAFFE PARSER

class `tensorrt.IBlobNameToTensor`

This class is used to store and query *ITensor* s after they have been extracted from a Caffe model using the *CaffeParser* .

find(*self*: `tensorrt.tensorrt.IBlobNameToTensor`, *name*: *str*) → `tensorrt.tensorrt.ITensor`

Given a blob name, this function returns an *ITensor* object.

Parameters *name* – Caffe blob name for which the user wants the corresponding *ITensor* .

Returns A *ITensor* object corresponding to the queried name. If no such *ITensor* exists, then an empty object is returned.

class `tensorrt.CaffeParser`(*self*: `tensorrt.tensorrt.CaffeParser`) → None

This class is used for parsing Caffe models. It allows users to export models trained using Caffe to TRT.

Variables

- **plugin_factory_v2** – *ICaffePluginFactoryV2* The *ICaffePluginFactory* used to create the user defined plugins.
- **plugin_namespace** – *str* The namespace used to lookup and create plugins in the network.
- **protobuf_buffer_size** – *int* The buffer size for the parsing and storage of the learned model.
- **error_recorder** – *IErrorRecorder* Application-implemented error reporting interface for TensorRT objects.

__del__(*self*: `tensorrt.tensorrt.CaffeParser`) → 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.CaffeParser`) → None

parse(*self*: `tensorrt.tensorrt.CaffeParser`, *deploy*: *str*, *model*: *str*, *network*: `tensorrt.tensorrt.INetworkDefinition`, *dtype*: `tensorrt.tensorrt.DataType`) → `tensorrt.tensorrt.IBlobNameToTensor`

Parse a prototxt file and a binaryproto Caffe model to extract network definition and weights associated with the network, respectively.

Parameters

- **deploy** – The plain text, prototxt file used to define the network definition.
- **model** – The binaryproto Caffe model that contains the weights associated with the network.

- **network** – Network in which the CaffeParser will fill the layers.
- **dtype** – The type to which the weights will be transformed.

Returns An *IBlobNameToTensor* object that contains the extracted data.

parse_binary_proto(*self*: *tensorrt.tensorrt.CaffeParser*, *filename*: *str*) → *numpy.ndarray*

Parse and extract data stored in binaryproto file. The binaryproto file contains data stored in a binary blob.

parse_binary_proto() converts it to an *numpy.ndarray* object.

Parameters **filename** – Path to file containing binary proto.

Returns *numpy.ndarray* An array that contains the extracted data.

parse_buffer(*self*: *tensorrt.tensorrt.CaffeParser*, *deploy_buffer*: *buffer*, *model_buffer*: *buffer*, *network*: *tensorrt.tensorrt.INetworkDefinition*, *dtype*: *tensorrt.tensorrt.DataType*) → *tensorrt.tensorrt.IBlobNameToTensor*

Parse a prototxt file and a binaryproto Caffe model to extract network definition and weights associated with the network, respectively.

Parameters

- **deploy_buffer** – The memory buffer containing the plain text deploy prototxt used to define the network definition.
- **model_buffer** – The binaryproto Caffe memory buffer that contains the weights associated with the network.
- **network** – Network in which the CaffeParser will fill the layers.
- **dtype** – The type to which the weights will be transformed.

Returns An *IBlobNameToTensor* object that contains the extracted data.

tensorrt.shutdown_protobuf_library() → None

Shuts down protocol buffers library.

10.1 Plugins

class *tensorrt.ICaffePluginFactoryV2*

Plugin factory used to configure plugins.

create_plugin(*self*: *tensorrt.tensorrt.ICaffePluginFactoryV2*, *layer_name*: *str*, *weights*: *std::vector<nvinfer1::Weights>*, *std::allocator<nvinfer1::Weights>*) → *tensorrt.tensorrt.IPluginV2*

Creates a plugin.

arg layer_name Name of layer associated with the plugin.

arg weights Weights used for the layer.

Returns The newly created *IPluginV2* .

is_plugin_v2(*self*: *tensorrt.tensorrt.ICaffePluginFactoryV2*, *layer_name*: *str*) → bool

A user implemented function that determines if a layer configuration is provided by an *IPluginV2* .

Parameters **layer_name** – Name of the layer which the user wishes to validate.

Returns True if the the layer configuration is provided by an *IPluginV2* .

ONNX PARSER

```
class tensorrt.OnnxParser(self: tensorrt.tensorrt.OnnxParser, network: tensorrt.tensorrt.INetworkDefinition,  
                        logger: tensorrt.tensorrt.ILogger) → None
```

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()`

```
get_error(self: tensorrt.tensorrt.OnnxParser, index: int) → nvonnxparser::IParserError
```

