



NVIDIA TensorRT 8.6.13 API Reference

for DRIVE OS

TensorRT API Reference

8.6.13

January 2024

Table 1 Revision history

Date	Summary of change
TBD	Initial draft
TBD	Review
TBD	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	27
7.1	File List	27
8	Namespace Documentation	29
8.1	nvcaffeparser1 Namespace Reference	29
8.1.1	Detailed Description	29
8.1.2	Function Documentation	29
8.1.2.1	createCaffeParser()	30
8.1.2.2	shutdownProtobufLibrary()	30
8.2	nvinfer1 Namespace Reference	30
8.2.1	Detailed Description	40
8.2.2	Typedef Documentation	40
8.2.2.1	AllocatorFlags	40
8.2.2.2	AsciiChar	40
8.2.2.3	BuilderFlags	40
8.2.2.4	char_t	40
8.2.2.5	Dims	40
8.2.2.6	NetworkDefinitionCreationFlags	41
8.2.2.7	PluginFormat	41
8.2.2.8	QuantizationFlags	41
8.2.2.9	ResizeMode	42
8.2.2.10	SliceMode	42
8.2.2.11	TacticSources	42
8.2.2.12	TempfileControlFlags	42
8.2.2.13	TensorFormats	43
8.2.3	Enumeration Type Documentation	43
8.2.3.1	ActivationType	43
8.2.3.2	AllocatorFlag	43

8.2.3.3	BoundingBoxFormat	44
8.2.3.4	BuilderFlag	44
8.2.3.5	CalibrationAlgoType	46
8.2.3.6	DataType	46
8.2.3.7	DeviceType	47
8.2.3.8	DimensionOperation	47
8.2.3.9	ElementWiseOperation	48
8.2.3.10	EngineCapability	49
8.2.3.11	ErrorCode	49
8.2.3.12	FillOperation	51
8.2.3.13	GatherMode	51
8.2.3.14	HardwareCompatibilityLevel	52
8.2.3.15	InterpolationMode	52
8.2.3.16	LayerInformationFormat	52
8.2.3.17	LayerType	53
8.2.3.18	LoopOutput	54
8.2.3.19	MatrixOperation	55
8.2.3.20	MemoryPoolType	55
8.2.3.21	NetworkDefinitionCreationFlag	56
8.2.3.22	OptProfileSelector	56
8.2.3.23	PaddingMode	57
8.2.3.24	PluginFieldType	60
8.2.3.25	PluginVersion	60
8.2.3.26	PoolingType	61
8.2.3.27	PreviewFeature	61
8.2.3.28	ProfilingVerbosity	62
8.2.3.29	QuantizationFlag	62
8.2.3.30	ReduceOperation	63
8.2.3.31	ResizeCoordinateTransformation	63
8.2.3.32	ResizeRoundMode	64
8.2.3.33	ResizeSelector	64
8.2.3.34	RNNDirection	65
8.2.3.35	RNNGateType	65
8.2.3.36	RNNInputMode	66
8.2.3.37	RNNOperation	66
8.2.3.38	SampleMode	67
8.2.3.39	ScaleMode	68
8.2.3.40	ScatterMode	68
8.2.3.41	TacticSource	69
8.2.3.42	TempfileControlFlag	70
8.2.3.43	TensorFormat	70
8.2.3.44	TensorIOMode	72
8.2.3.45	TensorLocation	72
8.2.3.46	TopKOperation	72
8.2.3.47	TripLimit	73
8.2.3.48	UnaryOperation	73
8.2.3.49	WeightsRole	74
8.2.4	Function Documentation	74
8.2.4.1	EnumMax()	75
8.2.4.2	EnumMax< BoundingBoxFormat >()	75
8.2.4.3	EnumMax< BuilderFlag >()	75
8.2.4.4	EnumMax< CalibrationAlgoType >()	75
8.2.4.5	EnumMax< DeviceType >()	76
8.2.4.6	EnumMax< DimensionOperation >()	76

8.2.4.7	EnumMax< FillOperation >()	76
8.2.4.8	EnumMax< GatherMode >()	76
8.2.4.9	EnumMax< LayerInformationFormat >()	77
8.2.4.10	EnumMax< LayerType >()	77
8.2.4.11	EnumMax< LoopOutput >()	77
8.2.4.12	EnumMax< MatrixOperation >()	77
8.2.4.13	EnumMax< MemoryPoolType >()	78
8.2.4.14	EnumMax< NetworkDefinitionCreationFlag >()	78
8.2.4.15	EnumMax< OptProfileSelector >()	78
8.2.4.16	EnumMax< ProfilingVerbosity >()	78
8.2.4.17	EnumMax< QuantizationFlag >()	79
8.2.4.18	EnumMax< ReduceOperation >()	79
8.2.4.19	EnumMax< RNNDirection >()	79
8.2.4.20	EnumMax< RNNGateType >()	79
8.2.4.21	EnumMax< RNNInputMode >()	80
8.2.4.22	EnumMax< RNNOperation >()	80
8.2.4.23	EnumMax< SampleMode >()	80
8.2.4.24	EnumMax< ScaleMode >()	80
8.2.4.25	EnumMax< ScatterMode >()	81
8.2.4.26	EnumMax< TacticSource >()	81
8.2.4.27	EnumMax< TempfileControlFlag >()	81
8.2.4.28	EnumMax< TopKOperation >()	81
8.2.4.29	EnumMax< TripLimit >()	82
8.2.4.30	EnumMax< UnaryOperation >()	82
8.2.4.31	EnumMax< WeightsRole >()	82
8.2.4.32	getBuilderPluginRegistry()	82
8.2.4.33	getBuilderSafePluginRegistry()	83
8.3	nvinfer1::consistency Namespace Reference	83
8.4	nvinfer1::impl Namespace Reference	83
8.5	nvinfer1::plugin Namespace Reference	84
8.5.1	Enumeration Type Documentation	84
8.5.1.1	CodeTypeSSD	84
8.6	nvinfer1::safe Namespace Reference	85
8.6.1	Detailed Description	86
8.6.2	Enumeration Type Documentation	86
8.6.2.1	RuntimeErrorType	86
8.6.3	Function Documentation	86
8.6.3.1	createInferRuntime()	86
8.6.3.2	getSafePluginRegistry()	87
8.7	nvinfer1::utils Namespace Reference	87
8.7.1	Function Documentation	87
8.7.1.1	reorderSubBuffers()	87
8.7.1.2	reshapeWeights()	88
8.7.1.3	transposeSubBuffers()	89
8.8	nvonnxparser Namespace Reference	90
8.8.1	Detailed Description	91
8.8.2	Typedef Documentation	91
8.8.2.1	OnnxParserFlags	91
8.8.3	Enumeration Type Documentation	91
8.8.3.1	ErrorCode	91
8.8.3.2	OnnxParserFlag	92
8.8.4	Function Documentation	92
8.8.4.1	createONNXConfig()	92
8.8.4.2	EnumMax()	92

8.8.4.3	EnumMax< ErrorCode >()	92
8.8.4.4	EnumMax< OnnxParserFlag >()	93
8.9	nvuffparser Namespace Reference	93
8.9.1	Detailed Description	93
8.9.2	Enumeration Type Documentation	94
8.9.2.1	FieldType	94
8.9.2.2	UffInputOrder	95
8.9.3	Function Documentation	95
8.9.3.1	createUffParser()	95
8.9.3.2	shutdownProtobufLibrary()	96
9	Class Documentation	97
9.1	nvinfer1::plugin::DetectionOutputParameters Struct Reference	97
9.1.1	Detailed Description	97
9.1.2	Member Data Documentation	98
9.1.2.1	backgroundLabelId	98
9.1.2.2	codeType	98
9.1.2.3	confidenceThreshold	98
9.1.2.4	confSigmoid	99
9.1.2.5	inputOrder	99
9.1.2.6	isBatchAgnostic	99
9.1.2.7	isNormalized	99
9.1.2.8	keepTopK	99
9.1.2.9	nmsThreshold	99
9.1.2.10	numClasses	99
9.1.2.11	shareLocation	100
9.1.2.12	topK	100
9.1.2.13	varianceEncodedInTarget	100
9.2	Dims Class Reference	100
9.2.1	Detailed Description	100
9.3	nvinfer1::Dims2 Class Reference	101
9.3.1	Detailed Description	101
9.3.2	Constructor & Destructor Documentation	101
9.3.2.1	Dims2() [1/2]	101
9.3.2.2	Dims2() [2/2]	101
9.4	nvinfer1::Dims3 Class Reference	102
9.4.1	Detailed Description	102
9.4.2	Constructor & Destructor Documentation	102
9.4.2.1	Dims3() [1/2]	103
9.4.2.2	Dims3() [2/2]	103
9.5	nvinfer1::Dims32 Class Reference	103
9.5.1	Member Data Documentation	104
9.5.1.1	d	104
9.5.1.2	MAX_DIMS	104
9.5.1.3	nbDims	104
9.6	nvinfer1::Dims4 Class Reference	105
9.6.1	Detailed Description	105
9.6.2	Constructor & Destructor Documentation	105
9.6.2.1	Dims4() [1/2]	105
9.6.2.2	Dims4() [2/2]	105
9.7	nvinfer1::DimsExprs Class Reference	106
9.7.1	Detailed Description	106
9.7.2	Member Data Documentation	106
9.7.2.1	d	106

9.7.2.2	nbDims	107
9.8	nvinfer1::DimsHW Class Reference	107
9.8.1	Detailed Description	107
9.8.2	Constructor & Destructor Documentation	108
9.8.2.1	DimsHW() [1/2]	108
9.8.2.2	DimsHW() [2/2]	108
9.8.3	Member Function Documentation	108
9.8.3.1	h() [1/2]	108
9.8.3.2	h() [2/2]	109
9.8.3.3	w() [1/2]	109
9.8.3.4	w() [2/2]	109
9.9	nvinfer1::DynamicPluginTensorDesc Class Reference	109
9.9.1	Detailed Description	110
9.9.2	Member Data Documentation	110
9.9.2.1	desc	110
9.9.2.2	max	110
9.9.2.3	min	110
9.10	nvinfer1::impl::EnumMaxImpl< T > Struct Template Reference	111
9.10.1	Detailed Description	111
9.11	nvinfer1::impl::EnumMaxImpl< ActivationType > Struct Reference	111
9.11.1	Detailed Description	111
9.11.2	Member Data Documentation	111
9.11.2.1	kVALUE	111
9.12	nvinfer1::impl::EnumMaxImpl< AllocatorFlag > Struct Reference	112
9.12.1	Detailed Description	112
9.12.2	Member Data Documentation	112
9.12.2.1	kVALUE	112
9.13	nvinfer1::impl::EnumMaxImpl< DataType > Struct Reference	112
9.13.1	Detailed Description	113
9.13.2	Member Data Documentation	113
9.13.2.1	kVALUE	113
9.14	nvinfer1::impl::EnumMaxImpl< ElementWiseOperation > Struct Reference	113
9.14.1	Detailed Description	113
9.14.2	Member Data Documentation	113
9.14.2.1	kVALUE	114
9.15	nvinfer1::impl::EnumMaxImpl< EngineCapability > Struct Reference	114
9.15.1	Detailed Description	114
9.15.2	Member Data Documentation	114
9.15.2.1	kVALUE	114
9.16	nvinfer1::impl::EnumMaxImpl< ErrorCode > Struct Reference	115
9.16.1	Detailed Description	115
9.16.2	Member Data Documentation	115
9.16.2.1	kVALUE	115
9.17	nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel > Struct Reference	115
9.17.1	Detailed Description	116
9.17.2	Member Data Documentation	116
9.17.2.1	kVALUE	116
9.18	nvinfer1::impl::EnumMaxImpl< ILogger::Severity > Struct Reference	116
9.18.1	Detailed Description	116
9.18.2	Member Data Documentation	117
9.18.2.1	kVALUE	117
9.19	nvinfer1::impl::EnumMaxImpl< InterpolationMode > Struct Reference	117
9.19.1	Detailed Description	117
9.19.2	Member Data Documentation	117

9.19.2.1	kVALUE	117
9.20	nvinfer1::impl::EnumMaxImpl< PaddingMode > Struct Reference	118
9.20.1	Detailed Description	118
9.20.2	Member Data Documentation	118
9.20.2.1	kVALUE	118
9.21	nvinfer1::impl::EnumMaxImpl< PoolingType > Struct Reference	118
9.21.1	Detailed Description	118
9.21.2	Member Data Documentation	119
9.21.2.1	kVALUE	119
9.22	nvinfer1::impl::EnumMaxImpl< PreviewFeature > Struct Reference	119
9.22.1	Detailed Description	119
9.22.2	Member Data Documentation	119
9.22.2.1	kVALUE	119
9.23	nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation > Struct Reference	120
9.23.1	Detailed Description	120
9.23.2	Member Data Documentation	120
9.23.2.1	kVALUE	120
9.24	nvinfer1::impl::EnumMaxImpl< ResizeRoundMode > Struct Reference	120
9.24.1	Detailed Description	121
9.24.2	Member Data Documentation	121
9.24.2.1	kVALUE	121
9.25	nvinfer1::impl::EnumMaxImpl< ResizeSelector > Struct Reference	121
9.25.1	Detailed Description	121
9.25.2	Member Data Documentation	121
9.25.2.1	kVALUE	122
9.26	nvinfer1::impl::EnumMaxImpl< TensorFormat > Struct Reference	122
9.26.1	Detailed Description	122
9.26.2	Member Data Documentation	122
9.26.2.1	kVALUE	122
9.27	nvinfer1::impl::EnumMaxImpl< TensorIOMode > Struct Reference	123
9.27.1	Detailed Description	123
9.27.2	Member Data Documentation	123
9.27.2.1	kVALUE	123
9.28	nvinfer1::impl::EnumMaxImpl< TensorLocation > Struct Reference	123
9.28.1	Detailed Description	124
9.28.2	Member Data Documentation	124
9.28.2.1	kVALUE	124
9.29	nvuffparser::FieldCollection Struct Reference	124
9.29.1	Member Data Documentation	124
9.29.1.1	fields	124
9.29.1.2	nbFields	125
9.30	nvuffparser::FieldMap Class Reference	125
9.30.1	Detailed Description	125
9.30.2	Constructor & Destructor Documentation	125
9.30.2.1	FieldMap() [1/2]	126
9.30.2.2	FieldMap() [2/2]	126
9.30.3	Member Data Documentation	126
9.30.3.1	data	126
9.30.3.2	length	126
9.30.3.3	name	126
9.30.3.4	type	127
9.31	nvinfer1::plugin::GridAnchorParameters Struct Reference	127
9.31.1	Detailed Description	127
9.31.2	Member Data Documentation	128

9.31.2.1	aspectRatios	128
9.31.2.2	H	128
9.31.2.3	maxSize	128
9.31.2.4	minSize	128
9.31.2.5	numAspectRatios	128
9.31.2.6	variance	128
9.31.2.7	W	129
9.32	nvinfer1::IActivationLayer Class Reference	129
9.32.1	Detailed Description	130
9.32.2	Constructor & Destructor Documentation	130
9.32.2.1	~IActivationLayer()	130
9.32.3	Member Function Documentation	130
9.32.3.1	getActivationType()	130
9.32.3.2	getAlpha()	131
9.32.3.3	getBeta()	131
9.32.3.4	setActivationType()	131
9.32.3.5	setAlpha()	132
9.32.3.6	setBeta()	132
9.32.4	Member Data Documentation	132
9.32.4.1	mImpl	132
9.33	nvinfer1::IAlgorithm Class Reference	133
9.33.1	Detailed Description	133
9.33.2	Constructor & Destructor Documentation	134
9.33.2.1	~IAlgorithm()	134
9.33.3	Member Function Documentation	134
9.33.3.1	getAlgorithmIOInfo()	134
9.33.3.2	getAlgorithmIOInfoByIndex()	134
9.33.3.3	getAlgorithmVariant()	135
9.33.3.4	getTimingMSec()	135
9.33.3.5	getWorkspaceSize()	135
9.33.4	Member Data Documentation	135
9.33.4.1	mImpl	135
9.34	nvinfer1::IAlgorithmContext Class Reference	136
9.34.1	Detailed Description	136
9.34.2	Constructor & Destructor Documentation	137
9.34.2.1	~IAlgorithmContext()	137
9.34.3	Member Function Documentation	137
9.34.3.1	getDimensions()	137
9.34.3.2	getName()	137
9.34.3.3	getNbInputs()	137
9.34.3.4	getNbOutputs()	138
9.34.4	Member Data Documentation	138
9.34.4.1	mImpl	138
9.35	nvinfer1::IAlgorithmIOInfo Class Reference	138
9.35.1	Detailed Description	139
9.35.2	Constructor & Destructor Documentation	139
9.35.2.1	~IAlgorithmIOInfo()	139
9.35.3	Member Function Documentation	139
9.35.3.1	getComponentsPerElement()	139
9.35.3.2	getDataType()	140
9.35.3.3	getStrides()	140
9.35.3.4	getTensorFormat()	140
9.35.3.5	getVectorizedDim()	140
9.35.4	Member Data Documentation	141

9.35.4.1	mImpl	141
9.36	nvinfer1::IAlgorithmSelector Class Reference	141
9.36.1	Detailed Description	141
9.36.2	Constructor & Destructor Documentation	141
9.36.2.1	~IAlgorithmSelector()	142
9.36.3	Member Function Documentation	142
9.36.3.1	reportAlgorithms()	142
9.36.3.2	selectAlgorithms()	142
9.37	nvinfer1::IAlgorithmVariant Class Reference	143
9.37.1	Detailed Description	144
9.37.2	Constructor & Destructor Documentation	144
9.37.2.1	~IAlgorithmVariant()	144
9.37.3	Member Function Documentation	144
9.37.3.1	getImplementation()	144
9.37.3.2	getTactic()	145
9.37.4	Member Data Documentation	145
9.37.4.1	mImpl	145
9.38	nvinfer1::IAssertionLayer Class Reference	145
9.38.1	Detailed Description	146
9.38.2	Constructor & Destructor Documentation	146
9.38.2.1	~IAssertionLayer()	146
9.38.3	Member Function Documentation	146
9.38.3.1	getMessage()	146
9.38.3.2	setMessage()	147
9.38.4	Member Data Documentation	147
9.38.4.1	mImpl	147
9.39	nvcaffeparser1::IBinaryProtoBlob Class Reference	147
9.39.1	Detailed Description	147
9.39.2	Constructor & Destructor Documentation	148
9.39.2.1	~IBinaryProtoBlob()	148
9.39.3	Member Function Documentation	148
9.39.3.1	destroy()	148
9.39.3.2	getData()	148
9.39.3.3	getDataType()	148
9.39.3.4	getDimensions()	149
9.40	nvcaffeparser1::IBlobNameToTensor Class Reference	149
9.40.1	Detailed Description	149
9.40.2	Constructor & Destructor Documentation	149
9.40.2.1	~IBlobNameToTensor()	150
9.40.3	Member Function Documentation	150
9.40.3.1	find()	150
9.41	nvinfer1::IBuilder Class Reference	150
9.41.1	Detailed Description	152
9.41.2	Constructor & Destructor Documentation	152
9.41.2.1	~IBuilder()	152
9.41.3	Member Function Documentation	152
9.41.3.1	buildEngineWithConfig()	152
9.41.3.2	buildSerializedNetwork()	152
9.41.3.3	createBuilderConfig()	153
9.41.3.4	createNetworkV2()	153
9.41.3.5	createOptimizationProfile()	154
9.41.3.6	destroy()	154
9.41.3.7	getErrorRecorder()	155
9.41.3.8	getLogger()	155

9.41.3.9	getMaxBatchSize()	155
9.41.3.10	getMaxDLABatchSize()	155
9.41.3.11	getMaxThreads()	156
9.41.3.12	getNbDLACores()	156
9.41.3.13	getPluginRegistry()	156
9.41.3.14	isNetworkSupported()	156
9.41.3.15	platformHasFastFp16()	157
9.41.3.16	platformHasFastInt8()	157
9.41.3.17	platformHasTf32()	157
9.41.3.18	reset()	158
9.41.3.19	setErrorRecorder()	158
9.41.3.20	setGpuAllocator()	158
9.41.3.21	setMaxBatchSize()	159
9.41.3.22	setMaxThreads()	159
9.41.4	Member Data Documentation	160
9.41.4.1	mImpl	160
9.42	nvinfer1::IBuilderConfig Class Reference	160
9.42.1	Detailed Description	163
9.42.2	Constructor & Destructor Documentation	163
9.42.2.1	~IBuilderConfig()	163
9.42.3	Member Function Documentation	163
9.42.3.1	addOptimizationProfile()	163
9.42.3.2	canRunOnDLA()	164
9.42.3.3	clearFlag()	164
9.42.3.4	clearQuantizationFlag()	164
9.42.3.5	createTimingCache()	165
9.42.3.6	destroy()	165
9.42.3.7	getAlgorithmSelector()	166
9.42.3.8	getAvgTimingIterations()	166
9.42.3.9	getBuilderOptimizationLevel()	166
9.42.3.10	getCalibrationProfile()	166
9.42.3.11	getDefaultDeviceType()	167
9.42.3.12	getDeviceType()	167
9.42.3.13	getDLACore()	167
9.42.3.14	getEngineCapability()	167
9.42.3.15	getFlag()	168
9.42.3.16	getFlags()	168
9.42.3.17	getHardwareCompatibilityLevel()	168
9.42.3.18	getInt8Calibrator()	169
9.42.3.19	getMaxAuxStreams()	169
9.42.3.20	getMaxWorkspaceSize()	169
9.42.3.21	getMemoryPoolLimit()	169
9.42.3.22	getMinTimingIterations()	170
9.42.3.23	getNbOptimizationProfiles()	170
9.42.3.24	getNbPluginsToSerialize()	171
9.42.3.25	getPluginToSerialize()	171
9.42.3.26	getPreviewFeature()	171
9.42.3.27	getProfileStream()	172
9.42.3.28	getProfilingVerbosity()	172
9.42.3.29	getQuantizationFlag()	172
9.42.3.30	getQuantizationFlags()	173
9.42.3.31	getTacticSources()	173
9.42.3.32	getTimingCache()	173
9.42.3.33	isDeviceTypeSet()	174

9.42.3.34	reset()	174
9.42.3.35	resetDeviceType()	174
9.42.3.36	setAlgorithmSelector()	174
9.42.3.37	setAvgTimingIterations()	175
9.42.3.38	setBuilderOptimizationLevel()	175
9.42.3.39	setCalibrationProfile()	176
9.42.3.40	setDefaultDeviceType()	176
9.42.3.41	setDeviceType()	176
9.42.3.42	setDLACore()	177
9.42.3.43	setEngineCapability()	177
9.42.3.44	setFlag()	178
9.42.3.45	setFlags()	178
9.42.3.46	setHardwareCompatibilityLevel()	178
9.42.3.47	setInt8Calibrator()	179
9.42.3.48	setMaxAuxStreams()	179
9.42.3.49	setMaxWorkspaceSize()	180
9.42.3.50	setMemoryPoolLimit()	180
9.42.3.51	setMinTimingIterations()	181
9.42.3.52	setPluginsToSerialize()	181
9.42.3.53	setPreviewFeature()	182
9.42.3.54	setProfileStream()	182
9.42.3.55	setProfilingVerbosity()	182
9.42.3.56	setQuantizationFlag()	183
9.42.3.57	setQuantizationFlags()	183
9.42.3.58	setTacticSources()	184
9.42.3.59	setTimingCache()	184
9.42.4	Member Data Documentation	185
9.42.4.1	mImpl	185
9.43	nvcaffeparser1::ICaffeParser Class Reference	185
9.43.1	Detailed Description	186
9.43.2	Constructor & Destructor Documentation	186
9.43.2.1	~ICaffeParser()	186
9.43.3	Member Function Documentation	186
9.43.3.1	destroy()	186
9.43.3.2	getErrorRecorder()	187
9.43.3.3	parse()	187
9.43.3.4	parseBinaryProto()	188
9.43.3.5	parseBuffers()	188
9.43.3.6	setErrorRecorder()	189
9.43.3.7	setPluginFactoryV2()	189
9.43.3.8	setPluginNamespace()	190
9.43.3.9	setProtobufBufferSize()	190
9.44	nvinfer1::ICastLayer Class Reference	190
9.44.1	Detailed Description	191
9.44.2	Constructor & Destructor Documentation	191
9.44.2.1	~ICastLayer()	191
9.44.3	Member Function Documentation	191
9.44.3.1	getToType()	191
9.44.3.2	setToType()	192
9.44.4	Member Data Documentation	192
9.44.4.1	mImpl	192
9.45	nvinfer1::IConcatenationLayer Class Reference	192
9.45.1	Detailed Description	193
9.45.2	Constructor & Destructor Documentation	193

9.45.2.1	<code>~IConcatenationLayer()</code>	193
9.45.3	Member Function Documentation	193
9.45.3.1	<code>getAxis()</code>	193
9.45.3.2	<code>setAxis()</code>	194
9.45.4	Member Data Documentation	194
9.45.4.1	<code>mImpl</code>	194
9.46	<code>nvinfer1::IConditionLayer</code> Class Reference	194
9.46.1	Detailed Description	195
9.46.2	Constructor & Destructor Documentation	195
9.46.2.1	<code>~IConditionLayer()</code>	195
9.46.3	Member Data Documentation	195
9.46.3.1	<code>mImpl</code>	195
9.47	<code>nvinfer1::consistency::IConsistencyChecker</code> Class Reference	195
9.47.1	Detailed Description	196
9.47.2	Constructor & Destructor Documentation	196
9.47.2.1	<code>~IConsistencyChecker()</code>	196
9.47.2.2	<code>IConsistencyChecker()</code> [1/3]	196
9.47.2.3	<code>IConsistencyChecker()</code> [2/3]	197
9.47.2.4	<code>IConsistencyChecker()</code> [3/3]	197
9.47.3	Member Function Documentation	197
9.47.3.1	<code>operator=()</code> [1/2]	197
9.47.3.2	<code>operator=()</code> [2/2]	197
9.47.3.3	<code>validate()</code>	197
9.47.4	Member Data Documentation	197
9.47.4.1	<code>mImpl</code>	198
9.48	<code>nvinfer1::IConstantLayer</code> Class Reference	198
9.48.1	Detailed Description	198
9.48.2	Constructor & Destructor Documentation	199
9.48.2.1	<code>~IConstantLayer()</code>	199
9.48.3	Member Function Documentation	199
9.48.3.1	<code>getDimensions()</code>	199
9.48.3.2	<code>getWeights()</code>	199
9.48.3.3	<code>setDimensions()</code>	199
9.48.3.4	<code>setWeights()</code>	200
9.48.4	Member Data Documentation	200
9.48.4.1	<code>mImpl</code>	200
9.49	<code>nvinfer1::IConvolutionLayer</code> Class Reference	200
9.49.1	Detailed Description	202
9.49.2	Constructor & Destructor Documentation	202
9.49.2.1	<code>~IConvolutionLayer()</code>	203
9.49.3	Member Function Documentation	203
9.49.3.1	<code>getBiasWeights()</code>	203
9.49.3.2	<code>getDilation()</code>	203
9.49.3.3	<code>getDilationNd()</code>	203
9.49.3.4	<code>getKernelSize()</code>	204
9.49.3.5	<code>getKernelSizeNd()</code>	204
9.49.3.6	<code>getKernelWeights()</code>	204
9.49.3.7	<code>getNbGroups()</code>	204
9.49.3.8	<code>getNbOutputMaps()</code>	205
9.49.3.9	<code>getPadding()</code>	205
9.49.3.10	<code>getPaddingMode()</code>	205
9.49.3.11	<code>getPaddingNd()</code>	206
9.49.3.12	<code>getPostPadding()</code>	206
9.49.3.13	<code>getPrePadding()</code>	206

9.49.3.14	getStride()	206
9.49.3.15	getStrideNd()	207
9.49.3.16	setBiasWeights()	207
9.49.3.17	setDilation()	207
9.49.3.18	setDilationNd()	208
9.49.3.19	setInput()	208
9.49.3.20	setKernelSize()	208
9.49.3.21	setKernelSizeNd()	209
9.49.3.22	setKernelWeights()	209
9.49.3.23	setNbGroups()	210
9.49.3.24	setNbOutputMaps()	210
9.49.3.25	setPadding()	211
9.49.3.26	setPaddingMode()	211
9.49.3.27	setPaddingNd()	211
9.49.3.28	setPostPadding()	212
9.49.3.29	setPrePadding()	212
9.49.3.30	setStride()	212
9.49.3.31	setStrideNd()	213
9.49.4	Member Data Documentation	213
9.49.4.1	mImpl	213
9.50	nvinfer1::ICudaEngine Class Reference	213
9.50.1	Detailed Description	216
9.50.2	Constructor & Destructor Documentation	216
9.50.2.1	~ICudaEngine()	217
9.50.3	Member Function Documentation	217
9.50.3.1	bindingIsInput()	217
9.50.3.2	createEngineInspector()	217
9.50.3.3	createExecutionContext()	218
9.50.3.4	createExecutionContextWithoutDeviceMemory()	218
9.50.3.5	destroy()	218
9.50.3.6	getBindingBytesPerComponent()	218
9.50.3.7	getBindingComponentsPerElement()	219
9.50.3.8	getBindingDataType()	219
9.50.3.9	getBindingDimensions()	220
9.50.3.10	getBindingFormat()	221
9.50.3.11	getBindingFormatDesc()	221
9.50.3.12	getBindingIndex()	222
9.50.3.13	getBindingName()	222
9.50.3.14	getBindingVectorizedDim()	223
9.50.3.15	getDeviceMemorySize()	223
9.50.3.16	getEngineCapability()	224
9.50.3.17	getErrorRecorder()	224
9.50.3.18	getHardwareCompatibilityLevel()	224
9.50.3.19	getIOTensorName()	224
9.50.3.20	getLocation()	225
9.50.3.21	getMaxBatchSize()	225
9.50.3.22	getName()	226
9.50.3.23	getNbAuxStreams()	226
9.50.3.24	getNbBindings()	227
9.50.3.25	getNbIOTensors()	227
9.50.3.26	getNbLayers()	227
9.50.3.27	getNbOptimizationProfiles()	228
9.50.3.28	getProfileDimensions()	228
9.50.3.29	getProfileShape()	229

9.50.3.30	getProfileShapeValues()	229
9.50.3.31	getProfilingVerbosity()	230
9.50.3.32	getTacticSources()	230
9.50.3.33	getTensorBytesPerComponent() [1/2]	231
9.50.3.34	getTensorBytesPerComponent() [2/2]	232
9.50.3.35	getTensorComponentsPerElement() [1/2]	233
9.50.3.36	getTensorComponentsPerElement() [2/2]	233
9.50.3.37	getTensorDataType()	234
9.50.3.38	getTensorFormat() [1/2]	234
9.50.3.39	getTensorFormat() [2/2]	235
9.50.3.40	getTensorFormatDesc() [1/2]	235
9.50.3.41	getTensorFormatDesc() [2/2]	235
9.50.3.42	getTensorIOMode()	236
9.50.3.43	getTensorLocation()	236
9.50.3.44	getTensorShape()	237
9.50.3.45	getTensorVectorizedDim() [1/2]	237
9.50.3.46	getTensorVectorizedDim() [2/2]	238
9.50.3.47	hasImplicitBatchDimension()	238
9.50.3.48	isExecutionBinding()	239
9.50.3.49	isRefittable()	239
9.50.3.50	isShapeBinding()	240
9.50.3.51	isShapeInferenceIO()	240
9.50.3.52	serialize()	241
9.50.3.53	setErrorRecorder()	241
9.50.4	Member Data Documentation	241
9.50.4.1	mImpl	242
9.51	nvinfer1::safe::ICudaEngine Class Reference	242
9.51.1	Detailed Description	243
9.51.2	Constructor & Destructor Documentation	243
9.51.2.1	ICudaEngine() [1/3]	244
9.51.2.2	~ICudaEngine()	244
9.51.2.3	ICudaEngine() [2/3]	244
9.51.2.4	ICudaEngine() [3/3]	244
9.51.3	Member Function Documentation	244
9.51.3.1	bindingIsInput()	244
9.51.3.2	createExecutionContext()	245
9.51.3.3	createExecutionContextWithoutDeviceMemory()	246
9.51.3.4	getBindingBytesPerComponent()	246
9.51.3.5	getBindingComponentsPerElement()	247
9.51.3.6	getBindingDataType()	248
9.51.3.7	getBindingDimensions()	248
9.51.3.8	getBindingFormat()	249
9.51.3.9	getBindingIndex()	250
9.51.3.10	getBindingName()	250
9.51.3.11	getBindingVectorizedDim()	251
9.51.3.12	getDeviceMemorySize()	252
9.51.3.13	getErrorRecorder()	252
9.51.3.14	getIOTensorName()	253
9.51.3.15	getName()	253
9.51.3.16	getNbBindings()	254
9.51.3.17	getNbIOTensors()	254
9.51.3.18	getTensorBytesPerComponent()	254
9.51.3.19	getTensorComponentsPerElement()	255
9.51.3.20	getTensorDataType()	256

9.51.3.21	getTensorFormat()	257
9.51.3.22	getTensorIOMode()	258
9.51.3.23	getTensorShape()	258
9.51.3.24	getTensorVectorizedDim()	259
9.51.3.25	operator=() [1/2]	260
9.51.3.26	operator=() [2/2]	260
9.51.3.27	setErrorRecorder()	260
9.52	nvinfer1::IDeconvolutionLayer Class Reference	261
9.52.1	Detailed Description	263
9.52.2	Constructor & Destructor Documentation	263
9.52.2.1	~IDeconvolutionLayer()	263
9.52.3	Member Function Documentation	263
9.52.3.1	getBiasWeights()	263
9.52.3.2	getDilationNd()	263
9.52.3.3	getKernelSize()	264
9.52.3.4	getKernelSizeNd()	264
9.52.3.5	getKernelWeights()	264
9.52.3.6	getNbGroups()	264
9.52.3.7	getNbOutputMaps()	265
9.52.3.8	getPadding()	265
9.52.3.9	getPaddingMode()	265
9.52.3.10	getPaddingNd()	266
9.52.3.11	getPostPadding()	266
9.52.3.12	getPrePadding()	266
9.52.3.13	getStride()	266
9.52.3.14	getStrideNd()	267
9.52.3.15	setBiasWeights()	267
9.52.3.16	setDilationNd()	267
9.52.3.17	setInput()	267
9.52.3.18	setKernelSize()	268
9.52.3.19	setKernelSizeNd()	268
9.52.3.20	setKernelWeights()	269
9.52.3.21	setNbGroups()	269
9.52.3.22	setNbOutputMaps()	269
9.52.3.23	setPadding()	270
9.52.3.24	setPaddingMode()	270
9.52.3.25	setPaddingNd()	270
9.52.3.26	setPostPadding()	271
9.52.3.27	setPrePadding()	271
9.52.3.28	setStride()	271
9.52.3.29	setStrideNd()	272
9.52.4	Member Data Documentation	272
9.52.4.1	mImpl	272
9.53	nvinfer1::IDequantizeLayer Class Reference	272
9.53.1	Detailed Description	273
9.53.2	Constructor & Destructor Documentation	274
9.53.2.1	~IDequantizeLayer()	274
9.53.3	Member Function Documentation	274
9.53.3.1	getAxis()	274
9.53.3.2	setAxis()	274
9.53.4	Member Data Documentation	274
9.53.4.1	mImpl	275
9.54	nvinfer1::IDimensionExpr Class Reference	275
9.54.1	Detailed Description	275

9.54.2	Constructor & Destructor Documentation	276
9.54.2.1	~IDimensionExpr()	276
9.54.3	Member Function Documentation	276
9.54.3.1	getConstantValue()	276
9.54.3.2	isConstant()	276
9.54.4	Member Data Documentation	276
9.54.4.1	mImpl	276
9.55	nvinfer1::IEinsumLayer Class Reference	277
9.55.1	Detailed Description	277
9.55.2	Constructor & Destructor Documentation	278
9.55.2.1	~IEinsumLayer()	278
9.55.3	Member Function Documentation	278
9.55.3.1	getEquation()	278
9.55.3.2	setEquation()	278
9.55.4	Member Data Documentation	279
9.55.4.1	mImpl	279
9.56	nvinfer1::IElementWiseLayer Class Reference	279
9.56.1	Detailed Description	280
9.56.2	Constructor & Destructor Documentation	280
9.56.2.1	~IElementWiseLayer()	280
9.56.3	Member Function Documentation	280
9.56.3.1	getOperation()	280
9.56.3.2	setOperation()	281
9.56.4	Member Data Documentation	281
9.56.4.1	mImpl	281
9.57	nvinfer1::IEngineInspector Class Reference	281
9.57.1	Detailed Description	282
9.57.2	Constructor & Destructor Documentation	282
9.57.2.1	~IEngineInspector()	283
9.57.3	Member Function Documentation	283
9.57.3.1	getEngineInformation()	283
9.57.3.2	getErrorRecorder()	283
9.57.3.3	getExecutionContext()	284
9.57.3.4	getLayerInformation()	284
9.57.3.5	setErrorRecorder()	285
9.57.3.6	setExecutionContext()	285
9.57.4	Member Data Documentation	286
9.57.4.1	mImpl	286
9.58	nvinfer1::IErrorRecorder Class Reference	286
9.58.1	Detailed Description	287
9.58.2	Member Typedef Documentation	287
9.58.2.1	ErrorDesc	287
9.58.2.2	RefCount	287
9.58.3	Constructor & Destructor Documentation	287
9.58.3.1	IErrorRecorder()	288
9.58.3.2	~IErrorRecorder()	288
9.58.4	Member Function Documentation	288
9.58.4.1	clear()	288
9.58.4.2	decRefCount()	289
9.58.4.3	getErrorCode()	289
9.58.4.4	getErrorDesc()	290
9.58.4.5	getNbErrors()	291
9.58.4.6	hasOverflowed()	291
9.58.4.7	incRefCount()	292

9.58.4.8	reportError()	292
9.58.5	Member Data Documentation	293
9.58.5.1	kMAX_DESC_LENGTH	293
9.59	nvinfer1::IExecutionContext Class Reference	293
9.59.1	Detailed Description	296
9.59.2	Constructor & Destructor Documentation	296
9.59.2.1	~IExecutionContext()	296
9.59.3	Member Function Documentation	296
9.59.3.1	allInputDimensionsSpecified()	297
9.59.3.2	allInputShapesSpecified()	297
9.59.3.3	destroy()	297
9.59.3.4	enqueue()	298
9.59.3.5	enqueueV2()	299
9.59.3.6	enqueueV3()	299
9.59.3.7	execute()	300
9.59.3.8	executeV2()	301
9.59.3.9	getBindingDimensions()	301
9.59.3.10	getDebugSync()	302
9.59.3.11	getEngine()	302
9.59.3.12	getEnqueueEmitsProfile()	303
9.59.3.13	getErrorRecorder()	303
9.59.3.14	getInputConsumedEvent()	303
9.59.3.15	getMaxOutputSize()	303
9.59.3.16	getName()	304
9.59.3.17	getNvtxVerbosity()	304
9.59.3.18	getOptimizationProfile()	305
9.59.3.19	getOutputAllocator()	305
9.59.3.20	getOutputTensorAddress()	305
9.59.3.21	getPersistentCacheLimit()	306
9.59.3.22	getProfiler()	306
9.59.3.23	getShapeBinding()	306
9.59.3.24	getStrides()	307
9.59.3.25	getTemporaryStorageAllocator()	308
9.59.3.26	getTensorAddress()	308
9.59.3.27	getTensorShape()	308
9.59.3.28	getTensorStrides()	309
9.59.3.29	inferShapes()	310
9.59.3.30	reportToProfiler()	310
9.59.3.31	setAuxStreams()	311
9.59.3.32	setBindingDimensions()	312
9.59.3.33	setDebugSync()	313
9.59.3.34	setDeviceMemory()	313
9.59.3.35	setEnqueueEmitsProfile()	314
9.59.3.36	setErrorRecorder()	314
9.59.3.37	setInputConsumedEvent()	314
9.59.3.38	setInputShape()	315
9.59.3.39	setInputShapeBinding()	315
9.59.3.40	setInputTensorAddress()	316
9.59.3.41	setName()	317
9.59.3.42	setNvtxVerbosity()	317
9.59.3.43	setOptimizationProfile()	318
9.59.3.44	setOptimizationProfileAsync()	319
9.59.3.45	setOutputAllocator()	320
9.59.3.46	setPersistentCacheLimit()	321

9.59.3.47	setProfiler()	321
9.59.3.48	setTemporaryStorageAllocator()	321
9.59.3.49	setTensorAddress()	322
9.59.4	Member Data Documentation	323
9.59.4.1	mImpl	323
9.60	nvinfer1::safe::IExecutionContext Class Reference	323
9.60.1	Detailed Description	324
9.60.2	Constructor & Destructor Documentation	324
9.60.2.1	IExecutionContext() [1/3]	325
9.60.2.2	~IExecutionContext()	325
9.60.2.3	IExecutionContext() [2/3]	325
9.60.2.4	IExecutionContext() [3/3]	325
9.60.3	Member Function Documentation	325
9.60.3.1	enqueueV2()	325
9.60.3.2	enqueueV3()	326
9.60.3.3	getEngine()	327
9.60.3.4	getErrorBuffer()	328
9.60.3.5	getErrorRecorder()	328
9.60.3.6	getInputConsumedEvent()	329
9.60.3.7	getInputTensorAddress()	329
9.60.3.8	getName()	330
9.60.3.9	getOutputTensorAddress()	330
9.60.3.10	getStrides()	331
9.60.3.11	getTensorStrides()	332
9.60.3.12	operator=() [1/2]	332
9.60.3.13	operator=() [2/2]	333
9.60.3.14	setDeviceMemory()	333
9.60.3.15	setErrorBuffer()	334
9.60.3.16	setErrorRecorder()	334
9.60.3.17	setInputConsumedEvent()	335
9.60.3.18	setInputTensorAddress()	336
9.60.3.19	setName()	336
9.60.3.20	setOutputTensorAddress()	337
9.61	nvinfer1::IExprBuilder Class Reference	338
9.61.1	Detailed Description	339
9.61.2	Constructor & Destructor Documentation	339
9.61.2.1	~IExprBuilder()	339
9.61.3	Member Function Documentation	339
9.61.3.1	constant()	339
9.61.3.2	operation()	339
9.61.4	Member Data Documentation	340
9.61.4.1	mImpl	340
9.62	nvinfer1::IFillLayer Class Reference	340
9.62.1	Detailed Description	341
9.62.2	Constructor & Destructor Documentation	341
9.62.2.1	~IFillLayer()	342
9.62.3	Member Function Documentation	342
9.62.3.1	getAlpha()	342
9.62.3.2	getBeta()	342
9.62.3.3	getDimensions()	343
9.62.3.4	getOperation()	343
9.62.3.5	setAlpha()	343
9.62.3.6	setBeta()	344
9.62.3.7	setDimensions()	344

9.62.3.8	setInput()	345
9.62.3.9	setOperation()	346
9.62.4	Member Data Documentation	346
9.62.4.1	mImpl	346
9.63	nvinfer1::IFullyConnectedLayer Class Reference	346
9.63.1	Detailed Description	347
9.63.2	Constructor & Destructor Documentation	348
9.63.2.1	~IFullyConnectedLayer()	348
9.63.3	Member Function Documentation	348
9.63.3.1	getBiasWeights()	348
9.63.3.2	getKernelWeights()	348
9.63.3.3	getNbOutputChannels()	349
9.63.3.4	setBiasWeights()	349
9.63.3.5	setInput()	349
9.63.3.6	setKernelWeights()	350
9.63.3.7	setNbOutputChannels()	350
9.63.4	Member Data Documentation	350
9.63.4.1	mImpl	351
9.64	nvinfer1::IGatherLayer Class Reference	351
9.64.1	Detailed Description	352
9.64.2	Constructor & Destructor Documentation	353
9.64.2.1	~IGatherLayer()	353
9.64.3	Member Function Documentation	354
9.64.3.1	getGatherAxis()	354
9.64.3.2	getMode()	354
9.64.3.3	getNbElementWiseDims()	354
9.64.3.4	setGatherAxis()	354
9.64.3.5	setMode()	355
9.64.3.6	setNbElementWiseDims()	355
9.64.4	Member Data Documentation	356
9.64.4.1	mImpl	356
9.65	nvinfer1::IGpuAllocator Class Reference	356
9.65.1	Detailed Description	356
9.65.2	Constructor & Destructor Documentation	356
9.65.2.1	~IGpuAllocator()	357
9.65.2.2	IGpuAllocator()	357
9.65.3	Member Function Documentation	357
9.65.3.1	allocate()	357
9.65.3.2	deallocate()	358
9.65.3.3	free()	358
9.65.3.4	realloc()	359
9.66	nvinfer1::IGridSampleLayer Class Reference	360
9.66.1	Detailed Description	361
9.66.2	Constructor & Destructor Documentation	361
9.66.2.1	~IGridSampleLayer()	361
9.66.3	Member Function Documentation	362
9.66.3.1	getAlignCorners()	362
9.66.3.2	getInterpolationMode()	362
9.66.3.3	getSampleMode()	362
9.66.3.4	setAlignCorners()	363
9.66.3.5	setInterpolationMode()	363
9.66.3.6	setSampleMode()	363
9.66.4	Member Data Documentation	363
9.66.4.1	mImpl	364

9.67	nvinfer1::IHostMemory Class Reference	364
9.67.1	Detailed Description	364
9.67.2	Constructor & Destructor Documentation	365
9.67.2.1	~IHostMemory()	365
9.67.3	Member Function Documentation	365
9.67.3.1	data()	365
9.67.3.2	destroy()	365
9.67.3.3	size()	365
9.67.3.4	type()	366
9.67.4	Member Data Documentation	366
9.67.4.1	mImpl	366
9.68	nvinfer1::IIdentityLayer Class Reference	366
9.68.1	Detailed Description	367
9.68.2	Constructor & Destructor Documentation	367
9.68.2.1	~IIdentityLayer()	367
9.68.3	Member Data Documentation	367
9.68.3.1	mImpl	367
9.69	nvinfer1::IIfConditional Class Reference	368
9.69.1	Detailed Description	368
9.69.2	Constructor & Destructor Documentation	369
9.69.2.1	~IIfConditional()	369
9.69.3	Member Function Documentation	369
9.69.3.1	addInput()	369
9.69.3.2	addOutput()	369
9.69.3.3	getName()	370
9.69.3.4	setCondition()	370
9.69.3.5	setName()	370
9.69.4	Member Data Documentation	371
9.69.4.1	mImpl	371
9.70	nvinfer1::IIfConditionalBoundaryLayer Class Reference	371
9.70.1	Detailed Description	372
9.70.2	Constructor & Destructor Documentation	372
9.70.2.1	~IIfConditionalBoundaryLayer()	372
9.70.3	Member Function Documentation	372
9.70.3.1	getConditional()	372
9.70.4	Member Data Documentation	372
9.70.4.1	mBoundary	372
9.71	nvinfer1::IIfConditionalInputLayer Class Reference	373
9.71.1	Detailed Description	373
9.71.2	Constructor & Destructor Documentation	373
9.71.2.1	~IIfConditionalInputLayer()	373
9.71.3	Member Data Documentation	373
9.71.3.1	mImpl	374
9.72	nvinfer1::IIfConditionalOutputLayer Class Reference	374
9.72.1	Detailed Description	374
9.72.2	Constructor & Destructor Documentation	374
9.72.2.1	~IIfConditionalOutputLayer()	375
9.72.3	Member Data Documentation	375
9.72.3.1	mImpl	375
9.73	nvinfer1::IInt8Calibrator Class Reference	375
9.73.1	Detailed Description	376
9.73.2	Constructor & Destructor Documentation	376
9.73.2.1	~IInt8Calibrator()	376
9.73.3	Member Function Documentation	376

9.73.3.1	getAlgorithm()	376
9.73.3.2	getBatch()	376
9.73.3.3	getBatchSize()	377
9.73.3.4	readCalibrationCache()	377
9.73.3.5	writeCalibrationCache()	378
9.74	nvinfer1::IInt8EntropyCalibrator Class Reference	378
9.74.1	Detailed Description	379
9.74.2	Constructor & Destructor Documentation	379
9.74.2.1	~IInt8EntropyCalibrator()	379
9.74.3	Member Function Documentation	379
9.74.3.1	getAlgorithm()	379
9.75	nvinfer1::IInt8EntropyCalibrator2 Class Reference	379
9.75.1	Detailed Description	380
9.75.2	Constructor & Destructor Documentation	380
9.75.2.1	~IInt8EntropyCalibrator2()	380
9.75.3	Member Function Documentation	380
9.75.3.1	getAlgorithm()	380
9.76	nvinfer1::IInt8LegacyCalibrator Class Reference	380
9.76.1	Detailed Description	381
9.76.2	Constructor & Destructor Documentation	381
9.76.2.1	~IInt8LegacyCalibrator()	381
9.76.3	Member Function Documentation	381
9.76.3.1	getAlgorithm()	381
9.76.3.2	getQuantile()	382
9.76.3.3	getRegressionCutoff()	382
9.76.3.4	readHistogramCache()	382
9.76.3.5	writeHistogramCache()	382
9.77	nvinfer1::IInt8MinMaxCalibrator Class Reference	383
9.77.1	Detailed Description	383
9.77.2	Constructor & Destructor Documentation	383
9.77.2.1	~IInt8MinMaxCalibrator()	383
9.77.3	Member Function Documentation	384
9.77.3.1	getAlgorithm()	384
9.78	nvinfer1::IIteratorLayer Class Reference	384
9.78.1	Constructor & Destructor Documentation	385
9.78.1.1	~IIteratorLayer()	385
9.78.2	Member Function Documentation	385
9.78.2.1	getAxis()	385
9.78.2.2	getReverse()	385
9.78.2.3	setAxis()	385
9.78.2.4	setReverse()	386
9.78.3	Member Data Documentation	386
9.78.3.1	mImpl	386
9.79	nvinfer1::ILayer Class Reference	386
9.79.1	Detailed Description	388
9.79.2	Constructor & Destructor Documentation	388
9.79.2.1	~ILayer()	388
9.79.3	Member Function Documentation	389
9.79.3.1	getInput()	389
9.79.3.2	getMetadata()	389
9.79.3.3	getName()	389
9.79.3.4	getNbInputs()	390
9.79.3.5	getNbOutputs()	390
9.79.3.6	getOutput()	390

9.79.3.7	getOutputType()	390
9.79.3.8	getPrecision()	391
9.79.3.9	getType()	391
9.79.3.10	outputTypesIsSet()	391
9.79.3.11	precisionIsSet()	392
9.79.3.12	resetOutputType()	392
9.79.3.13	resetPrecision()	393
9.79.3.14	setInput()	393
9.79.3.15	setMetadata()	393
9.79.3.16	setName()	394
9.79.3.17	setOutputType()	394
9.79.3.18	setPrecision()	395
9.79.4	Member Data Documentation	396
9.79.4.1	mLayer	396
9.80	nvinfer1::ILogger Class Reference	396
9.80.1	Detailed Description	397
9.80.2	Member Enumeration Documentation	397
9.80.2.1	Severity	397
9.80.3	Constructor & Destructor Documentation	397
9.80.3.1	ILogger()	397
9.80.3.2	~ILogger()	397
9.80.4	Member Function Documentation	398
9.80.4.1	log()	398
9.81	nvinfer1::ILoggerFinder Class Reference	398
9.81.1	Detailed Description	399
9.81.2	Constructor & Destructor Documentation	399
9.81.2.1	~ILoggerFinder()	399
9.81.3	Member Function Documentation	399
9.81.3.1	findLogger()	399
9.82	nvinfer1::ILoop Class Reference	400
9.82.1	Detailed Description	400
9.82.2	Constructor & Destructor Documentation	400
9.82.2.1	~ILoop()	401
9.82.3	Member Function Documentation	401
9.82.3.1	addIterator()	401
9.82.3.2	addLoopOutput()	401
9.82.3.3	addRecurrence()	401
9.82.3.4	addTripLimit()	402
9.82.3.5	getName()	402
9.82.3.6	setName()	402
9.82.4	Member Data Documentation	403
9.82.4.1	mImpl	403
9.83	nvinfer1::ILoopBoundaryLayer Class Reference	403
9.83.1	Constructor & Destructor Documentation	403
9.83.1.1	~ILoopBoundaryLayer()	404
9.83.2	Member Function Documentation	404
9.83.2.1	getLoop()	404
9.83.3	Member Data Documentation	404
9.83.3.1	mBoundary	404
9.84	nvinfer1::ILoopOutputLayer Class Reference	404
9.84.1	Detailed Description	405
9.84.2	Constructor & Destructor Documentation	405
9.84.2.1	~ILoopOutputLayer()	405
9.84.3	Member Function Documentation	406

9.84.3.1	getAxis()	406
9.84.3.2	getLoopOutput()	406
9.84.3.3	setAxis()	406
9.84.3.4	setInput()	406
9.84.4	Member Data Documentation	407
9.84.4.1	mImpl	407
9.85	nvinfer1::ILRNLayer Class Reference	407
9.85.1	Detailed Description	408
9.85.2	Constructor & Destructor Documentation	408
9.85.2.1	~ILRNLayer()	408
9.85.3	Member Function Documentation	409
9.85.3.1	getAlpha()	409
9.85.3.2	getBeta()	409
9.85.3.3	getK()	409
9.85.3.4	getWindowSize()	410
9.85.3.5	setAlpha()	410
9.85.3.6	setBeta()	410
9.85.3.7	setK()	411
9.85.3.8	setWindowSize()	411
9.85.4	Member Data Documentation	411
9.85.4.1	mImpl	411
9.86	nvinfer1::IMatrixMultiplyLayer Class Reference	412
9.86.1	Detailed Description	412
9.86.2	Constructor & Destructor Documentation	413
9.86.2.1	~IMatrixMultiplyLayer()	413
9.86.3	Member Function Documentation	413
9.86.3.1	getOperation()	413
9.86.3.2	setOperation()	413
9.86.4	Member Data Documentation	414
9.86.4.1	mImpl	414
9.87	nvinfer1::INetworkDefinition Class Reference	414
9.87.1	Detailed Description	418
9.87.2	Constructor & Destructor Documentation	418
9.87.2.1	~INetworkDefinition()	418
9.87.3	Member Function Documentation	419
9.87.3.1	addActivation()	419
9.87.3.2	addAssertion()	419
9.87.3.3	addCast()	420
9.87.3.4	addConcatenation()	420
9.87.3.5	addConstant()	421
9.87.3.6	addConvolution()	422
9.87.3.7	addConvolutionNd()	422
9.87.3.8	addDeconvolution()	423
9.87.3.9	addDeconvolutionNd()	424
9.87.3.10	addDequantize()	425
9.87.3.11	addEinsum()	425
9.87.3.12	addElementWise()	426
9.87.3.13	addFill()	426
9.87.3.14	addFullyConnected()	427
9.87.3.15	addGather()	428
9.87.3.16	addGatherV2()	428
9.87.3.17	addGridSample()	429
9.87.3.18	addIdentity()	429
9.87.3.19	addIfConditional()	430

9.87.3.20	addInput()	430
9.87.3.21	addLoop()	431
9.87.3.22	addLRN()	431
9.87.3.23	addMatrixMultiply()	432
9.87.3.24	addNMS()	433
9.87.3.25	addNonZero()	433
9.87.3.26	addNormalization()	434
9.87.3.27	addOneHot()	435
9.87.3.28	addPadding()	435
9.87.3.29	addPaddingNd()	436
9.87.3.30	addParametricReLU()	436
9.87.3.31	addPluginV2()	437
9.87.3.32	addPooling()	438
9.87.3.33	addPoolingNd()	438
9.87.3.34	addQuantize()	439
9.87.3.35	addRaggedSoftMax()	439
9.87.3.36	addReduce()	440
9.87.3.37	addResize()	441
9.87.3.38	addReverseSequence()	441
9.87.3.39	addRNNV2()	442
9.87.3.40	addScale()	444
9.87.3.41	addScaleNd()	444
9.87.3.42	addScatter()	445
9.87.3.43	addSelect()	446
9.87.3.44	addShape()	447
9.87.3.45	addShuffle()	447
9.87.3.46	addSlice()	448
9.87.3.47	addSoftMax()	449
9.87.3.48	addTopK()	449
9.87.3.49	addUnary()	450
9.87.3.50	destroy()	451
9.87.3.51	getBuilder()	451
9.87.3.52	getErrorRecorder()	451
9.87.3.53	getInput()	451
9.87.3.54	getLayer()	452
9.87.3.55	getName()	452
9.87.3.56	getNbInputs()	453
9.87.3.57	getNbLayers()	453
9.87.3.58	getNbOutputs()	454
9.87.3.59	getOutput()	454
9.87.3.60	hasExplicitPrecision()	455
9.87.3.61	hasImplicitBatchDimension()	455
9.87.3.62	markOutput()	455
9.87.3.63	markOutputForShapes()	456
9.87.3.64	removeTensor()	456
9.87.3.65	setErrorRecorder()	457
9.87.3.66	setName()	457
9.87.3.67	setWeightsName()	458
9.87.3.68	unmarkOutput()	458
9.87.3.69	unmarkOutputForShapes()	459
9.87.4	Member Data Documentation	459
9.87.4.1	mImpl	459
9.88	nvinfer1::INMSLayer Class Reference	459
9.88.1	Detailed Description	460

9.88.2	Constructor & Destructor Documentation	461
9.88.2.1	~INMSLayer()	461
9.88.3	Member Function Documentation	461
9.88.3.1	getBoundingBoxFormat()	461
9.88.3.2	getTopKBoxLimit()	462
9.88.3.3	setBoundingBoxFormat()	462
9.88.3.4	setInput()	462
9.88.3.5	setTopKBoxLimit()	463
9.88.4	Member Data Documentation	463
9.88.4.1	mImpl	463
9.89	nvinfer1::INoCopy Class Reference	464
9.89.1	Detailed Description	464
9.89.2	Constructor & Destructor Documentation	465
9.89.2.1	INoCopy() [1/3]	465
9.89.2.2	~INoCopy()	465
9.89.2.3	INoCopy() [2/3]	465
9.89.2.4	INoCopy() [3/3]	465
9.89.3	Member Function Documentation	465
9.89.3.1	operator=() [1/2]	465
9.89.3.2	operator=() [2/2]	466
9.90	INonZero Class Reference	466
9.90.1	Detailed Description	466
9.91	nvinfer1::INonZeroLayer Class Reference	466
9.91.1	Constructor & Destructor Documentation	467
9.91.1.1	~INonZeroLayer()	467
9.91.2	Member Data Documentation	467
9.91.2.1	mImpl	467
9.92	nvinfer1::INormalizationLayer Class Reference	467
9.92.1	Detailed Description	468
9.92.2	Constructor & Destructor Documentation	468
9.92.2.1	~INormalizationLayer()	469
9.92.3	Member Function Documentation	469
9.92.3.1	getAxes()	469
9.92.3.2	getComputePrecision()	469
9.92.3.3	getEpsilon()	469
9.92.3.4	getNbGroups()	470
9.92.3.5	setAxes()	470
9.92.3.6	setComputePrecision()	470
9.92.3.7	setEpsilon()	471
9.92.3.8	setNbGroups()	471
9.92.4	Member Data Documentation	471
9.92.4.1	mImpl	471
9.93	nvinfer1::IOneHotLayer Class Reference	472
9.93.1	Detailed Description	472
9.93.2	Member Function Documentation	473
9.93.2.1	getAxis()	473
9.93.2.2	setAxis()	473
9.93.3	Member Data Documentation	473
9.93.3.1	mImpl	474
9.94	nvonnxparser::IOnnxConfig Class Reference	474
9.94.1	Detailed Description	475
9.94.2	Member Typedef Documentation	475
9.94.2.1	Verbosity	475
9.94.3	Constructor & Destructor Documentation	475

9.94.3.1	~IOnnxConfig()	475
9.94.4	Member Function Documentation	475
9.94.4.1	addVerbosity()	475
9.94.4.2	destroy()	475
9.94.4.3	getFullTextFileName()	476
9.94.4.4	getModelDtype()	476
9.94.4.5	getModelFileName()	477
9.94.4.6	getPrintLayerInfo()	477
9.94.4.7	getTextFileName()	477
9.94.4.8	getVerbosityLevel()	478
9.94.4.9	reduceVerbosity()	478
9.94.4.10	setFullTextFileName()	478
9.94.4.11	setModelDtype()	479
9.94.4.12	setModelFileName()	479
9.94.4.13	setPrintLayerInfo()	479
9.94.4.14	setTextFileName()	480
9.94.4.15	setVerbosityLevel()	480
9.95	nvinfer1::IOptimizationProfile Class Reference	481
9.95.1	Detailed Description	482
9.95.2	Constructor & Destructor Documentation	482
9.95.2.1	~IOptimizationProfile()	482
9.95.3	Member Function Documentation	482
9.95.3.1	getDimensions()	482
9.95.3.2	getExtraMemoryTarget()	483
9.95.3.3	getNbShapeValues()	483
9.95.3.4	getShapeValues()	483
9.95.3.5	isValid()	484
9.95.3.6	setDimensions()	484
9.95.3.7	setExtraMemoryTarget()	485
9.95.3.8	setShapeValues()	485
9.95.4	Member Data Documentation	486
9.95.4.1	mImpl	486
9.96	nvinfer1::IOutputAllocator Class Reference	487
9.96.1	Detailed Description	487
9.96.2	Constructor & Destructor Documentation	487
9.96.2.1	~IOutputAllocator()	487
9.96.3	Member Function Documentation	487
9.96.3.1	getInterfaceVersion()	488
9.96.3.2	notifyShape()	488
9.96.3.3	reallocateOutput()	488
9.97	nvinfer1::IPaddingLayer Class Reference	489
9.97.1	Detailed Description	490
9.97.2	Constructor & Destructor Documentation	490
9.97.2.1	~IPaddingLayer()	490
9.97.3	Member Function Documentation	490
9.97.3.1	getPostPadding()	490
9.97.3.2	getPostPaddingNd()	490
9.97.3.3	getPrePadding()	491
9.97.3.4	getPrePaddingNd()	491
9.97.3.5	setPostPadding()	492
9.97.3.6	setPostPaddingNd()	492
9.97.3.7	setPrePadding()	492
9.97.3.8	setPrePaddingNd()	493
9.97.4	Member Data Documentation	493

9.97.4.1	mImpl	493
9.98	nvinfer1::IParametricReLU Layer Class Reference	493
9.98.1	Detailed Description	494
9.98.2	Constructor & Destructor Documentation	494
9.98.2.1	~IParametricReLU Layer()	494
9.98.3	Member Data Documentation	494
9.98.3.1	mImpl	494
9.99	nvonnxparser::IParser Class Reference	495
9.99.1	Detailed Description	496
9.99.2	Constructor & Destructor Documentation	496
9.99.2.1	~IParser()	496
9.99.3	Member Function Documentation	496
9.99.3.1	clearErrors()	496
9.99.3.2	clearFlag()	496
9.99.3.3	destroy()	496
9.99.3.4	getError()	497
9.99.3.5	getFlag()	497
9.99.3.6	getFlags()	497
9.99.3.7	getNbErrors()	498
9.99.3.8	getUsedVCPluginLibraries()	498
9.99.3.9	parse()	498
9.99.3.10	parseFromFile()	499
9.99.3.11	parseWithWeightDescriptors()	499
9.99.3.12	setFlag()	500
9.99.3.13	setFlags()	500
9.99.3.14	supportsModel()	501
9.99.3.15	supportsOperator()	501
9.100	nvonnxparser::IParserError Class Reference	502
9.100.1	Detailed Description	502
9.100.2	Constructor & Destructor Documentation	502
9.100.2.1	~IParserError()	502
9.100.3	Member Function Documentation	503
9.100.3.1	code()	503
9.100.3.2	desc()	503
9.100.3.3	file()	503
9.100.3.4	func()	503
9.100.3.5	line()	503
9.100.3.6	node()	504
9.101	nvinfer1::consistency::IPluginChecker Class Reference	504
9.101.1	Detailed Description	504
9.101.2	Constructor & Destructor Documentation	505
9.101.2.1	IPluginChecker() [1/3]	505
9.101.2.2	~IPluginChecker()	505
9.101.2.3	IPluginChecker() [2/3]	505
9.101.2.4	IPluginChecker() [3/3]	505
9.101.3	Member Function Documentation	505
9.101.3.1	operator=() [1/2]	505
9.101.3.2	operator=() [2/2]	506
9.101.3.3	validate()	506
9.102	nvinfer1::IPluginCreator Class Reference	506
9.102.1	Detailed Description	507
9.102.2	Constructor & Destructor Documentation	507
9.102.2.1	IPluginCreator()	507
9.102.2.2	~IPluginCreator()	508

9.102.3 Member Function Documentation	508
9.102.3.1 createPlugin()	508
9.102.3.2 deserializePlugin()	508
9.102.3.3 getFieldNames()	509
9.102.3.4 getPluginName()	509
9.102.3.5 getPluginNamespace()	510
9.102.3.6 getPluginVersion()	510
9.102.3.7 getTensorRTVersion()	511
9.102.3.8 setPluginNamespace()	511
9.103 nvcaffeparser1::IPluginFactoryV2 Class Reference	512
9.103.1 Detailed Description	512
9.103.2 Constructor & Destructor Documentation	512
9.103.2.1 ~IPluginFactoryV2()	512
9.103.3 Member Function Documentation	512
9.103.3.1 createPlugin()	512
9.103.3.2 isPluginV2()	513
9.104 nvinfer1::IPluginRegistry Class Reference	513
9.104.1 Detailed Description	514
9.104.2 Member Typedef Documentation	515
9.104.2.1 PluginLibraryHandle	515
9.104.3 Constructor & Destructor Documentation	515
9.104.3.1 ~IPluginRegistry()	515
9.104.4 Member Function Documentation	515
9.104.4.1 deregisterCreator()	515
9.104.4.2 deregisterLibrary()	515
9.104.4.3 getErrorRecorder()	516
9.104.4.4 getPluginCreator()	516
9.104.4.5 getPluginCreatorList()	517
9.104.4.6 isParentSearchEnabled()	517
9.104.4.7 loadLibrary()	517
9.104.4.8 registerCreator()	518
9.104.4.9 setErrorRecorder()	518
9.104.4.10 setParentSearchEnabled()	519
9.105 nvinfer1::safe::IPluginRegistry Class Reference	519
9.105.1 Detailed Description	520
9.105.2 Constructor & Destructor Documentation	520
9.105.2.1 ~IPluginRegistry()	520
9.105.3 Member Function Documentation	521
9.105.3.1 deregisterCreator()	521
9.105.3.2 getErrorRecorder()	521
9.105.3.3 getPluginCreator()	522
9.105.3.4 getPluginCreatorList()	523
9.105.3.5 registerCreator()	523
9.105.3.6 setErrorRecorder()	524
9.106 nvinfer1::IPluginV2 Class Reference	525
9.106.1 Detailed Description	526
9.106.2 Member Function Documentation	526
9.106.2.1 clone()	527
9.106.2.2 configureWithFormat()	527
9.106.2.3 destroy()	528
9.106.2.4 enqueue()	528
9.106.2.5 getNbOutputs()	529
9.106.2.6 getOutputDimensions()	530
9.106.2.7 getPluginNamespace()	531

9.106.2.8	getPluginType()	532
9.106.2.9	getPluginVersion()	532
9.106.2.10	getSerializationSize()	533
9.106.2.11	getTensorRTVersion()	533
9.106.2.12	getWorkspaceSize()	533
9.106.2.13	initialize()	534
9.106.2.14	serialize()	534
9.106.2.15	setPluginNamespace()	535
9.106.2.16	supportsFormat()	536
9.106.2.17	terminate()	536
9.107	nvInfer1::IPluginV2DynamicExt Class Reference	537
9.107.1	Detailed Description	538
9.107.2	Constructor & Destructor Documentation	538
9.107.2.1	~IPluginV2DynamicExt()	538
9.107.3	Member Function Documentation	539
9.107.3.1	clone()	539
9.107.3.2	configurePlugin()	539
9.107.3.3	enqueue()	540
9.107.3.4	getOutputDimensions()	541
9.107.3.5	getTensorRTVersion()	541
9.107.3.6	getWorkspaceSize()	542
9.107.3.7	supportsFormatCombination()	542
9.107.4	Member Data Documentation	543
9.107.4.1	kFORMAT_COMBINATION_LIMIT	543
9.108	nvInfer1::IPluginV2Ext Class Reference	543
9.108.1	Detailed Description	544
9.108.2	Constructor & Destructor Documentation	544
9.108.2.1	IPluginV2Ext()	544
9.108.2.2	~IPluginV2Ext()	545
9.108.3	Member Function Documentation	545
9.108.3.1	attachToContext()	545
9.108.3.2	canBroadcastInputAcrossBatch()	546
9.108.3.3	clone()	546
9.108.3.4	configurePlugin()	547
9.108.3.5	configureWithFormat()	548
9.108.3.6	detachFromContext()	549
9.108.3.7	getOutputDataType()	549
9.108.3.8	getTensorRTVersion()	550
9.108.3.9	isOutputBroadcastAcrossBatch()	550
9.109	nvInfer1::IPluginV2IOExt Class Reference	551
9.109.1	Detailed Description	552
9.109.2	Member Function Documentation	552
9.109.2.1	configurePlugin()	552
9.109.2.2	getTensorRTVersion()	553
9.109.2.3	supportsFormatCombination()	553
9.110	nvInfer1::IPluginV2Layer Class Reference	554
9.110.1	Detailed Description	555
9.110.2	Constructor & Destructor Documentation	555
9.110.2.1	~IPluginV2Layer()	555
9.110.3	Member Function Documentation	555
9.110.3.1	getPlugin()	555
9.110.4	Member Data Documentation	556
9.110.4.1	mImpl	556
9.111	nvInfer1::IPoolingLayer Class Reference	556

9.111.1 Detailed Description	558
9.111.2 Constructor & Destructor Documentation	558
9.111.2.1 ~IPoolingLayer()	558
9.111.3 Member Function Documentation	558
9.111.3.1 getAverageCountExcludesPadding()	558
9.111.3.2 getBlendFactor()	558
9.111.3.3 getPadding()	559
9.111.3.4 getPaddingMode()	559
9.111.3.5 getPaddingNd()	559
9.111.3.6 getPoolingType()	560
9.111.3.7 getPostPadding()	560
9.111.3.8 getPrePadding()	560
9.111.3.9 getStride()	560
9.111.3.10 getStrideNd()	561
9.111.3.11 getWindowSize()	561
9.111.3.12 getWindowSizeNd()	561
9.111.3.13 setAverageCountExcludesPadding()	562
9.111.3.14 setBlendFactor()	562
9.111.3.15 setPadding()	562
9.111.3.16 setPaddingMode()	563
9.111.3.17 setPaddingNd()	563
9.111.3.18 setPoolingType()	563
9.111.3.19 setPostPadding()	564
9.111.3.20 setPrePadding()	564
9.111.3.21 setStride()	564
9.111.3.22 setStrideNd()	565
9.111.3.23 setWindowSize()	565
9.111.3.24 setWindowSizeNd()	565
9.111.4 Member Data Documentation	566
9.111.4.1 mImpl	566
9.112 nvinfer1::IProfiler Class Reference	566
9.112.1 Detailed Description	566
9.112.2 Constructor & Destructor Documentation	566
9.112.2.1 ~IProfiler()	566
9.112.3 Member Function Documentation	567
9.112.3.1 reportLayerTime()	567
9.113 nvinfer1::IQuantizeLayer Class Reference	568
9.113.1 Detailed Description	569
9.113.2 Constructor & Destructor Documentation	569
9.113.2.1 ~IQuantizeLayer()	570
9.113.3 Member Function Documentation	570
9.113.3.1 getAxis()	570
9.113.3.2 setAxis()	570
9.113.4 Member Data Documentation	570
9.113.4.1 mImpl	570
9.114 nvinfer1::IRaggedSoftMaxLayer Class Reference	571
9.114.1 Detailed Description	571
9.114.2 Constructor & Destructor Documentation	571
9.114.2.1 ~IRaggedSoftMaxLayer()	572
9.114.3 Member Data Documentation	572
9.114.3.1 mImpl	572
9.115 nvinfer1::IRecurrenceLayer Class Reference	572
9.115.1 Constructor & Destructor Documentation	573
9.115.1.1 ~IRecurrenceLayer()	573

9.115.2 Member Function Documentation	573
9.115.2.1 setInput()	573
9.115.3 Member Data Documentation	573
9.115.3.1 mImpl	574
9.116 nvinfer1::IReduceLayer Class Reference	574
9.116.1 Detailed Description	574
9.116.2 Constructor & Destructor Documentation	575
9.116.2.1 ~IReduceLayer()	575
9.116.3 Member Function Documentation	575
9.116.3.1 getKeepDimensions()	575
9.116.3.2 getOperation()	575
9.116.3.3 getReduceAxes()	576
9.116.3.4 setKeepDimensions()	576
9.116.3.5 setOperation()	576
9.116.3.6 setReduceAxes()	576
9.116.4 Member Data Documentation	577
9.116.4.1 mImpl	577
9.117 nvinfer1::IRefitter Class Reference	577
9.117.1 Detailed Description	578
9.117.2 Constructor & Destructor Documentation	578
9.117.2.1 ~IRefitter()	578
9.117.3 Member Function Documentation	578
9.117.3.1 destroy()	579
9.117.3.2 getAll()	579
9.117.3.3 getAllWeights()	579
9.117.3.4 getDynamicRangeMax()	580
9.117.3.5 getDynamicRangeMin()	580
9.117.3.6 getErrorRecorder()	581
9.117.3.7 getLogger()	581
9.117.3.8 getMaxThreads()	581
9.117.3.9 getMissing()	581
9.117.3.10 getMissingWeights()	582
9.117.3.11 getTensorsWithDynamicRange()	582
9.117.3.12 refitCudaEngine()	583
9.117.3.13 setDynamicRange()	583
9.117.3.14 setErrorRecorder()	584
9.117.3.15 setMaxThreads()	584
9.117.3.16 setNamedWeights()	585
9.117.3.17 setWeights()	585
9.117.4 Member Data Documentation	586
9.117.4.1 mImpl	586
9.118 nvinfer1::IResizeLayer Class Reference	586
9.118.1 Detailed Description	588
9.118.2 Constructor & Destructor Documentation	588
9.118.2.1 ~IResizeLayer()	589
9.118.3 Member Function Documentation	589
9.118.3.1 getAlignCorners()	589
9.118.3.2 getCoordinateTransformation()	589
9.118.3.3 getCubicCoeff()	589
9.118.3.4 getExcludeOutside()	590
9.118.3.5 getNearestRounding()	590
9.118.3.6 getOutputDimensions()	590
9.118.3.7 getResizeMode()	590
9.118.3.8 getScales()	590

9.118.3.9	getSelectorForSinglePixel()	591
9.118.3.10	setAlignCorners()	591
9.118.3.11	setCoordinateTransformation()	592
9.118.3.12	setCubicCoeff()	592
9.118.3.13	setExcludeOutside()	592
9.118.3.14	setInput()	592
9.118.3.15	setNearestRounding()	593
9.118.3.16	setOutputDimensions()	593
9.118.3.17	setResizeMode()	594
9.118.3.18	setScales()	594
9.118.3.19	setSelectorForSinglePixel()	595
9.118.4	Member Data Documentation	595
9.118.4.1	mImpl	595
9.119	nvinfer1::IReverseSequenceLayer Class Reference	596
9.119.1	Detailed Description	596
9.119.2	Constructor & Destructor Documentation	597
9.119.2.1	~IReverseSequenceLayer()	597
9.119.3	Member Function Documentation	597
9.119.3.1	getBatchAxis()	597
9.119.3.2	getSequenceAxis()	597
9.119.3.3	setBatchAxis()	598
9.119.3.4	setSequenceAxis()	598
9.119.4	Member Data Documentation	598
9.119.4.1	mImpl	598
9.120	nvinfer1::IRNNv2Layer Class Reference	599
9.120.1	Detailed Description	600
9.120.2	Constructor & Destructor Documentation	600
9.120.2.1	~IRNNv2Layer()	600
9.120.3	Member Function Documentation	600
9.120.3.1	getBiasForGate()	601
9.120.3.2	getCellState()	601
9.120.3.3	getDataLength()	601
9.120.3.4	getDirection()	601
9.120.3.5	getHiddenSize()	602
9.120.3.6	getHiddenState()	602
9.120.3.7	getInputMode()	602
9.120.3.8	getLayerCount()	602
9.120.3.9	getMaxSeqLength()	602
9.120.3.10	getOperation()	603
9.120.3.11	getSequenceLengths()	603
9.120.3.12	getWeightsForGate()	603
9.120.3.13	setBiasForGate()	603
9.120.3.14	setCellState()	604
9.120.3.15	setDirection()	604
9.120.3.16	setHiddenState()	605
9.120.3.17	setInputMode()	605
9.120.3.18	setOperation()	605
9.120.3.19	setSequenceLengths()	606
9.120.3.20	setWeightsForGate()	606
9.120.4	Member Data Documentation	607
9.120.4.1	mImpl	607
9.121	nvinfer1::IRuntime Class Reference	607
9.121.1	Detailed Description	609
9.121.2	Constructor & Destructor Documentation	609

9.121.2.1 ~IRuntime()	609
9.121.3 Member Function Documentation	609
9.121.3.1 deserializeCudaEngine() [1/2]	609
9.121.3.2 deserializeCudaEngine() [2/2]	610
9.121.3.3 destroy()	610
9.121.3.4 getDLACore()	611
9.121.3.5 getEngineHostCodeAllowed()	611
9.121.3.6 getErrorRecorder()	611
9.121.3.7 getLogger()	612
9.121.3.8 getMaxThreads()	612
9.121.3.9 getNbDLACores()	612
9.121.3.10 getPluginRegistry()	612
9.121.3.11 getTempfileControlFlags()	613
9.121.3.12 getTemporaryDirectory()	613
9.121.3.13 loadRuntime()	613
9.121.3.14 setDLACore()	614
9.121.3.15 setEngineHostCodeAllowed()	614
9.121.3.16 setErrorRecorder()	615
9.121.3.17 setGpuAllocator()	615
9.121.3.18 setMaxThreads()	615
9.121.3.19 setTempfileControlFlags()	616
9.121.3.20 setTemporaryDirectory()	616
9.121.4 Member Data Documentation	617
9.121.4.1 mImpl	617
9.122 nvinfer1::safe::IRuntime Class Reference	617
9.122.1 Detailed Description	618
9.122.2 Constructor & Destructor Documentation	618
9.122.2.1 IRuntime() [1/3]	618
9.122.2.2 ~IRuntime()	619
9.122.2.3 IRuntime() [2/3]	619
9.122.2.4 IRuntime() [3/3]	619
9.122.3 Member Function Documentation	619
9.122.3.1 deserializeCudaEngine()	619
9.122.3.2 getErrorRecorder()	620
9.122.3.3 operator=() [1/2]	621
9.122.3.4 operator=() [2/2]	621
9.122.3.5 setErrorRecorder()	621
9.122.3.6 setGpuAllocator()	621
9.123 nvinfer1::IScaleLayer Class Reference	622
9.123.1 Detailed Description	623
9.123.2 Constructor & Destructor Documentation	624
9.123.2.1 ~IScaleLayer()	624
9.123.3 Member Function Documentation	624
9.123.3.1 getChannelAxis()	624
9.123.3.2 getMode()	624
9.123.3.3 getPower()	625
9.123.3.4 getScale()	625
9.123.3.5 getShift()	625
9.123.3.6 setChannelAxis()	625
9.123.3.7 setMode()	626
9.123.3.8 setPower()	626
9.123.3.9 setScale()	626
9.123.3.10 setShift()	626
9.123.4 Member Data Documentation	627

9.123.4.1 mImpl	627
9.124nvinfer1::IScatterLayer Class Reference	627
9.124.1 Detailed Description	628
9.124.2 Constructor & Destructor Documentation	629
9.124.2.1 ~IScatterLayer()	629
9.124.3 Member Function Documentation	629
9.124.3.1 getAxis()	629
9.124.3.2 getMode()	629
9.124.3.3 setAxis()	629
9.124.3.4 setMode()	630
9.124.4 Member Data Documentation	630
9.124.4.1 mImpl	630
9.125nvinfer1::ISelectLayer Class Reference	630
9.125.1 Detailed Description	630
9.125.2 Constructor & Destructor Documentation	631
9.125.2.1 ~ISelectLayer()	631
9.125.3 Member Data Documentation	631
9.125.3.1 mImpl	631
9.126nvinfer1::IShapeLayer Class Reference	631
9.126.1 Detailed Description	632
9.126.2 Constructor & Destructor Documentation	632
9.126.2.1 ~IShapeLayer()	632
9.126.3 Member Data Documentation	632
9.126.3.1 mImpl	632
9.127nvinfer1::IShuffleLayer Class Reference	633
9.127.1 Detailed Description	634
9.127.2 Constructor & Destructor Documentation	634
9.127.2.1 ~IShuffleLayer()	634
9.127.3 Member Function Documentation	634
9.127.3.1 getFirstTranspose()	634
9.127.3.2 getReshapeDimensions()	635
9.127.3.3 getSecondTranspose()	635
9.127.3.4 getZeroIsPlaceholder()	635
9.127.3.5 setFirstTranspose()	635
9.127.3.6 setInput()	636
9.127.3.7 setReshapeDimensions()	637
9.127.3.8 setSecondTranspose()	637
9.127.3.9 setZeroIsPlaceholder()	638
9.127.4 Member Data Documentation	638
9.127.4.1 mImpl	638
9.128nvinfer1::ISliceLayer Class Reference	638
9.128.1 Detailed Description	639
9.128.2 Constructor & Destructor Documentation	640
9.128.2.1 ~ISliceLayer()	640
9.128.3 Member Function Documentation	640
9.128.3.1 getMode()	640
9.128.3.2 getSize()	641
9.128.3.3 getStart()	641
9.128.3.4 getStride()	641
9.128.3.5 setInput()	641
9.128.3.6 setMode()	642
9.128.3.7 setSize()	642
9.128.3.8 setStart()	643
9.128.3.9 setStride()	643

9.128.4 Member Data Documentation	644
9.128.4.1 mImpl	644
9.129 nvinfer1::ISoftMaxLayer Class Reference	644
9.129.1 Detailed Description	645
9.129.2 Constructor & Destructor Documentation	645
9.129.2.1 ~ISoftMaxLayer()	645
9.129.3 Member Function Documentation	645
9.129.3.1 getAxes()	645
9.129.3.2 setAxes()	646
9.129.4 Member Data Documentation	646
9.129.4.1 mImpl	646
9.130 nvinfer1::ITensor Class Reference	646
9.130.1 Detailed Description	648
9.130.2 Constructor & Destructor Documentation	648
9.130.2.1 ~ITensor()	648
9.130.3 Member Function Documentation	649
9.130.3.1 dynamicRangeIsSet()	649
9.130.3.2 getAllowedFormats()	649
9.130.3.3 getBroadcastAcrossBatch()	649
9.130.3.4 getDimensionName()	649
9.130.3.5 getDimensions()	650
9.130.3.6 getDynamicRangeMax()	650
9.130.3.7 getDynamicRangeMin()	651
9.130.3.8 getLocation()	651
9.130.3.9 getName()	651
9.130.3.10 getType()	652
9.130.3.11 isExecutionTensor()	652
9.130.3.12 isNetworkInput()	652
9.130.3.13 isNetworkOutput()	653
9.130.3.14 isShapeTensor()	653
9.130.3.15 resetDynamicRange()	653
9.130.3.16 setAllowedFormats()	654
9.130.3.17 setBroadcastAcrossBatch()	654
9.130.3.18 setDimensionName()	655
9.130.3.19 setDimensions()	655
9.130.3.20 setDynamicRange()	656
9.130.3.21 setLocation()	656
9.130.3.22 setName()	657
9.130.3.23 setType()	657
9.130.4 Member Data Documentation	657
9.130.4.1 mImpl	658
9.131 nvinfer1::ITimingCache Class Reference	658
9.131.1 Detailed Description	658
9.131.2 Constructor & Destructor Documentation	659
9.131.2.1 ~ITimingCache()	659
9.131.3 Member Function Documentation	659
9.131.3.1 combine()	659
9.131.3.2 reset()	660
9.131.3.3 serialize()	660
9.131.4 Member Data Documentation	660
9.131.4.1 mImpl	660
9.132 nvinfer1::ITopKLayer Class Reference	661
9.132.1 Detailed Description	661
9.132.2 Constructor & Destructor Documentation	662

9.132.2.1 ~ITopKLayer()	662
9.132.3 Member Function Documentation	662
9.132.3.1 getK()	662
9.132.3.2 getOperation()	662
9.132.3.3 getReduceAxes()	663
9.132.3.4 setInput()	663
9.132.3.5 setK()	663
9.132.3.6 setOperation()	664
9.132.3.7 setReduceAxes()	664
9.132.4 Member Data Documentation	664
9.132.4.1 mImpl	664
9.133 nvinfer1::ITripLimitLayer Class Reference	664
9.133.1 Constructor & Destructor Documentation	665
9.133.1.1 ~ITripLimitLayer()	665
9.133.2 Member Function Documentation	665
9.133.2.1 getTripLimit()	665
9.133.3 Member Data Documentation	665
9.133.3.1 mImpl	665
9.134 nvuffparser::IUffParser Class Reference	666
9.134.1 Detailed Description	666
9.134.2 Constructor & Destructor Documentation	666
9.134.2.1 ~IUffParser()	667
9.134.3 Member Function Documentation	667
9.134.3.1 destroy()	667
9.134.3.2 getErrorRecorder()	667
9.134.3.3 getUffRequiredVersionMajor()	667
9.134.3.4 getUffRequiredVersionMinor()	668
9.134.3.5 getUffRequiredVersionPatch()	668
9.134.3.6 parse()	668
9.134.3.7 parseBuffer()	668
9.134.3.8 registerInput()	669
9.134.3.9 registerOutput()	669
9.134.3.10 setErrorRecorder()	669
9.134.3.11 setPluginNamespace()	670
9.135 nvinfer1::IUnaryLayer Class Reference	670
9.135.1 Detailed Description	671
9.135.2 Constructor & Destructor Documentation	671
9.135.2.1 ~IUnaryLayer()	671
9.135.3 Member Function Documentation	671
9.135.3.1 getOperation()	671
9.135.3.2 setOperation()	672
9.135.4 Member Data Documentation	672
9.135.4.1 mImpl	672
9.136 nvinfer1::plugin::NMSPParameters Struct Reference	672
9.136.1 Detailed Description	672
9.136.2 Member Data Documentation	673
9.136.2.1 backgroundLabelId	673
9.136.2.2 iouThreshold	673
9.136.2.3 isNormalized	673
9.136.2.4 keepTopK	673
9.136.2.5 numClasses	674
9.136.2.6 scoreThreshold	674
9.136.2.7 shareLocation	674
9.136.2.8 topK	674

9.137	<code>nvinfer1::Permutation Struct Reference</code>	674
9.137.1	Member Data Documentation	674
9.137.1.1	order	675
9.138	<code>nvinfer1::PluginField Class Reference</code>	675
9.138.1	Detailed Description	675
9.138.2	Constructor & Destructor Documentation	675
9.138.2.1	<code>PluginField()</code>	676
9.138.3	Member Data Documentation	676
9.138.3.1	data	676
9.138.3.2	length	676
9.138.3.3	name	676
9.138.3.4	type	676
9.139	<code>nvinfer1::PluginFieldCollection Struct Reference</code>	677
9.139.1	Detailed Description	677
9.139.2	Member Data Documentation	677
9.139.2.1	fields	677
9.139.2.2	nbFields	677
9.140	<code>nvinfer1::PluginRegistrar< T > Class Template Reference</code>	678
9.140.1	Detailed Description	678
9.140.2	Constructor & Destructor Documentation	678
9.140.2.1	<code>PluginRegistrar()</code>	678
9.141	<code>nvinfer1::safe::PluginRegistrar< T > Class Template Reference</code>	678
9.141.1	Detailed Description	679
9.141.2	Constructor & Destructor Documentation	679
9.141.2.1	<code>PluginRegistrar()</code>	679
9.142	<code>nvinfer1::PluginTensorDesc Struct Reference</code>	679
9.142.1	Detailed Description	680
9.142.2	Member Data Documentation	680
9.142.2.1	dims	680
9.142.2.2	format	680
9.142.2.3	scale	680
9.142.2.4	type	680
9.143	<code>PluginVersion Struct Reference</code>	681
9.143.1	Detailed Description	681
9.144	<code>nvinfer1::plugin::PriorBoxParameters Struct Reference</code>	681
9.144.1	Detailed Description	681
9.144.2	Member Data Documentation	682
9.144.2.1	aspectRatios	682
9.144.2.2	clip	682
9.144.2.3	flip	682
9.144.2.4	imgH	683
9.144.2.5	imgW	683
9.144.2.6	maxSize	683
9.144.2.7	minSize	683
9.144.2.8	numAspectRatios	683
9.144.2.9	numMaxSize	683
9.144.2.10	numMinSize	683
9.144.2.11	loffset	684
9.144.2.12	stepH	684
9.144.2.13	stepW	684
9.144.2.14	variance	684
9.145	<code>nvinfer1::plugin::Quadruple Struct Reference</code>	684
9.145.1	Detailed Description	684
9.145.2	Member Data Documentation	685

9.145.2.1 data	685
9.146nvinfer1::plugin::RegionParameters Struct Reference	685
9.146.1 Detailed Description	685
9.146.2 Member Data Documentation	686
9.146.2.1 classes	686
9.146.2.2 coords	686
9.146.2.3 num	686
9.146.2.4 smTree	686
9.147nvinfer1::plugin::RPROIParams Struct Reference	686
9.147.1 Detailed Description	686
9.147.2 Member Data Documentation	687
9.147.2.1 anchorsRatioCount	687
9.147.2.2 anchorsScaleCount	687
9.147.2.3 featureStride	687
9.147.2.4 iouThreshold	687
9.147.2.5 minBoxSize	688
9.147.2.6 nmsMaxOut	688
9.147.2.7 poolingH	688
9.147.2.8 poolingW	688
9.147.2.9 preNmsTop	688
9.147.2.10spatialScale	688
9.148nvinfer1::safe::RuntimeErrorInformation Struct Reference	688
9.148.1 Detailed Description	689
9.148.2 Member Data Documentation	689
9.148.2.1 bitMask	689
9.149nvinfer1::plugin::softmaxTree Struct Reference	689
9.149.1 Detailed Description	690
9.149.2 Member Data Documentation	690
9.149.2.1 child	690
9.149.2.2 group	690
9.149.2.3 groupOffset	690
9.149.2.4 groups	690
9.149.2.5 groupSize	690
9.149.2.6 leaf	691
9.149.2.7 n	691
9.149.2.8 name	691
9.149.2.9 parent	691
9.150nvinfer1::Weights Class Reference	691
9.150.1 Detailed Description	692
9.150.2 Member Data Documentation	692
9.150.2.1 count	692
9.150.2.2 type	692
9.150.2.3 values	692
10 File Documentation	693
10.1 NvCaffeParser.h File Reference	693
10.1.1 Detailed Description	694
10.2 NvCaffeParser.h	694
10.3 NvInfer.h File Reference	695
10.3.1 Detailed Description	702
10.4 NvInfer.h	702
10.5 NvInferConsistency.h File Reference	748
10.5.1 Function Documentation	748
10.5.1.1 createConsistencyChecker_INTERNAL()	748

10.6	NvInferConsistency.h	749
10.7	NvInferLegacyDims.h File Reference	750
10.7.1	Detailed Description	750
10.8	NvInferLegacyDims.h	751
10.9	NvInferPlugin.h File Reference	752
10.9.1	Detailed Description	753
10.9.2	Function Documentation	753
10.9.2.1	createAnchorGeneratorPlugin()	754
10.9.2.2	createBatchedNMSPlugin()	754
10.9.2.3	createInstanceNormalizationPlugin()	754
10.9.2.4	createNMSPlugin()	755
10.9.2.5	createNormalizePlugin()	755
10.9.2.6	createPriorBoxPlugin()	756
10.9.2.7	createRegionPlugin()	756
10.9.2.8	createReorgPlugin()	756
10.9.2.9	createRPNROIPlugin()	757
10.9.2.10	createSplitPlugin()	758
10.9.2.11	initLibNvInferPlugins()	758
10.10	NvInferPlugin.h	758
10.11	NvInferPluginUtils.h File Reference	759
10.11.1	Detailed Description	760
10.12	NvInferPluginUtils.h	760
10.13	NvInferRuntime.h File Reference	762
10.13.1	Detailed Description	765
10.13.2	Macro Definition Documentation	765
10.13.2.1	REGISTER_TENSORRT_PLUGIN	765
10.13.3	Function Documentation	765
10.13.3.1	getLogger()	765
10.13.3.2	getPluginRegistry()	766
10.14	NvInferRuntime.h	766
10.15	NvInferRuntimeBase.h File Reference	782
10.15.1	Detailed Description	783
10.15.2	Macro Definition Documentation	784
10.15.2.1	NV_TENSORRT_VERSION	784
10.15.2.2	TENSORRTAPI	784
10.15.2.3	TRT_DEPRECATED	784
10.15.2.4	TRT_DEPRECATED_API	784
10.15.2.5	TRT_DEPRECATED_ENUM	784
10.15.2.6	TRTNOEXCEPT	784
10.15.3	Function Documentation	785
10.15.3.1	getInferLibVersion()	785
10.16	NvInferRuntimeBase.h	785
10.17	NvInferRuntimeCommon.h File Reference	789
10.17.1	Detailed Description	790
10.18	NvInferRuntimeCommon.h	790
10.19	NvInferRuntimePlugin.h File Reference	791
10.19.1	Detailed Description	792
10.20	NvInferRuntimePlugin.h	792
10.21	NvInferSafeRuntime.h File Reference	795
10.21.1	Detailed Description	797
10.21.2	Macro Definition Documentation	797
10.21.2.1	REGISTER_SAFE_TENSORRT_PLUGIN	797
10.22	NvInferSafeRuntime.h	797
10.23	NvInferVersion.h File Reference	800

10.23.1 Detailed Description	801
10.23.2 Macro Definition Documentation	801
10.23.2.1 NV_TENSORRT_BUILD	801
10.23.2.2 NV_TENSORRT_LWS_MAJOR	801
10.23.2.3 NV_TENSORRT_LWS_MINOR	801
10.23.2.4 NV_TENSORRT_LWS_PATCH	801
10.23.2.5 NV_TENSORRT_MAJOR	802
10.23.2.6 NV_TENSORRT_MINOR	802
10.23.2.7 NV_TENSORRT_PATCH	802
10.23.2.8 NV_TENSORRT_RELEASE_TYPE	802
10.23.2.9 NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS	802
10.23.2.10 NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY	802
10.23.2.11 NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE	803
10.23.2.12 NV_TENSORRT_SONAME_MAJOR	803
10.23.2.13 NV_TENSORRT_SONAME_MINOR	803
10.23.2.14 NV_TENSORRT_SONAME_PATCH	803
10.24 NvInferVersion.h	803
10.25 NvOnnxConfig.h File Reference	804
10.25.1 Detailed Description	804
10.26 NvOnnxConfig.h	805
10.27 NvUffParser.h File Reference	805
10.27.1 Detailed Description	806
10.27.2 Macro Definition Documentation	806
10.27.2.1 UFF_REQUIRED_VERSION_MAJOR	807
10.27.2.2 UFF_REQUIRED_VERSION_MINOR	807
10.27.2.3 UFF_REQUIRED_VERSION_PATCH	807
10.28 NvUffParser.h	807
10.29 NvUtils.h File Reference	808
10.29.1 Detailed Description	809
10.30 NvUtils.h	809
10.31 NvOnnxParser.h File Reference	809
10.31.1 Detailed Description	811
10.31.2 Macro Definition Documentation	811
10.31.2.1 NV_ONNX_PARSER_MAJOR	811
10.31.2.2 NV_ONNX_PARSER_MINOR	811
10.31.2.3 NV_ONNX_PARSER_PATCH	811
10.31.3 Typedef Documentation	811
10.31.3.1 SubGraph_t	811
10.31.3.2 SubGraphCollection_t	811
10.31.4 Function Documentation	812
10.31.4.1 createNvOnnxParser_INTERNAL()	812
10.31.4.2 getNvOnnxParserVersion()	812
10.32 NvOnnxParser.h	812
Index	815

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	789
NvInferSafeRuntime.h	795
NvInferVersion.h	800

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::IAlgorithmIOInfo::getTensorFormat](#) () const noexcept

Deprecated in TensorRT 8.6. The strides, data type, and vectorization information is sufficient to uniquely identify tensor formats.

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, OptProfileSelector 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::kENABLE_TACTIC_HEURISTIC](#)

Superseded by builder optimization level 2. Deprecated in TensorRT 8.6

Member [nvinfer1::kFASTER_DYNAMIC_SHAPES_0805](#)

Turning it off is deprecated in TensorRT 8.6. The flag `kFASTER_DYNAMIC_SHAPES_0805` will be removed in 9.0.

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

Legacy constructor, retained for ABI compatibility. Deprecated in TensorRT 8.6. Use the default constructor instead.

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	29
nvinfer1	
The TensorRT API version 1 namespace	30
nvinfer1::consistency	83
nvinfer1::impl	83
nvinfer1::plugin	84
nvinfer1::safe	
The safety subset of TensorRT's API version 1 namespace	85
nvinfer1::utils	87
nvonnxparser	
The TensorRT ONNX parser API namespace	90
nvuffparser	
The TensorRT UFF parser API namespace	93

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

<code>nvinfer1::plugin::DetectionOutputParameters</code>	97
<code>Dims</code>	100
<code>nvinfer1::Dims32</code>	103
<code>nvinfer1::Dims2</code>	101
<code>nvinfer1::Dims3</code>	102
<code>nvinfer1::Dims4</code>	105
<code>nvinfer1::DimsHW</code>	107
<code>nvinfer1::DimsExprs</code>	106
<code>nvinfer1::DynamicPluginTensorDesc</code>	109
<code>nvinfer1::impl::EnumMaxImpl< T ></code>	111
<code>nvinfer1::impl::EnumMaxImpl< ActivationType ></code>	111
<code>nvinfer1::impl::EnumMaxImpl< AllocatorFlag ></code>	112
<code>nvinfer1::impl::EnumMaxImpl< DataType ></code>	112
<code>nvinfer1::impl::EnumMaxImpl< ElementWiseOperation ></code>	113
<code>nvinfer1::impl::EnumMaxImpl< EngineCapability ></code>	114
<code>nvinfer1::impl::EnumMaxImpl< ErrorCode ></code>	115
<code>nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel ></code>	115
<code>nvinfer1::impl::EnumMaxImpl< ILogger::Severity ></code>	116
<code>nvinfer1::impl::EnumMaxImpl< InterpolationMode ></code>	117
<code>nvinfer1::impl::EnumMaxImpl< PaddingMode ></code>	118
<code>nvinfer1::impl::EnumMaxImpl< PoolingType ></code>	118
<code>nvinfer1::impl::EnumMaxImpl< PreviewFeature ></code>	119
<code>nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation ></code>	120
<code>nvinfer1::impl::EnumMaxImpl< ResizeRoundMode ></code>	120
<code>nvinfer1::impl::EnumMaxImpl< ResizeSelector ></code>	121
<code>nvinfer1::impl::EnumMaxImpl< TensorFormat ></code>	122
<code>nvinfer1::impl::EnumMaxImpl< TensorIOMode ></code>	123
<code>nvinfer1::impl::EnumMaxImpl< TensorLocation ></code>	123
<code>nvuffparser::FieldCollection</code>	124
<code>nvuffparser::FieldMap</code>	125
<code>nvinfer1::plugin::GridAnchorParameters</code>	127

nvinfer1::IAlgorithmSelector	141
nvcaffeparser1::IBinaryProtoBlob	147
nvcaffeparser1::IBlobNameToTensor	149
nvcaffeparser1::ICaffeParser	185
nvinfer1::consistency::IConsistencyChecker	195
nvinfer1::safe::ICudaEngine	242
nvinfer1::ErrorRecorder	286
nvinfer1::safe::IExecutionContext	323
nvinfer1::IGpuAllocator	356
nvinfer1::IInt8Calibrator	375
nvinfer1::IInt8EntropyCalibrator	378
nvinfer1::IInt8EntropyCalibrator2	379
nvinfer1::IInt8LegacyCalibrator	380
nvinfer1::IInt8MinMaxCalibrator	383
nvinfer1::ILogger	396
nvinfer1::ILoggerFinder	398
nvinfer1::INoCopy	464
nvinfer1::IAlgorithm	133
nvinfer1::IAlgorithmContext	136
nvinfer1::IAlgorithmIOInfo	138
nvinfer1::IAlgorithmVariant	143
nvinfer1::IBuilder	150
nvinfer1::IBuilderConfig	160
nvinfer1::ICudaEngine	213
nvinfer1::IDimensionExpr	275
nvinfer1::IEngineInspector	281
nvinfer1::IExecutionContext	293
nvinfer1::IExprBuilder	338
nvinfer1::IHostMemory	364
nvinfer1::IIfConditional	368
nvinfer1::ILayer	386
nvinfer1::IActivationLayer	129
nvinfer1::IAssertionLayer	145
nvinfer1::ICastLayer	190
nvinfer1::IConcatenationLayer	192
nvinfer1::IConstantLayer	198
nvinfer1::IConvolutionLayer	200
nvinfer1::IDeconvolutionLayer	261
nvinfer1::IDequantizeLayer	272
nvinfer1::IEinsumLayer	277
nvinfer1::IElementWiseLayer	279
nvinfer1::IFillLayer	340
nvinfer1::IFullyConnectedLayer	346
nvinfer1::IGatherLayer	351
nvinfer1::IGridSampleLayer	360
nvinfer1::IIdentityLayer	366
nvinfer1::IIfConditionalBoundaryLayer	371
nvinfer1::IConditionLayer	194
nvinfer1::IIfConditionalInputLayer	373
nvinfer1::IIfConditionalOutputLayer	374
nvinfer1::ILRNLayer	407
nvinfer1::ILoopBoundaryLayer	403
nvinfer1::IIteratorLayer	384
nvinfer1::ILoopOutputLayer	404

nvinfer1::IRecurrenceLayer	572
nvinfer1::ITripLimitLayer	664
nvinfer1::IMatrixMultiplyLayer	412
nvinfer1::INMSLayer	459
nvinfer1::INonZeroLayer	466
nvinfer1::INormalizationLayer	467
nvinfer1::IOneHotLayer	472
nvinfer1::IPaddingLayer	489
nvinfer1::IParametricReLULayer	493
nvinfer1::IPluginV2Layer	554
nvinfer1::IPoolingLayer	556
nvinfer1::IQuantizeLayer	568
nvinfer1::IRNNv2Layer	599
nvinfer1::IRaggedSoftMaxLayer	571
nvinfer1::IReduceLayer	574
nvinfer1::IResizeLayer	586
nvinfer1::IReverseSequenceLayer	596
nvinfer1::IScaleLayer	622
nvinfer1::IScatterLayer	627
nvinfer1::ISelectLayer	630
nvinfer1::IShapeLayer	631
nvinfer1::IShuffleLayer	633
nvinfer1::ISliceLayer	638
nvinfer1::ISoftMaxLayer	644
nvinfer1::ITopKLayer	661
nvinfer1::UnaryLayer	670
nvinfer1::ILoop	400
nvinfer1::INetworkDefinition	414
nvinfer1::IOptimizationProfile	481
nvinfer1::IRefitter	577
nvinfer1::IRuntime	607
nvinfer1::ITensor	646
nvinfer1::ITimingCache	658
INonZero	466
nvonnxparser::IONnxConfig	474
nvinfer1::IOutputAllocator	487
nvonnxparser::IParser	495
nvonnxparser::IParserError	502
nvinfer1::IPluginCreator	506
nvinfer1::consistency::IPluginChecker	504
nvcaffeparser1::IPluginFactoryV2	512
nvinfer1::IPluginRegistry	513
nvinfer1::safe::IPluginRegistry	519
nvinfer1::IPluginV2	525
nvinfer1::IPluginV2Ext	543
nvinfer1::IPluginV2DynamicExt	537
nvinfer1::IPluginV2IOExt	551
nvinfer1::IProfiler	566
nvinfer1::safe::IRuntime	617
nvuffparser::IUFFParser	666
nvinfer1::plugin::NMSPParameters	672
nvinfer1::Permutation	674
nvinfer1::PluginField	675

<code>nvinfer1::PluginFieldCollection</code>	677
<code>nvinfer1::PluginRegistrar< T ></code>	678
<code>nvinfer1::safe::PluginRegistrar< T ></code>	678
<code>nvinfer1::PluginTensorDesc</code>	679
<code>PluginVersion</code>	681
<code>nvinfer1::plugin::PriorBoxParameters</code>	681
<code>nvinfer1::plugin::Quadruple</code>	684
<code>nvinfer1::plugin::RegionParameters</code>	685
<code>nvinfer1::plugin::RPROIParams</code>	686
<code>nvinfer1::safe::RuntimeErrorInformation</code>	688
<code>nvinfer1::plugin::softmaxTree</code>	689
<code>nvinfer1::Weights</code>	691