Get an error that occurred during prior calls to `parse()`

Parameters `index` – Index of the error

```
parse(self: tensorrt.tensorrt.OnnxParser, model: 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_with_weight_descriptors(*self*: [tensorrt.tensorrt.OnnxParser](#), *model*: *buffer*) → bool

Parse a serialized ONNX model into the TensorRT network with consideration of user provided weights.

Parameters **model** – The serialized ONNX model.

Returns true if the model was parsed successfully

supports_model(*self*: [tensorrt.tensorrt.OnnxParser](#), *model*: *buffer*, *path*: *str* = *None*) → Tuple[bool, [tensorrt.tensorrt.SubGraphCollection](#)]

Check whether TensorRT supports a particular ONNX model.

Parameters

- **model** – The serialized ONNX model.
- **path** – The path to the model file. Only required if the model has externally stored weights.

Returns Tuple[bool, List[Tuple[NodeIndices, bool]]] The first element of the tuple indicates whether the model is supported. The second indicates subgraphs (by node index) in the model and whether they are 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

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

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

node(*self*: [tensorrt.tensorrt.ParserError](#)) → int

Returns Index of the Onnx model node in which the error occurred

UFF CONVERTER

The `uff` package contains a set of utilities to convert trained models from various frameworks to a common format.

12.1 Conversion Tools

12.1.1 Tensorflow Modelstream to UFF

`uff.from_tensorflow(graphdef, output_nodes=[], preprocessor=None, **kwargs)`

Converts a TensorFlow GraphDef to a UFF model.

Parameters

- **graphdef** (*tensorflow.GraphDef*) – The TensorFlow graph to convert.
- **output_nodes** (*list(str)*) – The names of the outputs of the graph. If not provided, `graphsurgeon` is used to automatically deduce output nodes.
- **output_filename** (*str*) – The UFF file to write.
- **preprocessor** (*str*) – The path to a preprocessing script that will be executed before the converter. This script should define a `preprocess` function which accepts a `graphsurgeon.DynamicGraph` and modifies it in place.
- **write_preprocessed** (*bool*) – If set to `True`, the converter will write out the preprocessed graph as well as a TensorBoard visualization. Must be used in conjunction with `output_filename`.
- **text** (*bool*) – If set to `True`, the converter will also write out a human readable UFF file. Must be used in conjunction with `output_filename`.
- **quiet** (*bool*) – If set to `True`, suppresses informational messages. Errors may still be printed.
- **debug_mode** (*bool*) – If set to `True`, the converter prints verbose debug messages.
- **return_graph_info** (*bool*) – If set to `True`, this function returns the graph input and output nodes in addition to the serialized UFF graph.

Returns

serialized UFF MetaGraph (*str*)

OR, if `return_graph_info` is set to `True`,

serialized UFF MetaGraph (*str*), graph inputs (*list(tensorflow.NodeDef)*), graph outputs (*list(tensorflow.NodeDef)*)

12.1.2 Tensorflow Frozen Protobuf Model to UFF

`uff.from_tensorflow_frozen_model(frozen_file, output_nodes=[], preprocessor=None, **kwargs)`

Converts a TensorFlow frozen graph to a UFF model.

Parameters

- **frozen_file** (*str*) – The path to the frozen TensorFlow graph to convert.
- **output_nodes** (*list(str)*) – The names of the outputs of the graph. If not provided, `graphsurgeon` is used to automatically deduce output nodes.
- **output_filename** (*str*) – The UFF file to write.
- **preprocessor** (*str*) – The path to a preprocessing script that will be executed before the converter. This script should define a `preprocess` function which accepts a `graphsurgeon.DynamicGraph` and modifies it in place.
- **write_preprocessed** (*bool*) – If set to True, the converter will write out the preprocessed graph as well as a TensorBoard visualization. Must be used in conjunction with `output_filename`.
- **text** (*bool*) – If set to True, the converter will also write out a human readable UFF file. Must be used in conjunction with `output_filename`.
- **quiet** (*bool*) – If set to True, suppresses informational messages. Errors may still be printed.
- **debug_mode** (*bool*) – If set to True, the converter prints verbose debug messages.
- **return_graph_info** (*bool*) – If set to True, this function returns the graph input and output nodes in addition to the serialized UFF graph.

Returns

serialized UFF MetaGraph (*str*)

OR, if `return_graph_info` is set to True,

serialized UFF MetaGraph (*str*), graph inputs (`list(tensorflow.NodeDef)`), graph outputs (`list(tensorflow.NodeDef)`)

UFF OPERATORS

All shapes include batch dimension, unless otherwise specified.

13.1 Input

An input to the network. Expects a CHW shape.

13.1.1 Supported Datatypes

float32, float16, int32, int8

13.2 Identity

Identity layer.

13.2.1 Inputs

Input0 [Tensor or Constant] Input0 to the identity.

13.2.2 Supported Datatypes

float32, float16, int32, int8

13.3 Const

A constant in the network. Should not include batch dimension.

13.3.1 Supported Datatypes

float32, float16, int32, int8

13.4 Conv

A convolution operation.

13.4.1 Inputs

Input0 [Tensor] The input to the convolution. Must be 4 dimensional. Automatically transposed to NCHW.

Kernel [Constant] The kernel weights for the convolution. Must be 4 dimensional. Automatically transposed to NCHW.

13.4.2 Attributes

dilation [List[int]] The HW dilations.

strides [List[int]] The HW strides.

padding [List[int]] The HW padding. Asymmetric padding is unsupported.

13.4.3 Supported Datatypes

float32, float16, int8

13.5 ConvTranspose

A transposed convolution, also known as deconvolution.

13.5.1 Inputs

Input0 [Tensor] The input to the transposed convolution. Must be 4 dimensional. Automatically transposed to NCHW.

Kernel [Constant] The kernel weights for the transposed convolution. Must be 4 dimensional. Automatically transposed to NCHW.

Shape [Constant] The HW dimensions of the output.

13.5.2 Attributes

strides [List[int]] The HW strides.

padding [List[int]] The HW padding. Asymmetric padding is unsupported.

13.5.3 Supported Datatypes

float32, float16, int8

13.6 Pool

A pooling layer.

13.6.1 Inputs

Input0 [Tensor] The input to the pooling layer. Must be 4 dimensional.

13.6.2 Attributes

func [Enum[max, avg]] The type of pooling to apply.

kernel [List[int]] The HW shape of the kernel.

strides [List[int]] The HW strides.

padding [List[int]] The HW padding.

13.6.3 Supported Datatypes

float32, float16, int8

13.7 FullyConnected

A fully connected layer.

13.7.1 Inputs

Input0 [Tensor] The input to the fully connected layer. Must be at least 4 dimensional. Automatically transposed to -NC-.

Weights [Constant] The weights for the fully connected layer. Must be 3 dimensional. Automatically transposed to CHW, where C is the number of output channels.

13.7.2 Supported Datatypes

float32, float16, int8

13.8 LRN

An LRN layer.

13.8.1 Inputs

Input0 [Tensor] The input to the LRN. Must be at least 4 dimensional.

13.8.2 Attributes

window_size [int] The window size.

alpha [double] The LRN alpha value.

beta [double] The LRN beta value.

k [double] The LRN k value.

13.8.3 Supported Datatypes

float32, float16, int8

13.9 Binary

A binary layer.

13.9.1 Inputs

Input0 [Tensor or Constant] The first input to the binary layer.

Input1 [Tensor or Constant] The second input to the binary layer.

If either input is a constant, then at least one of the inputs must be 4 dimensional.