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:	97
Dims	
Structure to define the dimensions of a tensor	100
nvinfer1::Dims2	
Descriptor for two-dimensional data	101
nvinfer1::Dims3	
Descriptor for three-dimensional data	102
nvinfer1::Dims32	103
nvinfer1::Dims4	
Descriptor for four-dimensional data	105
nvinfer1::DimsExprs	106
nvinfer1::DimsHW	
Descriptor for two-dimensional spatial data	107
nvinfer1::DynamicPluginTensorDesc	109
nvinfer1::impl::EnumMaxImpl< T >	
Declaration of EnumMaxImpl struct to store maximum number of elements in an enumeration type	111
nvinfer1::impl::EnumMaxImpl< ActivationType >	111
nvinfer1::impl::EnumMaxImpl< AllocatorFlag >	
Maximum number of elements in AllocatorFlag enum	112
nvinfer1::impl::EnumMaxImpl< DataType >	
Maximum number of elements in DataType enum	112
nvinfer1::impl::EnumMaxImpl< ElementWiseOperation >	113
nvinfer1::impl::EnumMaxImpl< EngineCapability >	
Maximum number of elements in EngineCapability enum	114
nvinfer1::impl::EnumMaxImpl< ErrorCode >	
Maximum number of elements in ErrorCode enum	115

nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel >	115
nvinfer1::impl::EnumMaxImpl< ILogger::Severity >	
Maximum number of elements in ILogger::Severity enum	116
nvinfer1::impl::EnumMaxImpl< InterpolationMode >	117
nvinfer1::impl::EnumMaxImpl< PaddingMode >	118
nvinfer1::impl::EnumMaxImpl< PoolingType >	118
nvinfer1::impl::EnumMaxImpl< PreviewFeature >	119
nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation >	120
nvinfer1::impl::EnumMaxImpl< ResizeRoundMode >	120
nvinfer1::impl::EnumMaxImpl< ResizeSelector >	121
nvinfer1::impl::EnumMaxImpl< TensorFormat >	
Maximum number of elements in TensorFormat enum	122
nvinfer1::impl::EnumMaxImpl< TensorIOMode >	
Maximum number of elements in TensorIOMode enum	123
nvinfer1::impl::EnumMaxImpl< TensorLocation >	
Maximum number of elements in TensorLocation enum	123
nvuffparser::FieldCollection	124
nvuffparser::FieldMap	
An array of field params used as a layer parameter for plugin layers	125
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:	127
nvinfer1::IActivationLayer	
An Activation layer in a network definition	129
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() .”	133
nvinfer1::IAlgorithmContext	
Describes the context and requirements, that could be fulfilled by one or more instances of IAlgorithm	136
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()	138
nvinfer1::IAlgorithmSelector	
Interface implemented by application for selecting and reporting algorithms of a layer provided by the builder	141
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()	143
nvinfer1::IAssertionLayer	
An assertion layer in a network	145
nvcaffeparser1::IBinaryProtoBlob	
Object used to store and query data extracted from a binaryproto file using the ICaffeParser	147
nvcaffeparser1::IBlobNameToTensor	
Object used to store and query Tensors after they have been extracted from a Caffe model using the ICaffeParser	149
nvinfer1::IBuilder	
Builds an engine from a network definition	150
nvinfer1::IBuilderConfig	
Holds properties for configuring a builder to produce an engine	160

nvcaffeparser1::ICaffeParser	
Class used for parsing Caffe models	185
nvinfer1::ICastLayer	
A cast layer in a network	190
nvinfer1::IConcatenationLayer	
A concatenation layer in a network definition	192
nvinfer1::IConditionLayer	
.	194
nvinfer1::consistency::IConsistencyChecker	
Validates a serialized engine blob	195
nvinfer1::IConstantLayer	
Layer that represents a constant value	198
nvinfer1::IConvolutionLayer	
A convolution layer in a network definition	200
nvinfer1::ICudaEngine	
An engine for executing inference on a built network, with functionally unsafe features	213
nvinfer1::safe::ICudaEngine	
A functionally safe engine for executing inference on a built network	242
nvinfer1::IDeconvolutionLayer	
A deconvolution layer in a network definition	261
nvinfer1::IDequantizeLayer	
A Dequantize layer in a network definition	272
nvinfer1::IDimensionExpr	
.	275
nvinfer1::IEinsumLayer	
An Einsum layer in a network	277
nvinfer1::IElementWiseLayer	
A elementwise layer in a network definition	279
nvinfer1::IEngineInspector	
An engine inspector which prints out the layer information of an engine or an execution context	281
nvinfer1::IErrorRecorder	
Reference counted application-implemented error reporting interface for TensorRT objects	286
nvinfer1::IExecutionContext	
Context for executing inference using an engine, with functionally unsafe features	293
nvinfer1::safe::IExecutionContext	
Functionally safe context for executing inference using an engine	323
nvinfer1::IExprBuilder	
.	338
nvinfer1::IFillLayer	
Generate an output tensor with specified mode	340
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:	346
nvinfer1::IGatherLayer	
A Gather layer in a network definition. Supports several kinds of gathering	351
nvinfer1::IGpuAllocator	
Application-implemented class for controlling allocation on the GPU	356
nvinfer1::IGridSampleLayer	
A GridSample layer in a network definition	360
nvinfer1::IHostMemory	
Class to handle library allocated memory that is accessible to the user	364
nvinfer1::IIdentityLayer	
A layer that represents the identity function	366
nvinfer1::IIfConditional	
.	368
nvinfer1::IIfConditionalBoundaryLayer	
.	371

nvinfer1::IfConditionalInputLayer	373
nvinfer1::IfConditionalOutputLayer	374
nvinfer1::Int8Calibrator	
Application-implemented interface for calibration	375
nvinfer1::Int8EntropyCalibrator	378
nvinfer1::Int8EntropyCalibrator2	379
nvinfer1::Int8LegacyCalibrator	380
nvinfer1::Int8MinMaxCalibrator	383
nvinfer1::IteratorLayer	384
nvinfer1::ILayer	
Base class for all layer classes in a network definition	386
nvinfer1::ILogger	
Application-implemented logging interface for the builder, refitter and runtime	396
nvinfer1::ILoggerFinder	
A virtual base class to find a logger. Allows a plugin to find an instance of a logger if it needs to emit a log message. A pointer to an instance of this class is passed to a plugin shared library on initialization when that plugin is serialized as part of a version-compatible plan. See the plugin chapter in the developer guide for details	398
nvinfer1::ILoop	400
nvinfer1::ILoopBoundaryLayer	403
nvinfer1::ILoopOutputLayer	404
nvinfer1::ILRNLayer	
A LRN layer in a network definition	407
nvinfer1::IMatrixMultiplyLayer	
Layer that represents a Matrix Multiplication	412
nvinfer1::INetworkDefinition	
A network definition for input to the builder	414
nvinfer1::INMSLayer	
A non-maximum suppression layer in a network definition	459
nvinfer1::INoCopy	
Forward declaration of IEngineInspector for use by other interfaces	464
INonZero	
A NonZero layer in a network	466
nvinfer1::INonZeroLayer	466
nvinfer1::INormalizationLayer	
A normalization layer in a network definition	467
nvinfer1::IOneHotLayer	
A OneHot layer in a network definition	472
nvonnxparser::IONnxConfig	
Configuration Manager Class	474
nvinfer1::IOptimizationProfile	
Optimization profile for dynamic input dimensions and shape tensors	481
nvinfer1::IOutputAllocator	
Callback from <code>ExecutionContext::enqueueV3()</code>	487
nvinfer1::IPaddingLayer	
Layer that represents a padding operation	489
nvinfer1::IParametricReLULayer	
Layer that represents a parametric ReLU operation	493
nvonnxparser::IParser	
Object for parsing ONNX models into a TensorRT network definition	495
nvonnxparser::IParserError	
Object containing information about an error	502
nvinfer1::consistency::IPluginChecker	
Consistency Checker plugin class for user implemented Plugins	504

nvinfer1::IPluginCreator	506
Plugin creator class for user implemented layers	
nvcaffeparser1::IPluginFactoryV2	512
Plugin factory used to configure plugins	
nvinfer1::IPluginRegistry	513
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	
nvinfer1::safe::IPluginRegistry	519
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 must also have a corresponding IPluginCreator implementation	
nvinfer1::IPluginV2	525
Plugin class for user-implemented layers	
nvinfer1::IPluginV2DynamicExt	537
nvinfer1::IPluginV2Ext	543
Plugin class for user-implemented layers	
nvinfer1::IPluginV2IOExt	551
Plugin class for user-implemented layers	
nvinfer1::IPluginV2Layer	554
Layer type for pluginV2	
nvinfer1::IPoolingLayer	556
A Pooling layer in a network definition	
nvinfer1::IProfiler	566
Application-implemented interface for profiling	
nvinfer1::IQuantizeLayer	568
A Quantize layer in a network definition	
nvinfer1::IRaggedSoftMaxLayer	571
A RaggedSoftmax layer in a network definition	
nvinfer1::IRecurrenceLayer	572
nvinfer1::IReduceLayer	574
Layer that represents a reduction across a non-bool tensor	
nvinfer1::IRefitter	577
Updates weights in an engine	
nvinfer1::IResizeLayer	586
A resize layer in a network definition	
nvinfer1::IReverseSequenceLayer	596
A ReverseSequence layer in a network definition	
nvinfer1::IRNNv2Layer	599
An RNN layer in a network definition, version 2	
nvinfer1::IRuntime	607
Allows a serialized functionally unsafe engine to be deserialized	
nvinfer1::safe::IRuntime	617
Allows a serialized functionally safe engine to be deserialized	
nvinfer1::IScaleLayer	622
A Scale layer in a network definition	
nvinfer1::IScatterLayer	627
A scatter layer in a network definition. Supports several kinds of scattering	
nvinfer1::ISelectLayer	630

nvinfer1::IShapeLayer	
Layer type for getting shape of a tensor	631
nvinfer1::IShuffleLayer	
Layer type for shuffling data	633
nvinfer1::ISliceLayer	
Slices an input tensor into an output tensor based on the offset and strides	638
nvinfer1::ISoftMaxLayer	
A Softmax layer in a network definition	644
nvinfer1::ITensor	
A tensor in a network definition	646
nvinfer1::ITimingCache	
Class to handle tactic timing info collected from builder	658
nvinfer1::ITopKLayer	
Layer that represents a TopK reduction	661
nvinfer1::ITripLimitLayer	664
nvuffparser::IUffParser	
Class used for parsing models described using the UFF format	666
nvinfer1::IUnaryLayer	
Layer that represents an unary operation	670
nvinfer1::plugin::NMSPParameters	
The NMSPParameters are used by the BatchedNMSPPlugin for performing the non_max_suppression operation over boxes for object detection networks	672
nvinfer1::Permutation	674
nvinfer1::PluginField	
Structure containing plugin attribute field names and associated data This information can be parsed to decode necessary plugin metadata	675
nvinfer1::PluginFieldCollection	
Plugin field collection struct	677
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	678
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	678
nvinfer1::PluginTensorDesc	
Fields that a plugin might see for an input or output	679
PluginVersion	
Definition of plugin versions	681
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:	681
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	684
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	685

nvinfer1::plugin::RPROIParams	
RPROIParams is used to create the RPROIPlugin instance. It contains:	686
nvinfer1::safe::RuntimeErrorInformation	
Space to record information about runtime errors	688
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	689
nvinfer1::Weights	
An array of weights used as a layer parameter	691

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

NvCaffeParser.h	693
NvInfer.h	695
NvInferConsistency.h	748
NvInferLegacyDims.h	750
NvInferPlugin.h	752
NvInferPluginUtils.h	759
NvInferRuntime.h	762
NvInferRuntimeBase.h	782
NvInferRuntimeCommon.h	789
NvInferRuntimePlugin.h	791
NvInferSafeRuntime.h	795
NvInferVersion.h	800
NvOnnxConfig.h	804
NvUffParser.h	805
NvUtils.h	808
NvOnnxParser.h	809

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 [ILoggerFinder](#)
A virtual base class to find a logger. Allows a plugin to find an instance of a logger if it needs to emit a log message. A pointer to an instance of this class is passed to a plugin shared library on initialization when that plugin is serialized as part of a version-compatible plan. See the plugin chapter in the developer guide for details.

- class [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 [IReverseSequenceLayer](#)
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 BuilderFlag 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 [AllocatorFlags](#) = uint32_t
- using [PluginFormat](#) = [TensorFormat](#)
PluginFormat is reserved for backward compatibility.

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](#) , [kTRANSPOSE](#) , [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 { [kLinspace](#) = 0 , [kRANDOM_UNIFORM](#) = 1 , [kRANDOM_NORMAL](#) = 2 }

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

- enum class [ScatterMode](#) : int32_t { [kELEMENT](#) = 0 , [kND](#) = 1 }

Control form of IScatterLayer.

- enum class [BoundingBoxFormat](#) : int32_t { [kCORNER_PAIRS](#) = 0 , [kCENTER_SIZES](#) = 1 }

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

- enum class [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 }

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 [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.

- 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 }

The possible field types for custom layer.

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 building a Standard engine, or nullptr if no registry exists.
- [nvinfer1::safe::IPluginRegistry](#) * [getBuilderSafePluginRegistry](#) ([nvinfer1::EngineCapability](#) capability) noexcept
Return the plugin registry for building a Safety engine, 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 widely used in automotive safety context.

8.2.2.3 BuilderFlags

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

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

See also

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

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

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	Legacy flag with effect similar to setting all of these three flags: * kPREFER_PRECISION_CONSTRAINTS * kDIRECT_IO * kREJECT_EMPTY_ALGORITHMS 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. \deprecated Deprecatd in TensorRT 8.2.
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	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. This flag is only supported by NVIDIA Ampere and later GPUs. Deprecated Superseded by builder optimization level 2. Deprecated in TensorRT 8.6
kVERSION_COMPATIBLE	Restrict to lean runtime operators to provide version forward compatibility for the plan. 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.

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. Warning kFP8 is not supported yet and will result in an error or undefined behavior.

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.

Enumerator

kINT8	Signed 8-bit integer representing a quantized floating-point value.
kINT32	Signed 32-bit integer format.
kBOOL	8-bit boolean. 0 = false, 1 = true, other values undefined.
kUINT8	Unsigned 8-bit integer format. Cannot be used to represent quantized floating-point values. Use the IdentityLayer to convert kUINT8 network-level inputs to {kFLOAT, kHALF} prior to use with other TensorRT layers, or to convert intermediate output before kUINT8 network-level outputs from {kFLOAT, kHALF} to kUINT8. kUINT8 conversions are only supported for {kFLOAT, kHALF}. kUINT8 to {kFLOAT, kHALF} conversion will convert the integer values to equivalent floating point values. {kFLOAT, kHALF} to kUINT8 conversion will convert the floating point values to integer values by truncating towards zero. This conversion has undefined behavior for floating point values outside the range [0.0F, 256.0F) after truncation. kUINT8 conversions are not supported for {kINT8, kINT32, kBOOL}.
kFP8	Signed 8-bit floating point with 1 sign bit, 4 exponent bits, 3 mantissa bits, and exponent-bias 7. Warning kFP8 is not supported yet and will result in an error or undefined behavior.

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.
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](#), [DataType::kHALF](#), or [DataType::kINT8](#).

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

Enumerator

kUNSUPPORTED_STATE	An error occurred due to the network not being supported on the device due to constraints of the hardware or system. An example is running an unsafe layer in a safety certified context, or a resource requirement for the current network is greater than the capabilities of the target device. The network is otherwise correct, but the network and hardware combination is problematic. This can be recoverable. Examples: • Scratch space requests larger than available device memory and can be recovered by increasing allowed workspace size. • Tensor size exceeds the maximum element count and can be recovered by reducing the maximum batch size.

8.2.3.12 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.
kRANDOM_UNIFORM	Generate a tensor with random values drawn from a uniform distribution.
kRANDOM_NORMAL	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. This option will disable cuDNN, cuBLAS, and cuBLAS LT as tactic sources.

8.2.3.15 InterpolationMode

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

Enumerates various modes of interpolation.

Enumerator

kNEAREST	ND ($0 < N \leq 8$) 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.

Enumerator

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.
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.

Enumerator

<code>kDLA_LOCAL_DRAM</code>	<code>kDLA_LOCAL_DRAM</code> 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.
<code>kDLA_GLOBAL_DRAM</code>	<code>kDLA_GLOBAL_DRAM</code> 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.
<code>kTACTIC_DRAM</code>	<code>kTACTIC_DRAM</code> is the device DRAM used by the optimizer to run tactics. On embedded devices, where host and device memory are unified, this includes all host memory required by TensorRT to build the network up to the point of each memory allocation. This defaults to 75% of <code>totalGlobalMem</code> as reported by <code>cudaGetDeviceProperties</code> when <code>cudaGetDeviceProperties.embedded</code> 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

<code>kEXPLICIT_BATCH</code>	Mark the network to be an explicit batch network. Dynamic shape support requires that the <code>kEXPLICIT_BATCH</code> 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.
<code>kEXPLICIT_PRECISION</code>	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 = ceil(I / S) = {3, 3}
P = floor((I - 1) / S) * S + F - I
  ==> floor((({6, 6} - {1, 1}) / {2, 2}) * {2, 2} + {3, 3} - {6, 6})
  ==> {4, 4} + {3, 3} - {6, 6}
  ==> {1, 1}
B = floor({1, 1} / {2, 2})
  ==> {0, 0}
A = {1, 1} - {0, 0}
  ==> {1, 1}
```

- **SAME_LOWER:**

Computation:

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

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

Pooling Example 2:

Given $I = \{6, 6\}$, $B = \{3, 3\}$, $A = \{3, 3\}$, $S = \{2, 2\}$, $F = \{3, 3\}$. What is O?

- **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_layer.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]
```

The possible field types for custom layer.

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. The default value for this flag is on. Deprecated Turning it off is deprecated in TensorRT 8.6. The flag kFASTER_DYNAMIC_SHAPES_0805 will be removed in 9.0.
kDISABLE_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 IBuilderConfig::setTacticSources(), but cudnnContext and cublasContext handles will still be passed to plugins via IPluginV2Ext::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 on.
TensorRT 8.6.13 API Reference	See also TacticSource

Enumerator

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.

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	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. <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.
```

$\text{ReLU}(X) := \max(X, 0)$
 $\tanh(X) := \text{hyperbolic tangent of } X$
 $\text{sigmoid}(X) := 1 / (1 + \exp(-X))$
 $\exp(X) := e^X$
 $A.B$ 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:

```

::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	

Enumerator

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_TACTIC_SOURCES_FOR_CORE_0805](#)

Enumerator

<code>kCUBLAS</code>	cuBLAS tactics. Enabled by default. Note Disabling kCUBLAS will cause the cuBLAS handle passed to plugins in <code>attachToContext</code> to be null. Setting kCUBLAS tactic source takes no effect for core library if PreviewFeature::kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805 is on.
<code>kCUBLAS_LT</code>	cuBLAS LT tactics. Enabled by default. Note Setting kCUBLAS_LT tactic source takes no effect for core library if PreviewFeature::kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805 is on.
<code>kCUDNN</code>	cuDNN tactics. Enabled by default. Note Disabling kCUDNN will cause the cuDNN handle passed to plugins in <code>attachToContext</code> to be null. Setting kCUDNN tactic source takes no effect for core library if PreviewFeature::kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805 is on.
<code>kEDGE_MASK_CONVOLUTIONS</code>	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.
<code>kJIT_CONVOLUTIONS</code>	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 {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].

Enumerator

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 less than or equal to 4. If used as DLA input and the build option kGPU_FALLBACK is not specified, it needs to meet line stride requirement of DLA format. Column stride in bytes must be a multiple of 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 $[n][c][h][w]$.
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 $[n][h][w][c]$.

Enumerator

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](#), [DataType::kHALF](#), [DataType::kINT8](#), or [DataType::kINT32](#).

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

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

Operations kSIGN and kROUND are not supported in implicit batch mode.

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.

Enumerator

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 building a Standard engine, or nullptr if no registry exists.

Also return nullptr if the input argument is not [EngineCapability::kSTANDARD](#). Engine capabilities [EngineCapability::kSTANDARD](#) and [EngineCapability::kSAFETY](#) have distinct plugin registries. When building a Safety engine, use [nvinfer1::getBuilderSafePluginRegistry\(\)](#). 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](#).

8.2.4.33 getBuilderSafePluginRegistry()

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

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

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

8.3 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

- 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 [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 must also have a corresponding [IPluginCreator](#) implementation.
- 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.
- struct [RuntimeErrorInformation](#)
Space to record information about runtime errors.

Enumerations

- enum class [RuntimeErrorType](#) : uint64_t {
[kNAN_CONSUMED](#) = 1ULL << 0 , [kINF_CONSUMED](#) = 1ULL << 1 , [kGATHER_OOB](#) = 1ULL << 2 ,
[kSCATTER_OOB](#) = 1ULL << 3 ,
[kSCATTER_RACE](#) = 1ULL << 4 }

Functions

- [IRuntime](#) * [createInferRuntime](#) ([ILogger](#) &logger) noexcept
Create an instance of a [safe::IRuntime](#) class.
- [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 Enumeration Type Documentation

8.6.2.1 RuntimeErrorType

```
enum class nvinfer1::safe::RuntimeErrorType : uint64_t [strong]
```

Enumerator

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

8.6.3 Function Documentation

8.6.3.1 createInferRuntime()

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

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

Parameters

<i>logger</i>	A logger object whose lifetime must exceed that of the returned runtime. Loggers must be thread-safe.

Returns

A safe runtime object that can be used for safe plan file deserialization.

This class is the logging class for the runtime.

Usage considerations

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

8.6.3.2 getSafePluginRegistry()

```
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) noex-
cept
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 fco 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

Typedefs

- using [OnnxParserFlags](#) = uint32_t
Represents one or more OnnxParserFlag values using binary OR operations, e.g., 1U << [OnnxParserFlag::kNATIVE_INSTANCENORM](#).

Enumerations

- enum class [ErrorCode](#) : int {
[kSUCCESS](#) = 0 , [kINTERNAL_ERROR](#) = 1 , [kMEM_ALLOC_FAILED](#) = 2 , [kMODEL_DESERIALIZE_FAILED](#) = 3 ,
[kINVALID_VALUE](#) = 4 , [kINVALID_GRAPH](#) = 5 , [kINVALID_NODE](#) = 6 , [kUNSUPPORTED_GRAPH](#) = 7 ,
[kUNSUPPORTED_NODE](#) = 8 }
The type of error that the parser may return.
- enum class [OnnxParserFlag](#) : int32_t { [kNATIVE_INSTANCENORM](#) = 0 }

Functions

- [IOnnxConfig](#) * [createONNXConfig](#) ()
- [template](#)<typename T >
constexpr int32_t [EnumMax](#) ()
- [template](#)<> constexpr int32_t [EnumMax](#)< [ErrorCode](#) > ()
- [template](#)<> constexpr int32_t [EnumMax](#)< [OnnxParserFlag](#) > ()

8.8.1 Detailed Description

The TensorRT ONNX parser API namespace.

8.8.2 Typedef Documentation

8.8.2.1 OnnxParserFlags

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

Represents one or more OnnxParserFlag values using binary OR operations, e.g., 1U << [OnnxParserFlag::kNATIVE_INSTANCENORM](#)

See also

[IParser::setFlags\(\)](#) and [IParser::getFlags\(\)](#)

8.8.3 Enumeration Type Documentation

8.8.3.1 ErrorCode

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

The type of error that the parser may return.

Enumerator

kSUCCESS	
kINTERNAL_ERROR	
kMEM_ALLOC_FAILED	
kMODEL_DESERIALIZE_FAILED	
kINVALID_VALUE	
kINVALID_GRAPH	
kINVALID_NODE	
kUNSUPPORTED_GRAPH	
kUNSUPPORTED_NODE	

8.8.3.2 OnnxParserFlag

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

Enumerator

KNATIVE_INSTANCENORM	Parse the ONNX model into the INetworkDefinition with the intention of using TensorRT's native layer implementation over the plugin implementation for InstanceNormalization nodes. This flag is planned to be deprecated in TensorRT 8.7 and removed in TensorRT 9.0. This flag is required when building version-compatible or hardware-compatible engines. There may be performance degradations when this flag is enabled.

8.8.4 Function Documentation

8.8.4.1 createONNXConfig()

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

8.8.4.2 EnumMax()

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

8.8.4.3 EnumMax< ErrorCode >()

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

Maximum number of flags in the ErrorCode enum.

See also

[ErrorCode](#)

8.8.4.4 EnumMax< OnnxParserFlag >()

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

Maximum number of flags in the OnnxParserFlag enum.

See also

[OnnxParserFlag](#)

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 <NvInferRuntimeBase.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:

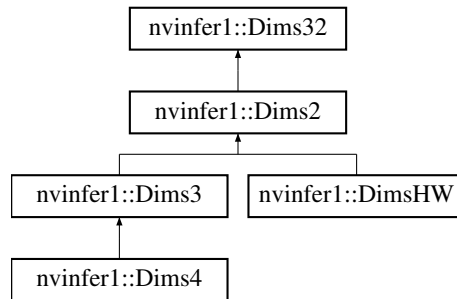
- [NvInferRuntimeBase.h](#)

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:

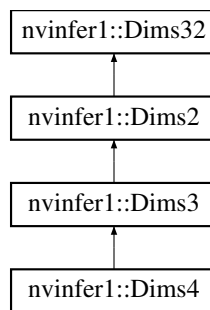
- [NvInferLegacyDims.h](#)

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.

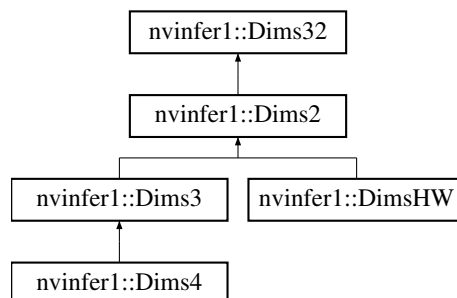
The documentation for this class was generated from the following file:

- [NvInferLegacyDims.h](#)

9.5 nvinfer1::Dims32 Class Reference

```
#include <NvInferRuntimeBase.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:

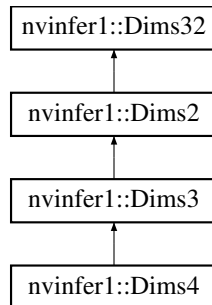
- [NvInferRuntimeBase.h](#)

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:

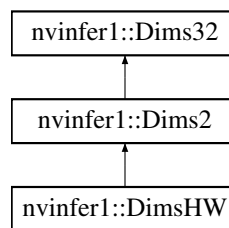
- [NvInferRuntime.h](#)

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:

- [NvInferRuntimeBase.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 <NvInferRuntimeBase.h>
```

Static Public Attributes

- static constexpr int32_t `kVALUE` = 1

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

9.12.1 Detailed Description

Maximum number of elements in `AllocatorFlag` enum.

See also

[AllocatorFlag](#)

9.12.2 Member Data Documentation

9.12.2.1 `kVALUE`

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

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

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

- [NvInferRuntimeBase.h](#)

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

Maximum number of elements in `DataType` enum.

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

Static Public Attributes

- static constexpr int32_t `kVALUE` = 7

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

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]
```

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

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

- [NvInferRuntimeBase.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 <NvInferRuntimeBase.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:

- [NvInferRuntimeBase.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 <NvInferRuntimeBase.h>
```

Static Public Attributes

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

9.18.1 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 the maximum number of elements in the [ILogger::Severity](#) enum.

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

- [NvInferRuntimeBase.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](#) = 3

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 = 3 [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 <NvInferRuntimeBase.h>
```

Static Public Attributes

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

9.26.1 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 the maximum number of elements in the TensorFormat enum.

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

- [NvInferRuntimeBase.h](#)

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

Maximum number of elements in `TensorIOMode` enum.

```
#include <NvInferRuntimeBase.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:

- [NvInferRuntimeBase.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

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

Default constructor.

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() [1/2]

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

Deprecated Legacy constructor, retained for ABI compatibility. Deprecated in TensorRT 8.6. Use the default constructor instead.

9.30.2.2 FieldMap() [2/2]

```
nvuffparser::FieldMap::FieldMap ( ) [default]
```

Default constructor.

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::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]

9.31.1 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.

9.31.2 Member Data Documentation

9.31.2.1 aspectRatios

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

9.31.2.2 H

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

9.31.2.3 maxSize

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

9.31.2.4 minSize

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

9.31.2.5 numAspectRatios

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

9.31.2.6 variance

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

9.31.2.7 W

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

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

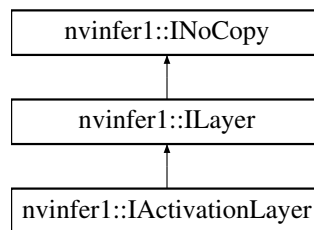
- [NvInferPluginUtils.h](#)

9.32 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`

9.32.1 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.

9.32.2 Constructor & Destructor Documentation

9.32.2.1 `~IActivationLayer()`

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

9.32.3 Member Function Documentation

9.32.3.1 `getActivationType()`

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

Get the type of activation to be performed.

See also

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

9.32.3.2 getAlpha()

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

Get the alpha parameter.

See also

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

9.32.3.3 getBeta()

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

Get the beta parameter.

See also

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

9.32.3.4 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](#)

9.32.3.5 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\(\)](#)

9.32.3.6 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\(\)](#)

9.32.4 Member Data Documentation

9.32.4.1 mImpl

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

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

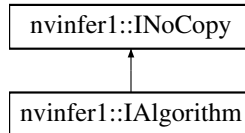
- [NvInfer.h](#)

9.33 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`

9.33.1 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.

9.33.2 Constructor & Destructor Documentation

9.33.2.1 ~IAgorithm()

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

9.33.3 Member Function Documentation

9.33.3.1 getAlgorithmIOInfo()

```
TRT_DEPRECATED IAlgorithmIOInfo const & nvinfer1::IAgorithm::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 [IAgorithm::getAlgorithmIOInfoByIndex\(\)](#).

9.33.3.2 getAlgorithmIOInfoByIndex()

```
IAlgorithmIOInfo const * nvinfer1::IAgorithm::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.

9.33.3.3 getAlgorithmVariant()

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

Returns the algorithm variant.

9.33.3.4 getTimingMSec()

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

The time in milliseconds to execute the algorithm.

9.33.3.5 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.

9.33.4 Member Data Documentation**9.33.4.1 mImpl**

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

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

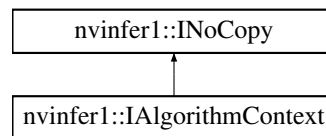
- [NvInfer.h](#)

9.34 nvinfer1::IAgorithmContext Class Reference

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

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

Inheritance diagram for nvinfer1::IAgorithmContext:



Public Member Functions

- `char const * getName () const noexcept`
Return name of the algorithm node. This is a unique identifier for the [IAgorithmContext](#).
- `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 ~IAgorithmContext () noexcept=default`

Protected Attributes

- `apiv::VAlgorithmContext * mImpl`

9.34.1 Detailed Description

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

See also

[IAgorithm](#)

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 ~IAlgorithmContext()

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

9.34.3 Member Function Documentation

9.34.3.1 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.

9.34.3.2 getName()

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

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

9.34.3.3 getNbInputs()

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

Return number of inputs of the algorithm.

9.34.3.4 getNbOutputs()

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

Return number of outputs of the algorithm.

9.34.4 Member Data Documentation

9.34.4.1 mImpl

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

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

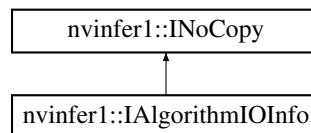
- [NvInfer.h](#)

9.35 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

- [TRT_DEPRECATED 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. For vectorized formats, strides are given in units of vectors.
- [int64_t getVectorizedDim \(\)](#) const noexcept
Return the index of the vectorized dimension or -1 for non-vectorized formats.
- [int64_t getComponentsPerElement \(\)](#) const noexcept
Return the number of components per element. This is always 1 for non-vectorized formats.

Protected Member Functions

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

Protected Attributes

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

9.35.1 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.

9.35.2 Constructor & Destructor Documentation

9.35.2.1 ~IAlgorithmIOInfo()

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

9.35.3 Member Function Documentation

9.35.3.1 getComponentsPerElement()

```
int64_t nvinfer1::IAlgorithmIOInfo::getComponentsPerElement ( ) const [inline], [noexcept]
```

Return the number of components per element. This is always 1 for non-vectorized formats.

Returns

the number of components per element.

9.35.3.2 `getDataType()`

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

Return DataType of the input/output of algorithm.

Returns

the data type.

9.35.3.3 `getStrides()`

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

Return strides of the input/output tensor of algorithm. For vectorized formats, strides are given in units of vectors.

Returns

the strides of the tensor.

9.35.3.4 `getTensorFormat()`

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

Return TensorFormat of the input/output of algorithm.

Deprecated Deprecated in TensorRT 8.6. The strides, data type, and vectorization information is sufficient to uniquely identify tensor formats.

Returns

the tensor format

9.35.3.5 `getVectorizedDim()`

```
int64_t nvinfer1::IAlgorithmIOInfo::getVectorizedDim ( ) const [inline], [noexcept]
```

Return the index of the vectorized dimension or -1 for non-vectorized formats.

Returns

the index of the vectorized dimension.

9.35.4 Member Data Documentation

9.35.4.1 mImpl

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

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

- [NvInfer.h](#)

9.36 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 *algoChoices, int32_t nbAlgorithms) noexcept=0
Called by TensorRT to report choices it made.
- virtual [~IAlgorithmSelector](#) () noexcept=default

9.36.1 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](#).

9.36.2 Constructor & Destructor Documentation

9.36.2.1 ~IAlgorithmSelector()

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

9.36.3 Member Function Documentation

9.36.3.1 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 algoContexts as well as algoChoices.

9.36.3.2 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:

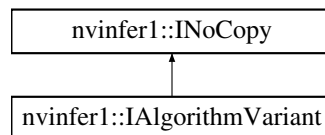
- [NvInfer.h](#)

9.37 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`

9.37.1 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.

9.37.2 Constructor & Destructor Documentation

9.37.2.1 `~IAlgorithmVariant()`

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

9.37.3 Member Function Documentation

9.37.3.1 `getImplementation()`

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

Return implementation of the algorithm.

9.37.3.2 getTactic()

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

Return tactic of the algorithm.

9.37.4 Member Data Documentation

9.37.4.1 mImpl

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

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

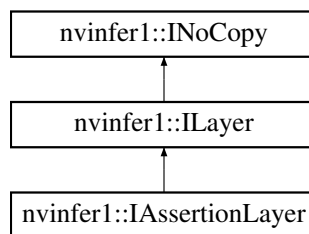
- [NvInfer.h](#)

9.38 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`

9.38.1 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.

9.38.2 Constructor & Destructor Documentation

9.38.2.1 `~IAssertionLayer()`

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

9.38.3 Member Function Documentation

9.38.3.1 `getMessage()`

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

Return the assertion message.

See also

[setMessage\(\)](#)

9.38.3.2 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\(\)](#)

9.38.4 Member Data Documentation

9.38.4.1 mImpl

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

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

- [NvInfer.h](#)

9.39 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

9.39.1 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.

9.39.2 Constructor & Destructor Documentation

9.39.2.1 ~IBinaryProtoBlob()

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

9.39.3 Member Function Documentation

9.39.3.1 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.

9.39.3.2 getData()

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

9.39.3.3 getDataType()

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

9.39.3.4 getDimensions()

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

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

- [NvCaffeParser.h](#)

9.40 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](#) ()

9.40.1 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.

9.40.2 Constructor & Destructor Documentation

9.40.2.1 ~IBlobNameToTensor()

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

9.40.3 Member Function Documentation

9.40.3.1 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:

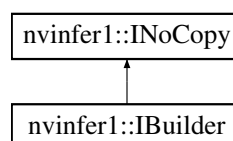
- [NvCaffeParser.h](#)

9.41 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

9.41.1 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.

9.41.2 Constructor & Destructor Documentation

9.41.2.1 ~IBuilder()

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

9.41.3 Member Function Documentation

9.41.3.1 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\(\)](#).

9.41.3.2 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](#)

9.41.3.3 createBuilderConfig()

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

Create a builder configuration object.

See also

[IBuilderConfig](#)

9.41.3.4 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 << NetworkDefinitionCreationFlag::kEXPLICIT_BATCH

See also

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

9.41.3.5 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](#)

9.41.3.6 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.

9.41.3.7 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\(\)](#)

9.41.3.8 getLogger()

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

get the logger with which the builder was created

Returns

the logger

9.41.3.9 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\(\)](#)

9.41.3.10 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.

9.41.3.11 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\(\)](#)

9.41.3.12 getNbDLACores()

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

Return the number of DLA engines available to this builder.

9.41.3.13 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.

9.41.3.14 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.

9.41.3.15 platformHasFastFp16()

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

Determine whether the platform has fast native fp16.

9.41.3.16 platformHasFastInt8()

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

Determine whether the platform has fast native int8.

9.41.3.17 platformHasTf32()

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

Determine whether the platform has TF32 support.

9.41.3.18 reset()

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

Resets the builder state to default values.

9.41.3.19 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\(\)](#)

9.41.3.20 setGpuAllocator()

```
void nvinfer1::IBuilder::setGpuAllocator (
    IGpuAllocator * 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.

9.41.3.21 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\(\)](#)

9.41.3.22 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.

9.41.4 Member Data Documentation

9.41.4.1 mImpl

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

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

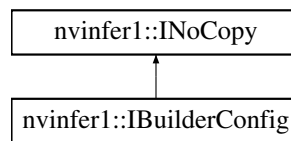
- [NvInfer.h](#)

9.42 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 version-compatible engines.
- char const * [getPluginToSerialize](#) (int32_t index) const noexcept
Get the plugin library path to be serialized with version-compatible engines.
- int32_t [getNbPluginsToSerialize](#) () const noexcept
Get the number of plugin library paths to be serialized with version-compatible engines.
- void [setMaxAuxStreams](#) (int32_t nbStreams) noexcept
Set the maximum number of auxiliary streams that TRT is allowed to use.
- int32_t [getMaxAuxStreams](#) () const noexcept
Get the maximum number of auxiliary streams that TRT is allowed to use.

Protected Attributes

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

Additional Inherited Members

9.42.1 Detailed Description

Holds properties for configuring a builder to produce an engine.

See also

[BuilderFlags](#)

9.42.2 Constructor & Destructor Documentation

9.42.2.1 ~IBuilderConfig()

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

9.42.3 Member Function Documentation

9.42.3.1 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.

9.42.3.2 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.

9.42.3.3 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\(\)](#)

9.42.3.4 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\(\)](#)

9.42.3.5 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

9.42.3.6 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.

9.42.3.7 `getAlgorithmSelector()`

```
IAlgorithmSelector * nvinfer1::IBuilderConfig::getAlgorithmSelector ( ) const [inline], [noexcept]
```

Get Algorithm Selector.

9.42.3.8 `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\(\)`](#)

9.42.3.9 `getBuilderOptimizationLevel()`

```
int32_t nvinfer1::IBuilderConfig::getBuilderOptimizationLevel ( ) [inline], [noexcept]
```

Get builder optimization level.

Returns

the current builder optimization level

See also

[`setBuilderOptimizationLevel`](#)

9.42.3.10 `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.

9.42.3.11 getDefaultDeviceType()

```
DeviceType nvinfer1::IBuilderConfig::getDefaultDeviceType ( ) const [inline], [noexcept]
```

Get the default DeviceType which was set by setDefaultDeviceType.

By default it returns DeviceType::kGPU.

9.42.3.12 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.

9.42.3.13 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.

9.42.3.14 getEngineCapability()

```
EngineCapability nvinfer1::IBuilderConfig::getEngineCapability ( ) const [inline], [noexcept]
```

Query EngineCapability flow configured for the builder.

By default it returns EngineCapability::kSTANDARD.

See also

[setEngineCapability\(\)](#)

9.42.3.15 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.

9.42.3.16 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\(\)](#)

9.42.3.17 getHardwareCompatibilityLevel()

```
HardwareCompatibilityLevel nvinfer1::IBuilderConfig::getHardwareCompatibilityLevel ( ) const [inline],
[noexcept]
```

Get the hardware compatibility level.

Returns

hardwareCompatibilityLevel The level of hardware compatibility.

See also

[setHardwareCompatibilityLevel\(\)](#)

9.42.3.18 getInt8Calibrator()

```
IInt8Calibrator * nvinfer1::IBuilderConfig::getInt8Calibrator ( ) const [inline], [noexcept]
```

Get Int8 Calibration interface.

9.42.3.19 getMaxAuxStreams()

```
int32_t nvinfer1::IBuilderConfig::getMaxAuxStreams ( ) const [inline], [noexcept]
```

Get the maximum number of auxiliary streams that TRT is allowed to use.

See also

[setMaxAuxStreams\(\)](#)

9.42.3.20 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](#)

9.42.3.21 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](#)

9.42.3.22 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\(\)](#).

9.42.3.23 getNbOptimizationProfiles()

```
int32_t nvinfer1::IBuilderConfig::getNbOptimizationProfiles ( ) const [inline], [noexcept]
```

Get number of optimization profiles.

This is one higher than the index of the last optimization profile that has be defined (or zero, if none has been defined yet).

Returns

The number of the optimization profiles.

9.42.3.24 getNbPluginsToSerialize()

```
int32_t nvinfer1::IBuilderConfig::getNbPluginsToSerialize ( ) const [inline], [noexcept]
```

Get the number of plugin library paths to be serialized with version-compatible engines.

Returns

The number of paths.

9.42.3.25 getPluginToSerialize()

```
char const * nvinfer1::IBuilderConfig::getPluginToSerialize (
    int32_t index ) const [inline], [noexcept]
```

Get the plugin library path to be serialized with version-compatible engines.

Parameters

<i>index</i>	Index of the plugin library path in the list. Should be in the range <code>[0, getNbPluginsToSerialize()]</code> .

Returns

The path to the plugin library.

9.42.3.26 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](#)

9.42.3.27 `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\(\)](#)

9.42.3.28 `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](#)

9.42.3.29 `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.

9.42.3.30 getQuantizationFlags()

```
QuantizationFlags nvinfer1::IBuilderConfig::getQuantizationFlags ( ) const [inline], [noexcept]
```

Get the quantization flags.

Returns

The quantization flags as a bitmask.

See also

[setQuantizationFlag\(\)](#)

9.42.3.31 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

9.42.3.32 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](#)

9.42.3.33 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\(\)](#)

9.42.3.34 reset()

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

Resets the builder configuration to defaults.

Useful for initializing a builder config object to its original state.

9.42.3.35 resetDeviceType()

```
void nvinfer1::IBuilderConfig::resetDeviceType (
    ILayer const * layer ) [inline], [noexcept]
```

reset the DeviceType for this layer

See also

[setDeviceType\(\)](#) [getDeviceType\(\)](#) [isDeviceTypeSet\(\)](#)

9.42.3.36 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.

9.42.3.37 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\(\)](#)

9.42.3.38 setBuilderOptimizationLevel()

```
void nvinfer1::IBuilderConfig::setBuilderOptimizationLevel (
    int32_t level ) [inline], [noexcept]
```

Set builder optimization level.

Set the builder optimization level. Setting a higher optimization level allows the optimizer to spend more time searching for optimization opportunities. The resulting engine may have better performance compared to an engine built with a lower optimization level.

The default optimization level is 3. Valid values include integers from 0 to the maximum optimization level, which is currently 5. Setting it to greater than the maximum level results in behavior identical to the maximum level.

Parameters

<i>level</i>	The optimization level to set to. Must be non-negative.

See also

[getBuilderOptimizationLevel](#)

9.42.3.39 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.

9.42.3.40 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\(\)](#)

9.42.3.41 setDeviceType()

```
void nvinfer1::IBuilderConfig::setDeviceType (
    ILayer const * layer,
    DeviceType deviceType ) [inline], [noexcept]
```

Set the device that this layer must execute on.

Parameters

<i>layer</i>	which layer to execute.
<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\(\)](#)

9.42.3.42 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\(\)](#)

9.42.3.43 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.

9.42.3.44 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\(\)](#)

9.42.3.45 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\(\)](#)

9.42.3.46 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.

9.42.3.47 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.

9.42.3.48 setMaxAuxStreams()

```
void nvinfer1::IBuilderConfig::setMaxAuxStreams (
    int32_t nbStreams ) [inline], [noexcept]
```

Set the maximum number of auxiliary streams that TRT is allowed to use.

If the network contains operators that can run in parallel, TRT can execute them using auxiliary streams in addition to the one provided to the [IExecutionContext::enqueueV3\(\)](#) call.

The default maximum number of auxiliary streams is determined by the heuristics in TensorRT on whether enabling multi-stream would improve the performance. This behavior can be overridden by calling this API to set the maximum number of auxiliary streams explicitly. Set this to 0 to enforce single-stream inference.

The resulting engine may use fewer auxiliary streams than the maximum if the network does not contain enough parallelism or if TensorRT determines that using more auxiliary streams does not help improve the performance.

Note

Allowing more auxiliary streams does not always give better performance since there will be synchronizations overhead between streams. Using CUDA graphs at runtime can help reduce the overhead caused by cross-stream synchronizations.

Using more auxiliary leads to more memory usage at runtime since some activation memory blocks will not be able to be reused.

Parameters

<i>nbStreams</i>	The maximum number of auxiliary streams that TRT is allowed to use.

See also

[getMaxAuxStreams\(\)](#), [ICudaEngine::getNbAuxStreams\(\)](#), [IExecutionContext::setAuxStreams\(\)](#)

9.42.3.49 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](#).

9.42.3.50 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](#)

9.42.3.51 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\(\)](#).

9.42.3.52 setPluginsToSerialize()

```
void nvinfer1::IBuilderConfig::setPluginsToSerialize (
    char const *const * paths,
    int32_t nbPaths ) [inline], [noexcept]
```

Set the plugin libraries to be serialized with version-compatible engines.

Each entry in the list of libraries must be unique.

Parameters

<i>paths</i>	The paths of plugin libraries.
<i>nbPaths</i>	The number of paths.

9.42.3.53 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](#)

9.42.3.54 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\(\)](#)

9.42.3.55 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](#)

9.42.3.56 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\(\)](#)

9.42.3.57 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\(\)](#)

9.42.3.58 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.

9.42.3.59 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.

9.42.4 Member Data Documentation

9.42.4.1 mImpl

apiv::VBuilderConfig* nvinfer1::IBuilderConfig::mImpl [protected]

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

- [NvInfer.h](#)

9.43 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.

9.43.1 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.

9.43.2 Constructor & Destructor Documentation

9.43.2.1 ~ICaffeParser()

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

9.43.3 Member Function Documentation

9.43.3.1 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.

9.43.3.2 `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\(\)](#)

9.43.3.3 `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](#)

9.43.3.4 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](#)

9.43.3.5 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](#)

9.43.3.6 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\(\)](#)

9.43.3.7 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 .

9.43.3.8 setPluginNamespace()

```
virtual void nvcaffeparser1::ICaffeParser::setPluginNamespace (
    char const * libNamespace ) [pure virtual], [noexcept]
```

Set the namespace used to lookup and create plugins in the network.

9.43.3.9 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:

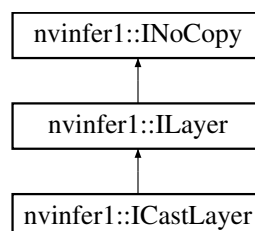
- [NvCaffeParser.h](#)

9.44 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](#)

9.44.1 Detailed Description

A cast layer in a network.

This layer casts a given tensor to the datatype specified by `toType`.

9.44.2 Constructor & Destructor Documentation

9.44.2.1 ~ICastLayer()

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

9.44.3 Member Function Documentation

9.44.3.1 getToType()

```
DataType nvinfer1::ICastLayer::getToType ( ) const [inline], [noexcept]
```

Return cast layer output type.

9.44.3.2 setToType()

```
void nvinfer1::ICastLayer::setToType (
    DataType toType ) [inline], [noexcept]
```

Set cast layer output type.

9.44.4 Member Data Documentation

9.44.4.1 mImpl

```
apiv::VCastLayer* nvinfer1::ICastLayer::mImpl [protected]
```

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

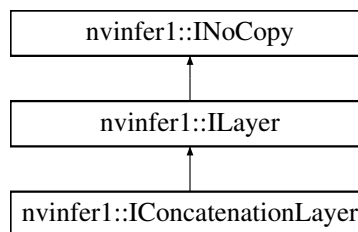
- [NvInfer.h](#)

9.45 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](#)

9.45.1 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.

9.45.2 Constructor & Destructor Documentation

9.45.2.1 ~IConcatenationLayer()

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

9.45.3 Member Function Documentation

9.45.3.1 getAxis()

```
int32_t nvinfer1::IConcatenationLayer::getAxis ( ) const [inline], [noexcept]
```

Get the axis along which concatenation occurs.

See also

[setAxis\(\)](#)

9.45.3.2 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.

9.45.4 Member Data Documentation

9.45.4.1 mImpl

```
apiv::VConcatenationLayer* nvinfer1::IConcatenationLayer::mImpl [protected]
```

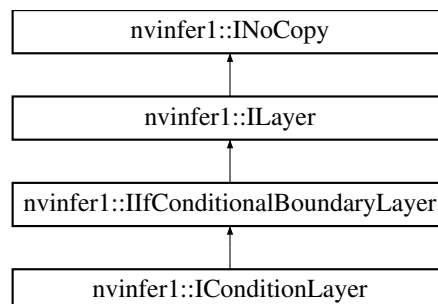
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.46 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

9.46.1 Detailed Description

This layer represents a condition input to an [IIfConditional](#).

9.46.2 Constructor & Destructor Documentation

9.46.2.1 ~IConditionLayer()

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

9.46.3 Member Data Documentation

9.46.3.1 mImpl

```
apiv::VConditionLayer* nvinfer1::IConditionLayer::mImpl [protected]
```

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

- [NvInfer.h](#)

9.47 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](#)

9.47.1 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.

9.47.2 Constructor & Destructor Documentation

9.47.2.1 ~IConsistencyChecker()

```
virtual nvinfer1::consistency::IConsistencyChecker::~~IConsistencyChecker ( ) [virtual], [default]
```

De-allocates any internally allocated memory.

9.47.2.2 IConsistencyChecker() [1/3]

```
nvinfer1::consistency::IConsistencyChecker::IConsistencyChecker ( ) [protected], [default]
```

9.47.2.3 IConsistencyChecker() [2/3]

```
nvinfer1::consistency::IConsistencyChecker::IConsistencyChecker (
    IConsistencyChecker const & other ) [protected], [delete]
```

9.47.2.4 IConsistencyChecker() [3/3]

```
nvinfer1::consistency::IConsistencyChecker::IConsistencyChecker (
    IConsistencyChecker && other ) [protected], [delete]
```

9.47.3 Member Function Documentation

9.47.3.1 operator=() [1/2]

```
IConsistencyChecker & nvinfer1::consistency::IConsistencyChecker::operator= (
    IConsistencyChecker && other ) [protected], [delete]
```

9.47.3.2 operator=() [2/2]

```
IConsistencyChecker & nvinfer1::consistency::IConsistencyChecker::operator= (
    IConsistencyChecker const & other ) [protected], [delete]
```

9.47.3.3 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.

9.47.4 Member Data Documentation

9.47.4.1 mImpl

```
apiv::VConsistencyChecker* nvinfer1::consistency::IConsistencyChecker::mImpl [protected]
```

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

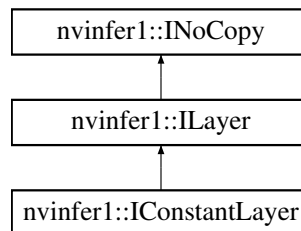
- [NvInferConsistency.h](#)

9.48 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](#)

9.48.1 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.

9.48.2 Constructor & Destructor Documentation

9.48.2.1 ~IConstantLayer()

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

9.48.3 Member Function Documentation

9.48.3.1 getDimensions()

```
Dims nvinfer1::IConstantLayer::getDimensions ( ) const [inline], [noexcept]
```

Get the dimensions for the layer.

Returns

the dimensions for the layer

See also

[getDimensions](#)

9.48.3.2 getWeights()

```
Weights nvinfer1::IConstantLayer::getWeights ( ) const [inline], [noexcept]
```

Get the weights for the layer.

See also

[setWeights](#)

9.48.3.3 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](#)

9.48.3.4 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\(\)](#)

9.48.4 Member Data Documentation

9.48.4.1 mImpl

```
apiv::VConstantLayer* nvinfer1::IConstantLayer::mImpl [protected]
```

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

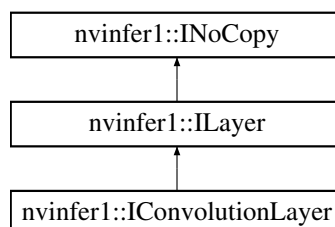
- [NvInfer.h](#)

9.49 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](#)

9.49.1 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.

9.49.2 Constructor & Destructor Documentation

9.49.2.1 ~IConvolutionLayer()

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

9.49.3 Member Function Documentation

9.49.3.1 getBiasWeights()

```
Weights nvinfer1::IConvolutionLayer::getBiasWeights ( ) const [inline], [noexcept]
```

Get the bias weights for the convolution.

See also

[setBiasWeights\(\)](#)

9.49.3.2 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

9.49.3.3 getDilationNd()

```
Dims nvinfer1::IConvolutionLayer::getDilationNd ( ) const [inline], [noexcept]
```

Get the multi-dimension dilation of the convolution.

See also

[setDilation\(\)](#)

9.49.3.4 `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

9.49.3.5 `getKernelSizeNd()`

```
Dims nvinfer1::IConvolutionLayer::getKernelSizeNd ( ) const [inline], [noexcept]
```

Get the multi-dimension kernel size of the convolution.

See also

[setKernelSizeNd\(\)](#)

9.49.3.6 `getKernelWeights()`

```
Weights nvinfer1::IConvolutionLayer::getKernelWeights ( ) const [inline], [noexcept]
```

Get the kernel weights of the convolution.

See also

[setKernelWeights\(\)](#)

9.49.3.7 `getNbGroups()`

```
int32_t nvinfer1::IConvolutionLayer::getNbGroups ( ) const [inline], [noexcept]
```

Get the number of groups of the convolution.

See also

[setNbGroups\(\)](#)

9.49.3.8 getNbOutputMaps()

```
int32_t nvinfer1::IConvolutionLayer::getNbOutputMaps ( ) const [inline], [noexcept]
```

Get the number of output maps for the convolution.

See also

[setNbOutputMaps\(\)](#)

9.49.3.9 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

9.49.3.10 getPaddingMode()

```
PaddingMode nvinfer1::IConvolutionLayer::getPaddingMode ( ) const [inline], [noexcept]
```

Get the padding mode.

Default: `kEXPLICIT_ROUND_DOWN`

See also

[setPaddingMode\(\)](#)

9.49.3.11 `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\(\)](#)

9.49.3.12 `getPostPadding()`

```
Dims nvinfer1::IConvolutionLayer::getPostPadding ( ) const [inline], [noexcept]
```

Get the post-padding.

See also

[setPostPadding\(\)](#)

9.49.3.13 `getPrePadding()`

```
Dims nvinfer1::IConvolutionLayer::getPrePadding ( ) const [inline], [noexcept]
```

Get the pre-padding.

See also

[setPrePadding\(\)](#)

9.49.3.14 `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

9.49.3.15 getStrideNd()

```
Dims nvinfer1::IConvolutionLayer::getStrideNd ( ) const [inline], [noexcept]
```

Get the multi-dimension stride of the convolution.

See also

[setStrideNd\(\)](#)

9.49.3.16 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\(\)](#)

9.49.3.17 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

9.49.3.18 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\(\)](#)

9.49.3.19 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\(\)](#)

9.49.3.20 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

9.49.3.21 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\(\)](#)

9.49.3.22 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\(\)](#)

9.49.3.23 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\(\)](#)

9.49.3.24 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\(\)](#)

9.49.3.25 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

9.49.3.26 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\(\)](#)

9.49.3.27 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\(\)](#)

9.49.3.28 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\(\)](#)

9.49.3.29 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\(\)](#)

9.49.3.30 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

9.49.3.31 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\(\)](#)

9.49.4 Member Data Documentation

9.49.4.1 mImpl

```
apiv::VConvolutionLayer* nvinfer1::IConvolutionLayer::mImpl [protected]
```

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

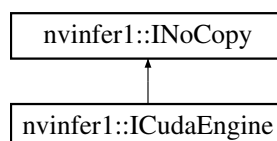
- [NvInfer.h](#)

9.50 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.

- `int32_t getTensorBytesPerComponent` (char const *tensorName, int32_t profileIndex) const noexcept
Return the number of bytes per component of an element of given profile, 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.
- `int32_t getTensorComponentsPerElement` (char const *tensorName, int32_t profileIndex) const noexcept
Return the number of components included in one element of given profile, or -1 if 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 tensor format, or `TensorFormat::kLINEAR` if the provided name does not map to an input or output tensor.
- `TensorFormat getTensorFormat` (char const *tensorName, int32_t profileIndex) const noexcept
Return the tensor format of given profile, or `TensorFormat::kLINEAR` if the provided name does not map to an input or output tensor.
- `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.
- `char const * getTensorFormatDesc` (char const *tensorName, int32_t profileIndex) const noexcept
Return the human readable description of the tensor format of given profile, or empty string if the provided name does not map to an input or output tensor.
- `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.
- `int32_t getTensorVectorizedDim` (char const *tensorName, int32_t profileIndex) const noexcept
Return the dimension index that the buffer is vectorized of given profile, or -1 if the provided name does not map to an input or output tensor.
- `char const * getName` () const noexcept
Returns the name of the network associated with the engine.
- `int32_t getNbOptimizationProfiles` () const noexcept
Get the number of optimization profiles defined for this engine.
- `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.
- int32_t [getNbAuxStreams](#) () const noexcept
Return the number of auxiliary streams used by this engine.

Protected Attributes

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

Additional Inherited Members

9.50.1 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.

9.50.2 Constructor & Destructor Documentation

9.50.2.1 ~ICudaEngine()

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

9.50.3 Member Function Documentation

9.50.3.1 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\(\)](#)

9.50.3.2 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](#).

9.50.3.3 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\(\)](#)

9.50.3.4 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.

9.50.3.5 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.

9.50.3.6 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\(\)](#)

9.50.3.7 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\(\)](#)

9.50.3.8 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\(\)](#)

9.50.3.9 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\(\)](#)

9.50.3.10 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\(\)](#)

9.50.3.11 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\(\)](#)

9.50.3.12 `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 $[0 \dots n-1]$ 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\(\)](#)

9.50.3.13 `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\(\)](#)

9.50.3.14 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 if 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\(\)](#)

9.50.3.15 getDeviceMemorySize()

```
size_t nvinfer1::ICudaEngine::getDeviceMemorySize ( ) const [inline], [noexcept]
```

Return the amount of device memory required by an execution context.

See also

[IExecutionContext::setDeviceMemory\(\)](#)

9.50.3.16 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.

9.50.3.17 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\(\)](#)

9.50.3.18 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.

9.50.3.19 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\(\)](#)

9.50.3.20 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\(\)](#)

9.50.3.21 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.

9.50.3.22 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.

9.50.3.23 getNbAuxStreams()

```
int32_t nvinfer1::ICudaEngine::getNbAuxStreams ( ) const [inline], [noexcept]
```

Return the number of auxiliary streams used by this engine.

This number will be less than or equal to the maximum allowed number of auxiliary streams set by [IBuilderConfig::setMaxAuxStreams\(\)](#) API call when the engine was built.

Returns

the number of auxiliary streams used by this engine.

See also

[IBuilderConfig::setMaxAuxStreams\(\)](#), [IExecutionContext::setAuxStreams\(\)](#)

9.50.3.24 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\(\)](#)

9.50.3.25 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\(\)](#)

9.50.3.26 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.

9.50.3.27 `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\(\)](#)

9.50.3.28 `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 <code>bindingsPerProfile-1</code> 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\(\)](#)

9.50.3.29 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.

9.50.3.30 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 getNbOptimizationProfiles()-1)
<i>inputIndex</i>	The input index (must be between 0 and getNbBindings() - 1)
<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\(\)](#)

9.50.3.31 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\(\)](#)

9.50.3.32 getTacticSources()

```
TacticSources nvinfer1::ICudaEngine::getTacticSources ( ) const [inline], [noexcept]
```

return the tactic sources required by this engine.

The value returned is equal to zero or more tactics sources set at build time via `setTacticSources()` in [IBuilderConfig](#). Sources set by the latter but not returned by [ICudaEngine::getTacticSources](#) do not reduce overall engine execution time, and can be removed from future builds to reduce build time.

See also

[IBuilderConfig::setTacticSources\(\)](#)

9.50.3.33 getTensorBytesPerComponent() [1/2]

```
int32_t nvinfer1::ICudaEngine::getTensorBytesPerComponent (
    char const * tensorName ) const [inline], [noexcept]
```

Return the number of bytes per component of an element, or -1 if the 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.

The function can only return the result of profile 0, and issues a warning message when there are multiple profiles in the engine, use `getTensorBytesPerComponent` with `profileIndex` when there are multiple profiles.

See also

[getTensorVectorizedDim\(\)](#)

`getTensorBytesPerComponent(tensorName, profileIndex)`

9.50.3.34 `getTensorBytesPerComponent()` [2/2]

```
int32_t nvinfer1::ICudaEngine::getTensorBytesPerComponent (
    char const * tensorName,
    int32_t profileIndex ) const [inline], [noexcept]
```

Return the number of bytes per component of an element of given profile, or -1 if the provided name does not map to an input or output tensor.

The vector component size is returned if `getTensorVectorizedDim(tensorName, profileIndex) != -1`.

Parameters

<i>tensorName</i>	The name of an input or output tensor.
<i>profileIndex</i>	The profile index to query

Warning

The string `tensorName` must be null-terminated, and be at most 4096 bytes including the terminator.

See also

`getTensorVectorizedDim(tensorName, profileIndex)`

9.50.3.35 getTensorComponentsPerElement() [1/2]

```
int32_t nvinfer1::ICudaEngine::getTensorComponentsPerElement (
    char const * tensorName ) const [inline], [noexcept]
```

Return the number of components included in one element, or -1 if 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.
The function can only return the result of profile 0, and issues a warning message when there are multiple profiles in the engine, use [getTensorComponentsPerElement](#) with *profileIndex* when there are multiple profiles.

See also

[getTensorVectorizedDim\(\)](#)

[getTensorComponentsPerElement\(tensorName, profileIndex\)](#)

9.50.3.36 getTensorComponentsPerElement() [2/2]

```
int32_t nvinfer1::ICudaEngine::getTensorComponentsPerElement (
    char const * tensorName,
    int32_t profileIndex ) const [inline], [noexcept]
```

Return the number of components included in one element of given profile, or -1 if the provided name does not map to an input or output tensor.

The number of elements in the vectors is returned if [getTensorVectorizedDim\(tensorName, profileIndex\)](#) != -1.

Parameters

<i>tensorName</i>	The name of an input or output tensor.
<i>profileIndex</i>	The profile index to query

Warning

The string `tensorName` must be null-terminated, and be at most 4096 bytes including the terminator.

See also

`getTensorVectorizedDim(tensorName, profileIndex)`

9.50.3.37 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.

9.50.3.38 getTensorFormat() [1/2]

```
TensorFormat nvinfer1::ICudaEngine::getTensorFormat (
    char const * tensorName ) const [inline], [noexcept]
```

Return the tensor format, or [TensorFormat::kLINEAR](#) if the provided name does not map to an input or output tensor.

Warning

The string `tensorName` must be null-terminated, and be at most 4096 bytes including the terminator.

This API can only return the tensor format of profile 0, and issues a warning message when there are multiple profiles in the engine, use `getTensorFormat` with `profileIndex` when there are multiple profiles.

See also

`getTensorFormat(tensorName, profileIndex)`

9.50.3.39 getTensorFormat() [2/2]

```
TensorFormat nvinfer1::ICudaEngine::getTensorFormat (
    char const * tensorName,
    int32_t profileIndex ) const [inline], [noexcept]
```

Return the tensor format of given profile, or [TensorFormat::kLINEAR](#) if the provided name does not map to an input or output tensor.

Parameters

<i>tensorName</i>	The name of an input or output tensor.
<i>profileIndex</i>	The profile index to query the format for.

Warning

The string *tensorName* must be null-terminated, and be at most 4096 bytes including the terminator.

9.50.3.40 getTensorFormatDesc() [1/2]

```
char const * nvinfer1::ICudaEngine::getTensorFormatDesc (
    char const * tensorName ) const [inline], [noexcept]
```

Return the human readable description of the tensor format, or empty string if the provided name does not map to an input or output tensor.

The description includes the order, vectorization, data type, and strides. Examples are shown as follows: Example 1: kCHW + FP32 "Row-major linear FP32 format" Example 2: kCHW2 + FP16 "Two-wide channel vectorized row-major FP16 format" Example 3: kHWC8 + FP16 + Line Stride = 32 "Channel major FP16 format where C % 8 == 0 and H Stride % 32 == 0"

Parameters

<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.

The function can only return the result of profile 0, and issues a warning message when there are multiple profiles in the engine, use `getTensorFormatDesc` with *profileIndex* when there are multiple profiles.

9.50.3.41 getTensorFormatDesc() [2/2]

```
char const * nvinfer1::ICudaEngine::getTensorFormatDesc (
```

```
char const * tensorName,
int32_t profileIndex ) const [inline], [noexcept]
```

Return the human readable description of the tensor format of given profile, or empty string if the provided name does not map to an input or output tensor.

The description includes the order, vectorization, data type, and strides. Examples are shown as follows: Example 1: kCHW + FP32 "Row-major linear FP32 format" Example 2: kCHW2 + FP16 "Two-wide channel vectorized row-major FP16 format" Example 3: kHWC8 + FP16 + Line Stride = 32 "Channel major FP16 format where C % 8 == 0 and H Stride % 32 == 0"

Parameters

<i>tensorName</i>	The name of an input or output tensor.
<i>profileIndex</i>	The profile index to query the format for.

Warning

The string `tensorName` must be null-terminated, and be at most 4096 bytes including the terminator.

9.50.3.42 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.

9.50.3.43 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.

9.50.3.44 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.

9.50.3.45 getTensorVectorizedDim() [1/2]

```
int32_t nvinfer1::ICudaEngine::getTensorVectorizedDim (
    char const * tensorName ) const [inline], [noexcept]
```

Return the dimension index that the buffer is vectorized, or -1 if the provided name does not map to an input or output tensor.

Specifically -1 is returned if scalars per vector is 1.

Parameters

<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.

The function can only return the result of profile 0, and issues a warning message when there are multiple profiles in the engine, use `getTensorVectorizedDim` with *profileIndex* when there are multiple profiles.

9.50.3.46 `getTensorVectorizedDim()` [2/2]

```
int32_t nvinfer1::ICudaEngine::getTensorVectorizedDim (
    char const * tensorName,
    int32_t profileIndex ) const [inline], [noexcept]
```

Return the dimension index that the buffer is vectorized of given profile, or -1 if the provided name does not map to an input or output tensor.

Specifically -1 is returned if scalars per vector is 1.

Parameters

<i>tensorName</i>	The name of an input.
<i>profileIndex</i>	The profile index to query the format for.

Warning

The string *tensorName* must be null-terminated, and be at most 4096 bytes including the terminator.

9.50.3.47 `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`

9.50.3.48 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\(\)](#)

9.50.3.49 isRefittable()

```
bool nvinfer1::ICudaEngine::isRefittable ( ) const [inline], [noexcept]
```

Return true if an engine can be refit.

See also

`nvinfer1::createInferRefitter()`

9.50.3.50 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\(\)](#)

9.50.3.51 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".

9.50.3.52 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\(\)](#)

9.50.3.53 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\(\)](#)

9.50.4 Member Data Documentation

9.50.4.1 mImpl

```
apiv::VCudaEngine* nvinfer1::ICudaEngine::mImpl [protected]
```

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

- [NvInferRuntime.h](#)

9.51 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 vector dimension index for a vectorized binding.*

 - virtual `AsciiChar` const * `getName` () const noexcept=0

Returns the name of the network associated with the engine.
- virtual void `setErrorRecorder` (`LErrorRecorder` *const recorder) noexcept=0

Set the ErrorRecorder for this interface.
- virtual `LErrorRecorder` * `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 *const tensorName) const noexcept=0

Get the extent of an input or output tensor.
- virtual `DataType` `getTensorDataType` (`AsciiChar` const *const tensorName) const noexcept=0

Determine the required data type for a buffer from its tensor name.
- virtual `TensorIOMode` `getTensorIOMode` (`AsciiChar` const *const tensorName) const noexcept=0

Determine whether a tensor is an input or output tensor.
- virtual `std::int32_t` `getTensorBytesPerComponent` (`AsciiChar` const *const tensorName) const noexcept=0

Return the size of the tensor data type in bytes for a vectorized tensor.
- virtual `std::int32_t` `getTensorComponentsPerElement` (`AsciiChar` const *const tensorName) const noexcept=0

Return the number of components included in one element for a vectorized tensor.
- virtual `TensorFormat` `getTensorFormat` (`AsciiChar` const *const tensorName) const noexcept=0

Return the tensor format.
- virtual `std::int32_t` `getTensorVectorizedDim` (`AsciiChar` const *const tensorName) const noexcept=0

Return the dimension index along which the 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.

9.51.1 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.

9.51.2 Constructor & Destructor Documentation

9.51.2.1 ICudaEngine() [1/3]

```
nvinfer1::safe::ICudaEngine::ICudaEngine ( ) [default]
```

9.51.2.2 ~ICudaEngine()

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

9.51.2.3 ICudaEngine() [2/3]

```
nvinfer1::safe::ICudaEngine::ICudaEngine (
    ICudaEngine const & ) [delete]
```

9.51.2.4 ICudaEngine() [3/3]

```
nvinfer1::safe::ICudaEngine::ICudaEngine (
    ICudaEngine && ) [delete]
```

9.51.3 Member Function Documentation**9.51.3.1 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 the range (between 0 and [getNbBindings\(\)-1](#)).
False if the index is out of range or does not correspond to an input binding.

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

9.51.3.2 createExecutionContext()

```
virtual IExecutionContext * nvinfer1::safe::ICudaEngine::createExecutionContext ( ) [pure virtual],  
[noexcept]
```

Create an execution context.

See also

`safe::IExecutionContext`.

Returns

An execution context object if it can be constructed, or `nullptr` if the construction fails.

Reasons for failure may include but not be limited to:

- Heap memory exhaustion
- Device memory exhaustion

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes; if `createExecutionContext` fails, users must treat this as a critical error and not perform any subsequent TensorRT operations apart from outputting the error logs.

9.51.3.3 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 by calling [safe::IExecutionContext::setDeviceMemory](#)

See also

[getDeviceMemorySize\(\)](#) [safe::IExecutionContext::setDeviceMemory\(\)](#)

Returns

An execution context object if it can be constructed, or nullptr if the construction fails.

Reasons for failure may include but not be limited to heap memory exhaustion.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes; if createExecutionContext fails, users must treat this as a critical error and not perform any subsequent TensorRT operations apart from outputting the error logs.

9.51.3.4 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.

Parameters

<i>bindingIndex</i>	The binding index.

Returns

The size of the tensor data type in bytes (4 for float and int32, 2 for half, 1 for int8) if the binding index is in range (between 0 and [getNbBindings\(\)-1](#)) and corresponds to a vectorized tensor. The value 0 is returned if

- bindingIndex is out of range, or
- bindingIndex corresponds to a scalar tensor.

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

9.51.3.5 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.

Parameters

<i>bindingIndex</i>	The binding index.

Returns

The vector length in scalars if the binding index is in range (between 0 and [getNbBindings\(\)-1](#)) and corresponds to a vectorized tensor ([getBindingVectorizedDim\(\)](#) != -1). Return 1 if the binding index corresponds to a scalar tensor ([getBindingVectorizedDim\(\)](#) == -1). The invalid value -1 is returned if

- the binding index is out of range, or
- the vector length does not fit in a signed int32 (integer overflow).

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

9.51.3.6 `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 tensor data type for the binding if the index is in range (between 0 and [getNbBindings\(\)-1](#)). The default value kFLOAT is returned if the index is out of range.

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

9.51.3.7 `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 the range (between 0 and [getNbBindings\(\)-1](#)). The invalid value [Dims{}](#) is returned if the index is out of range.

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

9.51.3.8 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.

Returns

The tensor format if *bindingIndex* is in range (between 0 and [getNbBindings\(\)-1](#)). The default value [TensorFormat::kLINEAR](#) is returned if *bindingIndex* is out of range.

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

9.51.3.9 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 input and output bindings.

Warning

Strings passed to the runtime must be NULL terminated and have a length of 1024 bytes or less including the NULL terminator.

Parameters

<i>name</i>	The tensor name.

Returns

The binding index for the named tensor, or -1 if the name is not found. Return values will lie between -1 and [getNbBindings\(\)](#)-1 inclusive.

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

9.51.3.10 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 if the index is in the range (between 0 and nbBindings()-1). nullptr is returned if the index is out of range. If a string is returned, it will have a length of 1024 bytes or less including the NULL terminator.

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

9.51.3.11 getBindingVectorizedDim()

```
virtual TRT\_DEPRECATED std::int32_t nvinfer1::safe::ICudaEngine::getBindingVectorizedDim (
    std::int32_t const bindingIndex ) const [pure virtual], [noexcept]
```

Return the vector dimension index for a vectorized binding.

Parameters

<i>bindingIndex</i>	The binding index.

Returns

The vector dimension index if bindingIndex is in range (between 0 and [getNbBindings\(\)-1](#)). The value -1 is returned if

- bindingIndex is out of range, or
- bindingIndex corresponds to a scalar tensor.

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

9.51.3.12 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\(\)](#)

Returns

Size of a contiguous memory buffer (in bytes) that users need to provide to [safe::IExecutionContext::setDeviceMemory\(\)](#) if the execution context has been created by calling [createExecutionContextWithoutDeviceMemory\(\)](#).

Usage considerations

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

9.51.3.13 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, or nullptr if none has been registered.

See also

[setErrorRecorder\(\)](#)

Usage considerations

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

9.51.3.14 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 IO tensor index.

Returns

The name of an IO tensor, which will be a NULL-terminated string of 1024 bytes or less (including the NULL terminator) if the index is in the range (between 0 and [getNbIOTensors\(\)-1](#)). nullptr will be returned if the index is not in range.

See also

[getNbIOTensors\(\)](#)

Usage considerations

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

9.51.3.15 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, which will have a length of 1024 bytes or less including the NULL terminator.

Usage considerations

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

9.51.3.16 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

9.51.3.17 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

9.51.3.18 getTensorBytesPerComponent()

```
virtual std::int32_t nvinfer1::safe::ICudaEngine::getTensorBytesPerComponent (   
    AsciiChar const *const tensorName ) const [pure virtual], [noexcept]
```

Return the size of the tensor data type in bytes for a vectorized tensor.

Parameters

<i>tensorName</i>	The name of an input or output tensor.

Warning

The string `tensorName` must be NULL terminated and have a length of 1024 bytes or less including the NULL terminator.

Returns

The size of the tensor data type in bytes if the tensor is vectorized (4 for float and int32, 2 for half, 1 for int8). 0 will be returned if

- name is not the name of an input or output tensor, or
- name is nullptr, or
- name exceeds the string length limit, or
- the tensor of the given name is not vectorized.

See also

[safe::ICudaEngine::getTensorVectorizedDim\(\)](#)

Usage considerations

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

9.51.3.19 getTensorComponentsPerElement()

```
virtual std::int32_t nvinfer1::safe::ICudaEngine::getTensorComponentsPerElement (
    AsciiChar const *const tensorName ) const [pure virtual], [noexcept]
```

Return the number of components included in one element for a vectorized tensor.

Parameters

<i>tensorName</i>	The name of an input or output tensor.

Warning

The string `tensorName` must be NULL terminated and have a length of 1024 bytes or less including the NULL terminator.

Returns

The vector length (in scalars) for a vectorized tensor, or 1 for a scalar tensor. The invalid value -1 will be returned if

- name is not the name of an input or output tensor, or
- name is nullptr, or
- name exceeds the string length limit.

See also

[safe::ICudaEngine::getTensorVectorizedDim\(\)](#)

Usage considerations

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

9.51.3.20 getTensorDataType()

```
virtual DataType nvinfer1::safe::ICudaEngine::getTensorDataType (
    AsciiChar const *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 NULL terminated and have a length of 1024 bytes or less including the NULL terminator.

Returns

The type of the data in the buffer. The default value [DataType::kFLOAT](#) will be returned if

- name is not the name of an input or output tensor, or
- name is nullptr, or
- name exceeds the string length limit.

Usage considerations

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

9.51.3.21 getTensorFormat()

```
virtual TensorFormat nvinfer1::safe::ICudaEngine::getTensorFormat (
    AsciiChar const *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 NULL terminated and have a length of 1024 bytes or less including the NULL terminator.

Returns

The tensor format. [TensorFormat::kLINEAR](#) will be returned if

- name is not the name of an input or output tensor, or
- name is nullptr, or
- name exceeds the string length limit.

Usage considerations

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

9.51.3.22 getTensorIOMode()

```
virtual TensorIOMode nvinfer1::safe::ICudaEngine::getTensorIOMode (
    AsciiChar const *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 NULL terminated and have a length of 1024 bytes or less including the NULL terminator.

Returns

kINPUT if *tensorName* is the name of an input tensor, kOUTPUT if *tensorName* is the name of an output tensor. The invalid value kNONE is returned if

- *tensorName* exceeds the string length limit, or
- *tensorName* is nullptr, or
- *tensorName* does not correspond to any input or output tensor.

Usage considerations

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

9.51.3.23 getTensorShape()

```
virtual Dims nvinfer1::safe::ICudaEngine::getTensorShape (
    AsciiChar const *const tensorName ) const [pure virtual], [noexcept]
```

Get the 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 NULL terminated and have a length of 1024 bytes or less including the NULL terminator.

Returns

Extent of the tensor. The invalid value `Dims{-1, {}}` will be returned if

- name is not the name of an input or output tensor, or
- name is nullptr, or
- name exceeds the string length limit.

Usage considerations

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

9.51.3.24 getTensorVectorizedDim()

```
virtual std::int32_t nvinfer1::safe::ICudaEngine::getTensorVectorizedDim (
    AsciiChar const *const tensorName ) const [pure virtual], [noexcept]
```

Return the dimension index along which the buffer is vectorized.

Specifically, -1 is returned if the tensor is scalar.

Parameters

<i>tensorName</i>	The name of an input or output tensor.

Warning

The string `tensorName` must be NULL terminated and have a length of 1024 bytes or less including the NULL terminator.

Returns

The dimension index along which the buffer is vectorized. -1 will be returned if

- name is not the name of an input or output tensor, or
- name is nullptr, or
- name exceeds the string length limit (1024 bytes or less including the NULL terminator), or

- the tensor of given name is not vectorized.

Usage considerations

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

9.51.3.25 operator=() [1/2]

```
ICudaEngine & nvinfer1::safe::ICudaEngine::operator= (
    ICudaEngine && ) & [delete]
```

9.51.3.26 operator=() [2/2]

```
ICudaEngine & nvinfer1::safe::ICudaEngine::operator= (
    ICudaEngine const & ) & [delete]
```

9.51.3.27 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. If the recorder is set to nullptr, the error code [ErrorCode::kINVALID_ARGUMENT](#) will be emitted if the recorder has been registered.

Parameters

<i>recorder</i>	The error recorder to register with this interface, or nullptr to deregister the current. error recorder.

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:

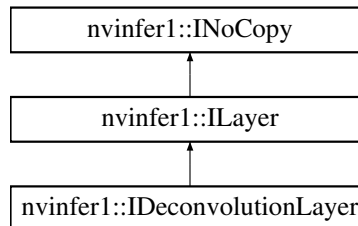
- [NvInferSafeRuntime.h](#)

9.52 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**

9.52.1 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.

9.52.2 Constructor & Destructor Documentation

9.52.2.1 ~IDeconvolutionLayer()

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

9.52.3 Member Function Documentation

9.52.3.1 getBiasWeights()

```
Weights nvinfer1::IDeconvolutionLayer::getBiasWeights ( ) const [inline], [noexcept]
```

Get the bias weights for the deconvolution.

See also

[getBiasWeights\(\)](#)

9.52.3.2 getDilationNd()

```
Dims nvinfer1::IDeconvolutionLayer::getDilationNd ( ) const [inline], [noexcept]
```

Get the multi-dimension dilation of the deconvolution.

See also

[setDilationNd\(\)](#)

9.52.3.3 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

9.52.3.4 getKernelSizeNd()

```
Dims nvinfer1::IDeconvolutionLayer::getKernelSizeNd ( ) const [inline], [noexcept]
```

Get the multi-dimension kernel size of the deconvolution.

See also

[setKernelSizeNd\(\)](#)

9.52.3.5 getKernelWeights()

```
Weights nvinfer1::IDeconvolutionLayer::getKernelWeights ( ) const [inline], [noexcept]
```

Get the kernel weights for the deconvolution.

See also

[setNbGroups\(\)](#)

9.52.3.6 getNbGroups()

```
int32_t nvinfer1::IDeconvolutionLayer::getNbGroups ( ) const [inline], [noexcept]
```

Get the number of groups for a deconvolution.

See also

[setNbGroups\(\)](#)

9.52.3.7 getNbOutputMaps()

```
int32_t nvinfer1::IDeconvolutionLayer::getNbOutputMaps ( ) const [inline], [noexcept]
```

Get the number of output feature maps for the deconvolution.

See also

[setNbOutputMaps\(\)](#)

9.52.3.8 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

9.52.3.9 getPaddingMode()

```
PaddingMode nvinfer1::IDeconvolutionLayer::getPaddingMode ( ) const [inline], [noexcept]
```

Get the padding mode.

Default: `kEXPLICIT_ROUND_DOWN`

See also

[setPaddingMode\(\)](#)

9.52.3.10 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\(\)](#)

9.52.3.11 getPostPadding()

```
Dims nvinfer1::IDeconvolutionLayer::getPostPadding ( ) const [inline], [noexcept]
```

Get the padding.

See also

[setPostPadding\(\)](#)

9.52.3.12 getPrePadding()

```
Dims nvinfer1::IDeconvolutionLayer::getPrePadding ( ) const [inline], [noexcept]
```

Get the pre-padding.

See also

[setPrePadding\(\)](#)

9.52.3.13 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

9.52.3.14 getStrideNd()

```
Dims nvinfer1::IDeconvolutionLayer::getStrideNd ( ) const [inline], [noexcept]
```

Get the multi-dimension stride of the deconvolution.

See also

[setStrideNd\(\)](#)

9.52.3.15 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\(\)](#)

9.52.3.16 setDilationNd()

```
void nvinfer1::IDeconvolutionLayer::setDilationNd (
    Dims dilation ) [inline], [noexcept]
```

Set the multi-dimension dilation of the deconvolution.

Default: (1, 1, ..., 1)

See also

[getDilationNd\(\)](#)

9.52.3.17 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\(\)](#)

9.52.3.18 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

9.52.3.19 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\(\)](#)

9.52.3.20 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\(\)](#)

9.52.3.21 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\(\)](#)

9.52.3.22 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\(\)](#)

9.52.3.23 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

9.52.3.24 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\(\)](#)

9.52.3.25 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\(\)](#)

9.52.3.26 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\(\)](#)

9.52.3.27 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\(\)](#)

9.52.3.28 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

9.52.3.29 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\(\)](#)

9.52.4 Member Data Documentation

9.52.4.1 mImpl

```
apiv::VDeconvolutionLayer* nvinfer1::IDeconvolutionLayer::mImpl [protected]
```

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

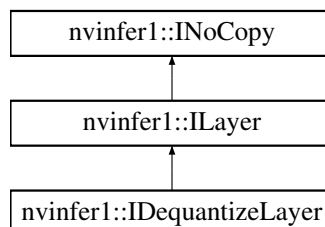
- [NvInfer.h](#)

9.53 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`

9.53.1 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.

9.53.2 Constructor & Destructor Documentation

9.53.2.1 ~IDequantizeLayer()

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

9.53.3 Member Function Documentation

9.53.3.1 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.

9.53.3.2 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).

9.53.4 Member Data Documentation

9.53.4.1 mImpl

```
apiv::VDequantizeLayer* nvinfer1::IDequantizeLayer::mImpl [protected]
```

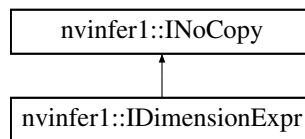
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.54 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](#)

9.54.1 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](#)

9.54.2 Constructor & Destructor Documentation

9.54.2.1 ~IDimensionExpr()

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

9.54.3 Member Function Documentation

9.54.3.1 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()`.

9.54.3.2 isConstant()

```
bool nvinfer1::IDimensionExpr::isConstant ( ) const [inline], [noexcept]
```

Return true if expression is a build-time constant.

9.54.4 Member Data Documentation

9.54.4.1 mImpl

```
apiv::VDimensionExpr* nvinfer1::IDimensionExpr::mImpl [protected]
```

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

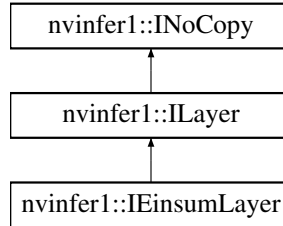
- [NvInferRuntime.h](#)

9.55 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](#)

9.55.1 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.

9.55.2 Constructor & Destructor Documentation

9.55.2.1 ~IEinsumLayer()

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

9.55.3 Member Function Documentation

9.55.3.1 getEquation()

```
char const * nvinfer1::IEinsumLayer::getEquation ( ) const [inline], [noexcept]
```

Return the equation.

See also

[setEquation\(\)](#)

9.55.3.2 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\(\)](#)

9.55.4 Member Data Documentation

9.55.4.1 mImpl

```
apiv::VEinsumLayer* nvinfer1::IEinsumLayer::mImpl [protected]
```

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

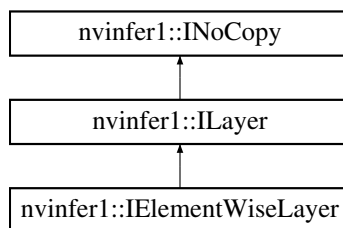
- [NvInfer.h](#)

9.56 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](#)

9.56.1 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.

9.56.2 Constructor & Destructor Documentation

9.56.2.1 ~IElementWiseLayer()

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

9.56.3 Member Function Documentation

9.56.3.1 getOperation()

```
ElementWiseOperation nvinfer1::IElementWiseLayer::getOperation ( ) const [inline], [noexcept]
```

Get the binary operation for the layer.

See also

[setOperation\(\)](#), [ElementWiseOperation](#)
[setBiasWeights\(\)](#)

9.56.3.2 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\(\)](#)

9.56.4 Member Data Documentation

9.56.4.1 mImpl

```
apiv::VElementWiseLayer* nvinfer1::IElementWiseLayer::mImpl [protected]
```

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

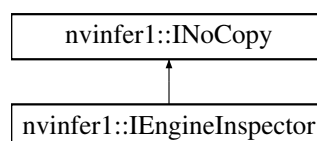
- [NvInfer.h](#)

9.57 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

9.57.1 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](#)

9.57.2 Constructor & Destructor Documentation

9.57.2.1 ~IEngineInspector()

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

9.57.3 Member Function Documentation

9.57.3.1 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>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](#)

9.57.3.2 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\(\)](#)

9.57.3.3 `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\(\)](#)

9.57.3.4 `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](#)

9.57.3.5 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\(\)](#)

9.57.3.6 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.

9.57.4 Member Data Documentation

9.57.4.1 mImpl

```
apiv::VEngineInspector* nvinfer1::IEngineInspector::mImpl [protected]
```

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

- [NvInferRuntime.h](#)

9.58 nvinfer1::IErrorRecorder Class Reference

Reference counted application-implemented error reporting interface for TensorRT objects.

```
#include <NvInferRuntimeBase.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}

9.58.1 Detailed Description

Reference counted application-implemented error reporting interface for TensorRT objects.

The error reporting mechanism is a user-defined object that interacts with the internal state of the object that it is assigned to in order to determine information about abnormalities in execution. The error recorder gets both an error enum that is more descriptive than pass/fail and also a string description that gives more detail on the exact failure modes. In the safety context, the error strings are all limited to 128 bytes or less in length, including the NULL terminator.

The ErrorRecorder gets passed along to any class that is created from another class that has an ErrorRecorder assigned to it. For example, assigning an ErrorRecorder to an `IBuilder` allows all `INetwork`'s, `ILayer`'s, and `ITensor`'s to use the same error recorder. For functions that have their own ErrorRecorder accessor functions. This allows registering a different error recorder or de-registering of the error recorder for that specific object.

ErrorRecorder objects that are used in the safety runtime must define an implementation-dependent upper limit of errors whose information can be stored, and drop errors above this upper limit. The limit must fit in `int32_t`. The `IErrorRecorder::hasOverflowed()` method is used to signal that one or more errors have been dropped.

The ErrorRecorder object implementation must be thread safe. All locking and synchronization is pushed to the interface implementation and TensorRT does not hold any synchronization primitives when calling the interface functions.

The lifetime of the ErrorRecorder object must exceed the lifetime of all TensorRT objects that use it.

9.58.2 Member Typedef Documentation

9.58.2.1 ErrorDesc

```
using nvinfer1::IErrorRecorder::ErrorDesc = char const*
```

A typedef of a C-style string for reporting error descriptions.

9.58.2.2 RefCount

```
using nvinfer1::IErrorRecorder::RefCount = int32_t
```

A typedef of a 32-bit integer for reference counting.

9.58.3 Constructor & Destructor Documentation

9.58.3.1 IErrorRecorder()

```
nvinfer1::IErrorRecorder::IErrorRecorder ( ) [default]
```

9.58.3.2 ~IErrorRecorder()

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

9.58.4 Member Function Documentation

9.58.4.1 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. The implementation must guarantee that after this function is called, and as long as no error occurs, the next call to `getNbErrors` will return zero and `hasOverflowed` will return false.

See also

[getNbErrors\(\)](#), [hasOverflowed\(\)](#)

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.

9.58.4.2 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.

9.58.4.3 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 if errorIdx is in range (between 0 and [getNbErrors\(\)](#)-1). [ErrorCode::kUNSPECIFIED_ERROR](#) must be returned if errorIdx is not in range.

See also

[getErrorDesc\(\)](#), [ErrorCode](#)

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.

9.58.4.4 `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 will be truncated if it exceeds `kMAX_DESC_LENGTH` bytes. The format of the string is "`<EnumAsStr> - <Description>`".

Returns

Returns a string representation of the error along with a description of the error if `errorIdx` is in range (between 0 and [getNbErrors\(\)](#)-1). An empty string will be returned if `errorIdx` is not in range.

See also

[getErrorCode\(\)](#)

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.

9.58.4.5 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 increment by 1 after each [reportError\(\)](#) call until [clear\(\)](#) is called, or the maximum number of errors that can be stored is exceeded.

Returns

Returns the number of errors detected, or 0 if there are no errors. If the upper bound of errors that can be stored is exceeded, the upper bound value must be returned.

For example, if the error recorder can store up to 16 error descriptions but [recordError\(\)](#) has been called 20 times, [getNbErrors\(\)](#) must return 16.

See also

[clear\(\)](#), [hasOverflowed\(\)](#)

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.

9.58.4.6 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.

9.58.4.7 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.

9.58.4.8 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, which will be a NULL-terminated string of kMAX_DESC_LENGTH bytes or less (excluding the NULL terminator). Descriptions that exceed this limit will be silently truncated.

Report an error to the user that has a given value and human readable description. The function returns false if processing can continue, which implies that the reported error is not fatal. This does not guarantee that processing continues, but provides a hint to TensorRT. The desc C-string data is only valid during the call to reportError and may be immediately deallocated by the caller when reportError returns. The implementation must not store the desc pointer in the ErrorRecorder object or otherwise access the data from desc after reportError returns.

Returns

True if the error is determined to be fatal and processing of the current function must end.

Warning

If the error recorder's maximum number of storable errors is exceeded, the error description will be silently dropped and the value returned by [getNbErrors\(\)](#) will not be incremented. However, the return value will still signal whether the error must be considered fatal.

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.

9.58.5 Member Data Documentation

9.58.5.1 kMAX_DESC_LENGTH

```
constexpr size_t nvinfer1::IErrorRecorder::kMAX_DESC_LENGTH {127U} [static], [constexpr]
```

The length limit for an error description in bytes, excluding the '\0' string terminator.

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

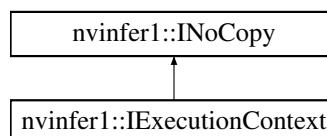
- [NvInferRuntimeBase.h](#)

9.59 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
Enqueue inference of a batch on a stream.
- 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

Enqueue inference on a stream.
- 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
Enqueue inference on a stream.
- void [setPersistentCacheLimit](#) (size_t size) noexcept
Set the maximum size for persistent cache usage.
- size_t [getPersistentCacheLimit](#) () const noexcept
Get the maximum size for persistent cache usage.
- bool [setNvtxVerbosity](#) ([ProfilingVerbosity](#) verbosity) noexcept
Set the verbosity of the NVTX markers in the execution context.
- [ProfilingVerbosity](#) [getNvtxVerbosity](#) () const noexcept
Get the NVTX verbosity of the execution context.
- void [setAuxStreams](#) (cudaStream_t *auxStreams, int32_t nbStreams) noexcept
Set the auxiliary streams that TensorRT should launch kernels on in the next [enqueueV3\(\)](#) call.

Protected Attributes

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

Additional Inherited Members

9.59.1 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.

9.59.2 Constructor & Destructor Documentation

9.59.2.1 ~IExecutionContext()

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

9.59.3 Member Function Documentation

9.59.3.1 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\(bindingIndex,dimensions\)](#)

9.59.3.2 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\(bindingIndex\)](#)

9.59.3.3 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.

9.59.3.4 enqueue()

```
TRT_DEPRECATED bool nvinfer1::IExecutionContext::enqueue (
    int32_t batchSize,
    void *const * bindings,
    cudaStream_t stream,
    cudaEvent_t * inputConsumed ) [inline], [noexcept]
```

Enqueue inference of a batch on a stream.

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.

9.59.3.5 enqueueV2()

```
TRT_DEPRECATED bool nvinfer1::IExecutionContext::enqueueV2 (
    void *const * bindings,
    cudaStream_t stream,
    cudaEvent_t * inputConsumed ) [inline], [noexcept]
```

Enqueue inference on a stream.

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.

9.59.3.6 enqueueV3()

```
bool nvinfer1::IExecutionContext::enqueueV3 (
    cudaStream_t stream ) [inline], [noexcept]
```

Enqueue inference on a stream.

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.

9.59.3.7 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()`

9.59.3.8 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\(\)](#)

9.59.3.9 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\(\)](#)

9.59.3.10 `getDebugSync()`

```
bool nvinfer1::IExecutionContext::getDebugSync ( ) const [inline], [noexcept]
```

Get the debug sync flag.

See also

[setDebugSync\(\)](#)

9.59.3.11 `getEngine()`

```
ICudaEngine const & nvinfer1::IExecutionContext::getEngine ( ) const [inline], [noexcept]
```

Get the associated engine.

See also

[ICudaEngine](#)

9.59.3.12 getEnqueueEmitsProfile()

```
bool nvinfer1::IExecutionContext::getEnqueueEmitsProfile ( ) const [inline], [noexcept]
```

Get the enqueueEmitsProfile state.

Returns

The enqueueEmitsProfile state.

See also

[IExecutionContext::setEnqueueEmitsProfile\(\)](#)

9.59.3.13 getErrorRecorder()

```
IErrRecorder * nvinfer1::IExecutionContext::getErrorRecorder ( ) const [inline], [noexcept]
```

Get the ErrRecorder 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\(\)](#)

9.59.3.14 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.

9.59.3.15 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.

9.59.3.16 `getName()`

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

Return the name of the execution context.

See also

[setName\(\)](#)

9.59.3.17 `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\(\)](#)

9.59.3.18 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.

9.59.3.19 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](#)

9.59.3.20 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\(\)](#)

9.59.3.21 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](#)

9.59.3.22 getProfiler()

```
IProfiler * nvinfer1::IExecutionContext::getProfiler ( ) const [inline], [noexcept]
```

Get the profiler.

See also

[IProfiler setProfiler\(\)](#)

9.59.3.23 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 ICudaEngine::isShapeBinding(bindingIndex) 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 getBindingDimensions(bindingIndex).

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\(\)](#)

9.59.3.24 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\(\)](#)

9.59.3.25 `getTemporaryStorageAllocator()`

```
IGpuAllocator * nvinfer1::IExecutionContext::getTemporaryStorageAllocator ( ) const [inline],
[noexcept]
```

Get allocator set by `setTemporaryStorageAllocator`.

Returns a nullptr if a nullptr was passed with `setTemporaryStorageAllocator()`.

9.59.3.26 `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\(\)](#)

9.59.3.27 `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.

9.59.3.28 `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.

9.59.3.29 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.

9.59.3.30 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\(\)](#)

9.59.3.31 setAuxStreams()

```
void nvinfer1::IExecutionContext::setAuxStreams (
    cudaStream_t * auxStreams,
    int32_t nbStreams ) [inline], [noexcept]
```

Set the auxiliary streams that TensorRT should launch kernels on in the next [enqueueV3\(\)](#) call.

If set, TensorRT will launch the kernels that are supposed to run on the auxiliary streams using the streams provided by the user with this API. If this API is not called before the [enqueueV3\(\)](#) call, then TensorRT will use the auxiliary streams created by TensorRT internally.

TensorRT will always insert event synchronizations between the main stream provided via [enqueueV3\(\)](#) call and the auxiliary streams:

- At the beginning of the [enqueueV3\(\)](#) call, TensorRT will make sure that all the auxiliary streams wait on the activities on the main stream.
- At the end of the [enqueueV3\(\)](#) call, TensorRT will make sure that the main stream wait on the activities on all the auxiliary streams.

Parameters

<i>auxStreams</i>	The pointer to an array of <code>cudaStream_t</code> with the array length equal to <code>nbStreams</code> .
<i>nbStreams</i>	The number of auxiliary streams provided. If <code>nbStreams</code> is greater than <code>engine->getNbAuxStreams()</code> , then only the first <code>engine->getNbAuxStreams()</code> streams will be used. If <code>nbStreams</code> is less than <code>engine->getNbAuxStreams()</code> , such as setting <code>nbStreams</code> to 0, then TensorRT will use the provided streams for the first <code>nbStreams</code> auxiliary streams, and will create additional streams internally for the rest of the auxiliary streams.

Note

The provided auxiliary streams must not be the default stream and must all be different to avoid deadlocks.

See also

[enqueueV3\(\)](#), [IBuilderConfig::setMaxAuxStreams\(\)](#), [ICudaEngine::getNbAuxStreams\(\)](#)

9.59.3.32 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\(\)](#)

9.59.3.33 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\(\)](#)

9.59.3.34 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\(\)](#)

9.59.3.35 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\(\)](#)

9.59.3.36 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\(\)](#)

9.59.3.37 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.

9.59.3.38 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.

9.59.3.39 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\(\)](#)

9.59.3.40 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\(\)](#)

9.59.3.41 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\(\)](#)

9.59.3.42 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\(\)](#)

9.59.3.43 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\(\)](#)

9.59.3.44 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 <code>cudaMemcpyAsync</code> 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\(\)](#)

9.59.3.45 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](#)

9.59.3.46 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](#)

9.59.3.47 setProfiler()

```
void nvinfer1::IExecutionContext::setProfiler (
    IProfiler * profiler ) [inline], [noexcept]
```

Set the profiler.

See also

[IProfiler getProfiler\(\)](#)

9.59.3.48 setTemporaryStorageAllocator()

```
bool nvinfer1::IExecutionContext::setTemporaryStorageAllocator (
    IGpuAllocator * allocator ) [inline], [noexcept]
```

Specify allocator to use for internal temporary storage.

This allocator is used only by [enqueueV3\(\)](#) for temporary storage whose size cannot be predicted ahead of [enqueueV3\(\)](#). It is not used for output tensors, because memory allocation for those is allocated by the allocator set by [setOutputAllocator\(\)](#). All memory allocated is freed by the time [enqueueV3\(\)](#) returns.

Parameters

<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\(\)](#)

9.59.3.49 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\(\)](#) [IOOutputAllocator](#)

9.59.4 Member Data Documentation

9.59.4.1 mImpl

```
apiv::VExecutionContext* nvinfer1::IExecutionContext::mImpl [protected]
```

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

- [NvInferRuntime.h](#)

9.60 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
Enqueue inference of a batch on a stream.
- `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` (`RuntimeErrorInformation` *const buffer) noexcept=0
Set error buffer output for runtime errors.
- virtual `RuntimeErrorInformation` * `getErrorBuffer` () const noexcept=0
Get error buffer output for runtime errors.
- virtual `Dims` `getTensorStrides` (`AsciiChar` const *const tensorName) const noexcept=0
Return the strides of the buffer for the given tensor name.
- virtual bool `setInputTensorAddress` (`AsciiChar` const *const tensorName, void const *const data) noexcept=0
Set memory address for given input tensor.
- virtual bool `setOutputTensorAddress` (`AsciiChar` const *const tensorName, void *const data) noexcept=0
Set memory address for given output tensor.
- virtual bool `setInputConsumedEvent` (cudaEvent_t const event) noexcept=0
Set the event to mark inputs as consumed.
- virtual cudaEvent_t `getInputConsumedEvent` () const noexcept=0
Return the event associated with consuming the input.
- virtual void const * `getInputTensorAddress` (`AsciiChar` const *const tensorName) const noexcept=0
Get memory address for given input tensor.
- virtual void * `getOutputTensorAddress` (`AsciiChar` const *const tensorName) const noexcept=0
Get memory address for given output tensor.
- virtual bool `enqueueV3` (cudaStream_t const stream) noexcept=0
Enqueue inference on a stream.

9.60.1 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.

9.60.2 Constructor & Destructor Documentation

9.60.2.1 IExecutionContext() [1/3]

```
nvinfer1::safe::IExecutionContext::IExecutionContext ( ) [default]
```

9.60.2.2 ~IExecutionContext()

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

9.60.2.3 IExecutionContext() [2/3]

```
nvinfer1::safe::IExecutionContext::IExecutionContext (
    IExecutionContext const & ) [delete]
```

9.60.2.4 IExecutionContext() [3/3]

```
nvinfer1::safe::IExecutionContext::IExecutionContext (
    IExecutionContext && ) [delete]
```

9.60.3 Member Function Documentation**9.60.3.1 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]
```

Enqueue inference of a batch on a stream.

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 device memory pointers to input and output buffers for the network, which must be of length <code>getEngine().getNbBindings()</code> . Users are responsible for ensuring that the buffer size for each binding has at least the expected length, which is the product of the tensor dimensions (with the vectorized dimension padded to a multiple of the vector length) times the data type size.
<i>stream</i>	A CUDA stream on which the inference kernels will be enqueued. Must be a valid CUDA stream.
<i>inputConsumed</i>	An optional event that will be signaled when the input buffers can be refilled with new data. Must be either nullptr or a valid CUDA event.

Returns

True if the kernels were enqueued successfully, else false. Errors may include but not be limited to:

- Internal errors during executing one engine layer
- CUDA errors
- Invalid input parameters such as nullptr being passed for the bindings.

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

9.60.3.2 enqueueV3()

```
virtual bool nvinfer1::safe::IExecutionContext::enqueueV3 (
    cudaStream_t const stream ) [pure virtual], [noexcept]
```

Enqueue inference on a stream.

Modifying or releasing memory that has been registered for the tensors before stream synchronization or the event passed to `setInputConsumedEvent` has been signaled 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. Errors may include but not be limited to:

- Internal errors during executing one engine layer
- CUDA errors
- Some input or output tensor addresses have not been set.

Usage considerations

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

9.60.3.3 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

9.60.3.4 `getErrorBuffer()`

```
virtual RuntimeErrorInformation * nvinfer1::safe::IExecutionContext::getErrorBuffer ( ) const  
[pure virtual], [noexcept]
```

Get error buffer output for runtime errors.

Returns

Pointer to device memory to use as runtime error buffer or nullptr if not set.

See also

[setErrorBuffer\(\)](#)

Usage considerations

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

9.60.3.5 `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, or nullptr if the error recorder has been deregistered or not set.

See also

[setErrorRecorder\(\)](#)

Usage considerations

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

9.60.3.6 getInputConsumedEvent()

```
virtual cudaEvent_t nvinfer1::safe::IExecutionContext::getInputConsumedEvent ( ) const [pure virtual], [noexcept]
```

Return the event associated with consuming the input.

Returns

The CUDA event that was passed to [setInputConsumedEvent\(\)](#). nullptr will be returned if the event is not set.

Usage considerations

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

9.60.3.7 getInputTensorAddress()

```
virtual void const * nvinfer1::safe::IExecutionContext::getInputTensorAddress (
    AsciiChar const *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 NULL terminated and have a length of 1024 bytes or less including the NULL terminator.

Returns

The device memory address for the given input tensor. nullptr will be returned if

- name is not the name of an input tensor, or
- name is nullptr, or
- name exceeds the string length limit, or
- the memory address for the given input tensor is not set.

Usage considerations

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

9.60.3.8 getName()

```
virtual AsciiChar const * nvinfer1::safe::IExecutionContext::getName ( ) const [pure virtual],
[noexcept]
```

Return the name of the execution context.

Returns

The name that was passed to [setName\(\)](#), as a NULL-terminated string of 1024 bytes or less including the NULL terminator. An empty string will be returned as the default value.

See also

[setName\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.60.3.9 getOutputTensorAddress()

```
virtual void * nvinfer1::safe::IExecutionContext::getOutputTensorAddress (
    AsciiChar const *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 NULL terminated and have a length of 1024 bytes or less including the NULL terminator.

Returns

The device memory address for the given output tensor. Return `nullptr` if

- name is not the name of an output tensor, or
- name is `nullptr`, or
- name exceeds the string length limit, or
- the memory address for the given output tensor is not set.

Usage considerations

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

9.60.3.10 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, which must be in the range (between 0 and getEngine().getNbBindings()-1).

Returns

The strides of the tensor corresponding to `bindingIndex` if the index is in range, or an invalid value of [Dims](#){-1, {}} if it is out of range.

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

9.60.3.11 getTensorStrides()

```
virtual Dims nvinfer1::safe::IExecutionContext::getTensorStrides (
    AsciiChar const *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. Elements are vectors (for a vectorized format) or scalars (for a scalar format). 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 NULL terminated and have a length of 1024 bytes or less including the NULL terminator.

Returns

The strides of the buffer for the given tensor name. `Dims{-1, {}}` will be returned if

- name is not the name of an input or output tensor, or
- name is nullptr, or
- name exceeds the string length limit.

Usage considerations

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

9.60.3.12 operator=() [1/2]

```
IExecutionContext & nvinfer1::safe::IExecutionContext::operator= (
    IExecutionContext && ) & [delete]
```

9.60.3.13 operator=() [2/2]

```
IExecutionContext & nvinfer1::safe::IExecutionContext::operator= (
    IExecutionContext const & ) & [delete]
```

9.60.3.14 setDeviceMemory()

```
virtual void nvinfer1::safe::IExecutionContext::setDeviceMemory (
    void *const memory ) [pure virtual], [noexcept]
```

Set the device memory for use by this execution context.

Parameters

<i>memory</i>	The start address of a device memory buffer whose size in bytes must be at least the value returned by getEngine().getDeviceMemorySize() .

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.

If the execution context has been created by calling `createExecutionContext()`, this function must not be used and will fail with an error message if called.

See also

[safe::ICudaEngine::getDeviceMemorySize\(\)](#) [safe::ICudaEngine::createExecutionContextWithoutDeviceMemory\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.60.3.15 setErrorBuffer()

```
virtual void nvinfer1::safe::IExecutionContext::setErrorBuffer (
    RuntimeErrorInformation *const buffer ) [pure virtual], [noexcept]
```

Set error buffer output for runtime errors.

The error buffer output must be allocated in device memory and will be used for subsequent calls to [enqueueV2\(\)](#) or [enqueueV3\(\)](#). 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 [RuntimeErrorInformation](#) struct.

Warning

The buffer is written if reportable errors are encountered during network execution. Releasing the buffer before network execution is complete will result in undefined behavior. Accessing the memory before network execution is complete may not correctly capture the error state.

Parameters

<i>buffer</i>	The device memory address of the runtime error information buffer.

See also

[getErrorBuffer\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.60.3.16 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. If the recorder is set to `nullptr`, the error code [ErrorCode::KINVALID_ARGUMENT](#) will be emitted if the recorder has been registered. The lifetime of the error recorder object must exceed the lifetime of the execution context.

Parameters

<i>recorder</i>	Either a pointer to a valid error recorder object to register with this interface, or nullptr to deregister the current recorder.

See also

[getErrorRecorder\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.60.3.17 setInputConsumedEvent()

```
virtual bool nvinfer1::safe::IExecutionContext::setInputConsumedEvent (
    cudaEvent_t const event ) [pure virtual], [noexcept]
```

Set the event to mark inputs as consumed.

Passing event==nullptr removes whatever event was set, if any.

Parameters

<i>event</i>	The CUDA event that is signaled after all input tensors have been consumed, or nullptr to remove an event that was previously set.

Returns

True on success, false if an error occurred.

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.60.3.18 setInputTensorAddress()

```
virtual bool nvinfer1::safe::IExecutionContext::setInputTensorAddress (
    AsciiChar const *const tensorName,
    void const *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 input tensor data, which is device memory owned by the user. Users are responsible for ensuring that the buffer size has at least the expected length, which is the product of the tensor dimensions (with the vectorized dimension padded to a multiple of the vector length) times the data type size.

Warning

The string *tensorName* must be NULL terminated and have a length of 1024 bytes or less including the NULL terminator.

The data pointer must have 256-byte alignment.

Returns

True on success, false if

- name is not the name of an input tensor, or
- name is nullptr, or
- name exceeds the string length limit, or
- pointer to the const data is nullptr or not correctly aligned.

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.60.3.19 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 NULL terminated and have a length of 1024 bytes or less including the NULL terminator. Otherwise, the operation will not change the execution context name, and an error message will be recorded via the error recorder.

See also

[getName\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.60.3.20 setOutputTensorAddress()

```
virtual bool nvinfer1::safe::IExecutionContext::setOutputTensorAddress (
    AsciiChar const *const tensorName,
    void *const 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 output tensor data, which is device memory owned by the user. Users are responsible for ensuring that the buffer size has at least the expected length, which is the product of the tensor dimensions (with the vectorized dimension padded to a multiple of the vector length) times the data type size.

Warning

The string *tensorName* must be NULL terminated and have a length of 1024 bytes or less including the NULL terminator.

The data pointer must have 256-byte alignment.

Returns

True on success. Return false if

- name is not the name of an output tensor, or
- name is nullptr, or
- name exceeds the string length limit, or
- pointer to data is nullptr or not aligned.

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

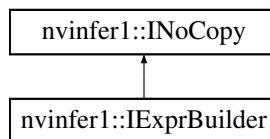
The documentation for this class was generated from the following file:

- [NvInferSafeRuntime.h](#)

9.61 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](#)

9.61.1 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](#)

9.61.2 Constructor & Destructor Documentation

9.61.2.1 ~IExprBuilder()

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

9.61.3 Member Function Documentation

9.61.3.1 constant()

```
IDimensionExpr const * nvinfer1::IExprBuilder::constant (
    int32_t value ) [inline], [noexcept]
```

Return pointer to [IDimensionExpr](#) for given value.

9.61.3.2 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](#).