13.9.2 Attributes

func [Enum[min, max, mul, sub, div, add, pow]] The type of operation to perform.

13.9.3 Supported Datatypes

float32, float16, int32, int8

13.10 Unary

A unary layer.

13.10.1 Inputs

Input0 [Tensor or Constant] The input to the unary layer.

The output of a unary layer with a Constant input is treated as a Constant, and therefore will not work with layers expecting a Tensor input.

13.10.2 Attributes

func [Enum[neg, exp, log, abs, sqrt, rsqrt, square, sin, cos, tan, sinh, cosh, asin, acos, atan, asinh, acosh, atanh, ceil, floor]]
The type of operation to perform.

13.10.3 Supported Datatypes

float32, float16, int32, int8

13.11 Reshape

A reshape layer. NOTE: this layer destroys order information. Therefore, subsequent layers will cease to automatically transpose their inputs to the correct format.

13.11.1 Inputs

Input0 [Tensor or Constant] The input to the reshape layer.

Shape [Constant] The desired shape. If the shape has fewer than 3 non-batch dimensions, 1s are inserted in the least significant dimensions. For example, if the shape specified is [1, 300, 5], it will be treated as [1, 300, 5, 1] instead. - 1 specifies that the dimension should be automatically deduced - this can only be used at most once in any given shape. - 0 specifies that the dimension should be copied from the input.

The output of a reshape layer with a Constant input is treated as a Constant, and therefore will not work with layers expecting a Tensor input.

13.11.2 Supported Datatypes

float32, float16, int32, int8

13.12 ExpandDims

An unsqueeze layer. NOTE: this layer destroys order information. Therefore, subsequent layers will cease to automatically transpose their inputs to the correct format.

13.12.1 Inputs

Input0 [Tensor] The input to unsqueeze.

If the resulting output has fewer than 3 non-batch dimensions, it is unsqueezed further with additional 1s inserted in the least significant dimensions. For example, with an input of shape [1, 300], and axis of 1, the shape of the output will be [1, 300, 1, 1] rather than [1, 300, 1].

13.12.2 Attributes

axis [int] The axis on which to unsqueeze, excluding batch dimension. For example, unsqueezing an NCHW tensor with an axis of 0 will result in a N1CHW tensor.

13.12.3 Supported Datatypes

float32, float16, int32, int8

13.13 ArgMax

An argmax layer.

13.13.1 Inputs

Input0 [Tensor] The input to the argmax layer.

13.13.2 Attributes

axis [int] The axis on which to perform the argmax, with 0 corresponding to the batch dimension. The specified dimension is removed. For example, performing argmax on an input of shape [1, 300, 150], with an axis of 1 would result in an output of shape [1, 150]. NOTE: argmax on the batch dimension is not supported.

13.13.3 Supported Datatypes

float32, float16, int8

13.14 ArgMin

An argmin layer.

13.14.1 Inputs

Input0 [Tensor] The input to the argmin layer.

13.14.2 Attributes

axis [int] The axis on which to perform the argmin, with 0 corresponding to the batch dimension. The specified dimension is removed. For example, performing argmin on an input of shape [1, 300, 150], with an axis of 1 would result in an output of shape [1, 150]. NOTE: argmin on the batch dimension is not supported.

13.14.3 Supported Datatypes

float32, float16, int8

13.15 Transpose

A transpose layer. A no-op in the network, this layer only modifies the UFF parser's internal order information. Therefore, when followed by any layer that destroys order information, the transpose will not be performed.

13.15.1 Inputs

Input0 [Tensor] The input to the transpose layer.

13.15.2 Attributes

permutation [int] The permutation to perform. Must be 4 dimensional.

13.15.3 Supported Datatypes

float32, float16, int32, int8

13.16 Reduce

A reduce layer. NOTE: this layer destroys order information. Therefore, subsequent layers will cease to automatically transpose their inputs to the correct format.

13.16.1 Inputs

Input0 [Tensor or Constant] The input to the reduce layer.

The output of a reduce layer with a Constant input is treated as a Constant, and therefore will not work with layers expecting a Tensor input.

13.16.2 Attributes

func [Enum[sum, prod, max, min, mean]] The reduction operation to perform.

axes [List[int]] The axes on which to reduce, with 0 corresponding to the batch dimension. Reduction on the batch dimension is unsupported.

keepdims [bool] Whether to keep the dimensions which were reduced. NOTE: The UFF parser ignored this value, and always keeps dimensions.

13.16.3 Supported Datatypes

float32, float16, int32, int8

13.17 Concat

A concatenation layer.

13.17.1 Inputs

Inputs (variadic) [List[Tensor]] The tensors to concatenate. All inputs are transposed to the same format as the first input. Inputs must be at least 4 dimensional.

13.17.2 Attributes

axis [int] The axis on which to perform the concatenation, with 0 corresponding to the batch dimension. Concatenating on the batch dimension is unsupported.

13.17.3 Supported Datatypes

float32, float16, int32, int8

13.18 MarkOutput

The output of the network.

13.18.1 Inputs

Inputs (variadic) [List[Tensor]] The inputs to this layer. Automatically transposed to the same order as the outputs of the original TensorFlow network.

13.18.2 Supported Datatypes

float32, float16, int32, int8

13.19 Activation

An activation layer.

13.19.1 Inputs

Input0 [Tensor] The input to the activation.

13.19.2 Attributes

func [Enum[relu, relu6, sigmoid, tanh, elu, selu, softsign, softplus]] The operation to perform.

13.19.3 Supported Datatypes

float32, float16, int8

13.20 Softmax

A softmax layer.

13.20.1 Inputs

Input0 [Tensor] The input to the softmax.

13.20.2 Attributes

axis [int] The axis on which to perform the reduction. NOTE: This value is ignored by the UFF parser.

13.20.3 Supported Datatypes

float32, float16, int8

13.21 BatchNorm

A batchnorm layer.

13.21.1 Inputs

Input0 [Tensor] The input to the batchnorm. Must be 4 dimensional.

Gamma [Constant] The gamma values.

Beta [Constant] The beta values.

Mean [Constant] The mean values.

Variance [Constant] The variance values.

13.21.2 Attributes

epsilon [double] The epsilon value.

13.21.3 Supported Datatypes

float32, float16, int8

13.22 Shape

A shape layer. Returns the shape of its input. NOTE: the output of this layer is a Constant, and therefore will not work with layers expecting a Tensor input.

13.22.1 Inputs

Input0 [Tensor] The input to the shape layer.

13.22.2 Supported Datatypes

float32, float16, int32, int8

13.23 StridedSlice

A strided slice layer.

13.23.1 Inputs

Input0 [Tensor or Constant] The input to the strided slice.

Begin [Constant] The indices at which to begin slicing.

End [Constant] The indices at which to end slicing.

Strides [Constant] Strides to use when slicing.

13.23.2 Attributes

begin_mask [int] See TensorFlow stridedSlice documentation.

end_mask [int] See TensorFlow stridedSlice documentation.

shrink_axis_mask [int] See TensorFlow stridedSlice documentation. This value is ignored unless the input is a constant.

13.23.3 Supported Datatypes

float32, float16, int32, int8

13.24 Stack

A stack layer. NOTE: the output of this layer is a Constant, and therefore will not work with layers expecting a Tensor input.

13.24.1 Inputs

Inputs (variadic) [**List**[**Constant**]] The inputs to the stack layer.

13.24.2 Attributes

axis [**int**] The axis on which to stack. NOTE: this value is ignored by the UFF parser.

13.24.3 Supported Datatypes

float32, float16, int32, int8

13.25 Squeeze

Not implemented

13.26 Flatten

A flatten layer. A no-op in the UFF parser.

13.26.1 Inputs