9.61.4 Member Data Documentation

9.61.4.1 mImpl

```
apiv::VExprBuilder* nvinfer1::IExprBuilder::mImpl [protected]
```

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

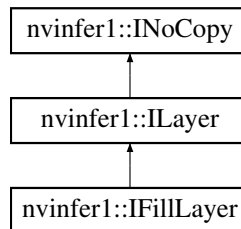
- [NvInferRuntime.h](#)

9.62 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](#)

9.62.1 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.

9.62.2 Constructor & Destructor Documentation

9.62.2.1 ~IFillLayer()

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

9.62.3 Member Function Documentation

9.62.3.1 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](#)

9.62.3.2 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](#)

9.62.3.3 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](#)

9.62.3.4 getOperation()

```
FillOperation nvinfer1::IFillLayer::getOperation ( ) const [inline], [noexcept]
```

Get the fill operation for the layer.

See also

[setOperation\(\)](#), [FillOperation](#)

9.62.3.5 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](#)

9.62.3.6 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 kLINSPEACE | the delta value, defaults to 1.0; kRANDOM_UNIFORM | the maximal value, defaults to 1.0; kRANDOM_NORMAL | the standard deviation of the normal distribution, default is 1.0;

If a third input had been used to create this layer, that input is reset to null by this method.

See also

[getBeta](#)

9.62.3.7 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](#)

9.62.3.8 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.

9.62.3.9 setOperation()

```
void nvinfer1::IFillLayer::setOperation (
    FillOperation op ) [inline], [noexcept]
```

Set the fill operation for the layer.

See also

[getOperation\(\)](#), [FillOperation](#)

9.62.4 Member Data Documentation

9.62.4.1 mImpl

```
apiv::VFillLayer* nvinfer1::IFillLayer::mImpl [protected]
```

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

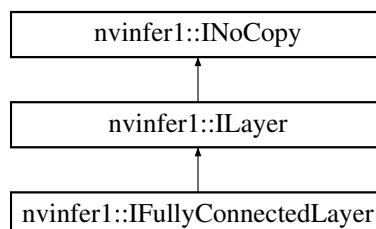
- [NvInfer.h](#)

9.63 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](#)

9.63.1 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](#).

9.63.2 Constructor & Destructor Documentation

9.63.2.1 ~IFullyConnectedLayer()

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

9.63.3 Member Function Documentation

9.63.3.1 getBiasWeights()

```
Weights nvinfer1::IFullyConnectedLayer::getBiasWeights ( ) const [inline], [noexcept]
```

Get the bias weights.

See also

[setBiasWeightsWeights\(\)](#)

9.63.3.2 getKernelWeights()

```
Weights nvinfer1::IFullyConnectedLayer::getKernelWeights ( ) const [inline], [noexcept]
```

Get the kernel weights.

See also

[setKernelWeights\(\)](#)

9.63.3.3 getNbOutputChannels()

```
int32_t nvinfer1::IFullyConnectedLayer::getNbOutputChannels ( ) const [inline], [noexcept]
```

Get the number of output channels K from the fully connected layer.

See also

[setNbOutputChannels\(\)](#)

9.63.3.4 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\(\)](#)

9.63.3.5 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.

9.63.3.6 [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\(\)](#)

9.63.3.7 [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\(\)](#)

9.63.4 Member Data Documentation

9.63.4.1 mImpl

```
apiv::VFullyConnectedLayer* nvinfer1::IFullyConnectedLayer::mImpl [protected]
```

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

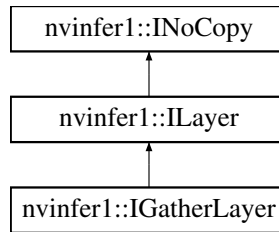
- [NvInfer.h](#)

9.64 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`

9.64.1 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.

9.64.2 Constructor & Destructor Documentation

9.64.2.1 ~IGatherLayer()

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

9.64.3 Member Function Documentation

9.64.3.1 `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\(\)](#)

9.64.3.2 `getMode()`

```
GatherMode nvinfer1::IGatherLayer::getMode ( ) const [inline], [noexcept]
```

Get the gather mode.

See also

[setMode\(\)](#)

9.64.3.3 `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\(\)](#)

9.64.3.4 `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\(\)](#)

9.64.3.5 setMode()

```
void nvinfer1::IGatherLayer::setMode (
    GatherMode mode ) [inline], [noexcept]
```

Set the gather mode.

See also

[getMode\(\)](#)

9.64.3.6 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\(\)](#)

9.64.4 Member Data Documentation

9.64.4.1 mImpl

```
apiv::VGatherLayer* nvinfer1::IGatherLayer::mImpl [protected]
```

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

- [NvInfer.h](#)

9.65 nvinfer1::IGpuAllocator Class Reference

Application-implemented class for controlling allocation on the GPU.

```
#include <NvInferRuntimeBase.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

9.65.1 Detailed Description

Application-implemented class for controlling allocation on the GPU.

Warning

The lifetime of an [IGpuAllocator](#) object must exceed that of all objects that use it.

9.65.2 Constructor & Destructor Documentation

9.65.2.1 ~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.

9.65.2.2 IGpuAllocator()

```
nvinfer1::IGpuAllocator::IGpuAllocator ( ) [default]
```

9.65.3 Member Function Documentation

9.65.3.1 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 block required (in bytes).
<i>alignment</i>	The required alignment of memory. Alignment will be zero or a power of 2 not exceeding the alignment guaranteed by <code>cudaMalloc</code> . Thus this allocator can be safely implemented with <code>cudaMalloc/cudaFree</code> . 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.

Returns

If the allocation was successful, the start address of a device memory block of the requested size. If an allocation request of size 0 is made, `nullptr` must be returned. If an allocation request cannot be satisfied, `nullptr` must be returned. If a non-null address is returned, it is guaranteed to have the specified alignment.

Note

The implementation must guarantee thread safety for concurrent `allocate/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.

9.65.3.2 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>	A memory address that was previously returned by an allocate() or reallocate() call of the same allocator object.

Returns

True if the acquired memory is released successfully.

Note

The implementation must guarantee thread safety for concurrent allocate/free/reallocate/deallocate requests.

If user-implemented [free\(\)](#) might hit an error condition, the user must 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.

9.65.3.3 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>	A memory address that was previously returned by calling allocate() or reallocate() on the same allocator object, or nullptr.

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.

9.65.3.4 [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, which will have been returned by previously calling allocate() or reallocate() on the same object.
<i>alignment</i>	The alignment used by the original allocation. This will be the same value that was previously passed to the allocate() or reallocate() call that returned baseAddr.
<i>newSize</i>	The new memory size required (in bytes).

Returns

The address of the reallocated memory, or nullptr. If a non-null address is returned, it is guaranteed to have the specified alignment.

Note

The implementation must guarantee thread safety for concurrent allocate/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:

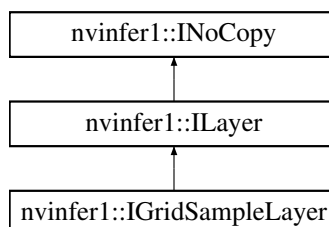
- [NvInferRuntimeBase.h](#)

9.66 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](#)

9.66.1 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.

9.66.2 Constructor & Destructor Documentation

9.66.2.1 [~IGridSampleLayer](#)()

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

9.66.3 Member Function Documentation

9.66.3.1 `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.

9.66.3.2 `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.

9.66.3.3 `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.

9.66.3.4 setAlignCorners()

```
void nvinfer1::IGridSampleLayer::setAlignCorners (
    bool alignCorners ) [inline], [noexcept]
```

Set the align corners mode.

See also

[getAlignCorners\(\)](#)

9.66.3.5 setInterpolationMode()

```
void nvinfer1::IGridSampleLayer::setInterpolationMode (
    InterpolationMode mode ) [inline], [noexcept]
```

Set the grid sample interpolation mode.

See also

[getInterpolationMode\(\)](#)

9.66.3.6 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.

9.66.4 Member Data Documentation

9.66.4.1 mImpl

```
apiv::VGridSampleLayer* nvinfer1::IGridSampleLayer::mImpl [protected]
```

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

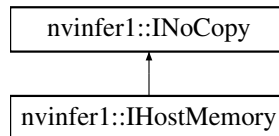
- [NvInfer.h](#)

9.67 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

9.67.1 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.

9.67.2 Constructor & Destructor Documentation

9.67.2.1 ~IHostMemory()

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

9.67.3 Member Function Documentation

9.67.3.1 data()

```
void * nvinfer1::IHostMemory::data ( ) const [inline], [noexcept]
```

A pointer to the raw data that is owned by the library.

9.67.3.2 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.

9.67.3.3 size()

```
std::size_t nvinfer1::IHostMemory::size ( ) const [inline], [noexcept]
```

The size in bytes of the data that was allocated.

9.67.3.4 type()

```
DataType nvinfer1::IHostMemory::type ( ) const [inline], [noexcept]
```

The type of the memory that was allocated.

9.67.4 Member Data Documentation

9.67.4.1 mImpl

```
apiv::VHostMemory* nvinfer1::IHostMemory::mImpl [protected]
```

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

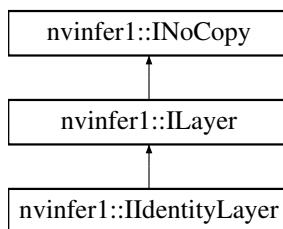
- [NvInferRuntime.h](#)

9.68 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

9.68.1 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.

9.68.2 Constructor & Destructor Documentation

9.68.2.1 ~IIdentityLayer()

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

9.68.3 Member Data Documentation

9.68.3.1 mImpl

```
apiv::VIdentityLayer* nvinfer1::IIdentityLayer::mImpl [protected]
```

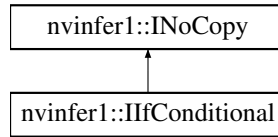
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.69 nvinfer1::IfConditional Class Reference

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

Inheritance diagram for nvinfer1::IfConditional:



Public Member Functions

- [IConditionLayer](#) * [setCondition](#) ([ITensor](#) &condition) noexcept
Set the condition tensor for this If-Conditional construct.
- [IfConditionalOutputLayer](#) * [addOutput](#) ([ITensor](#) &>trueSubgraphOutput, [ITensor](#) &>falseSubgraphOutput) noexcept
Add an If-conditional output.
- [IfConditionalInputLayer](#) * [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 [~IfConditional](#) () noexcept=default

Protected Attributes

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

9.69.1 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.

9.69.2 Constructor & Destructor Documentation

9.69.2.1 ~IIfConditional()

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

9.69.3 Member Function Documentation

9.69.3.1 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](#)

9.69.3.2 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](#)

9.69.3.3 getName()

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

Return the name of the conditional.

See also

[setName\(\)](#)

9.69.3.4 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](#)

9.69.3.5 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\(\)](#)

9.69.4 Member Data Documentation

9.69.4.1 mImpl

```
apiv::VIfConditional* nvinfer1::IIfConditional::mImpl [protected]
```

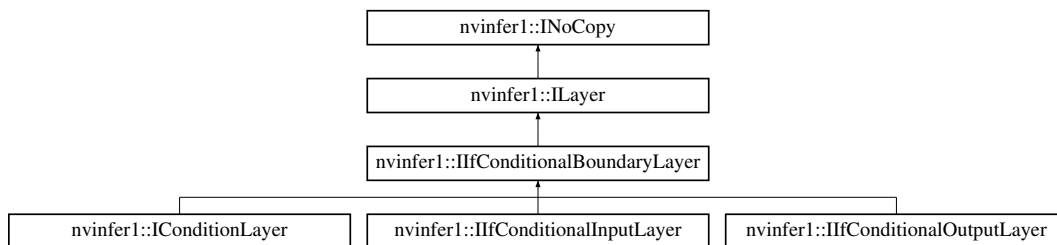
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.70 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`

9.70.1 Detailed Description

This is a base class for Conditional boundary layers.

Boundary layers are used to demarcate the boundaries of Conditionals.

9.70.2 Constructor & Destructor Documentation

9.70.2.1 `~IIfConditionalBoundaryLayer()`

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

9.70.3 Member Function Documentation

9.70.3.1 `getConditional()`

```
IIfConditional * nvinfer1::IIfConditionalBoundaryLayer::getConditional ( ) const [inline], [noexcept]
```

Return pointer to the [IIfConditional](#) associated with this boundary layer.

9.70.4 Member Data Documentation

9.70.4.1 `mBoundary`

```
apiv::VConditionalBoundaryLayer* nvinfer1::IIfConditionalBoundaryLayer::mBoundary [protected]
```

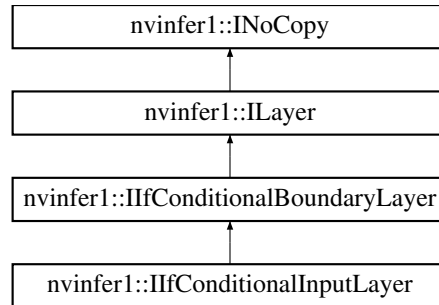
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.71 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

9.71.1 Detailed Description

This layer represents an input to an [IIfConditional](#).

9.71.2 Constructor & Destructor Documentation

9.71.2.1 ~IIfConditionalInputLayer()

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

9.71.3 Member Data Documentation

9.71.3.1 mImpl

```
apiv::VConditionalInputLayer* nvinfer1::IIfConditionalInputLayer::mImpl [protected]
```

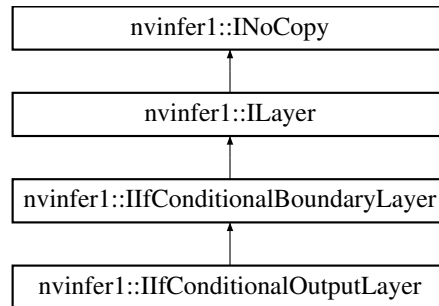
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.72 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

9.72.1 Detailed Description

This layer represents an output of an [IIfConditional](#).

An [IIfConditionalOutputLayer](#) has exactly one output.

9.72.2 Constructor & Destructor Documentation

9.72.2.1 ~IIfConditionalOutputLayer()

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

9.72.3 Member Data Documentation

9.72.3.1 mImpl

```
apiv::VConditionalOutputLayer* nvinfer1::IIfConditionalOutputLayer::mImpl [protected]
```

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

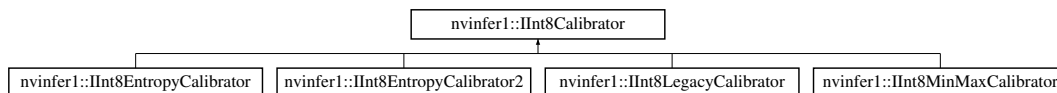
- [NvInfer.h](#)

9.73 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

9.73.1 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.

9.73.2 Constructor & Destructor Documentation

9.73.2.1 ~IInt8Calibrator()

```
virtual nvinfer1::IInt8Calibrator::~~IInt8Calibrator ( ) [virtual], [default], [noexcept]
```

9.73.3 Member Function Documentation

9.73.3.1 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](#).

9.73.3.2 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\(\)](#)

9.73.3.3 getBatchSize()

```
virtual int32_t nvinfer1::IInt8Calibrator::getBatchSize ( ) const [pure virtual], [noexcept]
```

Get the batch size used for calibration batches.

Returns

The batch size.

9.73.3.4 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.

9.73.3.5 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\(\)](#)

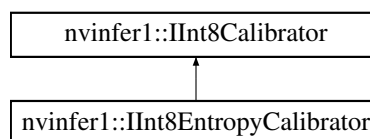
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.74 nvinfer1::IInt8EntropyCalibrator Class Reference

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

Inheritance diagram for nvinfer1::IInt8EntropyCalibrator:



Public Member Functions

- [CalibrationAlgoType getAlgorithm \(\)](#) noexcept override
- [virtual ~IInt8EntropyCalibrator \(\)](#) noexcept=default

9.74.1 Detailed Description

Entropy calibrator. This is the Legacy Entropy calibrator. It is less complicated than the legacy calibrator and produces better results.

9.74.2 Constructor & Destructor Documentation

9.74.2.1 ~IInt8EntropyCalibrator()

```
virtual nvinfer1::IInt8EntropyCalibrator::~~IInt8EntropyCalibrator ( ) [virtual], [default], [noexcept]
```

9.74.3 Member Function Documentation

9.74.3.1 getAlgorithm()

```
CalibrationAlgoType nvinfer1::IInt8EntropyCalibrator::getAlgorithm ( ) [inline], [override],  
[virtual], [noexcept]
```

Signal that this is the entropy calibrator.

Implements [nvinfer1::IInt8Calibrator](#).

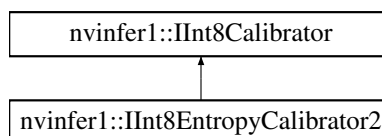
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.75 nvinfer1::IInt8EntropyCalibrator2 Class Reference

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

Inheritance diagram for nvinfer1::IInt8EntropyCalibrator2:



Public Member Functions

- [CalibrationAlgoType](#) `getAlgorithm()` noexcept override
- virtual `~IInt8EntropyCalibrator2()` noexcept=default

9.75.1 Detailed Description

Entropy calibrator 2. This is the preferred calibrator. This is the required calibrator for DLA, as it supports per activation tensor scaling.

9.75.2 Constructor & Destructor Documentation

9.75.2.1 `~IInt8EntropyCalibrator2()`

```
virtual nvinfer1::IInt8EntropyCalibrator2::~~IInt8EntropyCalibrator2 ( ) [virtual], [default],
[noexcept]
```

9.75.3 Member Function Documentation

9.75.3.1 `getAlgorithm()`

```
CalibrationAlgoType nvinfer1::IInt8EntropyCalibrator2::getAlgorithm ( ) [inline], [override],
[virtual], [noexcept]
```

Signal that this is the entropy calibrator 2.

Implements [nvinfer1::IInt8Calibrator](#).

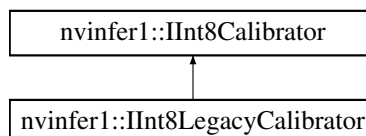
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.76 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

9.76.1 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.

9.76.2 Constructor & Destructor Documentation

9.76.2.1 [~IInt8LegacyCalibrator](#)()

```
virtual nvinfer1::IInt8LegacyCalibrator::~~IInt8LegacyCalibrator ( ) [virtual], [default], [noexcept]
```

9.76.3 Member Function Documentation

9.76.3.1 [getAlgorithm](#)()

```
CalibrationAlgoType nvinfer1::IInt8LegacyCalibrator::getAlgorithm ( ) [inline], [override], [virtual], [noexcept]
```

Signal that this is the legacy calibrator.

Implements [nvinfer1::IInt8Calibrator](#).

9.76.3.2 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.

9.76.3.3 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

9.76.3.4 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.

9.76.3.5 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\(\)](#)

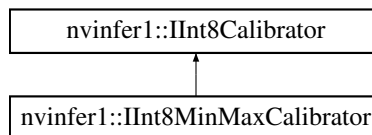
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.77 nvinfer1::IInt8MinMaxCalibrator Class Reference

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

Inheritance diagram for nvinfer1::IInt8MinMaxCalibrator:



Public Member Functions

- [CalibrationAlgoType getAlgorithm\(\)](#) noexcept override
- virtual [~IInt8MinMaxCalibrator\(\)](#) noexcept=default

9.77.1 Detailed Description

MinMax Calibrator. It supports per activation tensor scaling.

9.77.2 Constructor & Destructor Documentation

9.77.2.1 ~IInt8MinMaxCalibrator()

```
virtual nvinfer1::IInt8MinMaxCalibrator::~~IInt8MinMaxCalibrator ( ) [virtual], [default], [noexcept]
```

9.77.3 Member Function Documentation

9.77.3.1 getAlgorithm()

```
CalibrationAlgoType nvinfer1::IInt8MinMaxCalibrator::getAlgorithm ( ) [inline], [override], [virtual],
[noexcept]
```

Signal that this is the MinMax Calibrator.

Implements [nvinfer1::IInt8Calibrator](#).

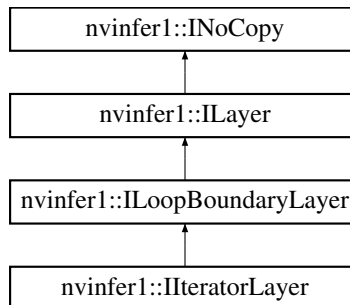
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.78 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](#)

9.78.1 Constructor & Destructor Documentation

9.78.1.1 ~IteratorLayer()

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

9.78.2 Member Function Documentation

9.78.2.1 getAxis()

```
int32_t nvinfer1::IteratorLayer::getAxis ( ) const [inline], [noexcept]
```

Get axis being iterated over.

9.78.2.2 getReverse()

```
bool nvinfer1::IteratorLayer::getReverse ( ) const [inline], [noexcept]
```

True if and only if reversing input.

9.78.2.3 setAxis()

```
void nvinfer1::IteratorLayer::setAxis (
    int32_t axis ) [inline], [noexcept]
```

Set axis to iterate over.

9.78.2.4 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.

9.78.3 Member Data Documentation

9.78.3.1 mImpl

```
apiv::VIteratorLayer* nvinfer1::IIteratorLayer::mImpl [protected]
```

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

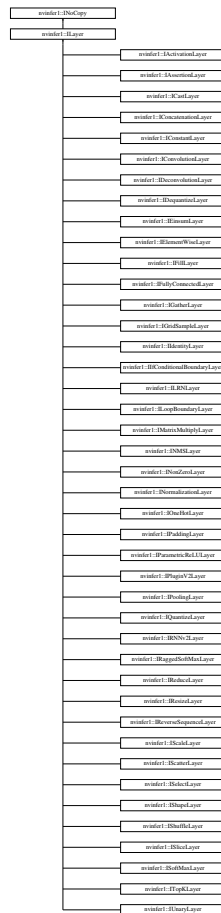
- [NvInfer.h](#)

9.79 nvinfer1::ILayer Class Reference

Base class for all layer classes in a network definition.

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

Inheritance diagram for nvinfer1::ILayer:



Public Member Functions

- `LayerType` `getType` () const noexcept
Return the type of a layer.
- void `setName` (char const *name) noexcept
Set the name of a layer.
- char const * `getName` () const noexcept
Return the name of a layer.
- int32_t `getNbInputs` () const noexcept
Get the number of inputs of a layer.
- `ITensor` * `getInput` (int32_t index) const noexcept
Get the layer input corresponding to the given index.
- int32_t `getNbOutputs` () const noexcept
Get the number of outputs of a layer.
- `ITensor` * `getOutput` (int32_t index) const noexcept
Get the layer output corresponding to the given index.
- void `setInput` (int32_t index, `ITensor` &tensor) noexcept
Replace an input of this layer with a specific tensor.
- 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
- void [setMetadata](#) (char const *metadata) noexcept
Set the metadata for this layer.
- char const * [getMetadata \(\)](#) const noexcept
Get the metadata of the layer.

Protected Member Functions

- virtual [~ILayer \(\)](#) noexcept=default

Protected Attributes

- apiv::VLayer * [mLayer](#)

9.79.1 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.

9.79.2 Constructor & Destructor Documentation

9.79.2.1 ~ILayer()

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

9.79.3 Member Function Documentation

9.79.3.1 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](#)).

9.79.3.2 getMetadata()

```
char const * nvinfer1::ILayer::getMetadata ( ) const [inline], [noexcept]
```

Get the metadata of the layer.

Returns

The metadata as a null-terminated C-style string. If [setMetadata\(\)](#) has not been called, an empty string "" will be returned as a default value.

See also

[setMetadata\(\)](#)

9.79.3.3 getName()

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

Return the name of a layer.

See also

[setName\(\)](#)

9.79.3.4 getNbInputs()

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

Get the number of inputs of a layer.

9.79.3.5 getNbOutputs()

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

Get the number of outputs of a layer.

9.79.3.6 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](#)).

9.79.3.7 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\(\)](#)

9.79.3.8 getPrecision()

```
DataType nvinfer1::ILayer::getPrecision ( ) const [inline], [noexcept]
```

get the computational precision of this layer

Returns

the computational precision

See also

[setPrecision\(\)](#) [precisionIsSet\(\)](#) [resetPrecision\(\)](#)

9.79.3.9 getType()

```
LayerType nvinfer1::ILayer::getType ( ) const [inline], [noexcept]
```

Return the type of a layer.

See also

[LayerType](#)

9.79.3.10 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\(\)](#)

9.79.3.11 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\(\)](#)

9.79.3.12 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\(\)](#)

9.79.3.13 resetPrecision()

```
void nvinfer1::ILayer::resetPrecision ( ) [inline], [noexcept]
```

reset the computational precision for this layer

See also

[setPrecision\(\)](#) [getPrecision\(\)](#) [precisionIsSet\(\)](#)

9.79.3.14 setInput()

```
void nvinfer1::ILayer::setInput (
    int32_t index,
    ITensor & tensor ) [inline], [noexcept]
```

Replace an input of this layer with a specific tensor.

Parameters

<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.

9.79.3.15 setMetadata()

```
void nvinfer1::ILayer::setMetadata (
    char const * metadata ) [inline], [noexcept]
```

Set the metadata for this layer.

The metadata is emitted in the JSON returned by [IEngineInspector](#) with `ProfilingVerbosity` set to `kDETAILED`.

Parameters

<i>metadata</i>	The per-layer metadata.

Warning

The string name must be null-terminated and be at most 4096 bytes including the terminator.

See also

[getMetadata\(\)](#)

[getLayerInformation\(\)](#)

9.79.3.16 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\(\)](#)

9.79.3.17 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\(\)](#)

9.79.3.18 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\(\)](#)

9.79.4 Member Data Documentation

9.79.4.1 mLayer

```
apiv::VLayer* nvinfer1::ILayer::mLayer [protected]
```

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

- [NvInfer.h](#)

9.80 nvinfer1::ILogger Class Reference

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

```
#include <NvInferRuntimeBase.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

9.80.1 Detailed Description

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

The logger used to create an instance of [IBuilder](#), [IRuntime](#) or [IRefitter](#) is used for all objects created through that interface. The logger must be valid until all objects created are released.

The Logger object implementation must be thread safe. All locking and synchronization is pushed to the interface implementation and TensorRT does not hold any synchronization primitives when calling the interface functions.

9.80.2 Member Enumeration Documentation

9.80.2.1 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.

9.80.3 Constructor & Destructor Documentation

9.80.3.1 ILogger()

```
nvinfer1::ILogger::ILogger ( ) [default]
```

9.80.3.2 ~ILogger()

```
virtual nvinfer1::ILogger::~ILogger ( ) [virtual], [default]
```


9.80.4 Member Function Documentation

9.80.4.1 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.

Warning

Loggers used in the safety certified runtime must set a maximum message length and truncate messages exceeding this length. It is up to the implementer of the derived class to define a suitable limit that will prevent buffer overruns, resource exhaustion, and other security vulnerabilities in their implementation. The TensorRT safety certified runtime will never emit messages longer than 1024 bytes.

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:

- [NvInferRuntimeBase.h](#)

9.81 nvinfer1::ILoggerFinder Class Reference

A virtual base class to find a logger. Allows a plugin to find an instance of a logger if it needs to emit a log message. A pointer to an instance of this class is passed to a plugin shared library on initialization when that plugin is serialized as part of a version-compatible plan. See the plugin chapter in the developer guide for details.

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

Public Member Functions

- virtual [ILogger](#) * [findLogger](#) ()=0

Get the logger used by the engine or execution context which called the plugin method.

Protected Member Functions

- virtual [~ILoggerFinder](#) ()=default

9.81.1 Detailed Description

A virtual base class to find a logger. Allows a plugin to find an instance of a logger if it needs to emit a log message. A pointer to an instance of this class is passed to a plugin shared library on initialization when that plugin is serialized as part of a version-compatible plan. See the plugin chapter in the developer guide for details.

9.81.2 Constructor & Destructor Documentation

9.81.2.1 ~ILoggerFinder()

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

9.81.3 Member Function Documentation

9.81.3.1 findLogger()

```
virtual ILogger * nvinfer1::ILoggerFinder::findLogger ( ) [pure virtual]
```

Get the logger used by the engine or execution context which called the plugin method.

Warning

Must be called from the thread in which the plugin method was called.

Returns

A pointer to the logger.

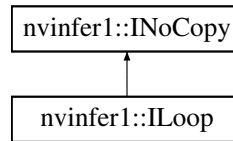
The documentation for this class was generated from the following file:

- [NvInferRuntime.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]
```

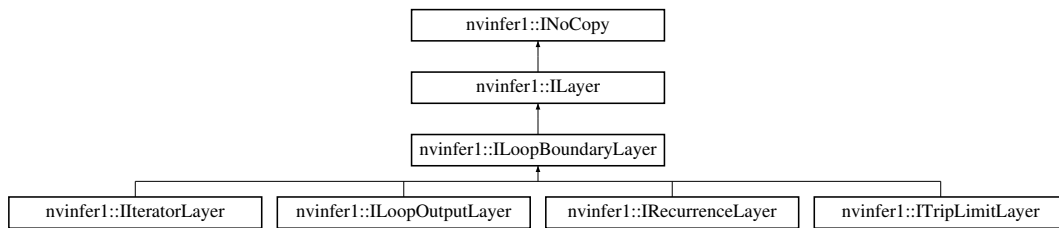
The documentation for this class was generated from the following file:

- [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]
```

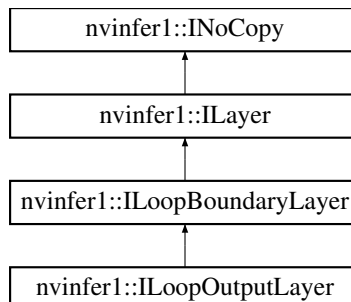
The documentation for this class was generated from the following file:

- [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 nvinfer1::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:

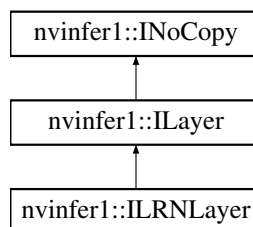
- [NvInfer.h](#)

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:

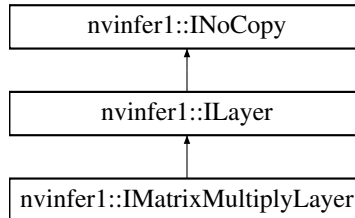
- [NvInfer.h](#)

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:

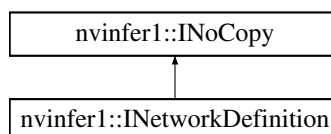
- [NvInfer.h](#)

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.
- virtual **IBuilder** & **getBuilder** () const noexcept
*Return the builder from which this **INetworkDefinition** was created.*

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 [RNNInputMode::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 [RNNOperation::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 [RNNDirection::kUNIDIRECTION](#), and `2x getLayerCount()` if `getDirection` is [RNNDirection::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 [ScaleMode::kUNIFORM](#), the number of weights equals 1. For [ScaleMode::kCHANNEL](#), the number of weights equals the channel dimension. For [ScaleMode::kELEMENTWISE](#), the number of weights equals the product of the last three dimensions of the input.

See also

[addScaleNd](#)

[IScaleLayer](#)

Warning

Int32 tensors are not valid input tensors.

Returns

The new Scale layer, or nullptr if it could not be created.

9.87.3.41 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 [ScaleMode::kUNIFORM](#), the number of weights equals 1. For [ScaleMode::kCHANNEL](#), the number of weights equals the channel dimension. For [ScaleMode::kELEMENTWISE](#), the number of weights equals the product of all input dimensions at channelAxis and beyond.

For example, if the inputs dimensions are [A,B,C,D,E,F], and channelAxis=2: For [ScaleMode::kUNIFORM](#), the number of weights is equal to 1. For [ScaleMode::kCHANNEL](#), the number of weights is C. For [ScaleMode::kELEMENTWISE](#), the number of weights is C*D*E*F.

channelAxis can also be set explicitly using setChannelAxis().

See also

[IScaleLayer](#)

setChannelAxis()

Warning

Int32 tensors are not valid input tensors.
Only 2D or 3D scale is supported.

Returns

The new Scale layer, or nullptr if it could not be created.

9.87.3.42 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>data</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.
<i>mode</i>	scatter mode.

See also

[IScatterLayer](#)

indices tensor data type must be [DataType::kINT32](#). *updates* tensor data type must be the same as *data*

Returns

The new Scatter layer, or nullptr if it could not be created.

9.87.3.43 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.

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

[IUnaryLayer](#)

Generally the input must have a floating-point type (or `kINT8` as a quantized float), except for the following operations:

- `kSIGN` accepts a floating-point or `Int32` tensor.
- `kNOT` requires a `Bool` tensor.

The input is a shape tensor if the output is a shape tensor.

Returns

The new unary layer, or `nullptr` if it could not be created

9.87.3.50 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 getBuilder()

virtual [IBuilder](#) & nvinfer1::INetworkDefinition::getBuilder () const [inline], [virtual], [noexcept]

Return the builder from which this [INetworkDefinition](#) was created.

See also

[IBuilder::createNetworkV2](#)

Returns

the builder

9.87.3.52 getErrorRecorder()

[IErrorRecorder](#) * nvinfer1::INetworkDefinition::getErrorRecorder () const [inline], [noexcept]

get the [ErrorRecorder](#) assigned to this interface.

Retrieves the assigned error recorder object for the given class. A nullptr will be returned if `setErrorRecorder` has not been called.

Returns

A pointer to the [IErrorRecorder](#) object that has been registered.

See also

[setErrorRecorder\(\)](#)

9.87.3.53 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\(\)](#)

9.87.3.54 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\(\)](#)

9.87.3.55 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.

9.87.3.56 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\(\)](#)

9.87.3.57 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\(\)](#)

9.87.3.58 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\(\)](#)

9.87.3.59 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\(\)](#)

9.87.3.60 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.

9.87.3.61 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`

9.87.3.62 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.

9.87.3.63 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\(\)](#)

9.87.3.64 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()`.

9.87.3.65 `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\(\)](#)

9.87.3.66 `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

9.87.3.67 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.

9.87.3.68 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\(\)](#)

9.87.3.69 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:

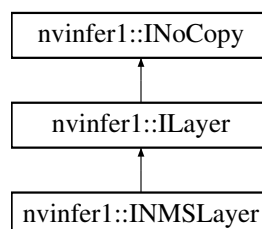
- [NvInfer.h](#)

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:

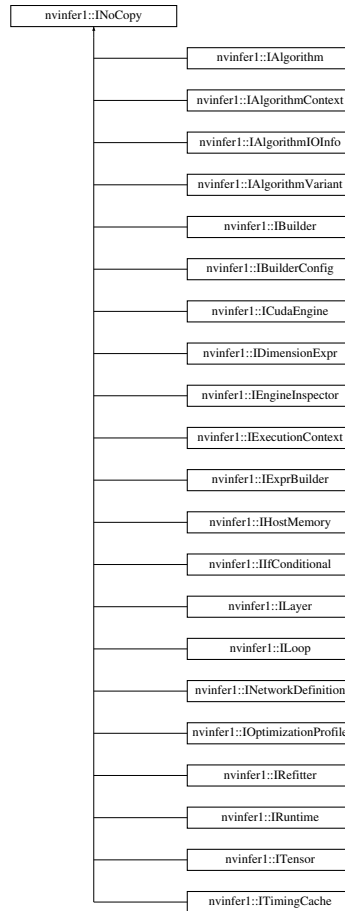
- [NvInfer.h](#)

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.

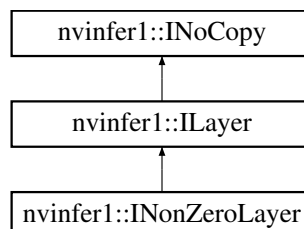
The documentation for this class was generated from the following file:

- [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:

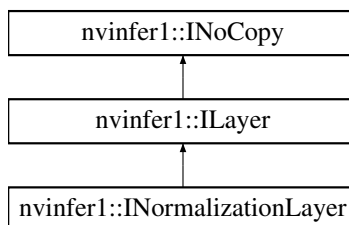
- [NvInfer.h](#)

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:

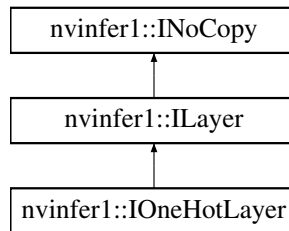
- [NvInfer.h](#)

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

the data type of the model.

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:

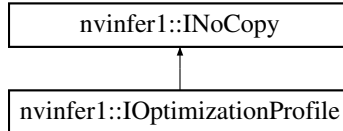
- [NvOnnxConfig.h](#)

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:

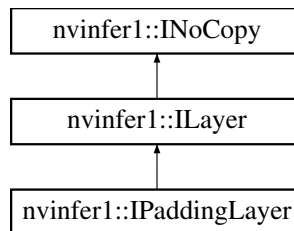
- [NvInferRuntime.h](#)

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:

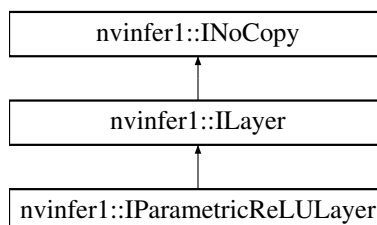
- [NvInfer.h](#)

9.98 nvinfer1::IParametricReLU Layer Class Reference

Layer that represents a parametric ReLU operation.

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

Inheritance diagram for nvinfer1::IParametricReLU Layer:



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
- virtual char const *const * [getUsedVCPluginLibraries](#) (int64_t &nbPluginLibs) const noexcept=0
Query the plugin libraries needed to implement operations used by the parser in a version-compatible engine.
- virtual void [setFlags](#) ([OnnxParserFlags](#) onnxParserFlags) noexcept=0
Set the parser flags.
- virtual [OnnxParserFlags](#) [getFlags](#) () const noexcept=0
Get the parser flags. Defaults to 0.
- virtual void [clearFlag](#) ([OnnxParserFlag](#) onnxParserFlag) noexcept=0
clear a parser flag.
- virtual void [setFlag](#) ([OnnxParserFlag](#) onnxParserFlag) noexcept=0
Set a single parser flag.
- virtual bool [getFlag](#) ([OnnxParserFlag](#) onnxParserFlag) const noexcept=0
Returns true if the parser flag is set.

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 clearFlag()

```
virtual void nvonnxparser::IParser::clearFlag (
    OnnxParserFlag onnxParserFlag ) [pure virtual], [noexcept]
```

clear a parser flag.

clears the parser flag from the enabled flags.

See also

[setFlags\(\)](#)

9.99.3.3 destroy()

```
virtual TRT\_DEPRECATED void nvonnxparser::IParser::destroy ( ) [pure virtual]
```

destroy this object

Warning

deprecated and planned on being removed in TensorRT 10.0

9.99.3.4 `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](#)

9.99.3.5 `getFlag()`

```
virtual bool nvonnxparser::IParser::getFlag (
    OnnxParserFlag onnxParserFlag ) const [pure virtual], [noexcept]
```

Returns true if the parser flag is set.

See also

[getFlags\(\)](#)

Returns

True if flag is set, false if unset.

9.99.3.6 `getFlags()`

```
virtual OnnxParserFlags nvonnxparser::IParser::getFlags ( ) const [pure virtual], [noexcept]
```

Get the parser flags. Defaults to 0.

Returns

The parser flags as a bitmask.

See also

[setFlags\(\)](#)

9.99.3.7 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](#)

9.99.3.8 getUsedVCPluginLibraries()

```
virtual char const *const * nvonnxparser::IParser::getUsedVCPluginLibraries (
    int64_t & nbPluginLibs ) const [pure virtual], [noexcept]
```

Query the plugin libraries needed to implement operations used by the parser in a version-compatible engine.

This provides a list of plugin libraries on the filesystem needed to implement operations in the parsed network. If you are building a version-compatible engine using this network, provide this list to `IBuilderConfig::setPluginsToSerialize` to serialize these plugins along with the version-compatible engine, or, if you want to ship these plugin libraries externally to the engine, ensure that `IPluginRegistry::loadLibrary` is used to load these libraries in the appropriate runtime before deserializing the corresponding engine.

Parameters

out	<i>nbPluginLibs</i>	Returns the number of plugin libraries in the array, or -1 if there was an error.

Returns

Array of `nbPluginLibs` C-strings describing plugin library paths on the filesystem if `nbPluginLibs > 0`, or `nullptr` otherwise. This array is owned by the [IParser](#), and the pointers in the array are only valid until the next call to [parse\(\)](#), [supportsModel\(\)](#), [parseFromFile\(\)](#), or [parseWithWeightDescriptors\(\)](#).

9.99.3.9 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\(\)](#)

9.99.3.10 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>onnxModelFile</i>	name
<i>verbosity</i>	Level

Returns

true if the model was parsed successfully

9.99.3.11 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\(\)](#)

9.99.3.12 setFlag()

```
virtual void nvonnxparser::IParser::setFlag (
    OnnxParserFlag onnxParserFlag ) [pure virtual], [noexcept]
```

Set a single parser flag.

Add the input parser flag to the already enabled flags.

See also

[setFlags\(\)](#)

9.99.3.13 setFlags()

```
virtual void nvonnxparser::IParser::setFlags (
    OnnxParserFlags onnxParserFlags ) [pure virtual], [noexcept]
```

Set the parser flags.

The flags are listed in the OnnxParserFlag enum.

Parameters

<i>OnnxParserFlag</i>	The flags used when parsing an ONNX model.

Note

This function will override the previous set flags, rather than bitwise ORing the new flag.

See also

[getFlags\(\)](#)

9.99.3.14 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

9.99.3.15 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:

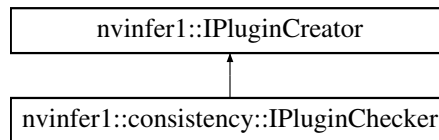
- [NvOnnxParser.h](#)

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:

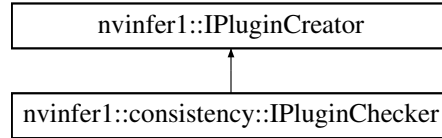
- [NvInferConsistency.h](#)

9.102 nvinfer1::IPluginCreator Class Reference

Plugin creator class for user implemented layers.

```
#include <NvInferRuntimePlugin.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 need to be passed to createPlugin.
- virtual `IPluginV2` * [createPlugin](#) (`AsciiChar` const *name, `PluginFieldCollection` const *fc) noexcept=0
Return a plugin object. Return nullptr in case of error.
- virtual `IPluginV2` * [deserializePlugin](#) (`AsciiChar` const *name, void const *serialData, size_t serialLength) noexcept=0
Called during deserialization of plugin layer. Return a plugin object.
- virtual void [setPluginNamespace](#) (`AsciiChar` const *pluginNamespace) noexcept=0
Set the namespace of the plugin creator based on the plugin library it belongs to. This can be set while registering the plugin creator.
- virtual `AsciiChar` const * [getPluginNamespace](#) () const noexcept=0
Return the namespace of the plugin creator object.
- `IPluginCreator` ()=default
- 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.

Parameters

<i>name</i>	A NULL-terminated name string of length 1024 or less, including the NULL terminator.
<i>fc</i>	A pointer to a collection of fields needed for constructing the plugin.

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.

Parameters

<i>name</i>	A NULL-terminated name string of length 1024 or less, including the NULL terminator.
<i>serialData</i>	The start address of a byte array with the serialized plugin representation.
<i>serialLength</i>	The length in bytes of the byte array with the serialized plugin representation.

Returns

A deserialized 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 need 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 NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

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 NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

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 NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

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.

Parameters

<i>pluginNamespace</i>	A NULL-terminated namespace string of length 1024 or less, including the NULL terminator

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:

- [NvInferRuntimePlugin.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 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.
- virtual bool [isParentSearchEnabled](#) () const =0
Return whether the parent registry will be searched if a plugin is not found in this registry default: true.
- virtual void [setParentSearchEnabled](#) (bool const enabled)=0
Set whether the parent registry will be searched if a plugin is not found in this registry.
- virtual [PluginLibraryHandle](#) [loadLibrary](#) ([AsciiChar](#) const *pluginPath) noexcept=0
Load and register a shared library of plugins.
- virtual void [deregisterLibrary](#) ([PluginLibraryHandle](#) handle) noexcept=0
Deregister plugins associated with a library. Any resources acquired when the library was loaded will be released.

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 `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

9.104.4.4 `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

9.104.4.5 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

9.104.4.6 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](#)

9.104.4.7 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.

9.104.4.8 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.

9.104.4.9 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

9.104.4.10 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::safe::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 must also have a corresponding [IPluginCreator](#) implementation.

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

Public Member Functions

- virtual bool [registerCreator](#) ([IPluginCreator](#) &creator, [AsciiChar](#) const *const pluginNamespace) noexcept=0
Register a plugin creator.
- 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 is 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 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

9.105.1 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 must 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.

[IPluginRegistry::setErrorRecorder\(\)](#) must be called to register an error recorder with the registry before using other methods in the registry.

9.105.2 Constructor & Destructor Documentation

9.105.2.1 [~IPluginRegistry\(\)](#)

```
virtual nvinferl1::safe::IPluginRegistry::~IPluginRegistry ( ) [protected], [virtual], [default],
[noexcept]
```

9.105.3 Member Function Documentation

9.105.3.1 deregisterCreator()

```
virtual bool nvinfer1::safe::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.

Parameters

<i>creator</i>	The plugin creator to deregister.

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

9.105.3.2 getErrorRecorder()

```
virtual IErrorRecorder * nvinfer1::safe::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, or nullptr if:

- no error recorder has been set, or
- the last error recorder has been deregistered via `setErrorRecorder(nullptr)`.

See also

[setErrorRecorder\(\)](#)

Usage considerations

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

9.105.3.3 `getPluginCreator()`

```
virtual IPluginCreator * nvinfer1::safe::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 NULL terminated and have a length of 1024 bytes or less including the NULL terminator.

Parameters

<i>pluginName</i>	The plugin name string
<i>pluginVersion</i>	The plugin version string
<i>pluginNamespace</i>	The plugin namespace (by default empty string)

Returns

If a plugin creator corresponding to the passed name, version and namespace can be found in the registry, it is returned. nullptr is returned in the following situations:

- Any of the input arguments is nullptr.
- Any of the input arguments exceeds the string length limit.

- No plugin creator corresponding to the input arguments can be found in the registry.
- A plugin creator can be found, but its stored namespace attribute does not match the pluginNamespace.

Usage considerations

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

9.105.3.4 getPluginCreatorList()

```
virtual IPluginCreator *const * nvinfer1::safe::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 is found.

Parameters

out	<i>numCreators</i>	If the call completes successfully, the number of registered plugin creators (which will be an integer between 0 and 100 inclusive)

Returns

The start address of an IPluginCreator* array of length numCreators if at least one plugin creator has been registered, or nullptr if there are no registered plugin creators.

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.105.3.5 registerCreator()

```
virtual bool nvinfer1::safe::IPluginRegistry::registerCreator (
    IPluginCreator & creator,
    AsciiChar const *const pluginNamespace ) [pure virtual], [noexcept]
```

Register a plugin creator.

Parameters

<i>creator</i>	The plugin creator to be registered.
<i>pluginNamespace</i>	A NULL-terminated namespace string, which must be 1024 bytes or less including the NULL terminator. It must be identical with the result of calling IPluginCreator::getPluginNamespace() on the creator object.

Returns

True if the registration succeeded, else false.

Registration may fail for any of the following reasons:

- The pluginNamespace string is a nullptr.
- The pluginNamespace string exceeds the maximum length.
- The pluginNamespace string does not match the result of creator.getPluginNamespace().
- There have already been 100 plugin creators registered (maximum number of plugins exceeded).
- Another plugin creator with the same combination of plugin name, version and namespace has already been registered.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes; calls to this method will be synchronized by a mutex.

9.105.3.6 setErrorRecorder()

```
virtual void nvinfer1::safe::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. If the recorder is set to nullptr, the error code [ErrorCode::kINVALID_ARGUMENT](#) will be emitted if the recorder has been registered.

Parameters

<i>recorder</i>	The error recorder to register with this interface, or nullptr to deregister the current recorder.

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:

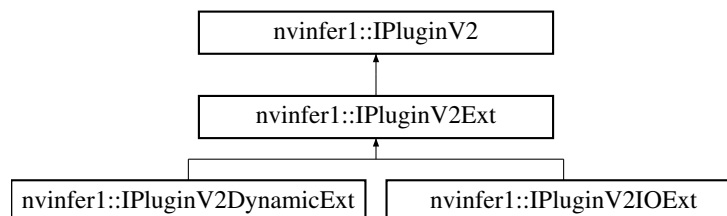
- [NvInferSafeRuntime.h](#)

9.106 nvinfer1::IPluginV2 Class Reference

Plugin class for user-implemented layers.

```
#include <NvInferRuntimePlugin.h>
```

Inheritance diagram for nvinfer1::IPluginV2:



Public Member Functions

- virtual `int32_t` [getTensorRTVersion](#) () const noexcept
Return the API version with which this plugin was built.
- virtual `AsciiChar` const * [getPluginType](#) () const noexcept=0
Return the plugin type. Should match the plugin name returned by the corresponding plugin creator.
- virtual `AsciiChar` const * [getPluginVersion](#) () const noexcept=0
Return the plugin version. Should match the plugin version returned by the corresponding plugin creator.
- virtual `int32_t` [getNbOutputs](#) () const noexcept=0
Get the number of outputs from the layer.
- virtual `Dims` [getOutputDimensions](#) (int32_t index, `Dims` const *inputs, int32_t nbInputDims) noexcept=0
Get the dimension of an output tensor.
- virtual bool [supportsFormat](#) (`DataType` type, `PluginFormat` format) const noexcept=0
Check format support.

- virtual void [configureWithFormat](#) ([Dims](#) const *inputDims, int32_t nbInputs, [Dims](#) const *outputDims, int32_t nbOutputs, [DataType](#) type, [PluginFormat](#) format, int32_t maxBatchSize) noexcept=0
Configure the layer.
- virtual int32_t [initialize](#) () noexcept=0
Initialize the layer for execution. This is called when the engine is created.
- virtual void [terminate](#) () noexcept=0
Release resources acquired during plugin layer initialization. This is called when the engine is destroyed.
- virtual size_t [getWorkspaceSize](#) (int32_t maxBatchSize) const noexcept=0
Find the workspace size required by the layer.
- virtual int32_t [enqueue](#) (int32_t batchSize, void const *const *inputs, void *const *outputs, void *workspace, cudaStream_t stream) noexcept=0
Execute the layer.
- virtual size_t [getSerializationSize](#) () const noexcept=0
Find the size of the serialization buffer required to store the plugin configuration in a binary file.
- virtual void [serialize](#) (void *buffer) const noexcept=0
Serialize the layer.
- virtual void [destroy](#) () noexcept=0
Destroy the plugin object. This will be called when the network, builder or engine is destroyed.
- virtual [IPluginV2](#) * [clone](#) () const noexcept=0
Clone the plugin object. This copies over internal plugin parameters and returns a new plugin object with these parameters.
- virtual void [setPluginNamespace](#) ([AsciiChar](#) const *pluginNamespace) noexcept=0
Set the namespace that this plugin object belongs to. Ideally, all plugin objects from the same plugin library must have the same namespace.
- virtual [AsciiChar](#) const * [getPluginNamespace](#) () const noexcept=0
Return the namespace of the plugin object.

9.106.1 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.

9.106.2 Member Function Documentation

9.106.2.1 clone()

```
virtual IPluginV2 * nvinfer1::IPluginV2::clone ( ) const [pure virtual], [noexcept]
```

Clone the plugin object. This copies over internal plugin parameters and returns a new plugin object with these parameters.

The TensorRT runtime calls [clone\(\)](#) to clone the plugin when an execution context is created for an engine, after the engine has been created. The runtime does not call [initialize\(\)](#) on the cloned plugin, so the cloned plugin must be created in an initialized state.

Returns

A cloned plugin object in an initialized state with the same parameters as the current object. nullptr must be returned if the cloning fails, e.g. because of resource exhaustion.

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](#).

9.106.2.2 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. Will be the start address of a Dims array of length nbInputs.
<i>nbInputs</i>	The number of inputs. Will be a non-negative integer.
<i>outputDims</i>	The output tensor dimensions. Will be the start address of a Dims array of length nbOutputs.
<i>nbOutputs</i>	The number of outputs. Will be a positive integer identical to the return value of getNbOutputs() .
<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. Will be a positive integer.

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 `PluginFormat::kV2IOExt` or `PluginFormat::kV2DynamicExt` for other `PluginFormat`s.

`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](#).

9.106.2.3 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.

9.106.2.4 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. Will be an array of device addresses corresponding to input tensors of length nbInputs, where nbInputs is the second parameter passed to configureWithFormat() . The i-th input tensor will have the dimensions inputDims[i], where inputDims is the first parameter that was passed to configureWithFormat() .
<i>outputs</i>	The memory for the output tensors. Will be an array of device addresses corresponding to output tensors of length getNbOutputs() .
<i>workspace</i>	Workspace for execution. Will be the start address of a device buffer whose length will be at least getWorkspaceSize(batchSize) .
<i>stream</i>	The stream in which to execute the kernels. This will be a valid CUDA stream.

Returns

0 for success, else non-zero (which will cause engine termination).

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.

9.106.2.5 getNbOutputs()

```
virtual int32_t nvinfer1::IPluginV2::getNbOutputs ( ) const [pure virtual], [noexcept]
```

Get the number of outputs from the layer.

Returns

The number of outputs, which is a positive integer.

This function is called by the implementations of [INetworkDefinition](#) and [IBuilder](#). In particular, it is called prior to any call to [initialize\(\)](#).

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.

9.106.2.6 `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. Will lie in the valid range (between 0 and getNbOutputs() -1 inclusive).
<i>inputs</i>	The input tensor dimensions. Will be the start address of a Dims array of length nbInputDims.
<i>nbInputDims</i>	The number of input tensors. Will be a non-negative integer.

Returns

The output tensor dimensions if the index is in the valid range. An invalid value of [Dims](#){-1, {}} must be returned if the index is not in the valid range.

This function is called by the implementations of [INetworkDefinition](#) and [IBuilder](#). In particular, it is called prior to any call to [initialize\(\)](#).

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 must not be included in the returned dimensions, even if the plugin is expected to be run in a network with explicit batch mode enabled. Please see the TensorRT Developer Guide for more details on how plugin inputs and outputs behave.

9.106.2.7 getPluginNamespace()

```
virtual AsciiChar const * nvinfer1::IPluginV2::getPluginNamespace ( ) const [pure virtual], [noexcept]
```

Return the namespace of the plugin object.

Returns

The namespace string that was passed to [setPluginNamespace\(\)](#), possibly after truncation to 1024 bytes if a longer string was passed. An empty string must be returned as default value.

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.

9.106.2.8 `getPluginType()`

```
virtual AsciiChar const * nvinfer1::IPluginV2::getPluginType ( ) const [pure virtual], [noexcept]
```

Return the plugin type. Should match the plugin name returned by the corresponding plugin creator.

See also

[IPluginCreator::getPluginName\(\)](#)

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

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.

9.106.2.9 `getPluginVersion()`

```
virtual AsciiChar const * nvinfer1::IPluginV2::getPluginVersion ( ) const [pure virtual], [noexcept]
```

Return the plugin version. Should match the plugin version returned by the corresponding plugin creator.

See also

[IPluginCreator::getPluginVersion\(\)](#)

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

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.

9.106.2.10 getSerializationSize()

```
virtual size_t nvinfer1::IPluginV2::getSerializationSize ( ) const [pure virtual], [noexcept]
```

Find the size of the serialization buffer required to store the plugin configuration in a binary file.

Returns

The size of the serialization buffer in bytes.

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.

9.106.2.11 getTensorRTVersion()

```
virtual int32_t nvinfer1::IPluginV2::getTensorRTVersion ( ) const [inline], [virtual], [noexcept]
```

Return the API version with which this plugin was built.

Do not override this method as it is used by the TensorRT library to maintain backwards-compatibility with plugins.

Returns

The TensorRT version in the format $1000 * \text{major} + 100 * \text{minor} + \text{patch}$.

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](#).

9.106.2.12 getWorkspaceSize()

```
virtual size_t nvinfer1::IPluginV2::getWorkspaceSize (
    int32_t batchSize ) const [pure virtual], [noexcept]
```

Find the workspace size required by the layer.

This function is called during engine startup, after [initialize\(\)](#). The workspace size returned must be sufficient for any batch size up to the maximum.

Parameters

<i>maxBatchSize</i>	The maximum batch size, which will be a positive integer.

Returns

The workspace size in bytes, i.e. the device memory size that the plugin requires for its internal computations.

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.

9.106.2.13 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.

9.106.2.14 serialize()

```
virtual void nvinfer1::IPluginV2::serialize (
    void * buffer ) const [pure virtual], [noexcept]
```

Serialize the layer.

Parameters

<i>buffer</i>	A pointer to a host buffer to serialize data. Size of buffer will be at least as large as the 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.

9.106.2.15 setPluginNamespace()

```
virtual void nvinfer1::IPluginV2::setPluginNamespace (
    AsciiChar const * pluginNamespace ) [pure virtual], [noexcept]
```

Set the namespace that this plugin object belongs to. Ideally, all plugin objects from the same plugin library must have the same namespace.

Parameters

<i>pluginNamespace</i>	The namespace for the plugin object.

Warning

The string `pluginNamespace` will be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

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.

9.106.2.16 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.

9.106.2.17 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.

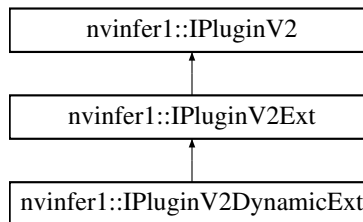
The documentation for this class was generated from the following file:

- [NvInferRuntimePlugin.h](#)

9.107 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 must also be pre-configured. The returned object must allow [attachToContext\(\)](#) with a new execution context. Cloned plugin objects can share the same per-engine immutable resource (e.g. weights) with the source object (e.g. via ref-counting) to avoid duplication.
- virtual [DimsExprs getOutputDimensions](#) (int32_t outputIndex, [DimsExprs](#) const *inputs, int32_t nbInputs, [IExprBuilder](#) &exprBuilder) noexcept=0
Get expressions for computing dimensions of an output tensor from dimensions of the input tensors.
- virtual bool [supportsFormatCombination](#) (int32_t pos, [PluginTensorDesc](#) const *inOut, int32_t nbInputs, int32_t nbOutputs) noexcept=0
Return true if plugin supports the format and datatype for the input/output indexed by pos.

- virtual void [configurePlugin](#) ([DynamicPluginTensorDesc](#) const *in, int32_t nbInputs, [DynamicPluginTensorDesc](#) const *out, int32_t nbOutputs) noexcept=0

Configure the plugin.

- virtual size_t [getWorkspaceSize](#) ([PluginTensorDesc](#) const *inputs, int32_t nbInputs, [PluginTensorDesc](#) const *outputs, int32_t nbOutputs) const noexcept=0

Find the workspace size required by the layer.

- virtual int32_t [enqueue](#) ([PluginTensorDesc](#) const *inputDesc, [PluginTensorDesc](#) const *outputDesc, void const *const *inputs, void *const *outputs, void *workspace, cudaStream_t stream) noexcept=0

Execute the layer.

Static Public Attributes

- static constexpr int32_t [kFORMAT_COMBINATION_LIMIT](#) = 100

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

9.107.1 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](#).

9.107.2 Constructor & Destructor Documentation

9.107.2.1 [~IPluginV2DynamicExt](#)()

```
virtual nvinfer1::IPluginV2DynamicExt::~~IPluginV2DynamicExt ( ) [inline], [protected], [virtual], [noexcept]
```

9.107.3 Member Function Documentation

9.107.3.1 clone()

```
IPluginV2DynamicExt * nvinfer1::IPluginV2DynamicExt::clone ( ) const [override], [pure virtual], [noexcept]
```

Clone the plugin object. This copies over internal plugin parameters as well and returns a new plugin object with these parameters. If the source plugin is pre-configured with [configurePlugin\(\)](#), the returned object must also be pre-configured. The returned object must allow [attachToContext\(\)](#) with a new execution context. Cloned plugin objects can share the same per-engine immutable resource (e.g. weights) with the source object (e.g. via ref-counting) to avoid duplication.

Returns

A pointer to a cloned plugin object if cloning was successful, otherwise nullptr.

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](#).

9.107.3.2 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.

9.107.3.3 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).

9.107.3.4 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;
```

9.107.3.5 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](#).

9.107.3.6 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.

9.107.3.7 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 $0 \leq \text{pos} < \text{nbInputs} + \text{nbOutputs}$.

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.

9.107.4 Member Data Documentation

9.107.4.1 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:

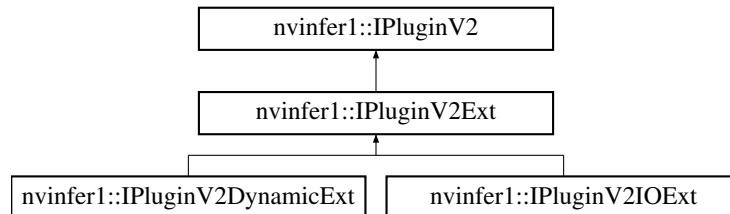
- [NvInferRuntime.h](#)

9.108 nvinfer1::IPluginV2Ext Class Reference

Plugin class for user-implemented layers.

```
#include <NvInferRuntimePlugin.h>
```

Inheritance diagram for nvinfer1::IPluginV2Ext:



Public Member Functions

- virtual [nvinfer1::DataType](#) [getOutputDataType](#) (int32_t index, [nvinfer1::DataType](#) const *inputTypes, int32_t nbInputs) const noexcept=0
Return the DataType of the plugin output at the requested index.
- virtual bool [isOutputBroadcastAcrossBatch](#) (int32_t outputIndex, bool const *inputIsBroadcasted, int32_t nbInputs) const noexcept=0
Return true if the output tensor is broadcast across a batch.
- virtual bool [canBroadcastInputAcrossBatch](#) (int32_t inputIndex) const noexcept=0
Return true if the plugin can use an input tensor 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 resources.
- virtual void `detachFromContext ()` noexcept
Detach the plugin object from its execution context.
- `IPluginV2Ext * clone ()` const noexcept override=0
Clone the plugin object. This copies over internal plugin parameters as well and returns a new plugin object with these parameters. If the source plugin is pre-configured with `configurePlugin()`, the returned object must also be pre-configured. The returned object must allow `attachToContext()` with a new execution context. Cloned plugin objects can share the same per-engine immutable resource (e.g. weights) with the source object (e.g. via ref-counting) to avoid duplication.

Protected Member Functions

- int32_t `getTensorRTVersion ()` const noexcept override
Return the API version with which this plugin was built. The upper byte reserved by TensorRT and is used to differentiate this from `IPluginV2`.
- void `configureWithFormat` (Dims const *, int32_t, Dims const *, int32_t, `DataType`, `PluginFormat`, int32_t) noexcept override
Derived classes must not implement this. In a C++11 API it would be override final.

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 `IPluginV2` interface by supporting different output data types and broadcast across batches.

See also

[IPluginV2](#)

Deprecated Deprecated in TensorRT 8.5. Implement `IPluginV2DynamicExt` or `IPluginV2IOExt` depending on your requirement.

9.108.2 Constructor & Destructor Documentation

9.108.2.1 IPluginV2Ext()

```
nvinfer1::IPluginV2Ext::IPluginV2Ext ( ) [default]
```

9.108.2.2 ~IPluginV2Ext()

```
nvinfer1::IPluginV2Ext::~~IPluginV2Ext ( ) [override], [default]
```

9.108.3 Member Function Documentation

9.108.3.1 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 resources.

Parameters

<i>cudnn</i>	The cuDNN context handle of the execution context. Will be a valid cuDNN context handle, or nullptr if the cuDNN backend is not used.
<i>cublas</i>	The cuBLAS context handle of the execution context. Will be a valid cuBLAS context handle, or nullptr if the cuBLAS backend is not used.
<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 are 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.

9.108.3.2 canBroadcastInputAcrossBatch()

```
virtual bool nvinfer1::IPluginV2Ext::canBroadcastInputAcrossBatch (
    int32_t inputIndex ) const [pure virtual], [noexcept]
```

Return true if the plugin can use an input tensor that is broadcast across batch without replication.

Parameters

<i>inputIndex</i>	Index of input that could be broadcast. Will be in the valid range between 0 and nbInputs - 1 where nbInputs is the maximum number of input tensors supported by this plugin.

Returns

true if the index is in the valid range and the plugin is able to broadcast a single copy of this input tensor across the batch. False otherwise.

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 must 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.

9.108.3.3 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 must also be pre-configured. The returned object must allow `attachToContext()` with a new execution context. Cloned plugin objects can share the same per-engine immutable resource (e.g. weights) with the source object (e.g. via ref-counting) to avoid duplication.

Returns

A pointer to a cloned plugin object if cloning was successful, otherwise nullptr.

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](#).

9.108.3.4 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. Will be an array of length nbInputs.
<i>nbInputs</i>	The number of inputs. Will be a non-negative integer.
<i>outputDims</i>	The output tensor dimensions. Will be an array of length nbOutputs.
<i>nbOutputs</i>	The number of outputs. Will be a positive integer.
<i>inputTypes</i>	The data types selected for the plugin inputs. Will be an array of length nbInputs.
<i>outputTypes</i>	The data types selected for the plugin outputs. Will be an array of length nbOutputs.
<i>inputIsBroadcast</i>	True for each input that the plugin must broadcast across the batch. Will be an array of length nbInputs.
<i>outputIsBroadcast</i>	True for each output that TensorRT will broadcast across the batch. Will be an array of length nbOutputs.
<i>floatFormat</i>	The format selected for the engine for the floating point inputs/outputs.
<i>maxBatchSize</i>	The maximum batch size. Will be a positive integer.

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 must be treated as if it is one. Index 'i' of `inputIsBroadcast` is true only if the input is semantically broadcast across the batch and calling `canBroadcastInputAcrossBatch` with argument 'i' returns true. Index 'i' of `outputIsBroadcast` is true only if calling `isOutputBroadcastAcrossBatch` with argument '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.

9.108.3.5 `configureWithFormat()`

```
void nvinfer1::IPluginV2Ext::configureWithFormat (
    Dims const * ,
    int32_t ,
    Dims const * ,
    int32_t ,
    DataType ,
    PluginFormat ,
    int32_t ) [inline], [override], [protected], [virtual], [noexcept]
```

Derived classes must not implement this. In a C++11 API it would be `override final`.

`IPluginV2Ext::configureWithFormat()` is a NOP operation for all classes derived from `IPluginV2Ext`. These classes call `configurePlugin()` instead.

Implements `nvinfer1::IPluginV2`.

9.108.3.6 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 an execution context is destroyed or the context resources are unassigned from the context.

If the plugin owns per-context resource, it can be released here.

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.

9.108.3.7 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.

Parameters

<i>index</i>	The output tensor index in the valid range between 0 and getNbOutputs()-1 .
<i>inputTypes</i>	The data types of the input tensors, stored in an array of length nbInputs.
<i>nbInputs</i>	The number of input tensors. Will be a non-negative integer.

Returns

The data type of the output tensor with the provided index if the input tensors have the data types provided in *inputTypes*, provided the output tensor index is in the valid range. [DataType::kFLOAT](#) must be returned if the index is not in the valid range.

The default behavior must be to return the type of the first input, or [DataType::kFLOAT](#) if the layer has no inputs. The returned data type must have a format that is supported by the plugin.

See also

[supportsFormat\(\)](#)

Warning

`DataType:kBOOL` and `DataType::kUINT8` are not supported.

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.

9.108.3.8 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](#).

Returns

In the lower three bytes, the TensorRT version in the format $1000 * \text{major} + 100 * \text{minor} + \text{patch}$. In the upper byte, the value 1.

Do not override this method as it is used by the TensorRT library to maintain backwards-compatibility with plugins.

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](#).

9.108.3.9 isOutputBroadcastAcrossBatch()

```
virtual bool nvinfer1::IPluginV2Ext::isOutputBroadcastAcrossBatch (
    int32_t outputIndex,
    bool const * inputIsBroadcasted,
    int32_t nbInputs ) const [pure virtual], [noexcept]
```

Return true if the output tensor is broadcast across a batch.

Parameters

<i>outputIndex</i>	The index of the output tensor, which will be in the valid range between 0 and nbOutputs()-1.
<i>inputIsBroadcasted</i>	A boolean array of length nbInputs. The i-th element will be true if and only if the tensor for the ith input is broadcast across a batch.
<i>nbInputs</i>	The number of inputs. Will be a non-negative integer.

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:

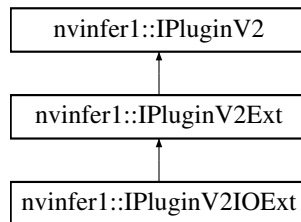
- [NvInferRuntimePlugin.h](#)

9.109 nvinfer1::IPluginV2IOExt Class Reference

Plugin class for user-implemented layers.

```
#include <NvInferRuntimePlugin.h>
```

Inheritance diagram for `nvinfer1::IPluginV2IOExt`:



Public Member Functions

- virtual void `configurePlugin` (`PluginTensorDesc` const *in, int32_t nbInput, `PluginTensorDesc` const *out, int32_t nbOutput) noexcept=0
Configure the layer.
- virtual bool `supportsFormatCombination` (int32_t pos, `PluginTensorDesc` const *inOut, int32_t nbInputs, int32_t nbOutputs) const noexcept=0
Return true if plugin supports the format and datatype for the input/output indexed by pos.

Protected Member Functions

- `int32_t getTensorRTVersion ()` const noexcept override

Return the API version with which this plugin was built. The upper byte is reserved by TensorRT and is used to differentiate this from [IPluginV2](#) and [IPluginV2Ext](#).

9.109.1 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](#)

9.109.2 Member Function Documentation

9.109.2.1 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.

9.109.2.2 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](#).

9.109.2.3 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 $0 \leq \text{pos} < \text{nbInputs} + \text{nbOutputs}$.

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 must return true if that format/datatype at `inOut[pos]` are supported by the plugin. If support is conditional on other input/output formats/datatypes, the plugin can make its result conditional on the formats/datatypes in `inOut[0..pos-1]`, which will be set to values that the plugin supports. The override must not inspect `inOut[pos+1..nbInputs+nbOutputs-1]`, which will have invalid values. In other words, the decision for pos must be based on `inOut[0..pos]` only.

Some examples:

- A definition for a plugin that supports only FP16 NCHW:

```
return inOut.format[pos] == TensorFormat::kLINEAR && inOut.type[pos] == DataType::kHALF;
```

- A definition for a plugin that supports only FP16 NCHW for its two inputs, and FP32 NCHW for its single output:

```
return inOut.format[pos] == TensorFormat::kLINEAR &&
       (inOut.type[pos] == (pos < 2 ? DataType::kHALF : DataType::kFLOAT));
```

- A definition for a "polymorphic" plugin with two inputs and one output that supports any format or type, but the inputs and output must have the same format and type:

```
return pos == 0 || (inOut.format[pos] == inOut.format[0] && inOut.type[pos] == inOut.type[0]);
```

Warning: TensorRT will stop asking for formats once it finds kFORMAT_COMBINATION_LIMIT on combinations.

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:

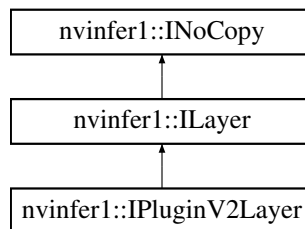
- [NvInferRuntimePlugin.h](#)

9.110 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](#)

9.110.1 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.

9.110.2 Constructor & Destructor Documentation

9.110.2.1 ~IPluginV2Layer()

```
virtual nvinfer1::IPluginV2Layer::~~IPluginV2Layer ( ) [protected], [virtual], [default], [noexcept]
```

9.110.3 Member Function Documentation

9.110.3.1 getPlugin()

```
IPluginV2 & nvinfer1::IPluginV2Layer::getPlugin ( ) [inline], [noexcept]
```

Get the plugin for the layer.

See also

[IPluginV2](#)

9.110.4 Member Data Documentation

9.110.4.1 mImpl

```
apiv::VPluginV2Layer* nvinfer1::IPluginV2Layer::mImpl [protected]
```

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

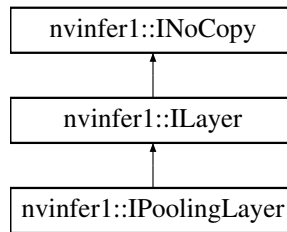
- [NvInfer.h](#)

9.111 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](#)

9.111.1 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.

9.111.2 Constructor & Destructor Documentation

9.111.2.1 ~IPoolingLayer()

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

9.111.3 Member Function Documentation

9.111.3.1 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\(\)](#)

9.111.3.2 getBlendFactor()

```
float nvinfer1::IPoolingLayer::getBlendFactor ( ) const [inline], [noexcept]
```

Get the blending factor for the `max_average_blend` mode: `max_average_blendPool = (1-blendFactor)*maxPool + blendFactor*avgPool` `blendFactor` is a user value in `[0,1]` with the default value of `0.0` In modes other than `kMAX_AVERAGE_BLEND`, `blendFactor` is ignored.

See also

[setBlendFactor\(\)](#)

9.111.3.3 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

9.111.3.4 getPaddingMode()

```
PaddingMode nvinfer1::IPoolingLayer::getPaddingMode ( ) const [inline], [noexcept]
```

Get the padding mode.

Default: kEXPLICIT_ROUND_DOWN

See also

[setPaddingMode\(\)](#)

9.111.3.5 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\(\)](#)

9.111.3.6 getPoolingType()

```
PoolingType nvinfer1::IPoolingLayer::getPoolingType ( ) const [inline], [noexcept]
```

Get the type of activation to be performed.

See also

[setPoolingType\(\)](#), [PoolingType](#)

9.111.3.7 getPostPadding()

```
Dims nvinfer1::IPoolingLayer::getPostPadding ( ) const [inline], [noexcept]
```

Get the padding.

See also

[setPostPadding\(\)](#)

9.111.3.8 getPrePadding()

```
Dims nvinfer1::IPoolingLayer::getPrePadding ( ) const [inline], [noexcept]
```

Get the pre-padding.

See also

[setPrePadding\(\)](#)

9.111.3.9 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

9.111.3.10 getStrideNd()

```
Dims nvinfer1::IPoolingLayer::getStrideNd ( ) const [inline], [noexcept]
```

Get the multi-dimension stride for pooling.

See also

[setStrideNd\(\)](#)

9.111.3.11 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

9.111.3.12 getWindowSizeNd()

```
Dims nvinfer1::IPoolingLayer::getWindowSizeNd ( ) const [inline], [noexcept]
```

Get the multi-dimension window size for pooling.

See also

[setWindowSizeNd\(\)](#)

9.111.3.13 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\(\)](#)

9.111.3.14 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\(\)](#)

9.111.3.15 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

9.111.3.16 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\(\)](#)

9.111.3.17 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\(\)](#)

9.111.3.18 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](#)

9.111.3.19 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\(\)](#)

9.111.3.20 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\(\)](#)

9.111.3.21 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

9.111.3.22 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\(\)](#)

9.111.3.23 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

9.111.3.24 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\(\)](#)

9.111.4 Member Data Documentation

9.111.4.1 mImpl

```
apiv::VPoolingLayer* nvinfer1::IPoolingLayer::mImpl [protected]
```

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

- [NvInfer.h](#)

9.112 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

9.112.1 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.

9.112.2 Constructor & Destructor Documentation

9.112.2.1 ~IProfiler()

```
virtual nvinfer1::IProfiler::~~IProfiler ( ) [inline], [virtual], [noexcept]
```

9.112.3 Member Function Documentation

9.112.3.1 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. If the engine is built with profiling verbosity set to kNONE, the layerName is the decimal index of the layer.
<i>ms</i>	The time in milliseconds to execute the layer.

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

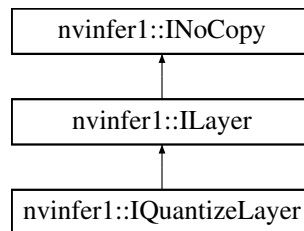
- [NvInferRuntime.h](#)

9.113 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`

9.113.1 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 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::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.

9.113.2 Constructor & Destructor Documentation

9.113.2.1 ~IQuantizeLayer()

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

9.113.3 Member Function Documentation

9.113.3.1 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.

9.113.3.2 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).

9.113.4 Member Data Documentation

9.113.4.1 mImpl

```
apiv::VQuantizeLayer* nvinfer1::IQuantizeLayer::mImpl [protected]
```

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

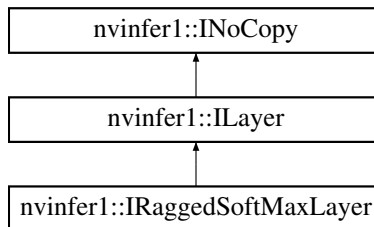
- [NvInfer.h](#)

9.114 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

9.114.1 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.

9.114.2 Constructor & Destructor Documentation

9.114.2.1 ~IRaggedSoftMaxLayer()

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

9.114.3 Member Data Documentation

9.114.3.1 mImpl

```
apiv::VRaggedSoftMaxLayer* nvinfer1::IRaggedSoftMaxLayer::mImpl [protected]
```

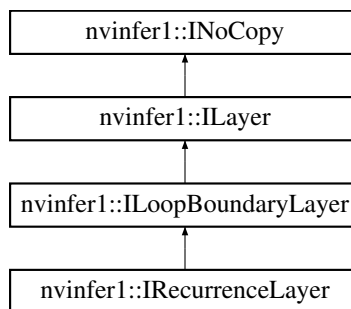
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.115 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`

9.115.1 Constructor & Destructor Documentation

9.115.1.1 ~IRecurrenceLayer()

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

9.115.2 Member Function Documentation

9.115.2.1 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.

9.115.3 Member Data Documentation

9.115.3.1 mImpl

```
apiv::VRecurrenceLayer* nvinfer1::IRecurrenceLayer::mImpl [protected]
```

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

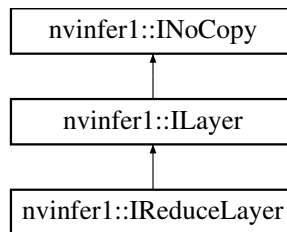
- [NvInfer.h](#)

9.116 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](#)

9.116.1 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.

9.116.2 Constructor & Destructor Documentation

9.116.2.1 ~IReduceLayer()

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

9.116.3 Member Function Documentation

9.116.3.1 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](#)

9.116.3.2 getOperation()

```
ReduceOperation nvinfer1::IReduceLayer::getOperation ( ) const [inline], [noexcept]
```

Get the reduce operation for the layer.

See also

[setOperation\(\)](#), [ReduceOperation](#)

9.116.3.3 `getReduceAxes()`

```
uint32_t nvinfer1::IReduceLayer::getReduceAxes ( ) const [inline], [noexcept]
```

Get the axes over which to reduce for the layer.

See also

[setReduceAxes](#)

9.116.3.4 `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](#)

9.116.3.5 `setOperation()`

```
void nvinfer1::IReduceLayer::setOperation (
    ReduceOperation op ) [inline], [noexcept]
```

Set the reduce operation for the layer.

See also

[getOperation\(\)](#), [ReduceOperation](#)

9.116.3.6 `setReduceAxes()`

```
void nvinfer1::IReduceLayer::setReduceAxes (
    uint32_t reduceAxes ) [inline], [noexcept]
```

Set the axes over which to reduce.

See also

[getReduceAxes](#)

9.116.4 Member Data Documentation

9.116.4.1 mImpl

```
apiv::VReduceLayer* nvinfer1::IReduceLayer::mImpl [protected]
```

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

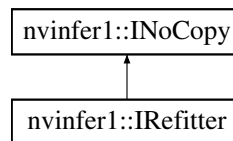
- [NvInfer.h](#)

9.117 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

9.117.1 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.

9.117.2 Constructor & Destructor Documentation

9.117.2.1 ~IRefitter()

```
virtual nvinferl::IRefitter::~IRefitter ( ) [virtual], [default], [noexcept]
```

9.117.3 Member Function Documentation

9.117.3.1 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.

9.117.3.2 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.

9.117.3.3 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.

9.117.3.4 `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.

9.117.3.5 `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.

9.117.3.6 getErrorRecorder()

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

See also

[setErrorRecorder\(\)](#)

9.117.3.7 getLogger()

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

get the logger with which the refitter was created

Returns

the logger

9.117.3.8 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\(\)](#)

9.117.3.9 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.

9.117.3.10 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.

9.117.3.11 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.

9.117.3.12 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.

9.117.3.13 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.

9.117.3.14 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\(\)](#)

9.117.3.15 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.

9.117.3.16 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.

9.117.3.17 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.

9.117.4 Member Data Documentation

9.117.4.1 mImpl

```
apiv::VRefitter* nvinfer1::IRefitter::mImpl [protected]
```

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

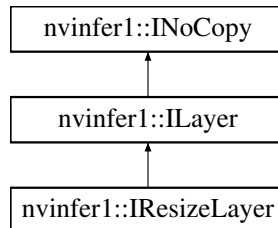
- [NvInferRuntime.h](#)

9.118 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](#)

9.118.1 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.

9.118.2 Constructor & Destructor Documentation

9.118.2.1 ~IResizeLayer()

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

9.118.3 Member Function Documentation

9.118.3.1 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\(\)](#).

9.118.3.2 getCoordinateTransformation()

```
ResizeCoordinateTransformation nvinfer1::IResizeLayer::getCoordinateTransformation ( ) const [inline], [noexcept]
```

Get coordinate transformation function.

Returns

The coordinate transformation function.

9.118.3.3 getCubicCoeff()

```
float nvinfer1::IResizeLayer::getCubicCoeff ( ) const [inline], [noexcept]
```

Get the coefficient 'A' used in cubic interpolation.

See also

[setCubicCoeff\(\)](#)

9.118.3.4 getExcludeOutside()

```
bool nvinfer1::IResizeLayer::getExcludeOutside ( ) const [inline], [noexcept]
```

Get the state for excluding outside pixels.

See also

[setExcludeOutside\(\)](#)

9.118.3.5 getNearestRounding()

```
ResizeRoundMode nvinfer1::IResizeLayer::getNearestRounding ( ) const [inline], [noexcept]
```

Get rounding mode for nearest neighbor resize.

Returns

The rounding mode.

9.118.3.6 getOutputDimensions()

```
Dims nvinfer1::IResizeLayer::getOutputDimensions ( ) const [inline], [noexcept]
```

Get the output dimensions.

Returns

The output dimensions.

9.118.3.7 getResizeMode()

```
ResizeMode nvinfer1::IResizeLayer::getResizeMode ( ) const [inline], [noexcept]
```

Get resize mode for an input tensor.

Returns

The resize mode.

9.118.3.8 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 size != nbScales, no scales will be copied.
<i>scales</i>	Pointer to where to copy the scales. Scales will be copied only if size == nbScales and scales != nullptr.

In case the size is not known consider using size = 0 and scales = nullptr. This method will return the number of resize scales.

Returns

The number of resize scales i.e. nbScales if scales were set. Return -1 in case no scales were set or resize layer is used in dynamic mode.

9.118.3.9 getSelectorForSinglePixel()

```
ResizeSelector nvinfer1::IResizeLayer::getSelectorForSinglePixel ( ) const [inline], [noexcept]
```

Get the coordinate selector function when resized to single pixel.

Returns

The selector function.

9.118.3.10 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\(\)](#).

9.118.3.11 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](#)

9.118.3.12 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.

9.118.3.13 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.

9.118.3.14 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.

9.118.3.15 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](#)

9.118.3.16 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](#)

9.118.3.17 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](#)

9.118.3.18 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](#)

9.118.3.19 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](#)

9.118.4 Member Data Documentation

9.118.4.1 mImpl

```
apiv::VResizeLayer* nvinfer1::IResizeLayer::mImpl [protected]
```

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

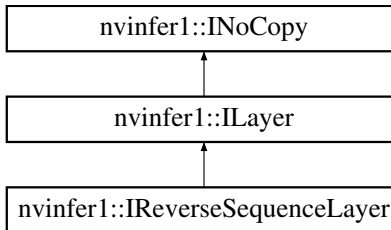
- [NvInfer.h](#)

9.119 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](#)

9.119.1 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.

9.119.2 Constructor & Destructor Documentation

9.119.2.1 ~IReverseSequenceLayer()

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

9.119.3 Member Function Documentation

9.119.3.1 getBatchAxis()

```
int32_t nvinfer1::IReverseSequenceLayer::getBatchAxis ( ) const [inline], [noexcept]
```

Return the batch axis. Return 1 if no batch axis was set.

See also

[getBatchAxis\(\)](#)

9.119.3.2 getSequenceAxis()

```
int32_t nvinfer1::IReverseSequenceLayer::getSequenceAxis ( ) const [inline], [noexcept]
```

Return the sequence axis. Return 0 if no sequence axis was set.

See also

[getSequenceAxis\(\)](#)

9.119.3.3 `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\(\)](#)

9.119.3.4 `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\(\)](#)

9.119.4 Member Data Documentation

9.119.4.1 `mImpl`

```
apiv::VReverseSequenceLayer* nvinfer1::IReverseSequenceLayer::mImpl [protected]
```

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

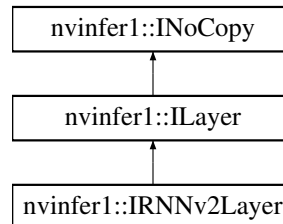
- [NvInfer.h](#)

9.120 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`

9.120.1 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.

9.120.2 Constructor & Destructor Documentation

9.120.2.1 `~IRNNv2Layer()`

```
virtual nvinfer1::IRNNv2Layer::~IRNNv2Layer ( ) [protected], [virtual], [default], [noexcept]
```

9.120.3 Member Function Documentation

9.120.3.1 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\(\)](#)

9.120.3.2 getCellState()

```
ITensor * nvinfer1::IRNNv2Layer::getCellState ( ) const [inline], [noexcept]
```

Get the initial cell state of the RNN.

See also

[setCellState\(\)](#)

9.120.3.3 getDataLength()

```
int32_t nvinfer1::IRNNv2Layer::getDataLength ( ) const [inline], [noexcept]
```

Get the embedding length of the RNN.

9.120.3.4 getDirection()

```
RNNDirection nvinfer1::IRNNv2Layer::getDirection ( ) const [inline], [noexcept]
```

Get the direction of the RNN layer.

See also

[setDirection\(\)](#), [RNNDirection](#)

9.120.3.5 getHiddenSize()

```
int32_t nvinfer1::IRNNv2Layer::getHiddenSize ( ) const [inline], [noexcept]
```

Get the hidden size of the RNN.

9.120.3.6 getHiddenState()

```
ITensor * nvinfer1::IRNNv2Layer::getHiddenState ( ) const [inline], [noexcept]
```

Get the initial hidden state of the RNN.

See also

[setHiddenState\(\)](#)

9.120.3.7 getInputMode()

```
RNNInputMode nvinfer1::IRNNv2Layer::getInputMode ( ) const [inline], [noexcept]
```

Get the input mode of the RNN layer.

See also

[setInputMode\(\)](#), [RNNInputMode](#)

9.120.3.8 getLayerCount()

```
int32_t nvinfer1::IRNNv2Layer::getLayerCount ( ) const [inline], [noexcept]
```

Get the layer count of the RNN.

9.120.3.9 getMaxSeqLength()

```
int32_t nvinfer1::IRNNv2Layer::getMaxSeqLength ( ) const [inline], [noexcept]
```

Get the maximum sequence length of the RNN.

9.120.3.10 getOperation()

```
RNNOperation nvinfer1::IRNNv2Layer::getOperation ( ) const [inline], [noexcept]
```

Get the operation of the RNN layer.

See also

[setOperation\(\)](#), [RNNOperation](#)

9.120.3.11 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\(\)](#)

9.120.3.12 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\(\)](#)

9.120.3.13 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 [DataType::kFLOAT](#) or [DataType::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 setWeightsForGate() 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() .

9.120.3.14 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 $\{N_1, \dots, N_p, L, H\}$, 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 [RNNDirection::kUNIDIRECTION](#), and $2 \times$ [getLayerCount\(\)](#) if `getDirection` is [RNNDirection::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](#).

9.120.3.15 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 [RNNDirection::kBIDIRECTION](#) case the output is concatenated together, resulting in output size of $2 \times$ [getHiddenSize\(\)](#).

See also

[getDirection\(\)](#), [RNNDirection](#)

9.120.3.16 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 [RNNDirection::kUNIDIRECTION](#), and $2 \times$ [getLayerCount\(\)](#) if [getDirection](#) is [RNNDirection::kBIDIRECTION](#). In the bi-directional case, layer l 's final forward hidden state is stored in $L = 2 \times l$, and final backward hidden state is stored in $L = 2 \times l + 1$.
- H is the hidden state for each layer, equal to [getHiddenSize\(\)](#).

9.120.3.17 setInputMode()

```
void nvinfer1::IRNNv2Layer::setInputMode (
    RNNInputMode op ) [inline], [noexcept]
```

Set the input mode of the RNN layer.

See also

[getInputMode\(\)](#), [RNNInputMode](#)

9.120.3.18 setOperation()

```
void nvinfer1::IRNNv2Layer::setOperation (
    RNNOperation op ) [inline], [noexcept]
```

Set the operation of the RNN layer.

See also

[getOperation\(\)](#), [RNNOperation](#)

9.120.3.19 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](#).

9.120.3.20 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 [DataType::kFLOAT](#) or [DataType::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 $K * \text{getHiddenSize}()$ -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.
<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 setWeightsForGate() for documentation on the expected dimensions of this matrix.

9.120.4 Member Data Documentation

9.120.4.1 mImpl

```
apiv::VRNNv2Layer* nvinfer1::IRNNv2Layer::mImpl [protected]
```

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

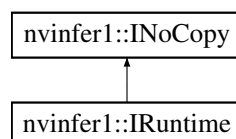
- [NvInfer.h](#)

9.121 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

9.121.1 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.

9.121.2 Constructor & Destructor Documentation

9.121.2.1 ~IRuntime()

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

9.121.3 Member Function Documentation

9.121.3.1 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.

9.121.3.2 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.

9.121.3.3 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.

9.121.3.4 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.

9.121.3.5 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.

9.121.3.6 getErrorRecorder()

```
IErrRecorder * nvinfer1::IRuntime::getErrorRecorder ( ) const [inline], [noexcept]
```

get the ErrRecorder 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\(\)](#)

9.121.3.7 getLogger()

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

get the logger with which the runtime was created

Returns

the logger

9.121.3.8 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\(\)](#)

9.121.3.9 getNbDLACores()

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

Returns number of DLA hardware cores accessible or 0 if DLA is unavailable.

9.121.3.10 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.

9.121.3.11 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\(\)](#)

9.121.3.12 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\(\)](#)

9.121.3.13 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 [BuilderFlag::kVERSION_COMPATIBLE](#) and [BuilderFlag::kEXCLUDE_LEAN_RUNTIME](#) both set and built with the same version of TensorRT as the loaded runtime library.

Parameters

<i>path</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.

9.121.3.14 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\(\)](#)

9.121.3.15 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.

9.121.3.16 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\(\)](#)

9.121.3.17 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.

9.121.3.18 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.

9.121.3.19 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\(\)](#)

9.121.3.20 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 the TMPDIR environment variable, then fall back to /tmp
- On Windows, TensorRT will try the TEMP environment variable.

See the TensorRT Developer Guide for more information.

The default value is nullptr.

Warning

If path is not nullptr, it must be a non-empty string representing a relative or absolute path in the format expected by the host operating system.

The string path must be null-terminated, and be at most 4096 bytes including the terminator. Note that the operating system may have stricter path length requirements.

The process using TensorRT must have rwx permissions for the temporary directory, and the directory shall be configured to disallow other users from modifying created files (e.g. on Linux, if the directory is shared with other users, the sticky bit must be set).

See also

[getTemporaryDirectory\(\)](#)

9.121.4 Member Data Documentation

9.121.4.1 mImpl

```
apiv::VRuntime* nvinfer1::IRuntime::mImpl [protected]
```

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

- [NvInferRuntime.h](#)

9.122 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 byte array.
- 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

9.122.1 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.

9.122.2 Constructor & Destructor Documentation

9.122.2.1 [IRuntime](#)() [1/3]

```
nvinfer1::safe::IRuntime::IRuntime ( ) [default]
```

9.122.2.2 ~IRuntime()

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

9.122.2.3 IRuntime() [2/3]

```
nvinfer1::safe::IRuntime::IRuntime (
    IRuntime const & ) [delete]
```

9.122.2.4 IRuntime() [3/3]

```
nvinfer1::safe::IRuntime::IRuntime (
    IRuntime && ) [delete]
```

9.122.3 Member Function Documentation

9.122.3.1 deserializeCudaEngine()

```
virtual ICudaEngine * nvinfer1::safe::IRuntime::deserializeCudaEngine (
    void const *const blob,
    std::size_t const size ) [pure virtual], [noexcept]
```

Deserialize an engine from a byte array.

If the serialized engine requires plugins the plugin creator must be registered by calling [IPluginRegistry::registerCreator\(\)](#) before calling [deserializeCudaEngine\(\)](#).

Parameters

<i>blob</i>	The memory that holds the serialized engine. The content must be a copy of the result of calling IHostMemory::data() on a serialized plan that was created via calling IBuilder::buildSerializedNetwork() on a network within the supported safety scope. Additionally, it must have been validated via IConsistencyChecker::validate() .
<i>size</i>	The size of the memory in bytes. This must be the result of calling IHostMemory::size() on the same IHostMemory object that is associated with the blob parameter.

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.

9.122.3.2 `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 or a previously assigned error recorder has been deregistered.

Returns

A pointer to the [IErrorRecorder](#) object that has been registered, or nullptr if no error recorder is set.

See also

[setErrorRecorder\(\)](#)

Usage considerations

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

9.122.3.3 operator=() [1/2]

```
IRuntime & nvinfer1::safe::IRuntime::operator= (
    IRuntime && ) & [delete]
```

9.122.3.4 operator=() [2/2]

```
IRuntime & nvinfer1::safe::IRuntime::operator= (
    IRuntime const & ) & [delete]
```

9.122.3.5 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. If the recorder is set to `nullptr`, an error code of [ErrorCode::kINVALID_ARGUMENT](#) will be emitted if the recorder has already been registered, or [ILogger::Severity::kERROR](#) will be logged if the recorder has not yet been registered.

Parameters

<i>recorder</i>	The error recorder to register with this interface, or <code>nullptr</code> to deregister the current error recorder.

See also

[getErrorRecorder\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.122.3.6 setGpuAllocator()

```
virtual void nvinfer1::safe::IRuntime::setGpuAllocator (
    IGpuAllocator *const allocator ) [pure virtual], [noexcept]
```

Set the GPU allocator.

Parameters

<i>allocator</i>	The GPU allocator to be used by the runtime. All GPU memory acquired will use this allocator. If nullptr is passed, the default allocator will be used, which calls cudaMalloc and cudaFree.

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

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

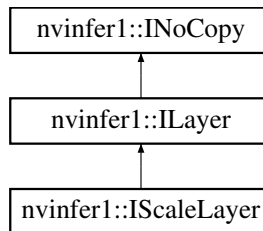
- [NvInferSafeRuntime.h](#)

9.123 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](#)

9.123.1 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.

9.123.2 Constructor & Destructor Documentation

9.123.2.1 ~IScaleLayer()

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

9.123.3 Member Function Documentation

9.123.3.1 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\(\)](#)

9.123.3.2 getMode()

```
ScaleMode nvinfer1::IScaleLayer::getMode ( ) const [inline], [noexcept]
```

Get the scale mode.

See also

[setMode\(\)](#)

9.123.3.3 `getPower()`

`Weights` `nvinfer1::IScaleLayer::getPower ( ) const [inline], [noexcept]`

Get the power value.

See also

[`setPower\(\)`](#)

9.123.3.4 `getScale()`

`Weights` `nvinfer1::IScaleLayer::getScale ( ) const [inline], [noexcept]`

Get the scale value.

See also

[`setScale\(\)`](#)

9.123.3.5 `getShift()`

`Weights` `nvinfer1::IScaleLayer::getShift ( ) const [inline], [noexcept]`

Get the shift value.

See also

[`setShift\(\)`](#)

9.123.3.6 `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\(\)`](#)

9.123.3.7 `setMode()`

```
void nvinfer1::IScaleLayer::setMode (
    ScaleMode mode ) [inline], [noexcept]
```

Set the scale mode.

See also

[getMode\(\)](#)

9.123.3.8 `setPower()`

```
void nvinfer1::IScaleLayer::setPower (
    Weights power ) [inline], [noexcept]
```

Set the power value.

See also

[getPower\(\)](#)

9.123.3.9 `setScale()`

```
void nvinfer1::IScaleLayer::setScale (
    Weights scale ) [inline], [noexcept]
```

Set the scale value.

See also

[getScale\(\)](#)

9.123.3.10 `setShift()`

```
void nvinfer1::IScaleLayer::setShift (
    Weights shift ) [inline], [noexcept]
```

Set the shift value.

See also

[getShift\(\)](#)

9.123.4 Member Data Documentation

9.123.4.1 mImpl

`apiv::VScaleLayer* nvinfer1::IScaleLayer::mImpl` [protected]

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

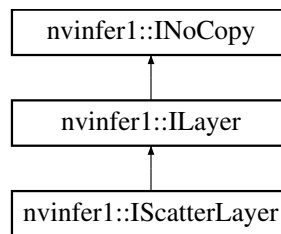
- [NvInfer.h](#)

9.124 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`

9.124.1 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.

9.124.2 Constructor & Destructor Documentation

9.124.2.1 ~IScatterLayer()

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

9.124.3 Member Function Documentation

9.124.3.1 getAxis()

```
int32_t nvinfer1::IScatterLayer::getAxis ( ) const [inline], [noexcept]
```

Get the axis.

9.124.3.2 getMode()

```
ScatterMode nvinfer1::IScatterLayer::getMode ( ) const [inline], [noexcept]
```

Get the scatter mode.

See also

[setMode\(\)](#)

9.124.3.3 setAxis()

```
void nvinfer1::IScatterLayer::setAxis (
    int32_t axis ) [inline], [noexcept]
```

Set the axis used by ScatterMode::kELEMENTS.

The axis defaults to 0.

9.124.3.4 setMode()

```
void nvinfer1::IScatterLayer::setMode (
    ScatterMode mode ) [inline], [noexcept]
```

Set the scatter mode.

See also

[getMode\(\)](#)

9.124.4 Member Data Documentation

9.124.4.1 mImpl

```
apiv::VScatterLayer* nvinfer1::IScatterLayer::mImpl [protected]
```

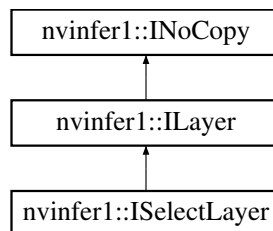
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.125 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

9.125.1 Detailed Description

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 ~ISelectLayer()**

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

9.125.3 Member Data Documentation**9.125.3.1 mImpl**

```
apiv::VSelectLayer* nvinfer1::ISelectLayer::mImpl [protected]
```

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

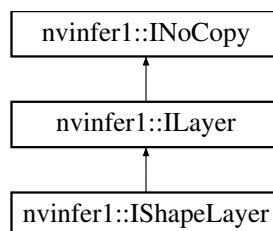
- [NvInfer.h](#)

9.126 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

9.126.1 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.

9.126.2 Constructor & Destructor Documentation

9.126.2.1 ~IShapeLayer()

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

9.126.3 Member Data Documentation

9.126.3.1 mImpl

```
apiv::VShapeLayer* nvinfer1::IShapeLayer::mImpl [protected]
```

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

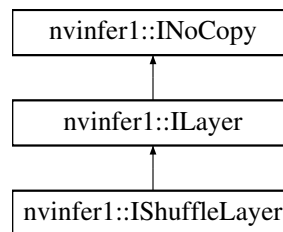
- [NvInfer.h](#)

9.127 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](#)

9.127.1 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.

9.127.2 Constructor & Destructor Documentation

9.127.2.1 ~IShuffleLayer()

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

9.127.3 Member Function Documentation

9.127.3.1 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](#)

9.127.3.2 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.

9.127.3.3 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](#)

9.127.3.4 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](#)

9.127.3.5 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](#)

9.127.3.6 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](#).

9.127.3.7 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.

9.127.3.8 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](#)

9.127.3.9 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\(\);](#)

9.127.4 Member Data Documentation

9.127.4.1 mImpl

```
apiv::VShuffleLayer* nvinfer1::IShuffleLayer::mImpl [protected]
```

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

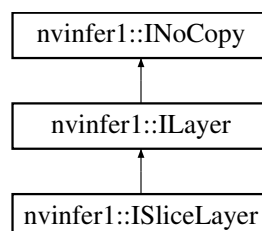
- [NvInfer.h](#)

9.128 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](#)

9.128.1 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 $2 * \text{Dims::MAX\_DIMS}$.

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.

9.128.2 Constructor & Destructor Documentation

9.128.2.1 ~ISliceLayer()

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

9.128.3 Member Function Documentation

9.128.3.1 getMode()

```
SliceMode nvinfer1::ISliceLayer::getMode ( ) const [inline], [noexcept]
```

Get the slice mode.

See also

[setMode\(\)](#)

9.128.3.2 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](#)

9.128.3.3 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](#)

9.128.3.4 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](#)

9.128.3.5 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.

9.128.3.6 setMode()

```
void nvinfer1::ISliceLayer::setMode (
    SliceMode mode ) [inline], [noexcept]
```

Set the slice mode.

See also

[getMode\(\)](#)

9.128.3.7 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](#)

9.128.3.8 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](#)

9.128.3.9 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](#)

9.128.4 Member Data Documentation

9.128.4.1 mImpl

```
apiv::VSliceLayer* nvinfer1::ISliceLayer::mImpl [protected]
```

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

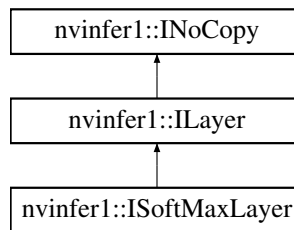
- [NvInfer.h](#)

9.129 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](#)

9.129.1 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.

9.129.2 Constructor & Destructor Documentation

9.129.2.1 ~ISoftMaxLayer()

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

9.129.3 Member Function Documentation

9.129.3.1 getAxes()

```
uint32_t nvinfer1::ISoftMaxLayer::getAxes ( ) const [inline], [noexcept]
```

Get the axis along which softmax occurs.

See also

[setAxes\(\)](#)

9.129.3.2 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.

9.129.4 Member Data Documentation

9.129.4.1 mImpl

```
apiv::VSoftMaxLayer* nvinfer1::ISoftMaxLayer::mImpl [protected]
```

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

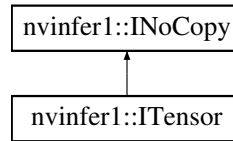
- [NvInfer.h](#)

9.130 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](#)

9.130.1 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.

9.130.2 Constructor & Destructor Documentation

9.130.2.1 [~ITensor\(\)](#)

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

9.130.3 Member Function Documentation

9.130.3.1 dynamicRangeIsSet()

```
bool nvinfer1::ITensor::dynamicRangeIsSet ( ) const [inline], [noexcept]
```

Query whether dynamic range is set.

Returns

True if dynamic range is set, false otherwise.

9.130.3.2 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\(\)](#)

9.130.3.3 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\(\)](#)

9.130.3.4 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\(\)](#)

9.130.3.5 `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\(\)](#)

9.130.3.6 `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.

9.130.3.7 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.

9.130.3.8 getLocation()

```
TensorLocation nvinfer1::ITensor::getLocation ( ) const [inline], [noexcept]
```

Get the storage location of a tensor.

Returns

The location of tensor data.

See also

[setLocation\(\)](#)

9.130.3.9 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\(\)](#)

9.130.3.10 `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\(\)](#)

9.130.3.11 `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.

9.130.3.12 `isNetworkInput()`

```
bool nvinfer1::ITensor::isNetworkInput ( ) const [inline], [noexcept]
```

Whether the tensor is a network input.

9.130.3.13 isNetworkOutput()

```
bool nvinfer1::ITensor::isNetworkOutput ( ) const [inline], [noexcept]
```

Whether the tensor is a network output.

9.130.3.14 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\(\)](#)

9.130.3.15 resetDynamicRange()

```
void nvinfer1::ITensor::resetDynamicRange ( ) [inline], [noexcept]
```

Undo effect of `setDynamicRange`.

9.130.3.16 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](#)

9.130.3.17 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\(\)](#)

9.130.3.18 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\(\)](#)

9.130.3.19 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\(\)](#)

9.130.3.20 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.

9.130.3.21 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\(\)](#)

9.130.3.22 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\(\)](#)

9.130.3.23 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\(\)](#)

9.130.4 Member Data Documentation

9.130.4.1 mImpl

```
apiv::VTensor* nvinfer1::ITensor::mImpl [protected]
```

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

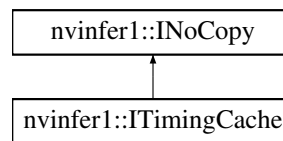
- [NvInfer.h](#)

9.131 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

9.131.1 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](#)

9.131.2 Constructor & Destructor Documentation

9.131.2.1 ~ITimingCache()

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

9.131.3 Member Function Documentation

9.131.3.1 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!

9.131.3.2 reset()

```
bool nvinfer1::ITimingCache::reset ( ) [inline], [noexcept]
```

Empty the timing cache.

Returns

True if reset successfully, false otherwise.

9.131.3.3 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](#)

9.131.4 Member Data Documentation

9.131.4.1 mImpl

```
apiv::VTimingCache* nvinfer1::ITimingCache::mImpl [protected]
```

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

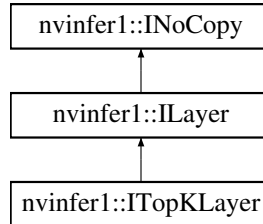
- [NvInfer.h](#)

9.132 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](#)

9.132.1 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.

9.132.2 Constructor & Destructor Documentation

9.132.2.1 ~ITopKLayer()

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

9.132.3 Member Function Documentation

9.132.3.1 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\(\)](#)

9.132.3.2 getOperation()

```
TopKOperation nvinfer1::ITopKLayer::getOperation ( ) const [inline], [noexcept]
```

Get the operation for the layer.

See also

[setOperation\(\)](#), [TopKOperation](#)

9.132.3.3 getReduceAxes()

```
uint32_t nvinfer1::ITopKLayer::getReduceAxes ( ) const [inline], [noexcept]
```

Get the axes to reduce for the layer.

See also

[setReduceAxes\(\)](#)

9.132.3.4 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.

9.132.3.5 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\(\)](#)

9.132.3.6 setOperation()

```
void nvinfer1::ITopKLayer::setOperation (
    TopKOperation op ) [inline], [noexcept]
```

Set the operation for the layer.

See also

[getOperation\(\)](#), [TopKOperation](#)

9.132.3.7 setReduceAxes()

```
void nvinfer1::ITopKLayer::setReduceAxes (
    uint32_t reduceAxes ) [inline], [noexcept]
```

Set which axes to reduce for the layer.

See also

[getReduceAxes\(\)](#)

9.132.4 Member Data Documentation

9.132.4.1 mImpl

```
apiv::VTopKLayer* nvinfer1::ITopKLayer::mImpl [protected]
```

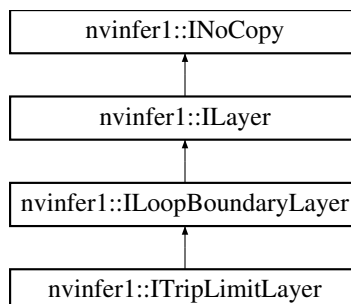
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.133 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`

9.133.1 Constructor & Destructor Documentation

9.133.1.1 ~ITripLimitLayer()

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

9.133.2 Member Function Documentation

9.133.2.1 getTripLimit()

```
TripLimit nvinfer1::ITripLimitLayer::getTripLimit ( ) const [inline], [noexcept]
```

9.133.3 Member Data Documentation

9.133.3.1 mImpl

```
apiv::VTripLimitLayer* nvinfer1::ITripLimitLayer::mImpl [protected]
```

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

- [NvInfer.h](#)

9.134 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.

9.134.1 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.

9.134.2 Constructor & Destructor Documentation

9.134.2.1 ~IUffParser()

```
virtual nvuffparser::IUffParser::~~IUffParser ( ) [virtual], [default], [noexcept]
```

9.134.3 Member Function Documentation

9.134.3.1 destroy()

```
virtual TRT\_DEPRECATED void nvuffparser::IUffParser::destroy ( ) [pure virtual], [noexcept]
```

Deprecated Use delete instead. Deprecated in TRT 8.0.

9.134.3.2 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\(\)](#)

9.134.3.3 getUffRequiredVersionMajor()

```
virtual int32_t nvuffparser::IUffParser::getUffRequiredVersionMajor ( ) [pure virtual], [noexcept]
```

Return Version Major of the UFF.

9.134.3.4 `getUffRequiredVersionMinor()`

```
virtual int32_t nvuffparser::IUffParser::getUffRequiredVersionMinor ( ) [pure virtual], [noexcept]
```

Return Version Minor of the UFF.

9.134.3.5 `getUffRequiredVersionPatch()`

```
virtual int32_t nvuffparser::IUffParser::getUffRequiredVersionPatch ( ) [pure virtual], [noexcept]
```

Return Patch Version of the UFF.

9.134.3.6 `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.

9.134.3.7 `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.

9.134.3.8 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.

9.134.3.9 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.

9.134.3.10 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\(\)](#)

9.134.3.11 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:

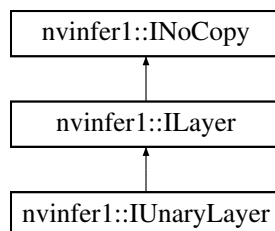
- [NvUffParser.h](#)

9.135 nvinfer1::IUnaryLayer Class Reference

Layer that represents an unary operation.

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

Inheritance diagram for `nvinfer1::IUnaryLayer`:



Public Member Functions

- void [setOperation](#) ([UnaryOperation](#) op) noexcept
Set the unary operation for the layer.
- [UnaryOperation](#) [getOperation](#) () const noexcept
Get the unary operation for the layer.

Protected Member Functions

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

Protected Attributes

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

9.135.1 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.

9.135.2 Constructor & Destructor Documentation

9.135.2.1 ~UnaryLayer()

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

9.135.3 Member Function Documentation

9.135.3.1 getOperation()

```
UnaryOperation nvinfer1::UnaryLayer::getOperation ( ) const [inline], [noexcept]
```

Get the unary operation for the layer.

See also

[setOperation\(\)](#), [UnaryOperation](#)

9.135.3.2 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](#)

9.135.4 Member Data Documentation

9.135.4.1 mImpl

```
apiv::VUnaryLayer* nvinfer1::UnaryLayer::mImpl [protected]
```

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

- [NvInfer.h](#)

9.136 nvinfer1::plugin::NMSPParameters Struct Reference

The [NMSPParameters](#) 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](#)

9.136.1 Detailed Description

The [NMSPParameters](#) 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.

9.136.2 Member Data Documentation

9.136.2.1 backgroundLabelId

```
int32_t nvinfer1::plugin::NMSParameters::backgroundLabelId
```

9.136.2.2 iouThreshold

```
float nvinfer1::plugin::NMSParameters::iouThreshold
```

9.136.2.3 isNormalized

```
bool nvinfer1::plugin::NMSParameters::isNormalized
```

9.136.2.4 keepTopK

```
int32_t nvinfer1::plugin::NMSParameters::keepTopK
```

9.136.2.5 numClasses

```
int32_t nvinfer1::plugin::NMSParameters::numClasses
```

9.136.2.6 scoreThreshold

```
float nvinfer1::plugin::NMSParameters::scoreThreshold
```

9.136.2.7 shareLocation

```
bool nvinfer1::plugin::NMSParameters::shareLocation
```

9.136.2.8 topK

```
int32_t nvinfer1::plugin::NMSParameters::topK
```

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

- [NvInferPluginUtils.h](#)

9.137 nvinfer1::Permutation Struct Reference

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

Public Attributes

- int32_t [order](#) [[Dims::MAX_DIMS](#)]

9.137.1 Member Data Documentation

9.137.1.1 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](#)

9.138 nvinfer1::PluginField Class Reference

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

```
#include <NvInferRuntimePlugin.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.