Input0 [**Tensor**] The tensor to flatten.

13.26.2 Supported Datatypes

float32, float16, int32, int8

13.27 Pad

A padding layer.

13.27.1 Inputs

Input0 [**Tensor or Constant**] The input to pad. The input is automatically transposed if padding is applied to non-HW dimensions.

Padding [**Constant**] The padding to apply. Padding is supported on 2 dimensions at most.

The output of a padding layer with a Constant input is treated as a Constant, and therefore will not work with layers expecting a Tensor input.

13.27.2 Supported Datatypes

float32, float16, int32, int8

13.28 Gather

A gather layer.

13.28.1 Inputs

Input0 [Tensor or Constant] The input to the gather layer. If the input is constant, it is assumed to be of shape NC11.

Indices [Tensor or Constant] The indices to gather along. These are assumed to be on the first non-batch dimension.

13.28.2 Supported Datatypes

float32, float16, int32, int8

13.29 GatherV2

A gatherV2 layer.

13.29.1 Inputs

Input0 [Tensor or Constant] The input to the gather layer. If the input is constant, it is assumed to be of shape NC11.

Indices [Tensor or Constant] The indices to gather along.

13.29.2 Attributes

axis [int] The axis along which to gather, excluding batch dimension.

13.29.3 Supported Datatypes

float32, float16, int32, int8

GRAPH SURGEON

`graphsurgeon` allows you to transform TensorFlow graphs. Its capabilities are broadly divided into two categories: search and manipulation. Search functions allow you to find nodes in a TensorFlow graph. Manipulation functions allow you to modify, add, or remove nodes.

14.1 Node Creation

Allow you to create free standing TensorFlow nodes, which can be used as stand-ins for plugins.

`graphsurgeon.create_node(name, op=None, trt_plugin=False, **kwargs)`
Creates a free-standing TensorFlow NodeDef with the specified properties.

Parameters

- **name** (*str*) – The name of the node.
- **op** (*str*) – The node’s operation.

Keyword Arguments

- **dtype** (*tensorflow.DType*) – TensorFlow dtype.
- **shape** (*tuple(int)*) – Iterable container (usually a tuple) describing the shape of a tensor.
- **inputs** (*list(tensorflow.NodeDef) or str*) – Iterable container (usually a tuple) of input nodes or input node names. Supports mixed-type lists.
- ****kwargs** (*AttrName=Value*) – Any additional fields that should be present in the node. Currently supports int, float, bool, list(int), list(float), str and NumPy arrays. NumPy arrays will be inserted into the “value” attribute of the node - this can be useful for creating constant nodes equivalent to those created by `tensorflow.constant`.

Returns `tensorflow.NodeDef`

`graphsurgeon.create_plugin_node(name, op=None, **kwargs)`
Creates a free-standing TensorFlow NodeDef with the specified properties. This is similar to `create_node`,

Parameters

- **name** (*str*) – The name of the node.
- **op** (*str*) – The node’s operation.
- **dtype** (*tensorflow.DType*) – TensorFlow dtype.
- **shape** (*tuple(int)*) – Iterable container (usually a tuple) describing the shape of a tensor.
- **inputs** (*list(tensorflow.NodeDef) or str*) – Iterable container (usually a tuple) of input nodes or input node names. Supports mixed-type lists.

- ****kwargs** (*AttrName=Value*) – Any additional fields that should be present in the node. Currently supports int, float, bool, list(int), list(float) and str.

Returns tensorflow.NodeDef

14.2 Static Graph

class graphsurgeon.StaticGraph(*graphdef=None*)

Acts as a thin wrapper for a read-only TensorFlow GraphDef. Supports indexing based on node name/index as well as iteration over nodes using Python's `for node in static_graph` syntax.

Parameters **graphdef** (*tensorflow.GraphDef/tensorflow.Graph OR graphsurgeon.StaticGraph/graphsurgeon.DynamicGraph OR str*) – A TensorFlow GraphDef/Graph or a StaticGraph from which to construct this graph, or a string containing a path to a frozen model.

node_outputs

A mapping of node names to their respective output nodes.

Type dict(str, list(tensorflow.NodeDef))

node_map

A mapping of node names to their corresponding nodes.

Type dict(str, tensorflow.NodeDef)

graph_outputs

A list of likely outputs of the graph.

Type list(tensorflow.NodeDef)

graph_inputs

A list of likely inputs of the graph.

Type list(tensorflow.NodeDef)

as_graph_def()

Returns this StaticGraph's internal TensorFlow GraphDef.

Parameters **None** –

Returns tensorflow.GraphDef

find_node_chains_by_op(chain)

Finds groups of nodes in this graph that match the specified sequence of ops. Returns a list of matching chains of nodes, with ordering preserved.

Parameters **chain** (*list(str)*) – The sequence of ops to look for. Should be ordered with the input of the chain as the first element, and the output as the last.

Returns list(list(tensorflow.NodeDef))

find_node_inputs(node)

Finds input nodes of a given node.

Parameters **node** (*tensorflow.NodeDef*) – The node in which to perform the search.

Returns list(tensorflow.NodeDef)

find_node_inputs_by_name(node, name)

Finds input nodes of a given node based on their names.

Parameters

- **node** (*tensorflow.NodeDef*) – The node in which to perform the search.
- **name** (*str OR list(str)*) – The name to look for. Also accepts iterable containers (preferably a list) to search for multiple names in a single pass. Supports regular expressions.

Returns *list(tensorflow.NodeDef)*

find_node_inputs_by_op(*node, op*)

Finds input nodes of a given node based on their ops.

Parameters

- **node** (*tensorflow.NodeDef*) – The node in which to perform the search.
- **op** (*str OR list(str)*) – The op to look for. Also accepts iterable containers (preferably a list) to search for multiple op in a single pass.

Returns *list(tensorflow.NodeDef)*

find_nodes_by_name(*name*)

Finds nodes in this graph based on their names.

Parameters **name** (*str OR list(str)*) – The name to look for. Also accepts iterable containers (preferably a list) to search for multiple names in a single pass of the graph. Supports regular expressions.

Returns *list(tensorflow.NodeDef)*

find_nodes_by_op(*op*)

Finds nodes in this graph based on their ops.

Parameters **op** (*str OR set(str)*) – The op to look for. Also accepts iterable containers (preferably hashsets) to search for multiple ops in a single pass of the graph.

Returns *list(tensorflow.NodeDef)*

find_nodes_by_path(*path*)

Finds nodes in this graph based on their full paths. This will only match exact paths.

Parameters **path** (*str OR list(str)*) – The path to look for. Also accepts iterable containers (preferably a list) to search for multiple paths in a single pass of the graph. Supports regular expressions.

Returns *list(tensorflow.NodeDef)*

read(*filename*)

Reads a frozen protobuf file into this StaticGraph.

Parameters **filename** (*str*) – Name of the protobuf file.

Returns *None*

write(*filename*)

Writes the StaticGraph's internal TensorFlow GraphDef into a frozen protobuf file.

Parameters **filename** (*str*) – Name of the protobuf file to write.

Returns *None*

write_tensorboard(*logdir*)

Writes the StaticGraph's internal TensorFlow GraphDef into the specified directory, which can then be visualized in TensorBoard.

Parameters **logdir** (*str*) – Name of the directory to write.

Returns None

Raises

- **Warning** – Passing a *GraphDef* to the SummaryWriter is deprecated. Pass a *Graph* object instead, such as *sess.graph*.
- **This is a known warning, but currently there is no alternative, since TensorFlow will not be able to convert invalid GraphDefs back to Graphs.** –

14.3 Dynamic Graph (Inherits from StaticGraph)

class graphsurgeon.**DynamicGraph**(graphdef=None)

A sub-class of StaticGraph that can search and modify a TensorFlow GraphDef.

Parameters **graphdef** (*tensorflow.GraphDef/tensorflow.Graph OR graphsurgeon.StaticGraph/graphsurgeon.DynamicGraph OR str*) – A TensorFlow GraphDef/Graph or a StaticGraph/DynamicGraph from which to construct this graph, or a string containing the path to a frozen model.

append(node)

Appends a node to this graph.

Parameters **node** (*tensorflow.NodeDef*) – TensorFlow NodeDef to add to the graph.

Returns None

collapse_namespaces(namespace_map, exclude_nodes=[], unique_inputs=True)

Collapses nodes in namespaces to single nodes specified by the user, except where those nodes are marked for exclusion.

Parameters

- **namespace_map** (*dict(str, tensorflow.NodeDef)*) – A dictionary specifying namespaces and their corresponding plugin nodes. These plugin nodes are typically used to specify attributes of the custom plugin, while inputs and outputs are automatically deduced. Multiple namespaces can be collapsed into a single plugin node, and nested namespaces are collapsed into plugin nodes outside their parent namespaces.
- **exclude_nodes** (*list(tensorflow.NodeDef)*) – Iterable container (usually a list) of nodes which should NOT be collapsed. These nodes will be present in the final graph as either inputs or outputs of the plugin nodes.
- **unique_inputs** (*bool*) – Whether inputs to the collapsed node should be unique. If this is false, plugin nodes may have duplicate inputs.

Returns None

extend(node_list)

Extends this graph's nodes based on the provided list.

Parameters **node_list** (*list(tensorflow.NodeDef)*) – List of TensorFlow NodeDefs to add to the graph.

Returns None

forward_inputs(nodes)

Removes nodes from this graph. Recursively forwards inputs, such that paths in the graph are preserved.

Warning: Nodes with control inputs are not removed, so as not to break the structure of the graph. If you need to forward these, remove their control inputs first.

Parameters **nodes** (*list(tensorflow.NodeDef)*) – Iterable container (usually a list) of nodes which should be removed and whose inputs forwarded.

Returns None

remove(*nodes, remove_exclusive_dependencies=False*)

Removes nodes from this graph. Does not forward inputs, so paths in the graph could be broken.

Parameters

- **nodes** (*list(tensorflow.NodeDef)*) – Iterable container (usually a list) of nodes which should be removed.
- **remove_exclusive_dependencies** (*bool*) – Whether to also remove dependencies exclusive to the nodes about to be removed. When set to True, all exclusive dependencies will be removed recursively, and the number of hanging nodes in the graph will remain constant. Defaults to False.

Returns None

Symbols

`__del__()` (*tensorrt.Builder* method), 21
`__del__()` (*tensorrt.CaffeParser* method), 115
`__del__()` (*tensorrt.IBuilderConfig* method), 17
`__del__()` (*tensorrt.ICudaEngine* method), 24
`__del__()` (*tensorrt.IExecutionContext* method), 31
`__del__()` (*tensorrt.IHostMemory* method), 8
`__del__()` (*tensorrt.INetworkDefinition* method), 45
`__del__()` (*tensorrt.OnnxParser* method), 117
`__del__()` (*tensorrt.Refitter* method), 37
`__del__()` (*tensorrt.Runtime* method), 36
`__del__()` (*tensorrt.UffParser* method), 111
`__exit__()` (*tensorrt.Builder* method), 21
`__exit__()` (*tensorrt.CaffeParser* method), 115
`__exit__()` (*tensorrt.IBuilderConfig* method), 17
`__exit__()` (*tensorrt.ICudaEngine* method), 24
`__exit__()` (*tensorrt.IExecutionContext* method), 31
`__exit__()` (*tensorrt.IHostMemory* method), 8
`__exit__()` (*tensorrt.INetworkDefinition* method), 45
`__exit__()` (*tensorrt.OnnxParser* method), 117
`__exit__()` (*tensorrt.Refitter* method), 37
`__exit__()` (*tensorrt.Runtime* method), 36
`__exit__()` (*tensorrt.UffParser* method), 111
`__getitem__()` (*tensorrt.ICudaEngine* method), 24
`__getitem__()` (*tensorrt.INetworkDefinition* method), 45
`__init__()` (*tensorrt.Builder* method), 21
`__init__()` (*tensorrt.CaffeParser* method), 115
`__init__()` (*tensorrt.EngineInspector* method), 43
`__init__()` (*tensorrt.IAlgorithm* method), 108
`__init__()` (*tensorrt.IAlgorithmContext* method), 107
`__init__()` (*tensorrt.IAlgorithmIOInfo* method), 107
`__init__()` (*tensorrt.IAlgorithmSelector* method), 108
`__init__()` (*tensorrt.IAlgorithmVariant* method), 107
`__init__()` (*tensorrt.IBuilderConfig* method), 17
`__init__()` (*tensorrt.ICudaEngine* method), 24
`__init__()` (*tensorrt.IExecutionContext* method), 31
`__init__()` (*tensorrt.IGpuAllocator* method), 42
`__init__()` (*tensorrt.IHostMemory* method), 8
`__init__()` (*tensorrt.INetworkDefinition* method), 45
`__init__()` (*tensorrt.IOutputAllocator* method), 29
`__init__()` (*tensorrt.OnnxParser* method), 117

`__init__()` (*tensorrt.Refitter* method), 37
`__init__()` (*tensorrt.Runtime* method), 36
`__init__()` (*tensorrt.UffParser* method), 111
`__len__()` (*tensorrt.ICudaEngine* method), 24
`__len__()` (*tensorrt.INetworkDefinition* method), 45

A

ActivationType (in module *tensorrt*), 67
 add_activation() (*tensorrt.INetworkDefinition* method), 45
 add_assertion() (*tensorrt.INetworkDefinition* method), 46
 add_cast() (*tensorrt.INetworkDefinition* method), 46
 add_concatenation() (*tensorrt.INetworkDefinition* method), 46
 add_constant() (*tensorrt.INetworkDefinition* method), 46
 add_convolution() (*tensorrt.INetworkDefinition* method), 46
 add_convolution_nd() (*tensorrt.INetworkDefinition* method), 47
 add_deconvolution() (*tensorrt.INetworkDefinition* method), 47
 add_deconvolution_nd() (*tensorrt.INetworkDefinition* method), 47
 add_dequantize() (*tensorrt.INetworkDefinition* method), 48
 add_einsum() (*tensorrt.INetworkDefinition* method), 48
 add_elementwise() (*tensorrt.INetworkDefinition* method), 48
 add_fill() (*tensorrt.INetworkDefinition* method), 48
 add_fully_connected() (*tensorrt.INetworkDefinition* method), 48
 add_gather() (*tensorrt.INetworkDefinition* method), 49
 add_gather_v2() (*tensorrt.INetworkDefinition* method), 49
 add_grid_sample() (*tensorrt.INetworkDefinition* method), 49
 add_identity() (*tensorrt.INetworkDefinition* method), 49
 add_if_conditional() (*tensorrt.INetworkDefinition* method), 49

`add_input()` (*tensorrt.IIfConditional method*), 90
`add_input()` (*tensorrt.INetworkDefinition method*), 50
`add_iterator()` (*tensorrt.ILoop method*), 85
`add_loop()` (*tensorrt.INetworkDefinition method*), 50
`add_loop_output()` (*tensorrt.ILoop method*), 85
`add_lrn()` (*tensorrt.INetworkDefinition method*), 50
`add_matrix_multiply()` (*tensorrt.INetworkDefinition method*), 50
`add_rms()` (*tensorrt.INetworkDefinition method*), 50
`add_non_zero()` (*tensorrt.INetworkDefinition method*), 51
`add_normalization()` (*tensorrt.INetworkDefinition method*), 51
`add_one_hot()` (*tensorrt.INetworkDefinition method*), 51
`add_optimization_profile()` (*tensorrt.IBuilderConfig method*), 17
`add_output()` (*tensorrt.IIfConditional method*), 90
`add_padding()` (*tensorrt.INetworkDefinition method*), 51
`add_padding_nd()` (*tensorrt.INetworkDefinition method*), 52
`add_parametric_relu()` (*tensorrt.INetworkDefinition method*), 52
`add_plugin_v2()` (*tensorrt.INetworkDefinition method*), 52
`add_pooling()` (*tensorrt.INetworkDefinition method*), 52
`add_pooling_nd()` (*tensorrt.INetworkDefinition method*), 52
`add_quantize()` (*tensorrt.INetworkDefinition method*), 53
`add_ragged_softmax()` (*tensorrt.INetworkDefinition method*), 53
`add_recurrence()` (*tensorrt.ILoop method*), 86
`add_reduce()` (*tensorrt.INetworkDefinition method*), 53