9.138.1 Detailed Description

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

9.138.2 Constructor & Destructor Documentation

9.138.2.1 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]
```

9.138.3 Member Data Documentation

9.138.3.1 data

```
void const* nvinfer1::PluginField::data
```

Plugin field attribute data.

9.138.3.2 length

```
int32_t nvinfer1::PluginField::length
```

Number of data entries in the Plugin attribute.

9.138.3.3 name

```
AsciiChar const* nvinfer1::PluginField::name
```

Plugin field attribute name.

9.138.3.4 type

```
PluginFieldType nvinfer1::PluginField::type
```

Plugin field attribute type.

See also

[PluginFieldType](#)

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

- [NvInferRuntimePlugin.h](#)

9.139 nvinfer1::PluginFieldCollection Struct Reference

Plugin field collection struct.

```
#include <NvInferRuntimePlugin.h>
```

Public Attributes

- `int32_t nbFields`
Number of [PluginField](#) entries.
- `PluginField const * fields`
Pointer to [PluginField](#) entries.

9.139.1 Detailed Description

Plugin field collection struct.

9.139.2 Member Data Documentation

9.139.2.1 fields

```
PluginField const* nvinfer1::PluginFieldCollection::fields
```

Pointer to [PluginField](#) entries.

9.139.2.2 nbFields

```
int32_t nvinfer1::PluginFieldCollection::nbFields
```

Number of [PluginField](#) entries.

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

- [NvInferRuntimePlugin.h](#)

9.140 `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\(\)](#)

9.140.1 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.

9.140.2 Constructor & Destructor Documentation

9.140.2.1 `PluginRegistrar()`

```
template<typename T>
nvinfer1::PluginRegistrar< T >::PluginRegistrar ( ) [inline]
```

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

- [NvInferRuntime.h](#)

9.141 `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](#) ()

9.141.1 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 must be avoided in the automotive safety context as the application developer must first register an error recorder with the plugin registry via [IPluginRegistry::setErrorRecorder\(\)](#) before using [IPluginRegistry::registerCreator\(\)](#) or other methods.

9.141.2 Constructor & Destructor Documentation

9.141.2.1 PluginRegistrar()

```
template<typename T >
nvinfer1::safe::PluginRegistrar< T >::PluginRegistrar ( ) [inline]
```

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

- [NvInferSafeRuntime.h](#)

9.142 nvinfer1::PluginTensorDesc Struct Reference

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

```
#include <NvInferRuntimePlugin.h>
```

Public Attributes

- [Dims](#) `dims`
Dimensions.
- [DataType](#) `type`
- [TensorFormat](#) `format`
Tensor format.
- float [scale](#)
Scale for INT8 data type.

9.142.1 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](#)

9.142.2 Member Data Documentation

9.142.2.1 dims

`Dims nvinfer1::PluginTensorDesc::dims`

Dimensions.

9.142.2.2 format

`TensorFormat nvinfer1::PluginTensorDesc::format`

Tensor format.

9.142.2.3 scale

`float nvinfer1::PluginTensorDesc::scale`

Scale for INT8 data type.

9.142.2.4 type

`DataType nvinfer1::PluginTensorDesc::type`

Warning

`DataType:kBOOL` and `DataType::kUINT8` are not supported.

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

- [NvInferRuntimePlugin.h](#)

9.143 PluginVersion Struct Reference

Definition of plugin versions.

```
#include <NvInferRuntimePlugin.h>
```

9.143.1 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:

- [NvInferRuntimePlugin.h](#)

9.144 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](#)

9.144.1 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.

9.144.2 Member Data Documentation

9.144.2.1 aspectRatios

```
float * nvinfer1::plugin::PriorBoxParameters::aspectRatios
```

9.144.2.2 clip

```
bool nvinfer1::plugin::PriorBoxParameters::clip
```

9.144.2.3 flip

```
bool nvinfer1::plugin::PriorBoxParameters::flip
```

9.144.2.4 imgH

```
int32_t nvinfer1::plugin::PriorBoxParameters::imgH
```

9.144.2.5 imgW

```
int32_t nvinfer1::plugin::PriorBoxParameters::imgW
```

9.144.2.6 maxSize

```
float * nvinfer1::plugin::PriorBoxParameters::maxSize
```

9.144.2.7 minSize

```
float* nvinfer1::plugin::PriorBoxParameters::minSize
```

9.144.2.8 numAspectRatios

```
int32_t nvinfer1::plugin::PriorBoxParameters::numAspectRatios
```

9.144.2.9 numMaxSize

```
int32_t nvinfer1::plugin::PriorBoxParameters::numMaxSize
```

9.144.2.10 numMinSize

```
int32_t nvinfer1::plugin::PriorBoxParameters::numMinSize
```

9.144.2.11 offset

```
float nvinfer1::plugin::PriorBoxParameters::offset
```

9.144.2.12 stepH

```
float nvinfer1::plugin::PriorBoxParameters::stepH
```

9.144.2.13 stepW

```
float nvinfer1::plugin::PriorBoxParameters::stepW
```

9.144.2.14 variance

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

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

- [NvInferPluginUtils.h](#)

9.145 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]

9.145.1 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.

9.145.2 Member Data Documentation

9.145.2.1 data

```
int32_t nvinfer1::plugin::Quadruple::data[4]
```

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

- [NvInferPluginUtils.h](#)

9.146 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](#)

9.146.1 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.

9.146.2 Member Data Documentation

9.146.2.1 classes

```
int32_t nvinfer1::plugin::RegionParameters::classes
```

9.146.2.2 coords

```
int32_t nvinfer1::plugin::RegionParameters::coords
```

9.146.2.3 num

```
int32_t nvinfer1::plugin::RegionParameters::num
```

9.146.2.4 smTree

```
softmaxTree* nvinfer1::plugin::RegionParameters::smTree
```

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

- [NvInferPluginUtils.h](#)

9.147 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](#)

9.147.1 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.

9.147.2 Member Data Documentation**9.147.2.1 anchorsRatioCount**

```
int32_t nvinfer1::plugin::RPROIParams::anchorsRatioCount
```

9.147.2.2 anchorsScaleCount

```
int32_t nvinfer1::plugin::RPROIParams::anchorsScaleCount
```

9.147.2.3 featureStride

```
int32_t nvinfer1::plugin::RPROIParams::featureStride
```

9.147.2.4 iouThreshold

```
float nvinfer1::plugin::RPROIParams::iouThreshold
```

9.147.2.5 minBoxSize

```
float nvinfer1::plugin::RPROIParams::minBoxSize
```

9.147.2.6 nmsMaxOut

```
int32_t nvinfer1::plugin::RPROIParams::nmsMaxOut
```

9.147.2.7 poolingH

```
int32_t nvinfer1::plugin::RPROIParams::poolingH
```

9.147.2.8 poolingW

```
int32_t nvinfer1::plugin::RPROIParams::poolingW
```

9.147.2.9 preNmsTop

```
int32_t nvinfer1::plugin::RPROIParams::preNmsTop
```

9.147.2.10 spatialScale

```
float nvinfer1::plugin::RPROIParams::spatialScale
```

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

- [NvInferPluginUtils.h](#)

9.148 nvinfer1::safe::RuntimeErrorInformation Struct Reference

Space to record information about runtime errors.

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

Public Attributes

- `uint64_t bitMask`

Each bit represents a `RuntimeErrorType` that has occurred during kernel execution.

9.148.1 Detailed Description

Space to record information about runtime errors.

`kNAN_CONSUMED` errors occur when NAN values are stored in an INT8 quantized datatype. `kINF_CONSUMED` errors occur when `+/-INF` values are stored in an INT8 quantized datatype. `kGATHER_OOB` errors occur when a gather index tensor contains a value that is outside of the data tensor. `kSCATTER_OOB` and `kSCATTER_RACE` are reserved for future use.

Mark the `RuntimeErrorType` that occurs during asynchronous kernel execution.

9.148.2 Member Data Documentation

9.148.2.1 bitMask

```
uint64_t nvinfer1::safe::RuntimeErrorInformation::bitMask
```

Each bit represents a `RuntimeErrorType` that has occurred during kernel execution.

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

- [NvInferSafeRuntime.h](#)

9.149 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`

9.149.1 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.

9.149.2 Member Data Documentation

9.149.2.1 child

```
int32_t* nvinfer1::plugin::softmaxTree::child
```

9.149.2.2 group

```
int32_t* nvinfer1::plugin::softmaxTree::group
```

9.149.2.3 groupOffset

```
int32_t* nvinfer1::plugin::softmaxTree::groupOffset
```

9.149.2.4 groups

```
int32_t nvinfer1::plugin::softmaxTree::groups
```

9.149.2.5 groupSize

```
int32_t* nvinfer1::plugin::softmaxTree::groupSize
```

9.149.2.6 leaf

```
int32_t* nvinfer1::plugin::softmaxTree::leaf
```

9.149.2.7 n

```
int32_t nvinfer1::plugin::softmaxTree::n
```

9.149.2.8 name

```
char** nvinfer1::plugin::softmaxTree::name
```

9.149.2.9 parent

```
int32_t* nvinfer1::plugin::softmaxTree::parent
```

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

- [NvInferPluginUtils.h](#)

9.150 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.

9.150.1 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`).

9.150.2 Member Data Documentation

9.150.2.1 `count`

```
int64_t nvinfer1::Weights::count
```

The number of weights in the array.

9.150.2.2 `type`

```
DataType nvinfer1::Weights::type
```

The type of the weights.

9.150.2.3 `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) 1993-2023 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::Int8Calibrator](#)
Application-implemented interface for calibration.
- class [nvinfer1::Int8EntropyCalibrator](#)
- class [nvinfer1::Int8EntropyCalibrator2](#)
- class [nvinfer1::Int8MinMaxCalibrator](#)
- class [nvinfer1::Int8LegacyCalibrator](#)
- 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](#)
- namespace [nvinfer1::safe](#)
The safety subset of TensorRT's API version 1 namespace.

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 BuilderFlag 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 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 }

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 building a Standard engine, or nullptr if no registry exists.

- `nvinfer1::safe::IPluginRegistry * nvinfer1::getBuilderSafePluginRegistry (nvinfer1::EngineCapability capability) noexcept`

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

10.3.1 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) 1993-2023 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
25     enum class LayerType : int32_t
26     {
27         kCONVOLUTION = 0,
28         kFULLY_CONNECTED = 1,
29         kACTIVATION = 2,
30         kPOOLING = 3,
31         kLRN = 4,
32         kSCALE = 5,
33         kSOFTMAX = 6,
34         kDECONVOLUTION = 7,
35         kCONCATENATION = 8,
36         kELEMENTWISE = 9,
37         kPLUGIN = 10,
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

```

```

65     kUNARY = 11,
66     kPADDING = 12,
67     kSHUFFLE = 13,
68     kREDUCE = 14,
69     kTOPK = 15,
70     kGATHER = 16,
71     kMATRIX.MULTIPLY = 17,
72     kRAGGED_SOFTMAX = 18,
73     kCONSTANT = 19,
74     kRNN.V2 = 20,
75     kIDENTITY = 21,
76     kPLUGIN.V2 = 22,
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

```

```

204     {
205         return mImpl->getName();
206     }
207
222     void setDimensions(Dims dimensions) noexcept
223     {
224         mImpl->setDimensions(dimensions);
225     }
226
235     Dims getDimensions() const noexcept
236     {
237         return mImpl->getDimensions();
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
788     void setMetadata(char const* metadata) noexcept
789     {
790         mLayer->setMetadata(metadata);
791     }
792
801     char const* getMetadata() const noexcept
802     {
803         return mLayer->getMetadata();
804     }
805
806 protected:
807     virtual ~ILayer() noexcept = default;
808     apiv::VLayer* mLayer;
809 };
810
1033 enum class PaddingMode : int32_t
1034 {
1035     kEXPLICIT_ROUND_DOWN = 0,
1036     kEXPLICIT_ROUND_UP = 1,
1037     kSAME_UPPER = 2,
1038     kSAME_LOWER = 3,
1039     kCAFFE_ROUND_DOWN = 4,
1040     kCAFFE_ROUND_UP = 5
1041 };
1042
1043 namespace impl
1044 {
1050     template <>
1051     struct EnumMaxImpl<PaddingMode>
1052     {
1053         static constexpr int32_t kVALUE = 6;
1054     };
1055 } // namespace impl
1056
1069 class IConvolutionLayer : public ILayer
1070 {
1071 public:
1081     TRT_DEPRECATED void setKernelSize(DimsHW kernelSize) noexcept
1082     {
1083         mImpl->setKernelSize(kernelSize);
1084     }
1085
1093     TRT_DEPRECATED DimsHW getKernelSize() const noexcept
1094     {
1095         return mImpl->getKernelSize();
1096     }

```

```

1097
1105 void setNbOutputMaps(int32_t nbOutputMaps) noexcept
1106 {
1107     mImpl->setNbOutputMaps(nbOutputMaps);
1108 }
1109
1115 int32_t getNbOutputMaps() const noexcept
1116 {
1117     return mImpl->getNbOutputMaps();
1118 }
1119
1131 TRT_DEPRECATED void setStride(DimsHW stride) noexcept
1132 {
1133     mImpl->setStride(stride);
1134 }
1135
1141 TRT_DEPRECATED DimsHW getStride() const noexcept
1142 {
1143     return mImpl->getStride();
1144 }
1145
1161 TRT_DEPRECATED void setPadding(DimsHW padding) noexcept
1162 {
1163     return mImpl->setPadding(padding);
1164 }
1165
1173 TRT_DEPRECATED DimsHW getPadding() const noexcept
1174 {
1175     return mImpl->getPadding();
1176 }
1177
1193 void setNbGroups(int32_t nbGroups) noexcept
1194 {
1195     mImpl->setNbGroups(nbGroups);
1196 }
1197
1203 int32_t getNbGroups() const noexcept
1204 {
1205     return mImpl->getNbGroups();
1206 }
1207
1217 void setKernelWeights(Weights weights) noexcept
1218 {
1219     mImpl->setKernelWeights(weights);
1220 }
1221
1227 Weights getKernelWeights() const noexcept
1228 {
1229     return mImpl->getKernelWeights();
1230 }
1231
1242 void setBiasWeights(Weights weights) noexcept
1243 {
1244     mImpl->setBiasWeights(weights);
1245 }
1246
1252 Weights getBiasWeights() const noexcept
1253 {
1254     return mImpl->getBiasWeights();
1255 }
1256
1268 TRT_DEPRECATED void setDilation(DimsHW dilation) noexcept
1269 {
1270     return mImpl->setDilation(dilation);
1271 }
1272
1280 TRT_DEPRECATED DimsHW getDilation() const noexcept
1281 {
1282     return mImpl->getDilation();
1283 }
1284
1297 void setPrePadding(Dims padding) noexcept
1298 {
1299     mImpl->setPrePadding(padding);
1300 }
1301
1307 Dims getPrePadding() const noexcept
1308 {
1309     return mImpl->getPrePadding();
1310 }
1311

```

```

1324 void setPostPadding(Dims padding) noexcept
1325 {
1326     mImpl->setPostPadding(padding);
1327 }
1328
1334 Dims getPostPadding() const noexcept
1335 {
1336     return mImpl->getPostPadding();
1337 }
1338
1348 void setPaddingMode(PaddingMode paddingMode) noexcept
1349 {
1350     mImpl->setPaddingMode(paddingMode);
1351 }
1352
1360 PaddingMode getPaddingMode() const noexcept
1361 {
1362     return mImpl->getPaddingMode();
1363 }
1364
1373 void setKernelSizeNd(Dims kernelSize) noexcept
1374 {
1375     mImpl->setKernelSizeNd(kernelSize);
1376 }
1377
1383 Dims getKernelSizeNd() const noexcept
1384 {
1385     return mImpl->getKernelSizeNd();
1386 }
1387
1398 void setStrideNd(Dims stride) noexcept
1399 {
1400     mImpl->setStrideNd(stride);
1401 }
1402
1408 Dims getStrideNd() const noexcept
1409 {
1410     return mImpl->getStrideNd();
1411 }
1412
1426 void setPaddingNd(Dims padding) noexcept
1427 {
1428     mImpl->setPaddingNd(padding);
1429 }
1430
1438 Dims getPaddingNd() const noexcept
1439 {
1440     return mImpl->getPaddingNd();
1441 }
1442
1452 void setDilationNd(Dims dilation) noexcept
1453 {
1454     mImpl->setDilationNd(dilation);
1455 }
1456
1462 Dims getDilationNd() const noexcept
1463 {
1464     return mImpl->getDilationNd();
1465 }
1466
1480 using ILayer::setInput;
1481
1482 protected:
1483     virtual ~IConvolutionLayer() noexcept = default;
1484     apiv::VConvolutionLayer* mImpl;
1485 };
1486
1518 class TRT_DEPRECATED IFullyConnectedLayer : public ILayer
1519 {
1520 public:
1528     void setNbOutputChannels(int32_t nbOutputs) noexcept
1529     {
1530         mImpl->setNbOutputChannels(nbOutputs);
1531     }
1532
1538     int32_t getNbOutputChannels() const noexcept
1539     {
1540         return mImpl->getNbOutputChannels();
1541     }
1542
1548     void setKernelWeights(Weights weights) noexcept

```

```

1549     {
1550         mImpl->setKernelWeights(weights);
1551     }
1552
1558     Weights getKernelWeights() const noexcept
1559     {
1560         return mImpl->getKernelWeights();
1561     }
1562
1570     void setBiasWeights(Weights weights) noexcept
1571     {
1572         mImpl->setBiasWeights(weights);
1573     }
1574
1580     Weights getBiasWeights() const noexcept
1581     {
1582         return mImpl->getBiasWeights();
1583     }
1584
1606     using ILayer::setInput;
1607
1608 protected:
1609     virtual ~IFullyConnectedLayer() noexcept = default;
1610     apiv::VFullyConnectedLayer* mImpl;
1611 };
1612
1626 class IActivationLayer : public ILayer
1627 {
1628 public:
1636     void setActivationType(ActivationType type) noexcept
1637     {
1638         mImpl->setActivationType(type);
1639     }
1640
1646     ActivationType getActivationType() const noexcept
1647     {
1648         return mImpl->getActivationType();
1649     }
1650
1661     void setAlpha(float alpha) noexcept
1662     {
1663         mImpl->setAlpha(alpha);
1664     }
1665
1675     void setBeta(float beta) noexcept
1676     {
1677         mImpl->setBeta(beta);
1678     }
1679
1684     float getAlpha() const noexcept
1685     {
1686         return mImpl->getAlpha();
1687     }
1688
1693     float getBeta() const noexcept
1694     {
1695         return mImpl->getBeta();
1696     }
1697
1698 protected:
1699     virtual ~IActivationLayer() noexcept = default;
1700     apiv::VActivationLayer* mImpl;
1701 };
1702
1708 enum class PoolingType : int32_t
1709 {
1710     kMAX = 0,                // Maximum over elements
1711     kAVERAGE = 1,          // Average over elements. If the tensor is padded, the count includes the
padding
1712     kMAX_AVERAGE_BLEND = 2 // Blending between max and average pooling: (1-blendFactor)*maxPool +
blendFactor*avgPool
1713 };
1714
1715 namespace impl
1716 {
1722 template <>
1723 struct EnumMaxImpl<PoolingType>
1724 {
1725     static constexpr int32_t kVALUE = 3;
1726 };
1727 } // namespace impl

```



```

1728
1740 class IPoolingLayer : public ILayer
1741 {
1742 public:
1750     void setPoolingType(PoolingType type) noexcept
1751     {
1752         mImpl->setPoolingType(type);
1753     }
1754
1760     PoolingType getPoolingType() const noexcept
1761     {
1762         return mImpl->getPoolingType();
1763     }
1764
1774     TRT_DEPRECATED void setWindowSize(DimsHW windowSize) noexcept
1775     {
1776         mImpl->setWindowSize(windowSize);
1777     }
1778
1786     TRT_DEPRECATED DimsHW getWindowSize() const noexcept
1787     {
1788         return mImpl->getWindowSize();
1789     }
1790
1802     TRT_DEPRECATED void setStride(DimsHW stride) noexcept
1803     {
1804         mImpl->setStride(stride);
1805     }
1806
1814     TRT_DEPRECATED DimsHW getStride() const noexcept
1815     {
1816         return mImpl->getStride();
1817     }
1818
1830     TRT_DEPRECATED void setPadding(DimsHW padding) noexcept
1831     {
1832         mImpl->setPadding(padding);
1833     }
1834
1844     TRT_DEPRECATED DimsHW getPadding() const noexcept
1845     {
1846         return mImpl->getPadding();
1847     }
1848
1859     void setBlendFactor(float blendFactor) noexcept
1860     {
1861         mImpl->setBlendFactor(blendFactor);
1862     }
1863
1872     float getBlendFactor() const noexcept
1873     {
1874         return mImpl->getBlendFactor();
1875     }
1876
1889     void setAverageCountExcludesPadding(bool exclusive) noexcept
1890     {
1891         mImpl->setAverageCountExcludesPadding(exclusive);
1892     }
1893
1900     bool getAverageCountExcludesPadding() const noexcept
1901     {
1902         return mImpl->getAverageCountExcludesPadding();
1903     }
1904
1918     void setPrePadding(Dims padding) noexcept
1919     {
1920         mImpl->setPrePadding(padding);
1921     }
1922
1928     Dims getPrePadding() const noexcept
1929     {
1930         return mImpl->getPrePadding();
1931     }
1932
1946     void setPostPadding(Dims padding) noexcept
1947     {
1948         mImpl->setPostPadding(padding);
1949     }
1950
1956     Dims getPostPadding() const noexcept
1957     {

```

```

1958         return mImpl->getPostPadding();
1959     }
1960
1969     void setPaddingMode(PaddingMode paddingMode) noexcept
1970     {
1971         mImpl->setPaddingMode(paddingMode);
1972     }
1973
1980     PaddingMode getPaddingMode() const noexcept
1981     {
1982         return mImpl->getPaddingMode();
1983     }
1984
1993     void setWindowSizeNd(Dims windowSize) noexcept
1994     {
1995         mImpl->setWindowSizeNd(windowSize);
1996     }
1997
2003     Dims getWindowSizeNd() const noexcept
2004     {
2005         return mImpl->getWindowSizeNd();
2006     }
2007
2018     void setStrideNd(Dims stride) noexcept
2019     {
2020         mImpl->setStrideNd(stride);
2021     }
2022
2028     Dims getStrideNd() const noexcept
2029     {
2030         return mImpl->getStrideNd();
2031     }
2032
2047     void setPaddingNd(Dims padding) noexcept
2048     {
2049         mImpl->setPaddingNd(padding);
2050     }
2051
2059     Dims getPaddingNd() const noexcept
2060     {
2061         return mImpl->getPaddingNd();
2062     }
2063
2064 protected:
2065     virtual ~IPoolingLayer() noexcept = default;
2066     apiv::VPoolingLayer* mImpl;
2067 };
2068
2078 class ILRNLayer : public ILayer
2079 {
2080 public:
2090     void setWindowSize(int32_t windowSize) noexcept
2091     {
2092         mImpl->setWindowSize(windowSize);
2093     }
2094
2100     int32_t getWindowSize() const noexcept
2101     {
2102         return mImpl->getWindowSize();
2103     }
2104
2111     void setAlpha(float alpha) noexcept
2112     {
2113         mImpl->setAlpha(alpha);
2114     }
2115
2121     float getAlpha() const noexcept
2122     {
2123         return mImpl->getAlpha();
2124     }
2125
2132     void setBeta(float beta) noexcept
2133     {
2134         mImpl->setBeta(beta);
2135     }
2136
2142     float getBeta() const noexcept
2143     {
2144         return mImpl->getBeta();
2145     }
2146

```

```

2153     void setK(float k) noexcept
2154     {
2155         mImpl->setK(k);
2156     }
2157
2163     float getK() const noexcept
2164     {
2165         return mImpl->getK();
2166     }
2167
2168 protected:
2169     virtual ~ILRNLayer() noexcept = default;
2170     apiv::VLRNLayer* mImpl;
2171 };
2172
2178 enum class ScaleMode : int32_t
2179 {
2180     kUNIFORM = 0,
2181     kCHANNEL = 1,
2182     kELEMENTWISE = 2
2183 };
2184
2190 template <>
2191 constexpr inline int32_t EnumMax<ScaleMode>() noexcept
2192 {
2193     return 3;
2194 }
2195
2222 class IScaleLayer : public ILayer
2223 {
2224 public:
2230     void setMode(ScaleMode mode) noexcept
2231     {
2232         mImpl->setMode(mode);
2233     }
2234
2240     ScaleMode getMode() const noexcept
2241     {
2242         return mImpl->getMode();
2243     }
2244
2250     void setShift(Weights shift) noexcept
2251     {
2252         mImpl->setShift(shift);
2253     }
2254
2260     Weights getShift() const noexcept
2261     {
2262         return mImpl->getShift();
2263     }
2264
2270     void setScale(Weights scale) noexcept
2271     {
2272         mImpl->setScale(scale);
2273     }
2274
2280     Weights getScale() const noexcept
2281     {
2282         return mImpl->getScale();
2283     }
2284
2290     void setPower(Weights power) noexcept
2291     {
2292         mImpl->setPower(power);
2293     }
2294
2300     Weights getPower() const noexcept
2301     {
2302         return mImpl->getPower();
2303     }
2304
2315     int32_t getChannelAxis() const noexcept
2316     {
2317         return mImpl->getChannelAxis();
2318     }
2319
2336     void setChannelAxis(int32_t channelAxis) noexcept
2337     {
2338         mImpl->setChannelAxis(channelAxis);
2339     }
2340

```

```

2341 protected:
2342     virtual ~IScaleLayer() noexcept = default;
2343     apiv::VScaleLayer* mImpl;
2344 };
2345
2367 class ISoftMaxLayer : public ILayer
2368 {
2369 public:
2400     void setAxes(uint32_t axes) noexcept
2401     {
2402         mImpl->setAxes(axes);
2403     }
2404
2410     uint32_t getAxes() const noexcept
2411     {
2412         return mImpl->getAxes();
2413     }
2414
2415 protected:
2416     virtual ~ISoftMaxLayer() noexcept = default;
2417     apiv::VSoftMaxLayer* mImpl;
2418 };
2419
2432 class IConcatenationLayer : public ILayer
2433 {
2434 public:
2447     void setAxis(int32_t axis) noexcept
2448     {
2449         mImpl->setAxis(axis);
2450     }
2451
2457     int32_t getAxis() const noexcept
2458     {
2459         return mImpl->getAxis();
2460     }
2461
2462 protected:
2463     virtual ~IConcatenationLayer() noexcept = default;
2464     apiv::VConcatenationLayer* mImpl;
2465 };
2466
2474 class IDeconvolutionLayer : public ILayer
2475 {
2476 public:
2488     TRT_DEPRECATED void setKernelSize(DimsHW kernelSize) noexcept
2489     {
2490         mImpl->setKernelSize(kernelSize);
2491     }
2492
2500     TRT_DEPRECATED DimsHW getKernelSize() const noexcept
2501     {
2502         return mImpl->getKernelSize();
2503     }
2504
2512     void setNbOutputMaps(int32_t nbOutputMaps) noexcept
2513     {
2514         mImpl->setNbOutputMaps(nbOutputMaps);
2515     }
2516
2522     int32_t getNbOutputMaps() const noexcept
2523     {
2524         return mImpl->getNbOutputMaps();
2525     }
2526
2538     TRT_DEPRECATED void setStride(DimsHW stride) noexcept
2539     {
2540         mImpl->setStride(stride);
2541     }
2542
2550     TRT_DEPRECATED DimsHW getStride() const noexcept
2551     {
2552         return mImpl->getStride();
2553     }
2554
2570     TRT_DEPRECATED void setPadding(DimsHW padding) noexcept
2571     {
2572         mImpl->setPadding(padding);
2573     }
2574
2584     TRT_DEPRECATED DimsHW getPadding() const noexcept
2585     {

```

```

2586         return mImpl->getPadding();
2587     }
2588
2604     void setNbGroups(int32_t nbGroups) noexcept
2605     {
2606         mImpl->setNbGroups(nbGroups);
2607     }
2608
2614     int32_t getNbGroups() const noexcept
2615     {
2616         return mImpl->getNbGroups();
2617     }
2618
2628     void setKernelWeights(Weights weights) noexcept
2629     {
2630         mImpl->setKernelWeights(weights);
2631     }
2632
2638     Weights getKernelWeights() const noexcept
2639     {
2640         return mImpl->getKernelWeights();
2641     }
2642
2653     void setBiasWeights(Weights weights) noexcept
2654     {
2655         mImpl->setBiasWeights(weights);
2656     }
2657
2663     Weights getBiasWeights() const noexcept
2664     {
2665         return mImpl->getBiasWeights();
2666     }
2667
2681     void setPrePadding(Dims padding) noexcept
2682     {
2683         mImpl->setPrePadding(padding);
2684     }
2685
2691     Dims getPrePadding() const noexcept
2692     {
2693         return mImpl->getPrePadding();
2694     }
2695
2709     void setPostPadding(Dims padding) noexcept
2710     {
2711         mImpl->setPostPadding(padding);
2712     }
2713
2719     Dims getPostPadding() const noexcept
2720     {
2721         return mImpl->getPostPadding();
2722     }
2723
2733     void setPaddingMode(PaddingMode paddingMode) noexcept
2734     {
2735         mImpl->setPaddingMode(paddingMode);
2736     }
2737
2745     PaddingMode getPaddingMode() const noexcept
2746     {
2747         return mImpl->getPaddingMode();
2748     }
2749
2760     void setKernelSizeNd(Dims kernelSize) noexcept
2761     {
2762         mImpl->setKernelSizeNd(kernelSize);
2763     }
2764
2770     Dims getKernelSizeNd() const noexcept
2771     {
2772         return mImpl->getKernelSizeNd();
2773     }
2774
2787     void setStrideNd(Dims stride) noexcept
2788     {
2789         mImpl->setStrideNd(stride);
2790     }
2791
2797     Dims getStrideNd() const noexcept
2798     {
2799         return mImpl->getStrideNd();

```

```

2800     }
2801
2815     void setPaddingNd(Dims padding) noexcept
2816     {
2817         mImpl->setPaddingNd(padding);
2818     }
2819
2827     Dims getPaddingNd() const noexcept
2828     {
2829         return mImpl->getPaddingNd();
2830     }
2831
2843     using ILayer::setInput;
2844
2851     void setDilationNd(Dims dilation) noexcept
2852     {
2853         mImpl->setDilationNd(dilation);
2854     }
2855
2861     Dims getDilationNd() const noexcept
2862     {
2863         return mImpl->getDilationNd();
2864     }
2865
2866 protected:
2867     virtual ~IDEconvolutionLayer() noexcept = default;
2868     apiv::VDeconvolutionLayer* mImpl;
2869 };
2870
2884 enum class ElementWiseOperation : int32_t
2885 {
2886     kSUM = 0,
2887     kPROD = 1,
2888     kMAX = 2,
2889     kMIN = 3,
2890     kSUB = 4,
2891     kDIV = 5,
2892     kPOW = 6,
2893     kFLOOR_DIV = 7,
2894     kAND = 8,
2895     kOR = 9,
2896     kXOR = 10,
2897     kEQUAL = 11,
2898     kGREATER = 12,
2899     kLESS = 13
2900 };
2901
2902 namespace impl
2903 {
2904     template <>
2910     struct EnumMaxImpl<ElementWiseOperation>
2911     {
2912         static constexpr int32_t kVALUE = 14;
2913     };
2914 } // namespace impl
2915
2935 class IElementWiseLayer : public ILayer
2936 {
2937 public:
2947     void setOperation(ElementWiseOperation op) noexcept
2948     {
2949         return mImpl->setOperation(op);
2950     }
2951
2959     ElementWiseOperation getOperation() const noexcept
2960     {
2961         return mImpl->getOperation();
2962     }
2963
2964 protected:
2965     apiv::VElementWiseLayer* mImpl;
2966     virtual ~IElementWiseLayer() noexcept = default;
2967 };
2968
2974 enum class GatherMode : int32_t
2975 {
2976     kDEFAULT = 0,
2977     kELEMENT = 1,
2978     kND = 2
2979 };
2980

```

```

2986 template <>
2987 constexpr inline int32_t EnumMax<GatherMode>() noexcept
2988 {
2989     return 3;
2990 }
2991
3070 class IGatherLayer : public ILayer
3071 {
3072 public:
3082     void setGatherAxis(int32_t axis) noexcept
3083     {
3084         mImpl->setGatherAxis(axis);
3085     }
3086
3093     int32_t getGatherAxis() const noexcept
3094     {
3095         return mImpl->getGatherAxis();
3096     }
3097
3114     void setNbElementWiseDims(int32_t elementWiseDims) noexcept
3115     {
3116         mImpl->setNbElementWiseDims(elementWiseDims);
3117     }
3118
3124     int32_t getNbElementWiseDims() const noexcept
3125     {
3126         return mImpl->getNbElementWiseDims();
3127     }
3128
3134     void setMode(GatherMode mode) noexcept
3135     {
3136         mImpl->setMode(mode);
3137     }
3138
3144     GatherMode getMode() const noexcept
3145     {
3146         return mImpl->getMode();
3147     }
3148
3149 protected:
3150     apiv::VGatherLayer* mImpl;
3151     virtual ~IGatherLayer() noexcept = default;
3152 };
3153
3233 enum class RNNOperation : int32_t
3234 {
3235     kRELU = 0,
3236     kTANH = 1,
3237     kLSTM = 2,
3238     kGRU = 3
3239 };
3240
3246 template <>
3247 constexpr inline int32_t EnumMax<RNNOperation>() noexcept
3248 {
3249     return 4;
3250 }
3251
3259 enum class RNNDirection : int32_t
3260 {
3261     kUNIDIRECTION = 0,
3262     kBIDIRECTION = 1
3263 };
3264
3270 template <>
3271 constexpr inline int32_t EnumMax<RNNDirection>() noexcept
3272 {
3273     return 2;
3274 }
3275
3291 enum class RNNInputMode : int32_t
3292 {
3293     kLINEAR = 0,
3294     kSKIP = 1
3295 };
3296
3302 template <>
3303 constexpr inline int32_t EnumMax<RNNInputMode>() noexcept
3304 {
3305     return 2;
3306 }

```

```

3307
3315 enum class RNNGateType : int32_t
3316 {
3317     kINPUT = 0,
3318     kOUTPUT = 1,
3319     kFORGET = 2,
3320     kUPDATE = 3,
3321     kRESET = 4,
3322     kCELL = 5,
3323     kHIDDEN = 6
3324 };
3325
3331 template <>
3332 constexpr inline int32_t EnumMax<RNNGateType>() noexcept
3333 {
3334     return 7;
3335 }
3336
3349 class TRT_DEPRECATED IRNNv2Layer : public ILayer
3350 {
3351 public:
3352     int32_t getLayerCount() const noexcept
3353     {
3354         return mImpl->getLayerCount();
3355     }
3356     int32_t getHiddenSize() const noexcept
3357     {
3358         return mImpl->getHiddenSize();
3359     }
3360     int32_t getMaxSeqLength() const noexcept
3361     {
3362         return mImpl->getMaxSeqLength();
3363     }
3364     int32_t getDataLength() const noexcept
3365     {
3366         return mImpl->getDataLength();
3367     }
3368
3383 void setSequenceLengths(ITensor& seqLengths) noexcept
3384 {
3385     return mImpl->setSequenceLengths(seqLengths);
3386 }
3387
3395 ITensor* getSequenceLengths() const noexcept
3396 {
3397     return mImpl->getSequenceLengths();
3398 }
3399
3405 void setOperation(RNNOperation op) noexcept
3406 {
3407     mImpl->setOperation(op);
3408 }
3409
3415 RNNOperation getOperation() const noexcept
3416 {
3417     return mImpl->getOperation();
3418 }
3419
3425 void setInputMode(RNNInputMode op) noexcept
3426 {
3427     mImpl->setInputMode(op);
3428 }
3429
3435 RNNInputMode getInputMode() const noexcept
3436 {
3437     return mImpl->getInputMode();
3438 }
3439
3450 void setDirection(RNNDirection op) noexcept
3451 {
3452     mImpl->setDirection(op);
3453 }
3454
3460 RNNDirection getDirection() const noexcept
3461 {
3462     return mImpl->getDirection();
3463 }
3464
3518 void setWeightsForGate(int32_t layerIndex, RNNGateType gate, bool isW, Weights weights) noexcept
3519 {
3520     mImpl->setWeightsForGate(layerIndex, gate, isW, weights);

```



```

3521     }
3522
3523     Weights getWeightsForGate(int32_t layerIndex, RNNGateType gate, bool isW) const noexcept
3524     {
3525         return mImpl->getWeightsForGate(layerIndex, gate, isW);
3526     }
3527
3528     void setBiasForGate(int32_t layerIndex, RNNGateType gate, bool isW, Weights bias) noexcept
3529     {
3530         mImpl->setBiasForGate(layerIndex, gate, isW, bias);
3531     }
3532
3533     Weights getBiasForGate(int32_t layerIndex, RNNGateType gate, bool isW) const noexcept
3534     {
3535         return mImpl->getBiasForGate(layerIndex, gate, isW);
3536     }
3537
3538     void setHiddenState(ITensor& hidden) noexcept
3539     {
3540         mImpl->setHiddenState(hidden);
3541     }
3542
3543     ITensor* getHiddenState() const noexcept
3544     {
3545         return mImpl->getHiddenState();
3546     }
3547
3548     void setCellState(ITensor& cell) noexcept
3549     {
3550         mImpl->setCellState(cell);
3551     }
3552
3553     ITensor* getCellState() const noexcept
3554     {
3555         return mImpl->getCellState();
3556     }
3557
3558 protected:
3559     apiv::VRNNv2Layer* mImpl;
3560     virtual ~IRNNv2Layer() noexcept = default;
3561 };
3562
3563 class IPluginV2Layer : public ILayer
3564 {
3565 public:
3566     IPluginV2& getPlugin() noexcept
3567     {
3568         return mImpl->getPlugin();
3569     }
3570
3571 protected:
3572     apiv::VPluginV2Layer* mImpl;
3573     virtual ~IPluginV2Layer() noexcept = default;
3574 };
3575
3576 enum class UnaryOperation : int32_t
3577 {
3578     kEXP = 0,
3579     kLOG = 1,
3580     kSQRT = 2,
3581     kRECIP = 3,
3582     kABS = 4,
3583     kNEG = 5,
3584     kSIN = 6,
3585     kCOS = 7,
3586     kTAN = 8,
3587     kSINH = 9,
3588     kCOSH = 10,
3589     kASIN = 11,
3590     kACOS = 12,
3591     kATAN = 13,
3592     kASINH = 14,
3593     kACOSH = 15,
3594     kATANH = 16,
3595     kCEIL = 17,
3596     kFLOOR = 18,
3597     kERF = 19,
3598     kNOT = 20,
3599     kSIGN = 21,
3600     kROUND = 22,
3601     kISINF = 23,

```

```

3700 };
3701
3702 template <>
3703 constexpr inline int32_t EnumMax<UnaryOperation>() noexcept
3704 {
3705     return 24;
3706 }
3707
3708 class IUnaryLayer : public ILayer
3709 {
3710 public:
3711     void setOperation(UnaryOperation op) noexcept
3712     {
3713         mImpl->setOperation(op);
3714     }
3715
3716     UnaryOperation getOperation() const noexcept
3717     {
3718         return mImpl->getOperation();
3719     }
3720
3721 protected:
3722     apiv::VUnaryLayer* mImpl;
3723     virtual ~IUnaryLayer() noexcept = default;
3724 };
3725
3726 enum class ReduceOperation : int32_t
3727 {
3728     kSUM = 0,
3729     kPROD = 1,
3730     kMAX = 2,
3731     kMIN = 3,
3732     kAVG = 4
3733 };
3734
3735 template <>
3736 constexpr inline int32_t EnumMax<ReduceOperation>() noexcept
3737 {
3738     return 5;
3739 }
3740
3741 class IReduceLayer : public ILayer
3742 {
3743 public:
3744     void setOperation(ReduceOperation op) noexcept
3745     {
3746         mImpl->setOperation(op);
3747     }
3748
3749     ReduceOperation getOperation() const noexcept
3750     {
3751         return mImpl->getOperation();
3752     }
3753
3754     void setReduceAxes(uint32_t reduceAxes) noexcept
3755     {
3756         mImpl->setReduceAxes(reduceAxes);
3757     }
3758
3759     uint32_t getReduceAxes() const noexcept
3760     {
3761         return mImpl->getReduceAxes();
3762     }
3763
3764     void setKeepDimensions(bool keepDimensions) noexcept
3765     {
3766         mImpl->setKeepDimensions(keepDimensions);
3767     }
3768
3769     bool getKeepDimensions() const noexcept
3770     {
3771         return mImpl->getKeepDimensions();
3772     }
3773
3774 protected:
3775     apiv::VReduceLayer* mImpl;
3776     virtual ~IReduceLayer() noexcept = default;
3777 };
3778
3779 class IPaddingLayer : public ILayer
3780 {

```

```

3875 public:
3885     TRT_DEPRECATED void setPrePadding(DimsHW padding) noexcept
3886     {
3887         mImpl->setPrePadding(padding);
3888     }
3889
3897     TRT_DEPRECATED DimsHW getPrePadding() const noexcept
3898     {
3899         return mImpl->getPrePadding();
3900     }
3901
3911     TRT_DEPRECATED void setPostPadding(DimsHW padding) noexcept
3912     {
3913         mImpl->setPostPadding(padding);
3914     }
3915
3923     TRT_DEPRECATED DimsHW getPostPadding() const noexcept
3924     {
3925         return mImpl->getPostPadding();
3926     }
3927
3937     void setPrePaddingNd(Dims padding) noexcept
3938     {
3939         mImpl->setPrePaddingNd(padding);
3940     }
3941
3949     Dims getPrePaddingNd() const noexcept
3950     {
3951         return mImpl->getPrePaddingNd();
3952     }
3953
3963     void setPostPaddingNd(Dims padding) noexcept
3964     {
3965         mImpl->setPostPaddingNd(padding);
3966     }
3967
3975     Dims getPostPaddingNd() const noexcept
3976     {
3977         return mImpl->getPostPaddingNd();
3978     }
3979
3980 protected:
3981     apiv::VPaddingLayer* mImpl;
3982     virtual ~IPaddingLayer() noexcept = default;
3983 };
3984
3985 struct Permutation
3986 {
3993     int32_t order[Dims::MAX_DIMS];
3994 };
3995
4008 class IShuffleLayer : public ILayer
4009 {
4010 public:
4020     void setFirstTranspose(Permutation permutation) noexcept
4021     {
4022         mImpl->setFirstTranspose(permutation);
4023     }
4024
4032     Permutation getFirstTranspose() const noexcept
4033     {
4034         return mImpl->getFirstTranspose();
4035     }
4036
4057     void setReshapeDimensions(Dims dimensions) noexcept
4058     {
4059         mImpl->setReshapeDimensions(dimensions);
4060     }
4061
4070     Dims getReshapeDimensions() const noexcept
4071     {
4072         return mImpl->getReshapeDimensions();
4073     }
4074
4080     //
4103     using ILayer::setInput;
4104
4117     void setSecondTranspose(Permutation permutation) noexcept
4118     {
4119         mImpl->setSecondTranspose(permutation);
4120     }

```

```

4121
4129     Permutation getSecondTranspose() const noexcept
4130     {
4131         return mImpl->getSecondTranspose();
4132     }
4133
4145     void setZeroIsPlaceholder(bool zeroIsPlaceholder) noexcept
4146     {
4147         return mImpl->setZeroIsPlaceholder(zeroIsPlaceholder);
4148     }
4149
4158     bool getZeroIsPlaceholder() const noexcept
4159     {
4160         return mImpl->getZeroIsPlaceholder();
4161     }
4162
4163 protected:
4164     apiv::VShuffleLayer* mImpl;
4165     virtual ~IShuffleLayer() noexcept = default;
4166 };
4167
4173 enum class SampleMode : int32_t
4174 {
4175     kSTRICT_BOUNDS = 0,
4176     kDEFAULT_TRT_DEPRECATED_ENUM = kSTRICT_BOUNDS,
4177     kWRAP = 1,
4178     kCLAMP = 2,
4179     kFILL = 3,
4180     kREFLECT = 4,
4183 };
4184
4186 using SliceMode = SampleMode;
4187
4193 template <>
4194 constexpr inline int32_t EnumMax<SampleMode>() noexcept
4195 {
4196     return 5;
4197 }
4198
4241 class ISliceLayer : public ILayer
4242 {
4243 public:
4253     void setStart(Dims start) noexcept
4254     {
4255         mImpl->setStart(start);
4256     }
4257
4268     Dims getStart() const noexcept
4269     {
4270         return mImpl->getStart();
4271     }
4272
4282     void setSize(Dims size) noexcept
4283     {
4284         return mImpl->setSize(size);
4285     }
4286
4297     Dims getSize() const noexcept
4298     {
4299         return mImpl->getSize();
4300     }
4301
4311     void setStride(Dims stride) noexcept
4312     {
4313         mImpl->setStride(stride);
4314     }
4315
4326     Dims getStride() const noexcept
4327     {
4328         return mImpl->getStride();
4329     }
4330
4336     void setMode(SliceMode mode) noexcept
4337     {
4338         mImpl->setMode(mode);
4339     }
4340
4346     SliceMode getMode() const noexcept
4347     {
4348         return mImpl->getMode();
4349     }

```

```

4350
4374     using ILayer::setInput;
4375
4376 protected:
4377     apiv::VSliceLayer* mImpl;
4378     virtual ~ISliceLayer() noexcept = default;
4379 };
4380
4393 class IShapeLayer : public ILayer
4394 {
4395 protected:
4396     apiv::VShapeLayer* mImpl;
4397     virtual ~IShapeLayer() noexcept = default;
4398 };
4399
4405 enum class TopKOperation : int32_t
4406 {
4407     kMAX = 0,
4408     kMIN = 1,
4409 };
4410
4416 template <>
4417 constexpr inline int32_t EnumMax<TopKOperation>() noexcept
4418 {
4419     return 2;
4420 }
4421
4433 class ITopKLayer : public ILayer
4434 {
4435 public:
4441     void setOperation(TopKOperation op) noexcept
4442     {
4443         mImpl->setOperation(op);
4444     }
4445
4451     TopKOperation getOperation() const noexcept
4452     {
4453         return mImpl->getOperation();
4454     }
4455
4465     void setK(int32_t k) noexcept
4466     {
4467         mImpl->setK(k);
4468     }
4469
4479     int32_t getK() const noexcept
4480     {
4481         return mImpl->getK();
4482     }
4483
4489     void setReduceAxes(uint32_t reduceAxes) noexcept
4490     {
4491         mImpl->setReduceAxes(reduceAxes);
4492     }
4493
4499     uint32_t getReduceAxes() const noexcept
4500     {
4501         return mImpl->getReduceAxes();
4502     }
4503
4518     using ILayer::setInput;
4519
4520 protected:
4521     apiv::VTopKLayer* mImpl;
4522     virtual ~ITopKLayer() noexcept = default;
4523 };
4524
4531 enum class MatrixOperation : int32_t
4532 {
4533     kNONE,
4534
4539     kTRANPOSE,
4540
4551     kVECTOR
4552 };
4553
4559 template <>
4560 constexpr inline int32_t EnumMax<MatrixOperation>() noexcept
4561 {
4562     return 3;
4563 }

```

```

4564
4590 class IMatrixMultiplyLayer : public ILayer
4591 {
4592 public:
4599     void setOperation(int32_t index, MatrixOperation op) noexcept
4600     {
4601         mImpl->setOperation(index, op);
4602     }
4603
4611     MatrixOperation getOperation(int32_t index) const noexcept
4612     {
4613         return mImpl->getOperation(index);
4614     }
4615
4616 protected:
4617     apiv::VMatrixMultiplyLayer* mImpl;
4618     virtual ~IMatrixMultiplyLayer() noexcept = default;
4619 };
4620
4642 class INonZeroLayer : public ILayer
4643 {
4644 protected:
4645     virtual ~INonZeroLayer() noexcept = default;
4646     apiv::VNonZeroLayer* mImpl;
4647 };
4648
4663 class IRaggedSoftMaxLayer : public ILayer
4664 {
4665 protected:
4666     apiv::VRaggedSoftMaxLayer* mImpl;
4667     virtual ~IRaggedSoftMaxLayer() noexcept = default;
4668 };
4669
4697 class IIdentityLayer : public ILayer
4698 {
4699 protected:
4700     apiv::VIdentityLayer* mImpl;
4701     virtual ~IIdentityLayer() noexcept = default;
4702 };
4703
4710 class ICastLayer : public ILayer
4711 {
4712 public:
4716     void setToType(DataType toType) noexcept
4717     {
4718         mImpl->setToType(toType);
4719     }
4720
4724     DataType getToType() const noexcept
4725     {
4726         return mImpl->getToType();
4727     }
4728
4729 protected:
4730     apiv::VCastLayer* mImpl;
4731     virtual ~ICastLayer() noexcept = default;
4732 };
4733
4742 class IConstantLayer : public ILayer
4743 {
4744 public:
4754     void setWeights(Weights weights) noexcept
4755     {
4756         mImpl->setWeights(weights);
4757     }
4758
4764     Weights getWeights() const noexcept
4765     {
4766         return mImpl->getWeights();
4767     }
4768
4776     void setDimensions(Dims dimensions) noexcept
4777     {
4778         mImpl->setDimensions(dimensions);
4779     }
4780
4788     Dims getDimensions() const noexcept
4789     {
4790         return mImpl->getDimensions();
4791     }
4792

```

```

4793 protected:
4794     apiv::VConstantLayer* mImpl;
4795     virtual ~IConstantLayer() noexcept = default;
4796 };
4797
4807 class IParametricReLULayer : public ILayer
4808 {
4809 protected:
4810     apiv::VParametricReLULayer* mImpl;
4811     virtual ~IParametricReLULayer() noexcept = default;
4812 };
4813
4819 enum class InterpolationMode : int32_t
4820 {
4821     kNEAREST = 0,
4822     kLINEAR = 1,
4823     kCUBIC = 2
4824 };
4825
4827 using ResizeMode = InterpolationMode;
4828
4829 namespace impl
4830 {
4836 template <>
4837 struct EnumMaxImpl<InterpolationMode>
4838 {
4839     static constexpr int32_t kVALUE = 3;
4840 };
4841 } // namespace impl
4842
4850 enum class ResizeCoordinateTransformation : int32_t
4851 {
4864     kALIGN_CORNERS = 0,
4865     kASYMMETRIC = 1,
4873     kHALF_PIXEL = 2,
4881 };
4882
4883 namespace impl
4884 {
4890 template <>
4891 struct EnumMaxImpl<ResizeCoordinateTransformation>
4892 {
4893     static constexpr int32_t kVALUE = 3;
4894 };
4895 } // namespace impl
4896
4904 enum class ResizeSelector : int32_t
4905 {
4907     kFORMULA = 0,
4910     kUPPER = 1,
4911 };
4912
4913 namespace impl
4914 {
4920 template <>
4921 struct EnumMaxImpl<ResizeSelector>
4922 {
4923     static constexpr int32_t kVALUE = 2;
4924 };
4925 } // namespace impl
4926
4934 enum class ResizeRoundMode : int32_t
4935 {
4937     kHALF_UP = 0,
4938     kHALF_DOWN = 1,
4941     kFLOOR = 2,
4944     kCEIL = 3,
4947 };
4948
4949 namespace impl
4950 {
4956 template <>
4957 struct EnumMaxImpl<ResizeRoundMode>
4958 {
4959     static constexpr int32_t kVALUE = 4;