`add_resize()` (*tensorrt.INetworkDefinition method*), 53
`add_reverse_sequence()` (*tensorrt.INetworkDefinition method*), 54
`add_rnn_v2()` (*tensorrt.INetworkDefinition method*), 54
`add_scale()` (*tensorrt.INetworkDefinition method*), 55
`add_scale_nd()` (*tensorrt.INetworkDefinition method*), 55
`add_scatter()` (*tensorrt.INetworkDefinition method*), 56
`add_select()` (*tensorrt.INetworkDefinition method*), 56
`add_shape()` (*tensorrt.INetworkDefinition method*), 56
`add_shuffle()` (*tensorrt.INetworkDefinition method*), 56
`add_slice()` (*tensorrt.INetworkDefinition method*), 56
`add_softmax()` (*tensorrt.INetworkDefinition method*), 57
`add_topk()` (*tensorrt.INetworkDefinition method*), 57
`add_trip_limit()` (*tensorrt.ILoop method*), 86

`add_unary()` (*tensorrt.INetworkDefinition method*), 57
`allocate()` (*tensorrt.IGpuAllocator method*), 42
`AllocatorFlag` (in module *tensorrt*), 42
`append()` (*graphsurgeon.DynamicGraph method*), 140
`append()` (*tensorrt.PluginFieldCollection method*), 96
`as_graph_def()` (*graphsurgeon.StaticGraph method*), 138

B

`binding_is_input()` (*tensorrt.ICudaEngine method*), 24
`build_engine()` (*tensorrt.Builder method*), 21
`build_serialized_network()` (*tensorrt.Builder method*), 21
`Builder` (class in *tensorrt*), 21
`BuilderFlag` (in module *tensorrt*), 14

C

`CaffeParser` (class in *tensorrt*), 115
`CalibrationAlgoType` (in module *tensorrt*), 99
`can_run_on_DLA()` (*tensorrt.IBuilderConfig method*), 17
`clear()` (*tensorrt.IErrorRecorder method*), 40
`clear()` (*tensorrt.PluginFieldCollection method*), 96
`clear_errors()` (*tensorrt.OnnxParser method*), 117
`clear_flag()` (*tensorrt.IBuilderConfig method*), 17
`clear_quantization_flag()` (*tensorrt.IBuilderConfig method*), 17
`code()` (*tensorrt.ParserError method*), 118
`collapse_namespaces()` (*graphsurgeon.DynamicGraph method*), 140
`combine()` (*tensorrt.ITimingCache method*), 41
`create_builder_config()` (*tensorrt.Builder method*), 22
`create_engine_inspector()` (*tensorrt.ICudaEngine method*), 24
`create_execution_context()` (*tensorrt.ICudaEngine method*), 24
`create_execution_context_without_device_memory()` (*tensorrt.ICudaEngine method*), 24
`create_network()` (*tensorrt.Builder method*), 22
`create_node()` (in module *graphsurgeon*), 137
`create_optimization_profile()` (*tensorrt.Builder method*), 22
`create_plugin()` (*tensorrt.ICaffePluginFactoryV2 method*), 116
`create_plugin()` (*tensorrt.IPluginCreator method*), 96
`create_plugin_node()` (in module *graphsurgeon*), 137
`create_timing_cache()` (*tensorrt.IBuilderConfig method*), 17

D

`DataType` (in module *tensorrt*), 5
`deallocate()` (*tensorrt.IGpuAllocator method*), 42

deregister_creator() (*tensorrt.IPluginRegistry method*), 97
deregister_library() (*tensorrt.IPluginRegistry method*), 97
desc() (*tensorrt.ParserError method*), 118
deserialize_cuda_engine() (*tensorrt.Runtime method*), 36
deserialize_library() (*tensorrt.IPluginRegistry method*), 97
deserialize_plugin() (*tensorrt.IPluginCreator method*), 96
DeviceType (*in module tensorrt*), 13
Dims (*class in tensorrt*), 7
Dims2 (*class in tensorrt*), 7
Dims3 (*class in tensorrt*), 8
Dims4 (*class in tensorrt*), 8
DimsHW (*class in tensorrt*), 7
DynamicGraph (*class in graphsurgeon*), 140
E
ElementWiseOperation (*in module tensorrt*), 72
EngineCapability (*in module tensorrt*), 13
EngineInspector (*class in tensorrt*), 43
ErrorCode (*in module tensorrt*), 118
ErrorCodeTRT (*in module tensorrt*), 39
execute() (*tensorrt.IExecutionContext method*), 31
execute_async() (*tensorrt.IExecutionContext method*), 31
execute_async_v2() (*tensorrt.IExecutionContext method*), 32
execute_async_v3() (*tensorrt.IExecutionContext method*), 32
execute_v2() (*tensorrt.IExecutionContext method*), 32
extend() (*graphsurgeon.DynamicGraph method*), 140
extend() (*tensorrt.PluginFieldCollection method*), 96
F
FieldCollection (*class in tensorrt*), 113
FieldMap (*class in tensorrt*), 112
FieldType (*in module tensorrt*), 112
file() (*tensorrt.ParserError method*), 118
FillOperation (*in module tensorrt*), 88
find() (*tensorrt.IBlobNameToTensor method*), 115
find_node_chains_by_op() (*graphsurgeon.StaticGraph method*), 138
find_node_inputs() (*graphsurgeon.StaticGraph method*), 138
find_node_inputs_by_name() (*graphsurgeon.StaticGraph method*), 138
find_node_inputs_by_op() (*graphsurgeon.StaticGraph method*), 139
find_nodes_by_name() (*graphsurgeon.StaticGraph method*), 139
find_nodes_by_op() (*graphsurgeon.StaticGraph method*), 139
find_nodes_by_path() (*graphsurgeon.StaticGraph method*), 139
forward_inputs() (*graphsurgeon.DynamicGraph method*), 140
free() (*tensorrt.IGpuAllocator method*), 42
from_tensorflow() (*in module uff*), 121
from_tensorflow_frozen_model() (*in module uff*), 122
func() (*tensorrt.ParserError method*), 118
G
get_algorithm() (*tensorrt.IInt8Calibrator method*), 99
get_algorithm() (*tensorrt.IInt8EntropyCalibrator method*), 102
get_algorithm() (*tensorrt.IInt8EntropyCalibrator2 method*), 104
get_algorithm() (*tensorrt.IInt8LegacyCalibrator method*), 101
get_algorithm() (*tensorrt.IInt8MinMaxCalibrator method*), 105
get_algorithm_io_info() (*tensorrt.IAlgorithm method*), 108
get_all() (*tensorrt.Refit method*), 37
get_all_weights() (*tensorrt.Refit method*), 37
get_batch() (*tensorrt.IInt8Calibrator method*), 99
get_batch() (*tensorrt.IInt8EntropyCalibrator method*), 102
get_batch() (*tensorrt.IInt8EntropyCalibrator2 method*), 104
get_batch() (*tensorrt.IInt8LegacyCalibrator method*), 101
get_batch() (*tensorrt.IInt8MinMaxCalibrator method*), 105
get_batch_size() (*tensorrt.IInt8Calibrator method*), 100
get_batch_size() (*tensorrt.IInt8EntropyCalibrator method*), 103
get_batch_size() (*tensorrt.IInt8EntropyCalibrator2 method*), 104
get_batch_size() (*tensorrt.IInt8LegacyCalibrator method*), 101
get_batch_size() (*tensorrt.IInt8MinMaxCalibrator method*), 106
get_bias_for_gate() (*tensorrt.IRNNv2Layer method*), 76
get_binding_bytes_per_component() (*tensorrt.ICudaEngine method*), 24
get_binding_components_per_element() (*tensorrt.ICudaEngine method*), 24
get_binding_dtype() (*tensorrt.ICudaEngine method*), 25

`get_binding_format()` (*tensorrt.ICudaEngine method*), 25
`get_binding_format_desc()` (*tensorrt.ICudaEngine method*), 25
`get_binding_index()` (*tensorrt.ICudaEngine method*), 25
`get_binding_name()` (*tensorrt.ICudaEngine method*), 25
`get_binding_shape()` (*tensorrt.ICudaEngine method*), 26
`get_binding_shape()` (*tensorrt.IExecutionContext method*), 32
`get_binding_vectorized_dim()` (*tensorrt.ICudaEngine method*), 26
`get_builder_plugin_registry()` (*in module tensorrt*), 98
`get_calibration_profile()` (*tensorrt.IBuilderConfig method*), 17
`get_device_type()` (*tensorrt.IBuilderConfig method*), 17
`get_dimension_name()` (*tensorrt.ITensor method*), 61
`get_dynamic_range()` (*tensorrt.Refit method*), 37
`get_engine_information()` (*tensorrt.EngineInspector method*), 43
`get_error()` (*tensorrt.OnnxParser method*), 117
`get_error_code()` (*tensorrt.IErrorRecorder method*), 40
`get_error_desc()` (*tensorrt.IErrorRecorder method*), 40
`get_flag()` (*tensorrt.IBuilderConfig method*), 18
`get_input()` (*tensorrt.ILayer method*), 63
`get_input()` (*tensorrt.INetworkDefinition method*), 57
`get_input_consumed_event()` (*tensorrt.IExecutionContext method*), 33
`get_layer()` (*tensorrt.INetworkDefinition method*), 57
`get_layer_information()` (*tensorrt.EngineInspector method*), 43
`get_location()` (*tensorrt.ICudaEngine method*), 26
`get_max_output_size()` (*tensorrt.IExecutionContext method*), 33
`get_memory_pool_limit()` (*tensorrt.IBuilderConfig method*), 18
`get_missing()` (*tensorrt.Refit method*), 37
`get_missing_weights()` (*tensorrt.Refit method*), 38
`get_output()` (*tensorrt.ILayer method*), 63
`get_output()` (*tensorrt.INetworkDefinition method*), 58
`get_output_allocator()` (*tensorrt.IExecutionContext method*), 33
`get_output_type()` (*tensorrt.ILayer method*), 63
`get_plugin_creator()` (*tensorrt.IPluginRegistry method*), 97
`get_plugin_registry()` (*in module tensorrt*), 98
`get_plugin_registry()` (*tensorrt.Builder method*), 22
`get_plugin_registry()` (*tensorrt.Runtime method*), 36
`get_preview_feature()` (*tensorrt.IBuilderConfig method*), 18
`get_profile_shape()` (*tensorrt.ICudaEngine method*), 26
`get_profile_shape_input()` (*tensorrt.ICudaEngine method*), 27
`get_quantization_flag()` (*tensorrt.IBuilderConfig method*), 18
`get_shape()` (*tensorrt.IAlgorithmContext method*), 107
`get_shape()` (*tensorrt.IExecutionContext method*), 33
`get_shape()` (*tensorrt.IOptimizationProfile method*), 11
`get_shape_input()` (*tensorrt.IOptimizationProfile method*), 11
`get_strides()` (*tensorrt.IExecutionContext method*), 33
`get_tactic_sources()` (*tensorrt.IBuilderConfig method*), 18
`get_tensor_address()` (*tensorrt.IExecutionContext method*), 33
`get_tensor_bytes_per_component()` (*tensorrt.ICudaEngine method*), 27
`get_tensor_components_per_element()` (*tensorrt.ICudaEngine method*), 27
`get_tensor_dtype()` (*tensorrt.ICudaEngine method*), 27
`get_tensor_format()` (*tensorrt.ICudaEngine method*), 27
`get_tensor_format_desc()` (*tensorrt.ICudaEngine method*), 28
`get_tensor_location()` (*tensorrt.ICudaEngine method*), 28
`get_tensor_mode()` (*tensorrt.ICudaEngine method*), 28
`get_tensor_name()` (*tensorrt.ICudaEngine method*), 28
`get_tensor_profile_shape()` (*tensorrt.ICudaEngine method*), 28
`get_tensor_shape()` (*tensorrt.ICudaEngine method*), 28
`get_tensor_shape()` (*tensorrt.IExecutionContext method*), 33
`get_tensor_strides()` (*tensorrt.IExecutionContext method*), 33
`get_tensor_vectorized_dim()` (*tensorrt.ICudaEngine method*), 28
`get_tensors_with_dynamic_range()` (*tensorrt.Refit method*), 38
`get_timing_cache()` (*tensorrt.IBuilderConfig method*), 18
`get_weights_for_gate()` (*tensorrt.IRNNv2Layer method*), 76
`graph_inputs` (*graphsurgeon.StaticGraph attribute*), 138

`graph_outputs` (*graphsurgeon.StaticGraph* attribute), 138

H

`has_overflowed()` (*tensorrt.IErrorRecorder* method), 40

I

`IActivationLayer` (class in *tensorrt*), 67
`IAlgorithm` (class in *tensorrt*), 108
`IAlgorithmContext` (class in *tensorrt*), 107
`IAlgorithmIOInfo` (class in *tensorrt*), 107
`IAlgorithmSelector` (class in *tensorrt*), 108
`IAlgorithmVariant` (class in *tensorrt*), 107
`IAssertionLayer` (class in *tensorrt*), 92
`IBlobNameToTensor` (class in *tensorrt*), 115
`IBuilderConfig` (class in *tensorrt*), 16
`ICaffePluginFactoryV2` (class in *tensorrt*), 116
`IConcatenationLayer` (class in *tensorrt*), 71
`IConditionLayer` (class in *tensorrt*), 91
`IConstantLayer` (class in *tensorrt*), 83
`IConvolutionLayer` (class in *tensorrt*), 65
`ICudaEngine` (class in *tensorrt*), 23
`IDEconvolutionLayer` (class in *tensorrt*), 71
`IDEquantizeLayer` (class in *tensorrt*), 89
`IEinsumLayer` (class in *tensorrt*), 91
`IElementWiseLayer` (class in *tensorrt*), 72
`IErrorRecorder` (class in *tensorrt*), 40
`IExecutionContext` (class in *tensorrt*), 30
`IFillLayer` (class in *tensorrt*), 88
`IFullyConnectedLayer` (class in *tensorrt*), 66
`IGatherLayer` (class in *tensorrt*), 72
`IGpuAllocator` (class in *tensorrt*), 42
`IGridSampleLayer` (class in *tensorrt*), 67
`IHostMemory` (class in *tensorrt*), 8
`IIdentityLayer` (class in *tensorrt*), 83
`IIfConditional` (class in *tensorrt*), 90
`IIfConditionalInputLayer` (class in *tensorrt*), 91
`IIfConditionalOutputLayer` (class in *tensorrt*), 91
`IInt8Calibrator` (class in *tensorrt*), 99
`IInt8EntropyCalibrator` (class in *tensorrt*), 102
`IInt8EntropyCalibrator2` (class in *tensorrt*), 104
`IInt8LegacyCalibrator` (class in *tensorrt*), 101
`IInt8MinMaxCalibrator` (class in *tensorrt*), 105
`IIteratorLayer` (class in *tensorrt*), 87
`ILayer` (class in *tensorrt*), 63
`ILogger` (class in *tensorrt*), 9
`ILogger.Severity` (class in *tensorrt*), 9
`ILoop` (class in *tensorrt*), 85
`ILoopBoundaryLayer` (class in *tensorrt*), 86
`ILoopOutputLayer` (class in *tensorrt*), 87
`ILRNLayer` (class in *tensorrt*), 69
`IMatrixMultiplyLayer` (class in *tensorrt*), 82
`INetworkDefinition` (class in *tensorrt*), 45

`infer_shapes()` (*tensorrt.IExecutionContext* method), 33

`init_libnvinfer_plugins()` (in module *tensorrt*), 98
`INMSLayer` (class in *tensorrt*), 92
`INonZeroLayer` (class in *tensorrt*), 92
`INormalizationLayer` (class in *tensorrt*), 94
`insert()` (*tensorrt.PluginFieldCollection* method), 96
`InterpolationMode` (in module *tensorrt*), 66
`IOneHotLayer` (class in *tensorrt*), 92
`IOptimizationProfile` (class in *tensorrt*), 11
`IOutputAllocator` (class in *tensorrt*), 29
`IPaddingLayer` (class in *tensorrt*), 78
`IParametricReLULayer` (class in *tensorrt*), 79
`IPluginCreator` (class in *tensorrt*), 96
`IPluginRegistry` (class in *tensorrt*), 97
`IPluginV2Layer` (class in *tensorrt*), 77
`IPoolingLayer` (class in *tensorrt*), 68
`IProfiler` (class in *tensorrt*), 10
`IQuantizeLayer` (class in *tensorrt*), 89
`IRaggedSoftMaxLayer` (class in *tensorrt*), 83
`IRecurrenceLayer` (class in *tensorrt*), 87
`IReduceLayer` (class in *tensorrt*), 78
`IResizeLayer` (class in *tensorrt*), 84
`IRReverseSequenceLayer` (class in *tensorrt*), 94
`IRNNv2Layer` (class in *tensorrt*), 75
`is_device_type_set()` (*tensorrt.IBuilderConfig* method), 18
`is_execution_binding()` (*tensorrt.ICudaEngine* method), 28
`is_network_supported()` (*tensorrt.Builder* method), 22
`is_plugin_v2()` (*tensorrt.ICaffePluginFactoryV2* method), 116
`is_shape_binding()` (*tensorrt.ICudaEngine* method), 29
`is_shape_inference_io()` (*tensorrt.ICudaEngine* method), 29
`IScaleLayer` (class in *tensorrt*), 69
`IScatterLayer` (class in *tensorrt*), 90
`ISelectLayer` (class in *tensorrt*), 79
`IShapeLayer` (class in *tensorrt*), 81
`IShuffleLayer` (class in *tensorrt*), 79
`ISliceLayer` (class in *tensorrt*), 80
`ISoftMaxLayer` (class in *tensorrt*), 70
`ITensor` (class in *tensorrt*), 60
`ITimingCache` (class in *tensorrt*), 41
`ITopKLayer` (class in *tensorrt*), 82
`ITripLimitLayer` (class in *tensorrt*), 86
`IUnaryLayer` (class in *tensorrt*), 78

L

`LayerType` (in module *tensorrt*), 62
`line()` (*tensorrt.ParserError* method), 119
`load_library()` (*tensorrt.IPluginRegistry* method), 98

`log()` (*tensorrt.ILogger method*), 9
`log()` (*tensorrt.Logger method*), 10
`Logger` (*class in tensorrt*), 10
`LoopOutput` (*in module tensorrt*), 87

M

`mark_output()` (*tensorrt.INetworkDefinition method*), 58
`mark_output_for_shapes()` (*tensorrt.INetworkDefinition method*), 58
`MatrixOperation` (*in module tensorrt*), 82
`MAX_DIMS` (*tensorrt.Dims property*), 7
`MemoryPoolType` (*in module tensorrt*), 15

N

`name` (*tensorrt.ILogger.Severity property*), 9
`NetworkDefinitionCreationFlag` (*in module tensorrt*), 20
`node()` (*tensorrt.ParserError method*), 119
`node_map` (*graphsurgeon.StaticGraph attribute*), 138
`node_outputs` (*graphsurgeon.StaticGraph attribute*), 138
`notify_shape()` (*tensorrt.IOutputAllocator method*), 30
`nptype()` (*in module tensorrt*), 5
`num_errors()` (*tensorrt.IErrorRecorder method*), 40
`numpy()` (*tensorrt.Weights method*), 6

O

`OnnxParser` (*class in tensorrt*), 117
`output_type_is_set()` (*tensorrt.ILayer method*), 63

P

`PaddingMode` (*in module tensorrt*), 64
`parse()` (*tensorrt.CaffeParser method*), 115
`parse()` (*tensorrt.OnnxParser method*), 117
`parse()` (*tensorrt.UffParser method*), 111
`parse_binary_proto()` (*tensorrt.CaffeParser method*), 116
`parse_buffer()` (*tensorrt.CaffeParser method*), 116
`parse_buffer()` (*tensorrt.UffParser method*), 111
`parse_from_file()` (*tensorrt.OnnxParser method*), 117
`parse_with_weight_descriptors()` (*tensorrt.OnnxParser method*), 118
`ParserError` (*class in tensorrt*), 118
`Permutation` (*class in tensorrt*), 79
`PluginField` (*class in tensorrt*), 95
`PluginFieldCollection` (*class in tensorrt*), 95
`PluginFieldType` (*in module tensorrt*), 95
`PoolingType` (*in module tensorrt*), 68
`pop()` (*tensorrt.PluginFieldCollection method*), 96
`PreviewFeature` (*in module tensorrt*), 15

`Profiler` (*class in tensorrt*), 10
`ProfilingVerbosity` (*in module tensorrt*), 13

Q

`QuantizationFlag` (*in module tensorrt*), 13

R

`read()` (*graphsurgeon.StaticGraph method*), 139
`read_calibration_cache()` (*tensorrt.IInt8Calibrator method*), 100
`read_calibration_cache()` (*tensorrt.IInt8EntropyCalibrator method*), 103
`read_calibration_cache()` (*tensorrt.IInt8EntropyCalibrator2 method*), 104
`read_calibration_cache()` (*tensorrt.IInt8LegacyCalibrator method*), 101
`read_calibration_cache()` (*tensorrt.IInt8MinMaxCalibrator method*), 106
`reallocate()` (*tensorrt.IGpuAllocator method*), 42
`reallocate_output()` (*tensorrt.IOutputAllocator method*), 30
`ReduceOperation` (*in module tensorrt*), 78
`refit_cuda_engine()` (*tensorrt.Refit method*), 38
`Refitter` (*class in tensorrt*), 37
`register_creator()` (*tensorrt.IPluginRegistry method*), 98
`register_input()` (*tensorrt.UffParser method*), 112
`register_output()` (*tensorrt.UffParser method*), 112
`remove()` (*graphsurgeon.DynamicGraph method*), 141
`remove_tensor()` (*tensorrt.INetworkDefinition method*), 58
`report_algorithms()` (*tensorrt.IAlgorithmSelector method*), 108
`report_error()` (*tensorrt.IErrorRecorder method*), 41
`report_layer_time()` (*tensorrt.IProfiler method*), 10
`report_layer_time()` (*tensorrt.Profiler method*), 10
`report_to_profiler()` (*tensorrt.IExecutionContext method*), 34
`reset()` (*tensorrt.Builder method*), 22
`reset()` (*tensorrt.IBuilderConfig method*), 18
`reset()` (*tensorrt.ITimingCache method*), 41
`reset_device_type()` (*tensorrt.IBuilderConfig method*), 18
`reset_dynamic_range()` (*tensorrt.ITensor method*), 61
`reset_output_type()` (*tensorrt.ILayer method*), 64
`reset_precision()` (*tensorrt.ILayer method*), 64
`ResizeMode` (*in module tensorrt*), 84
`RNNDirection` (*in module tensorrt*), 74
`RNNGateType` (*in module tensorrt*), 75
`RNNInputMode` (*in module tensorrt*), 74
`RNNOperation` (*in module tensorrt*), 73
`Runtime` (*class in tensorrt*), 36

S

SampleMode (in module *tensorrt*), 66
 ScaleMode (in module *tensorrt*), 69
 select_algorithms() (*tensorrt.IAlgorithmSelector* method), 109
 serialize() (*tensorrt.ICudaEngine* method), 29
 serialize() (*tensorrt.ITimingCache* method), 41
 set_bias_for_gate() (*tensorrt.IRNNv2Layer* method), 76
 set_binding_shape() (*tensorrt.IExecutionContext* method), 34
 set_calibration_profile() (*tensorrt.IBuilderConfig* method), 18
 set_condition() (*tensorrt.IIfConditional* method), 90
 set_device_type() (*tensorrt.IBuilderConfig* method), 19
 set_dimension_name() (*tensorrt.ITensor* method), 61
 set_dynamic_range() (*tensorrt.ITensor* method), 61
 set_dynamic_range() (*tensorrt.Refitter* method), 38
 set_flag() (*tensorrt.IBuilderConfig* method), 19
 set_input() (*tensorrt.IFillLayer* method), 88
 set_input() (*tensorrt.ILayer* method), 64
 set_input() (*tensorrt.ILoopOutputLayer* method), 88
 set_input() (*tensorrt.INMSLayer* method), 93
 set_input() (*tensorrt.IRecurrenceLayer* method), 87
 set_input() (*tensorrt.IResizeLayer* method), 85
 set_input() (*tensorrt.IShuffleLayer* method), 80
 set_input() (*tensorrt.ISliceLayer* method), 81
 set_input() (*tensorrt.ITopKLayer* method), 82
 set_input_consumed_event() (*tensorrt.IExecutionContext* method), 34
 set_input_shape() (*tensorrt.IExecutionContext* method), 35
 set_memory_pool_limit() (*tensorrt.IBuilderConfig* method), 19
 set_named_weights() (*tensorrt.Refitter* method), 38
 set_optimization_profile_async() (*tensorrt.IExecutionContext* method), 35
 set_output_allocator() (*tensorrt.IExecutionContext* method), 35
 set_output_type() (*tensorrt.ILayer* method), 64
 set_preview_feature() (*tensorrt.IBuilderConfig* method), 19
 set_quantization_flag() (*tensorrt.IBuilderConfig* method), 20
 set_shape() (*tensorrt.IOptimizationProfile* method), 11
 set_shape_input() (*tensorrt.IExecutionContext* method), 35
 set_shape_input() (*tensorrt.IOptimizationProfile* method), 12
 set_tactic_sources() (*tensorrt.IBuilderConfig* method), 20
 set_tensor_address() (*tensorrt.IExecutionContext* method), 36

set_timing_cache() (*tensorrt.IBuilderConfig* method), 20
 set_weights() (*tensorrt.Refitter* method), 38
 set_weights_for_gate() (*tensorrt.IRNNv2Layer* method), 76
 set_weights_name() (*tensorrt.INetworkDefinition* method), 58
 shutdown_protobuf_library() (in module *tensorrt*), 116
 SliceMode (in module *tensorrt*), 80
 StaticGraph (class in *graphsurgeon*), 138
 supports_model() (*tensorrt.OnnxParser* method), 118
 supports_operator() (*tensorrt.OnnxParser* method), 118

T

TacticSource (in module *tensorrt*), 13
 TensorFormat (in module *tensorrt*), 59
 TensorIOMode (in module *tensorrt*), 23
 TensorLocation (in module *tensorrt*), 59
 TopKOperation (in module *tensorrt*), 82
 TripLimit (in module *tensorrt*), 86

U

UffInputOrder (in module *tensorrt*), 111
 UffParser (class in *tensorrt*), 111
 UnaryOperation (in module *tensorrt*), 77
 unmark_output() (*tensorrt.INetworkDefinition* method), 58
 unmark_output_for_shapes() (*tensorrt.INetworkDefinition* method), 58

V

volume() (in module *tensorrt*), 6

W

Weights (class in *tensorrt*), 6
 WeightsRole (in module *tensorrt*), 6
 write() (*graphsurgeon.StaticGraph* method), 139
 write_calibration_cache() (*tensorrt.IInt8Calibrator* method), 100
 write_calibration_cache() (*tensorrt.IInt8EntropyCalibrator* method), 103
 write_calibration_cache() (*tensorrt.IInt8EntropyCalibrator2* method), 105
 write_calibration_cache() (*tensorrt.IInt8LegacyCalibrator* method), 102
 write_calibration_cache() (*tensorrt.IInt8MinMaxCalibrator* method), 106
 write_tensorboard() (*graphsurgeon.StaticGraph* method), 139

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

ARM

ARM, AMBA and ARM Powered are registered trademarks of ARM Limited. Cortex, MPCore and Mali are trademarks of ARM Limited. "ARM" is used to represent ARM Holdings plc; its operating company ARM Limited; and the regional subsidiaries ARM Inc.; ARM KK; ARM Korea Limited.; ARM Taiwan Limited; ARM France SAS; ARM Consulting (Shanghai) Co. Ltd.; ARM Germany GmbH; ARM Embedded Technologies Pvt. Ltd.; ARM Norway, AS and ARM Sweden AB.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

BlackBerry/QNX

Copyright © 2020 BlackBerry Limited. All rights reserved.

Trademarks, including but not limited to BLACKBERRY, EMBLEM Design, QNX, AVIAGE, MOMENTICS, NEUTRINO and QNX CAR are the trademarks or registered trademarks of BlackBerry Limited, used under license, and the exclusive rights to such trademarks are expressly reserved.

Google

Android, Android TV, Google Play and the Google Play logo are trademarks of Google, Inc.

Trademarks

NVIDIA, the NVIDIA logo, and CUDA, DALI, DGX-1, DRIVE, JetPack, Orin, Pegasus, TensorRT, Triton and Xavier are trademarks and/or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2017-2022 NVIDIA Corporation & affiliates. All rights reserved.