```

```

4960 };
4961 } // namespace impl
4962
4999 class IResizeLayer : public ILayer
5000 {
5001 public:
5020     void setOutputDimensions(Dims dimensions) noexcept
5021     {
5022         return mImpl->setOutputDimensions(dimensions);
5023     }
5024
5030     Dims getOutputDimensions() const noexcept
5031     {
5032         return mImpl->getOutputDimensions();
5033     }
5034
5060     void setScales(float const* scales, int32_t nbScales) noexcept
5061     {
5062         mImpl->setScales(scales, nbScales);
5063     }
5064
5079     int32_t getScales(int32_t size, float* scales) const noexcept
5080     {
5081         return mImpl->getScales(size, scales);
5082     }
5083
5091     void setResizeMode(ResizeMode resizeMode) noexcept
5092     {
5093         mImpl->setResizeMode(resizeMode);
5094     }
5095
5101     ResizeMode getResizeMode() const noexcept
5102     {
5103         return mImpl->getResizeMode();
5104     }
5105
5117     TRT_DEPRECATED void setAlignCorners(bool alignCorners) noexcept
5118     {
5119         mImpl->setAlignCorners(alignCorners);
5120     }
5121
5129     TRT_DEPRECATED bool getAlignCorners() const noexcept
5130     {
5131         return mImpl->getAlignCorners();
5132     }
5133
5153     using ILayer::setInput;
5154
5164     void setCoordinateTransformation(ResizeCoordinateTransformation coordTransform) noexcept
5165     {
5166         mImpl->setCoordinateTransformation(coordTransform);
5167     }
5168
5174     ResizeCoordinateTransformation getCoordinateTransformation() const noexcept
5175     {
5176         return mImpl->getCoordinateTransformation();
5177     }
5178
5189     void setSelectorForSinglePixel(ResizeSelector selector) noexcept
5190     {
5191         mImpl->setSelectorForSinglePixel(selector);
5192     }
5193
5199     ResizeSelector getSelectorForSinglePixel() const noexcept
5200     {
5201         return mImpl->getSelectorForSinglePixel();
5202     }
5203
5213     void setNearestRounding(ResizeRoundMode value) noexcept
5214     {
5215         mImpl->setNearestRounding(value);
5216     }
5217
5223     ResizeRoundMode getNearestRounding() const noexcept
5224     {
5225         return mImpl->getNearestRounding();
5226     }
5227
5245     void setCubicCoeff(float A) noexcept
5246     {
5247         mImpl->setCubicCoeff(A);

```



```

5248     }
5249
5250     float getCubicCoeff() const noexcept
5251     {
5252         return mImpl->getCubicCoeff();
5253     }
5254
5255     void setExcludeOutside(bool excludeFlag) noexcept
5256     {
5257         mImpl->setExcludeOutside(excludeFlag);
5258     }
5259
5260     bool getExcludeOutside() const noexcept
5261     {
5262         return mImpl->getExcludeOutside();
5263     }
5264
5265 protected:
5266     virtual ~IResizeLayer() noexcept = default;
5267     apiv::VResizeLayer* mImpl;
5268 };
5269
5270 enum class LoopOutput : int32_t
5271 {
5272     kLAST_VALUE = 0,
5273     kCONCATENATE = 1,
5274     kREVERSE = 2
5275 };
5276
5277 template <>
5278 constexpr inline int32_t EnumMax<LoopOutput>() noexcept
5279 {
5280     return 3;
5281 }
5282
5283 enum class TripLimit : int32_t
5284 {
5285     kCOUNT = 0,
5286     kWHILE = 1
5287 };
5288
5289 template <>
5290 constexpr inline int32_t EnumMax<TripLimit>() noexcept
5291 {
5292     return 2;
5293 }
5294
5295 class ILoop;
5296
5297 class ILoopBoundaryLayer : public ILayer
5298 {
5299 public:
5300     ILoop* getLoop() const noexcept
5301     {
5302         return mBoundary->getLoop();
5303     }
5304
5305 protected:
5306     virtual ~ILoopBoundaryLayer() noexcept = default;
5307     apiv::VLoopBoundaryLayer* mBoundary;
5308 };
5309
5310 class IIfConditionalBoundaryLayer : public ILayer
5311 {
5312 public:
5313     IIfConditional* getConditional() const noexcept
5314     {
5315         return mBoundary->getConditional();
5316     }
5317
5318 protected:
5319     virtual ~IIfConditionalBoundaryLayer() noexcept = default;
5320     apiv::VConditionalBoundaryLayer* mBoundary;
5321 };
5322
5323 class IConditionLayer : public IIfConditionalBoundaryLayer
5324 {
5325 public:

```

```

5372 protected:
5373     virtual ~IConditionLayer() noexcept = default;
5374     apiv::VConditionLayer* mImpl;
5375 };
5376
5382 class IIfConditionalOutputLayer : public IIfConditionalBoundaryLayer
5383 {
5384 public:
5385 protected:
5386     virtual ~IIfConditionalOutputLayer() noexcept = default;
5387     apiv::VConditionalOutputLayer* mImpl;
5388 };
5389
5393 class IIfConditionalInputLayer : public IIfConditionalBoundaryLayer
5394 {
5395 public:
5396 protected:
5397     virtual ~IIfConditionalInputLayer() noexcept = default;
5398     apiv::VConditionalInputLayer* mImpl;
5399 };
5400
5422 class IIfConditional : public INoCopy
5423 {
5424 public:
5425     IConditionLayer* setCondition(ITensor& condition) noexcept
5426     {
5427         return mImpl->setCondition(condition);
5428     }
5429
5430     IIfConditionalOutputLayer* addOutput(ITensor& trueSubgraphOutput, ITensor& falseSubgraphOutput)
5431     noexcept
5432     {
5433         return mImpl->addOutput(trueSubgraphOutput, falseSubgraphOutput);
5434     }
5435
5436     IIfConditionalInputLayer* addInput(ITensor& input) noexcept
5437     {
5438         return mImpl->addInput(input);
5439     }
5440
5441     void setName(char const* name) noexcept
5442     {
5443         mImpl->setName(name);
5444     }
5445
5446     char const* getName() const noexcept
5447     {
5448         return mImpl->getName();
5449     }
5450
5451 protected:
5452     virtual ~IIfConditional() noexcept = default;
5453     apiv::VIfConditional* mImpl;
5454 };
5455
5498 class IRecurrenceLayer : public ILoopBoundaryLayer
5499 {
5500 public:
5501     //
5502     using ILayer::setInput;
5503
5504 protected:
5505     virtual ~IRecurrenceLayer() noexcept = default;
5506     apiv::VRecurrenceLayer* mImpl;
5507 };
5508
5543 class ILoopOutputLayer : public ILoopBoundaryLayer
5544 {
5545 public:
5546     LoopOutput getLoopOutput() const noexcept
5547     {
5548         return mImpl->getLoopOutput();
5549     }
5550
5551     void setAxis(int32_t axis) noexcept
5552     {
5553         mImpl->setAxis(axis);
5554     }
5555
5556     int32_t getAxis() const noexcept

```

```

5570     {
5571         return mImpl->getAxis();
5572     }
5573
5574     //
5575     using ILayer::setInput;
5576
5577 protected:
5578     virtual ~ILoopOutputLayer() noexcept = default;
5579     apiv::VLoopOutputLayer* mImpl;
5580 };
5581
5582 class ITripLimitLayer : public ILoopBoundaryLayer
5583 {
5584 public:
5585     TripLimit getTripLimit() const noexcept
5586     {
5587         return mImpl->getTripLimit();
5588     }
5589
5590 protected:
5591     virtual ~ITripLimitLayer() noexcept = default;
5592     apiv::VTripLimitLayer* mImpl;
5593 };
5594
5595 class IIteratorLayer : public ILoopBoundaryLayer
5596 {
5597 public:
5598     void setAxis(int32_t axis) noexcept
5599     {
5600         mImpl->setAxis(axis);
5601     }
5602
5603     int32_t getAxis() const noexcept
5604     {
5605         return mImpl->getAxis();
5606     }
5607
5608     void setReverse(bool reverse) noexcept
5609     {
5610         mImpl->setReverse(reverse);
5611     }
5612
5613     bool getReverse() const noexcept
5614     {
5615         return mImpl->getReverse();
5616     }
5617
5618 protected:
5619     virtual ~IIteratorLayer() noexcept = default;
5620     apiv::VIteratorLayer* mImpl;
5621 };
5622
5623 class ILoop : public INoCopy
5624 {
5625 public:
5626     IRecurrenceLayer* addRecurrence(ITensor& initialValue) noexcept
5627     {
5628         return mImpl->addRecurrence(initialValue);
5629     }
5630
5631     ITripLimitLayer* addTripLimit(ITensor& tensor, TripLimit limit) noexcept
5632     {
5633         return mImpl->addTripLimit(tensor, limit);
5634     }
5635
5636     IIteratorLayer* addIterator(ITensor& tensor, int32_t axis = 0, bool reverse = false) noexcept
5637     {
5638         return mImpl->addIterator(tensor, axis, reverse);
5639     }
5640
5641     ILoopOutputLayer* addLoopOutput(ITensor& tensor, LoopOutput outputKind, int32_t axis = 0) noexcept
5642     {
5643         return mImpl->addLoopOutput(tensor, outputKind, axis);
5644     }
5645
5646     void setName(char const* name) noexcept
5647     {
5648         mImpl->setName(name);
5649     }
5650 };

```

```

5735     char const* getName() const noexcept
5736     {
5737         return mImpl->getName();
5738     }
5739
5740 protected:
5741     virtual ~ILoop() noexcept = default;
5742     apiv::VLoop* mImpl;
5743 };
5744
5745 class ISelectLayer : public ILayer
5746 {
5747 protected:
5748     virtual ~ISelectLayer() noexcept = default;
5749     apiv::VSelectLayer* mImpl;
5750 };
5751
5752 class IAssertionLayer : public ILayer
5753 {
5754 public:
5755     void setMessage(char const* message) noexcept
5756     {
5757         mImpl->setMessage(message);
5758     }
5759
5760     char const* getMessage() const noexcept
5761     {
5762         return mImpl->getMessage();
5763     }
5764
5765 protected:
5766     virtual ~IAssertionLayer() noexcept = default;
5767     apiv::VAssertionLayer* mImpl;
5768 };
5769
5770 enum class FillOperation : int32_t
5771 {
5772     kLINSPEACE = 0,
5773     kRANDOM_UNIFORM = 1,
5774     kRANDOM_NORMAL = 2
5775 };
5776
5777 template <>
5778 constexpr inline int32_t EnumMax<FillOperation>() noexcept
5779 {
5780     return 3;
5781 }
5782
5783 class IFillLayer : public ILayer
5784 {
5785 public:
5786     //
5787     void setDimensions(Dims dimensions) noexcept
5788     {
5789         mImpl->setDimensions(dimensions);
5790     }
5791
5792     Dims getDimensions() const noexcept
5793     {
5794         return mImpl->getDimensions();
5795     }
5796
5797     void setOperation(FillOperation op) noexcept
5798     {
5799         mImpl->setOperation(op);
5800     }
5801
5802     FillOperation getOperation() const noexcept
5803     {
5804         return mImpl->getOperation();
5805     }
5806
5807     //
5808     void setAlpha(double alpha) noexcept
5809     {
5810         mImpl->setAlpha(alpha);
5811     }
5812
5813     double getAlpha() const noexcept
5814     {

```

```

5934         return mImpl->getAlpha();
5935     }
5936
5951     void setBeta(double beta) noexcept
5952     {
5953         mImpl->setBeta(beta);
5954     }
5955
5966     double getBeta() const noexcept
5967     {
5968         return mImpl->getBeta();
5969     }
5970
6003     using ILayer::setInput;
6004
6005 protected:
6006     virtual ~IFillLayer() noexcept = default;
6007     apiv::VFillLayer* mImpl;
6008 };
6009
6067 class IQuantizeLayer : public ILayer
6068 {
6069 public:
6078     int32_t getAxis() const noexcept
6079     {
6080         return mImpl->getAxis();
6081     }
6089     void setAxis(int32_t axis) noexcept
6090     {
6091         mImpl->setAxis(axis);
6092     }
6093
6094 protected:
6095     virtual ~IQuantizeLayer() noexcept = default;
6096     apiv::VQuantizeLayer* mImpl;
6097 };
6098
6153 class IDequantizeLayer : public ILayer
6154 {
6155 public:
6164     int32_t getAxis() const noexcept
6165     {
6166         return mImpl->getAxis();
6167     }
6175     void setAxis(int32_t axis) noexcept
6176     {
6177         mImpl->setAxis(axis);
6178     }
6179
6180 protected:
6181     virtual ~IDequantizeLayer() noexcept = default;
6182     apiv::VDequantizeLayer* mImpl;
6183 };
6184
6220 class IEinsumLayer : public ILayer
6221 {
6222 public:
6232     bool setEquation(char const* equation) noexcept
6233     {
6234         return mImpl->setEquation(equation);
6235     }
6236
6242     char const* getEquation() const noexcept
6243     {
6244         return mImpl->getEquation();
6245     }
6246
6247 protected:
6248     virtual ~IEinsumLayer() noexcept = default;
6249     apiv::VEinsumLayer* mImpl;
6250 };
6251
6257 enum class ScatterMode : int32_t
6258 {
6259     kELEMENT = 0,
6260     kND = 1,
6261 };
6262
6268 template <>
6269 constexpr inline int32_t EnumMax<ScatterMode>() noexcept
6270 {

```

```

6271     return 2;
6272 }
6273
6330 class IScatterLayer : public ILayer
6331 {
6332 public:
6333     void setMode(ScatterMode mode) noexcept
6334     {
6340         mImpl->setMode(mode);
6341     }
6342
6348     ScatterMode getMode() const noexcept
6349     {
6350         return mImpl->getMode();
6351     }
6352
6358     void setAxis(int32_t axis) noexcept
6359     {
6360         mImpl->setAxis(axis);
6361     }
6362
6366     int32_t getAxis() const noexcept
6367     {
6368         return mImpl->getAxis();
6369     }
6370
6371 protected:
6372     apiv::VScatterLayer* mImpl;
6373     virtual ~IScatterLayer() noexcept = default;
6374 }; // class IScatterLayer
6375
6403 class IOneHotLayer : public ILayer
6404 {
6405 public:
6411     void setAxis(int32_t axis) noexcept
6412     {
6413         mImpl->setAxis(axis);
6414     }
6415
6419     int32_t getAxis() const noexcept
6420     {
6421         return mImpl->getAxis();
6422     }
6423
6424 protected:
6425     apiv::VOneHotLayer* mImpl;
6426 };
6427
6438 class IGridSampleLayer : public ILayer
6439 {
6440 public:
6446     void setInterpolationMode(InterpolationMode mode) noexcept
6447     {
6448         mImpl->setInterpolationMode(mode);
6449     }
6450
6458     InterpolationMode getInterpolationMode() const noexcept
6459     {
6460         return mImpl->getInterpolationMode();
6461     }
6462
6468     void setAlignCorners(bool alignCorners) noexcept
6469     {
6470         mImpl->setAlignCorners(alignCorners);
6471     }
6472
6480     bool getAlignCorners() const noexcept
6481     {
6482         return mImpl->getAlignCorners();
6483     }
6484
6492     bool setSampleMode(SampleMode mode) noexcept
6493     {
6494         return mImpl->setSampleMode(mode);
6495     }
6496
6504     SampleMode getSampleMode() const noexcept
6505     {
6506         return mImpl->getSampleMode();
6507     }
6508

```

```

6509 protected:
6510     apiv::VGridSampleLayer* mImpl;
6511     virtual ~IGridSampleLayer() noexcept = default;
6512 }; // class IGridSampleLayer
6513
6519 enum class BoundingBoxFormat : int32_t
6520 {
6522     kCORNER_PAIRS = 0,
6524     kCENTER_SIZES = 1
6525 };
6526
6532 template <>
6533 constexpr inline int32_t EnumMax<BoundingBoxFormat>() noexcept
6534 {
6535     return 2;
6536 }
6537
6581 class INMSLayer : public ILayer
6582 {
6583 public:
6593     void setBoundingBoxFormat(BoundingBoxFormat fmt) noexcept
6594     {
6595         mImpl->setBoundingBoxFormat(fmt);
6596     }
6597
6605     BoundingBoxFormat getBoundingBoxFormat() const noexcept
6606     {
6607         return mImpl->getBoundingBoxFormat();
6608     }
6609
6619     void setTopKBoxLimit(int32_t limit) noexcept
6620     {
6621         mImpl->setTopKBoxLimit(limit);
6622     }
6623
6629     int32_t getTopKBoxLimit() const noexcept
6630     {
6631         return mImpl->getTopKBoxLimit();
6632     }
6633
6652     using ILayer::setInput;
6653
6654 protected:
6655     apiv::VNMSLayer* mImpl;
6656     virtual ~INMSLayer() noexcept = default;
6657 }; // class INMSLayer
6658
6670 class IReverseSequenceLayer: public ILayer
6671 {
6672 public:
6681     void setBatchAxis(int32_t batchAxis) noexcept
6682     {
6683         mImpl->setBatchAxis(batchAxis);
6684     }
6685
6691     int32_t getBatchAxis() const noexcept
6692     {
6693         return mImpl->getBatchAxis();
6694     }
6695
6704     void setSequenceAxis(int32_t sequenceAxis) noexcept
6705     {
6706         mImpl->setSequenceAxis(sequenceAxis);
6707     }
6708
6714     int32_t getSequenceAxis() const noexcept
6715     {
6716         return mImpl->getSequenceAxis();
6717     }
6718
6719 protected:
6720     apiv::VReverseSequenceLayer* mImpl;
6721     virtual ~IReverseSequenceLayer() noexcept = default;
6722 }; // class IReverseSequenceLayer
6723
6740
6741 class INormalizationLayer : public ILayer
6742 {
6743 public:
6750     void setEpsilon(float eps) noexcept
6751     {

```

```

6752         return mImpl->setEpsilon(eps);
6753     }
6754
6759     float getEpsilon() const noexcept
6760     {
6761         return mImpl->getEpsilon();
6762     }
6763
6768     void setAxes(uint32_t axesMask) noexcept
6769     {
6770         return mImpl->setAxes(axesMask);
6771     }
6772
6777     uint32_t getAxes() const noexcept
6778     {
6779         return mImpl->getAxes();
6780     }
6781
6797     void setNbGroups(int32_t nbGroups) noexcept
6798     {
6799         return mImpl->setNbGroups(nbGroups);
6800     }
6801
6806     int32_t getNbGroups() const noexcept
6807     {
6808         return mImpl->getNbGroups();
6809     }
6810
6824     void setComputePrecision(DataType type) noexcept
6825     {
6826         return mImpl->setComputePrecision(type);
6827     }
6828
6833     DataType getComputePrecision() const noexcept
6834     {
6835         return mImpl->getComputePrecision();
6836     }
6837
6838 protected:
6839     apiv::VNormalizationLayer* mImpl;
6840     virtual ~INormalizationLayer() noexcept = default;
6841 };
6842
6862 class INetworkDefinition : public INoCopy
6863 {
6864 public:
6865     virtual ~INetworkDefinition() noexcept = default;
6866
6904     ITensor* addInput(char const* name, DataType type, Dims dimensions) noexcept
6905     {
6906         return mImpl->addInput(name, type, dimensions);
6907     }
6908
6918     void markOutput(ITensor& tensor) noexcept
6919     {
6920         mImpl->markOutput(tensor);
6921     }
6922
6941     TRT_DEPRECATED IConvolutionLayer* addConvolution(
6942         ITensor& input, int32_t nbOutputMaps, DimsHW kernelSize, Weights kernelWeights, Weights
        biasWeights) noexcept
6943     {
6944         return mImpl->addConvolution(input, nbOutputMaps, kernelSize, kernelWeights, biasWeights);
6945     }
6946
6964     TRT_DEPRECATED IFullyConnectedLayer* addFullyConnected(
6965         ITensor& input, int32_t nbOutputs, Weights kernelWeights, Weights biasWeights) noexcept
6966     {
6967         return mImpl->addFullyConnected(input, nbOutputs, kernelWeights, biasWeights);
6968     }
6969
6984     IActivationLayer* addActivation(ITensor& input, ActivationType type) noexcept
6985     {
6986         return mImpl->addActivation(input, type);
6987     }
6988
7003     TRT_DEPRECATED IPoolingLayer* addPooling(ITensor& input, PoolingType type, DimsHW windowSize) noexcept
7004     {
7005         return mImpl->addPooling(input, type, windowSize);
7006     }
7007

```



```

7022     ILRNLayer* addLRN(ITensor& input, int32_t window, float alpha, float beta, float k) noexcept
7023     {
7024         return mImpl->addLRN(input, window, alpha, beta, k);
7025     }
7026
7049     IScaleLayer* addScale(ITensor& input, ScaleMode mode, Weights shift, Weights scale, Weights power)
noexcept
7050     {
7051         return mImpl->addScale(input, mode, shift, scale, power);
7052     }
7053
7062     ISoftMaxLayer* addSoftMax(ITensor& input) noexcept
7063     {
7064         return mImpl->addSoftMax(input);
7065     }
7066
7079     IConcatenationLayer* addConcatenation(ITensor* const* inputs, int32_t nbInputs) noexcept
7080     {
7081         return mImpl->addConcatenation(inputs, nbInputs);
7082     }
7083
7102     TRT_DEPRECATED IDeconvolutionLayer* addDeconvolution(
7103     ITensor& input, int32_t nbOutputMaps, DimsHW kernelSize, Weights kernelWeights, Weights
biasWeights) noexcept
7104     {
7105         return mImpl->addDeconvolution(input, nbOutputMaps, kernelSize, kernelWeights, biasWeights);
7106     }
7107
7130     IElementWiseLayer* addElementWise(ITensor& input1, ITensor& input2, ElementWiseOperation op) noexcept
7131     {
7132         return mImpl->addElementWise(input1, input2, op);
7133     }
7134
7152     IUnaryLayer* addUnary(ITensor& input, UnaryOperation operation) noexcept
7153     {
7154         return mImpl->addUnary(input, operation);
7155     }
7156
7169     TRT_DEPRECATED IPaddingLayer* addPadding(ITensor& input, DimsHW prePadding, DimsHW postPadding)
noexcept
7170     {
7171         return mImpl->addPadding(input, prePadding, postPadding);
7172     }
7173
7183     IShuffleLayer* addShuffle(ITensor& input) noexcept
7184     {
7185         return mImpl->addShuffle(input);
7186     }
7187
7200     IOneHotLayer* addOneHot(ITensor& indices, ITensor& values, ITensor& depth, int32_t axis) noexcept
7201     {
7202         return mImpl->addOneHot(indices, values, depth, axis);
7203     }
7204
7212     int32_t getNbLayers() const noexcept
7213     {
7214         return mImpl->getNbLayers();
7215     }
7216
7226     ILayer* getLayer(int32_t index) const noexcept
7227     {
7228         return mImpl->getLayer(index);
7229     }
7230
7238     int32_t getNbInputs() const noexcept
7239     {
7240         return mImpl->getNbInputs();
7241     }
7242
7254     ITensor* getInput(int32_t index) const noexcept
7255     {
7256         return mImpl->getInput(index);
7257     }
7258
7268     int32_t getNbOutputs() const noexcept
7269     {
7270         return mImpl->getNbOutputs();
7271     }
7272
7284     ITensor* getOutput(int32_t index) const noexcept
7285     {

```

```

7286         return mImpl->getOutput(index);
7287     }
7288
7296     TRT_DEPRECATED void destroy() noexcept
7297     {
7298         delete this;
7299     }
7300
7323     IReduceLayer* addReduce(
7324         ITensor& input, ReduceOperation operation, uint32_t reduceAxes, bool keepDimensions) noexcept
7325     {
7326         return mImpl->addReduce(input, operation, reduceAxes, keepDimensions);
7327     }
7328
7358     ITopKLayer* addTopK(ITensor& input, TopKOperation op, int32_t k, uint32_t reduceAxes) noexcept
7359     {
7360         return mImpl->addTopK(input, op, k, reduceAxes);
7361     }
7362
7374     IGatherLayer* addGather(ITensor& data, ITensor& indices, int32_t axis) noexcept
7375     {
7376         return mImpl->addGather(data, indices, axis);
7377     }
7378
7390     IGatherLayer* addGatherV2(ITensor& data, ITensor& indices, GatherMode mode) noexcept
7391     {
7392         return mImpl->addGatherV2(data, indices, mode);
7393     }
7394
7408     IRaggedSoftMaxLayer* addRaggedSoftMax(ITensor& input, ITensor& bounds) noexcept
7409     {
7410         return mImpl->addRaggedSoftMax(input, bounds);
7411     }
7412
7429     IMatrixMultiplyLayer* addMatrixMultiply(
7430         ITensor& input0, MatrixOperation op0, ITensor& input1, MatrixOperation op1) noexcept
7431     {
7432         return mImpl->addMatrixMultiply(input0, op0, input1, op1);
7433     }
7434
7444     INonZeroLayer* addNonZero(ITensor& input) noexcept
7445     {
7446         return mImpl->addNonZero(input);
7447     }
7448
7471     IConstantLayer* addConstant(Dims dimensions, Weights weights) noexcept
7472     {
7473         return mImpl->addConstant(dimensions, weights);
7474     }
7475
7541     TRT_DEPRECATED IRNNv2Layer* addRNNv2(
7542         ITensor& input, int32_t layerCount, int32_t hiddenSize, int32_t maxSeqLen, RNNOperation op) noexcept
7543     {
7544         return mImpl->addRNNv2(input, layerCount, hiddenSize, maxSeqLen, op);
7545     }
7546
7556     IIdentityLayer* addIdentity(ITensor& input) noexcept
7557     {
7558         return mImpl->addIdentity(input);
7559     }
7560
7571     ICastLayer* addCast(ITensor& input, DataType toType) noexcept
7572     {
7573         return mImpl->addCast(input, toType);
7574     }
7575
7586     void removeTensor(ITensor& tensor) noexcept
7587     {
7588         mImpl->removeTensor(tensor);
7589     }
7590
7598     void unmarkOutput(ITensor& tensor) noexcept
7599     {
7600         mImpl->unmarkOutput(tensor);
7601     }
7602
7617     IPluginV2Layer* addPluginV2(ITensor* const* inputs, int32_t nbInputs, IPluginV2& plugin) noexcept
7618     {
7619         return mImpl->addPluginV2(inputs, nbInputs, plugin);
7620     }
7621

```

```

7636     ISliceLayer* addSlice(ITensor& input, Dims start, Dims size, Dims stride) noexcept
7637     {
7638         return mImpl->addSlice(input, start, size, stride);
7639     }
7640
7660     void setName(char const* name) noexcept
7661     {
7662         mImpl->setName(name);
7663     }
7664
7674     char const* getName() const noexcept
7675     {
7676         return mImpl->getName();
7677     }
7678
7690     IShapeLayer* addShape(ITensor& input) noexcept
7691     {
7692         return mImpl->addShape(input);
7693     }
7694
7708     bool hasImplicitBatchDimension() const noexcept
7709     {
7710         return mImpl->hasImplicitBatchDimension();
7711     }
7712
7726     bool markOutputForShapes(ITensor& tensor) noexcept
7727     {
7728         return mImpl->markOutputForShapes(tensor);
7729     }
7730
7738     bool unmarkOutputForShapes(ITensor& tensor) noexcept
7739     {
7740         return mImpl->unmarkOutputForShapes(tensor);
7741     }
7742
7756     IParametricReLULayer* addParametricReLU(ITensor& input, ITensor& slope) noexcept
7757     {
7758         return mImpl->addParametricReLU(input, slope);
7759     }
7760
7778     IConvolutionLayer* addConvolutionNd(
7779         ITensor& input, int32_t nbOutputMaps, Dims kernelSize, Weights kernelWeights, Weights biasWeights)
7780     noexcept
7781     {
7782         return mImpl->addConvolutionNd(input, nbOutputMaps, kernelSize, kernelWeights, biasWeights);
7783     }
7798     IPoolingLayer* addPoolingNd(ITensor& input, PoolingType type, Dims windowSize) noexcept
7799     {
7800         return mImpl->addPoolingNd(input, type, windowSize);
7801     }
7802
7817     //
7820     IDeconvolutionLayer* addDeconvolutionNd(
7821         ITensor& input, int32_t nbOutputMaps, Dims kernelSize, Weights kernelWeights, Weights biasWeights)
7822     noexcept
7823     {
7824         return mImpl->addDeconvolutionNd(input, nbOutputMaps, kernelSize, kernelWeights, biasWeights);
7825     }
7857     IScaleLayer* addScaleNd(
7858         ITensor& input, ScaleMode mode, Weights shift, Weights scale, Weights power, int32_t channelAxis)
7859     noexcept
7860     {
7861         return mImpl->addScaleNd(input, mode, shift, scale, power, channelAxis);
7862     }
7873     IResizeLayer* addResize(ITensor& input) noexcept
7874     {
7875         return mImpl->addResize(input);
7876     }
7877
7887     TRT_DEPRECATED bool hasExplicitPrecision() const noexcept
7888     {
7889         return mImpl->hasExplicitPrecision();
7890     }
7891
7903     ILoop* addLoop() noexcept
7904     {
7905         return mImpl->addLoop();
7906     }

```

```

7907
7943 ISelectLayer* addSelect(ITensor& condition, ITensor& thenInput, ITensor& elseInput) noexcept
7944 {
7945     return mImpl->addSelect(condition, thenInput, elseInput);
7946 }
7947
7960 IAssertionLayer* addAssertion(ITensor& condition, char const* message) noexcept
7961 {
7962     return mImpl->addAssertion(condition, message);
7963 }
7964
7983 IFillLayer* addFill(Dims dimensions, FillOperation op) noexcept
7984 {
7985     return mImpl->addFill(dimensions, op);
7986 }
7987
8000 TRT_DEPRECATED IPaddingLayer* addPaddingNd(ITensor& input, Dims prePadding, Dims postPadding) noexcept
8001 {
8002     return mImpl->addPaddingNd(input, prePadding, postPadding);
8003 }
8004
8023 bool setWeightsName(Weights weights, char const* name) noexcept
8024 {
8025     return mImpl->setWeightsName(weights, name);
8026 }
8027
8039 //
8042 void setErrorRecorder(IErrorRecorder* recorder) noexcept
8043 {
8044     mImpl->setErrorRecorder(recorder);
8045 }
8046
8057 IErrorRecorder* getErrorRecorder() const noexcept
8058 {
8059     return mImpl->getErrorRecorder();
8060 }
8061
8076 IDequantizeLayer* addDequantize(ITensor& input, ITensor& scale) noexcept
8077 {
8078     return mImpl->addDequantize(input, scale);
8079 }
8080
8096 IScatterLayer* addScatter(ITensor& data, ITensor& indices, ITensor& updates, ScatterMode mode) noexcept
8097 {
8098     return mImpl->addScatter(data, indices, updates, mode);
8099 }
8100
8115 IQuantizeLayer* addQuantize(ITensor& input, ITensor& scale) noexcept
8116 {
8117     return mImpl->addQuantize(input, scale);
8118 }
8119
8130 IIfConditional* addIfConditional() noexcept
8131 {
8132     return mImpl->addIfConditional();
8133 }
8134
8144 IEinsumLayer* addEinsum(ITensor* const* inputs, int32_t nbInputs, char const* equation) noexcept
8145 {
8146     return mImpl->addEinsum(inputs, nbInputs, equation);
8147 }
8148
8160 IGridSampleLayer* addGridSample(ITensor& input, ITensor& grid) noexcept
8161 {
8162     return mImpl->addGridSample(input, grid);
8163 }
8164
8178 INMSLayer* addNMS(ITensor& boxes, ITensor& scores, ITensor& maxOutputBoxesPerClass) noexcept
8179 {
8180     return mImpl->addNMS(boxes, scores, maxOutputBoxesPerClass);
8181 }
8182
8195 IReverseSequenceLayer* addReverseSequence(ITensor& input, ITensor& sequenceLens) noexcept
8196 {
8197     return mImpl->addReverseSequence(input, sequenceLens);
8198 }
8199
8221 INormalizationLayer* addNormalization(
8222     ITensor& input, ITensor& scale, ITensor& bias, uint32_t axesMask) noexcept
8223 {
8224     return mImpl->addNormalization(input, scale, bias, axesMask);

```

```

8225     }
8226
8233     virtual IBuilder& getBuilder() const noexcept
8234     {
8235         return mImpl->getBuilder();
8236     }
8237
8238 protected:
8239     apiv::VNetworkDefinition* mImpl;
8240 };
8241
8247 enum class CalibrationAlgoType : int32_t
8248 {
8249     kLEGACY_CALIBRATION = 0,
8250     kENTROPY_CALIBRATION = 1,
8251     kENTROPY_CALIBRATION_2 = 2,
8252     kMINMAX_CALIBRATION = 3,
8253 };
8254
8260 template <>
8261 constexpr inline int32_t EnumMax<CalibrationAlgoType>() noexcept
8262 {
8263     return 4;
8264 }
8265
8277 class IInt8Calibrator
8278 {
8279 public:
8285     virtual int32_t getBatchSize() const noexcept = 0;
8286
8300     virtual bool getBatch(void* bindings[], char const* names[], int32_t nbBindings) noexcept = 0;
8301
8316     virtual void const* readCalibrationCache(std::size_t& length) noexcept = 0;
8317
8326     virtual void writeCalibrationCache(void const* ptr, std::size_t length) noexcept = 0;
8327
8333     virtual CalibrationAlgoType getAlgorithm() noexcept = 0;
8334
8335     virtual ~IInt8Calibrator() noexcept = default;
8336 };
8337
8342 class IInt8EntropyCalibrator : public IInt8Calibrator
8343 {
8344 public:
8348     CalibrationAlgoType getAlgorithm() noexcept override
8349     {
8350         return CalibrationAlgoType::kENTROPY_CALIBRATION;
8351     }
8352
8353     virtual ~IInt8EntropyCalibrator() noexcept = default;
8354 };
8355
8360 class IInt8EntropyCalibrator2 : public IInt8Calibrator
8361 {
8362 public:
8366     CalibrationAlgoType getAlgorithm() noexcept override
8367     {
8368         return CalibrationAlgoType::kENTROPY_CALIBRATION_2;
8369     }
8370
8371     virtual ~IInt8EntropyCalibrator2() noexcept = default;
8372 };
8373
8377 class IInt8MinMaxCalibrator : public IInt8Calibrator
8378 {
8379 public:
8383     CalibrationAlgoType getAlgorithm() noexcept override
8384     {
8385         return CalibrationAlgoType::kMINMAX_CALIBRATION;
8386     }
8387
8388     virtual ~IInt8MinMaxCalibrator() noexcept = default;
8389 };
8390
8395 class IInt8LegacyCalibrator : public IInt8Calibrator
8396 {
8397 public:
8401     CalibrationAlgoType getAlgorithm() noexcept override
8402     {
8403         return CalibrationAlgoType::kLEGACY_CALIBRATION;
8404     }

```

```

8405
8412     virtual double getQuantile() const noexcept = 0;
8413
8420     virtual double getRegressionCutoff() const noexcept = 0;
8421
8434     virtual void const* readHistogramCache(std::size_t& length) noexcept = 0;
8435
8444     virtual void writeHistogramCache(void const* ptr, std::size_t length) noexcept = 0;
8445
8446     virtual ~IInt8LegacyCalibrator() noexcept = default;
8447 };
8448
8459 class IAlgorithmIOInfo : public INoCopy
8460 {
8461 public:
8470     TRT_DEPRECATED TensorFormat getTensorFormat() const noexcept
8471     {
8472         return mImpl->getTensorFormat();
8473     }
8474
8480     DataType getDataType() const noexcept
8481     {
8482         return mImpl->getDataType();
8483     }
8484
8491     Dims getStrides() const noexcept
8492     {
8493         return mImpl->getStrides();
8494     }
8495
8501     int64_t getVectorizedDim() const noexcept
8502     {
8503         return mImpl->getVectorizedDim();
8504     }
8505
8512     int64_t getComponentsPerElement() const noexcept
8513     {
8514         return mImpl->getComponentsPerElement();
8515     }
8516
8517 protected:
8518     virtual ~IAlgorithmIOInfo() noexcept = default;
8519     apiv::VAlgorithmIOInfo* mImpl;
8520 };
8521
8533 class IAlgorithmVariant : public INoCopy
8534 {
8535 public:
8539     int64_t getImplementation() const noexcept
8540     {
8541         return mImpl->getImplementation();
8542     }
8543
8547     int64_t getTactic() const noexcept
8548     {
8549         return mImpl->getTactic();
8550     }
8551
8552 protected:
8553     virtual ~IAlgorithmVariant() noexcept = default;
8554     apiv::VAlgorithmVariant* mImpl;
8555 };
8556
8565 class IAlgorithmContext : public INoCopy
8566 {
8567 public:
8572     char const* getName() const noexcept
8573     {
8574         return mImpl->getName();
8575     }
8576
8583     Dims getDimensions(int32_t index, OptProfileSelector select) const noexcept
8584     {
8585         return mImpl->getDimensions(index, select);
8586     }
8587
8591     int32_t getNbInputs() const noexcept
8592     {
8593         return mImpl->getNbInputs();
8594     }
8595

```

```

8599     int32_t getNbOutputs() const noexcept
8600     {
8601         return mImpl->getNbOutputs();
8602     }
8603
8604 protected:
8605     virtual ~IAlgorithmContext() noexcept = default;
8606     apiv::VAlgorithmContext* mImpl;
8607 };
8608
8609 class IAlgorithm : public INoCopy
8610 {
8611 public:
8612     TRT_DEPRECATED IAlgorithmIOInfo const& getAlgorithmIOInfo(int32_t index) const noexcept
8613     {
8614         return mImpl->getAlgorithmIOInfo(index);
8615     }
8616
8617     IAlgorithmVariant const& getAlgorithmVariant() const noexcept
8618     {
8619         return mImpl->getAlgorithmVariant();
8620     }
8621
8622     float getTimingMSec() const noexcept
8623     {
8624         return mImpl->getTimingMSec();
8625     }
8626
8627     std::size_t getWorkspaceSize() const noexcept
8628     {
8629         return mImpl->getWorkspaceSize();
8630     }
8631
8632     IAlgorithmIOInfo const* getAlgorithmIOInfoByIndex(int32_t index) const noexcept
8633     {
8634         return mImpl->getAlgorithmIOInfoByIndex(index);
8635     }
8636
8637 protected:
8638     virtual ~IAlgorithm() noexcept = default;
8639     apiv::VAlgorithm* mImpl;
8640 }; // IAlgorithm
8641
8642 class IAlgorithmSelector
8643 {
8644 public:
8645     virtual int32_t selectAlgorithms(IAlgorithmContext const& context, IAlgorithm const* const* choices,
8646                                     int32_t nbChoices, int32_t* selection) noexcept = 0;
8647     virtual void reportAlgorithms(IAlgorithmContext const* const* algoContexts, IAlgorithm const* const*
8648                                   algoChoices,
8649                                   int32_t nbAlgorithms) noexcept = 0;
8650
8651     virtual ~IAlgorithmSelector() noexcept = default;
8652 };
8653
8654 using QuantizationFlags = uint32_t;
8655
8656 enum class QuantizationFlag : int32_t
8657 {
8658     kCALIBRATE_BEFORE_FUSION = 0
8659 };
8660
8661 template <>
8662 constexpr inline int32_t EnumMax<QuantizationFlag>() noexcept
8663 {
8664     return 1;
8665 }
8666
8667 using BuilderFlags = uint32_t;
8668
8669 enum class BuilderFlag : int32_t
8670 {
8671     kFP16 = 0,
8672     kINT8 = 1,
8673     kDEBUG = 2,
8674     kGPU_FALLBACK = 3,
8675
8676     kSTRICT_TYPES TRT_DEPRECATED_ENUM = 4,
8677
8678     kREFIT = 5,
8679     kDISABLE_TIMING_CACHE = 6,

```

```

8792
8796     kTF32 = 7,
8797
8799     kSPARSE_WEIGHTS = 8,
8800
8807     kSAFETY_SCOPE = 9,
8808
8810     kOBEY_PRECISION_CONSTRAINTS = 10,
8811
8814     kPREFER_PRECISION_CONSTRAINTS = 11,
8815
8819     kDIRECT_IO = 12,
8820
8822     kREJECT_EMPTY_ALGORITHMS = 13,
8823
8829     kENABLE_TACTIC_HEURISTIC = 14,
8830
8839     kVERSION_COMPATIBLE = 15,
8840
8848     kEXCLUDE_LEAN_RUNTIME = 16,
8849
8852     kFP8 = 17
8853 };
8854
8860 template <>
8861 constexpr inline int32_t EnumMax<BuilderFlag>() noexcept
8862 {
8863     return 18;
8864 }
8865
8876 class ITimingCache : public INoCopy
8877 {
8878 public:
8879     virtual ~ITimingCache() noexcept = default;
8880
8890     nvinfer1::IHostMemory* serialize() const noexcept
8891     {
8892         return mImpl->serialize();
8893     }
8894
8914     bool combine(ITimingCache const& inputCache, bool ignoreMismatch) noexcept
8915     {
8916         return mImpl->combine(inputCache, ignoreMismatch);
8917     }
8918
8924     bool reset() noexcept
8925     {
8926         return mImpl->reset();
8927     }
8928
8929 protected:
8930     apiv::VTimingCache* mImpl;
8931 };
8932
8940 enum class MemoryPoolType : int32_t
8941 {
8949     kWORKSPACE = 0,
8950
8957     kDLA_MANAGED_SRAM = 1,
8958
8964     kDLA_LOCAL_DRAM = 2,
8965
8971     kDLA_GLOBAL_DRAM = 3,
8972
8980     kTACTIC_DRAM = 4,
8981 };
8982
8988 template <>
8989 constexpr inline int32_t EnumMax<MemoryPoolType>() noexcept
8990 {
8991     return 5;
8992 }
8993
9002 enum class PreviewFeature : int32_t
9003 {
9014     kFASTER_DYNAMIC_SHAPES_0805 TRT_DEPRECATED_ENUM = 0,
9015
9030     kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805 = 1,
9031
9037     kPROFILE_SHARING_0806 = 2,
9038 };

```



```

9039 namespace impl
9040 {
9041     template <>
9042     struct EnumMaxImpl<PreviewFeature>
9043     {
9044         static constexpr int32_t kVALUE = 3;
9045     };
9046 } // namespace impl
9047
9048 enum class HardwareCompatibilityLevel : int32_t
9049 {
9050     kNONE = 0,
9051     kAMPERE_PLUS = 1,
9052 };
9053
9054 namespace impl
9055 {
9056     template <>
9057     struct EnumMaxImpl<HardwareCompatibilityLevel>
9058     {
9059         static constexpr int32_t kVALUE = 2;
9060     };
9061 } // namespace impl
9062
9063 class IBuilderConfig : public INoCopy
9064 {
9065 public:
9066     virtual ~IBuilderConfig() noexcept = default;
9067
9068     TRT_DEPRECATED virtual void setMinTimingIterations(int32_t minTiming) noexcept
9069     {
9070         mImpl->setMinTimingIterations(minTiming);
9071     }
9072
9073     TRT_DEPRECATED virtual int32_t getMinTimingIterations() const noexcept
9074     {
9075         return mImpl->getMinTimingIterations();
9076     }
9077
9078     virtual void setAvgTimingIterations(int32_t avgTiming) noexcept
9079     {
9080         mImpl->setAvgTimingIterations(avgTiming);
9081     }
9082
9083     int32_t getAvgTimingIterations() const noexcept
9084     {
9085         return mImpl->getAvgTimingIterations();
9086     }
9087
9088     void setEngineCapability(EngineCapability capability) noexcept
9089     {
9090         mImpl->setEngineCapability(capability);
9091     }
9092
9093     EngineCapability getEngineCapability() const noexcept
9094     {
9095         return mImpl->getEngineCapability();
9096     }
9097
9098     void setInt8Calibrator(IInt8Calibrator* calibrator) noexcept
9099     {
9100         mImpl->setInt8Calibrator(calibrator);
9101     }
9102
9103     IInt8Calibrator* getInt8Calibrator() const noexcept
9104     {
9105         return mImpl->getInt8Calibrator();
9106     }
9107
9108     TRT_DEPRECATED void setMaxWorkspaceSize(std::size_t workspaceSize) noexcept
9109     {
9110         mImpl->setMaxWorkspaceSize(workspaceSize);
9111     }
9112
9113     TRT_DEPRECATED std::size_t getMaxWorkspaceSize() const noexcept
9114     {
9115         return mImpl->getMaxWorkspaceSize();
9116     }
9117
9118     void setFlags(BuilderFlags builderFlags) noexcept

```

```

9239     {
9240         mImpl->setFlags(builderFlags);
9241     }
9242
9250     BuilderFlags getFlags() const noexcept
9251     {
9252         return mImpl->getFlags();
9253     }
9254
9262     void clearFlag(BuilderFlag builderFlag) noexcept
9263     {
9264         mImpl->clearFlag(builderFlag);
9265     }
9266
9274     void setFlag(BuilderFlag builderFlag) noexcept
9275     {
9276         mImpl->setFlag(builderFlag);
9277     }
9278
9286     bool getFlag(BuilderFlag builderFlag) const noexcept
9287     {
9288         return mImpl->getFlag(builderFlag);
9289     }
9290
9302     void setDeviceType(ILayer const* layer, DeviceType deviceType) noexcept
9303     {
9304         mImpl->setDeviceType(layer, deviceType);
9305     }
9306
9311     DeviceType getDeviceType(ILayer const* layer) const noexcept
9312     {
9313         return mImpl->getDeviceType(layer);
9314     }
9315
9321     bool isDeviceTypeSet(ILayer const* layer) const noexcept
9322     {
9323         return mImpl->isDeviceTypeSet(layer);
9324     }
9325
9331     void resetDeviceType(ILayer const* layer) noexcept
9332     {
9333         mImpl->resetDeviceType(layer);
9334     }
9335
9340     bool canRunOnDLA(ILayer const* layer) const noexcept
9341     {
9342         return mImpl->canRunOnDLA(layer);
9343     }
9344
9355     void setDLACore(int32_t dlaCore) noexcept
9356     {
9357         mImpl->setDLACore(dlaCore);
9358     }
9359
9364     int32_t getDLACore() const noexcept
9365     {
9366         return mImpl->getDLACore();
9367     }
9368
9374     void setDefaultDeviceType(DeviceType deviceType) noexcept
9375     {
9376         mImpl->setDefaultDeviceType(deviceType);
9377     }
9378
9384     DeviceType getDefaultDeviceType() const noexcept
9385     {
9386         return mImpl->getDefaultDeviceType();
9387     }
9388
9394     void reset() noexcept
9395     {
9396         mImpl->reset();
9397     }
9398
9408     TRT_DEPRECATED void destroy() noexcept
9409     {
9410         delete this;
9411     }
9412
9420     void setProfileStream(const cudaStream_t stream) noexcept
9421     {

```

```

9422         return mImpl->setProfileStream(stream);
9423     }
9424
9432     cudaStream_t getProfileStream() const noexcept
9433     {
9434         return mImpl->getProfileStream();
9435     }
9436
9448     int32_t addOptimizationProfile(IOptimizationProfile const* profile) noexcept
9449     {
9450         return mImpl->addOptimizationProfile(profile);
9451     }
9452
9461     int32_t getNbOptimizationProfiles() const noexcept
9462     {
9463         return mImpl->getNbOptimizationProfiles();
9464     }
9465
9473     void setProfilingVerbosity(ProfilingVerbosity verbosity) noexcept
9474     {
9475         mImpl->setProfilingVerbosity(verbosity);
9476     }
9477
9486     ProfilingVerbosity getProfilingVerbosity() const noexcept
9487     {
9488         return mImpl->getProfilingVerbosity();
9489     }
9490
9495     void setAlgorithmSelector(IAgorithmSelector* selector) noexcept
9496     {
9497         mImpl->setAlgorithmSelector(selector);
9498     }
9499
9503     IAgorithmSelector* getAlgorithmSelector() const noexcept
9504     {
9505         return mImpl->getAlgorithmSelector();
9506     }
9507
9518     bool setCalibrationProfile(IOptimizationProfile const* profile) noexcept
9519     {
9520         return mImpl->setCalibrationProfile(profile);
9521     }
9522
9528     IOptimizationProfile const* getCalibrationProfile() noexcept
9529     {
9530         return mImpl->getCalibrationProfile();
9531     }
9532
9545     void setQuantizationFlags(QuantizationFlags flags) noexcept
9546     {
9547         mImpl->setQuantizationFlags(flags);
9548     }
9549
9557     QuantizationFlags getQuantizationFlags() const noexcept
9558     {
9559         return mImpl->getQuantizationFlags();
9560     }
9561
9569     void clearQuantizationFlag(QuantizationFlag flag) noexcept
9570     {
9571         mImpl->clearQuantizationFlag(flag);
9572     }
9573
9581     void setQuantizationFlag(QuantizationFlag flag) noexcept
9582     {
9583         mImpl->setQuantizationFlag(flag);
9584     }
9585
9593     bool getQuantizationFlag(QuantizationFlag flag) const noexcept
9594     {
9595         return mImpl->getQuantizationFlag(flag);
9596     }
9597
9615     bool setTacticSources(TacticSources tacticSources) noexcept
9616     {
9617         return mImpl->setTacticSources(tacticSources);
9618     }
9619
9630     TacticSources getTacticSources() const noexcept
9631     {
9632         return mImpl->getTacticSources();

```

```

9633     }
9634
9649     nvinfer1::ITimingCache* createTimingCache(void const* blob, std::size_t size) const noexcept
9650     {
9651         return mImpl->createTimingCache(blob, size);
9652     }
9653
9672     bool setTimingCache(ITimingCache const& cache, bool ignoreMismatch) noexcept
9673     {
9674         return mImpl->setTimingCache(cache, ignoreMismatch);
9675     }
9676
9682     nvinfer1::ITimingCache const* getTimingCache() const noexcept
9683     {
9684         return mImpl->getTimingCache();
9685     }
9686
9714     void setMemoryPoolLimit(MemoryPoolType pool, std::size_t poolSize) noexcept
9715     {
9716         mImpl->setMemoryPoolLimit(pool, poolSize);
9717     }
9718
9733     std::size_t getMemoryPoolLimit(MemoryPoolType pool) const noexcept
9734     {
9735         return mImpl->getMemoryPoolLimit(pool);
9736     }
9737
9751     void setPreviewFeature(PreviewFeature feature, bool enable) noexcept
9752     {
9753         mImpl->setPreviewFeature(feature, enable);
9754     }
9755
9765     bool getPreviewFeature(PreviewFeature feature) const noexcept
9766     {
9767         return mImpl->getPreviewFeature(feature);
9768     }
9769
9785     void setBuilderOptimizationLevel(int32_t level) noexcept
9786     {
9787         mImpl->setBuilderOptimizationLevel(level);
9788     }
9789
9797     int32_t getBuilderOptimizationLevel() noexcept
9798     {
9799         return mImpl->getBuilderOptimizationLevel();
9800     }
9801
9813     void setHardwareCompatibilityLevel(HardwareCompatibilityLevel hardwareCompatibilityLevel) noexcept
9814     {
9815         mImpl->setHardwareCompatibilityLevel(hardwareCompatibilityLevel);
9816     }
9817
9826     HardwareCompatibilityLevel getHardwareCompatibilityLevel() const noexcept
9827     {
9828         return mImpl->getHardwareCompatibilityLevel();
9829     }
9830
9839     void setPluginsToSerialize(char const* const* paths, int32_t nbPaths) noexcept
9840     {
9841         mImpl->setPluginsToSerialize(paths, nbPaths);
9842     }
9843
9852     char const* getPluginToSerialize(int32_t index) const noexcept
9853     {
9854         return mImpl->getPluginToSerialize(index);
9855     }
9856
9862     int32_t getNbPluginsToSerialize() const noexcept
9863     {
9864         return mImpl->getNbPluginsToSerialize();
9865     }
9866
9891     void setMaxAuxStreams(int32_t nbStreams) noexcept
9892     {
9893         mImpl->setMaxAuxStreams(nbStreams);
9894     }
9895
9901     int32_t getMaxAuxStreams() const noexcept
9902     {
9903         return mImpl->getMaxAuxStreams();
9904     }

```

```

9905
9906 protected:
9907     apiv::VBuilderConfig* mImpl;
9908 };
9909
9916 using NetworkDefinitionCreationFlags = uint32_t;
9917
9926 enum class NetworkDefinitionCreationFlag : int32_t
9927 {
9933     kEXPLICIT_BATCH = 0,
9934
9937     kEXPLICIT_PRECISION TRT_DEPRECATED_ENUM = 1,
9938 };
9939
9945 template <>
9946 constexpr inline int32_t EnumMax<NetworkDefinitionCreationFlag>() noexcept
9947 {
9948     return 2;
9949 }
9950
9958 class IBuilder : public INoCopy
9959 {
9960 public:
9961     virtual ~IBuilder() noexcept = default;
9962
9973     TRT_DEPRECATED void setMaxBatchSize(int32_t batchSize) noexcept
9974     {
9975         mImpl->setMaxBatchSize(batchSize);
9976     }
9977
9988     TRT_DEPRECATED int32_t getMaxBatchSize() const noexcept
9989     {
9990         return mImpl->getMaxBatchSize();
9991     }
9992
9996     bool platformHasFastFp16() const noexcept
9997     {
9998         return mImpl->platformHasFastFp16();
9999     }
10000
10004     bool platformHasFastInt8() const noexcept
10005     {
10006         return mImpl->platformHasFastInt8();
10007     }
10008
10016     TRT_DEPRECATED void destroy() noexcept
10017     {
10018         delete this;
10019     }
10020
10028     int32_t getMaxDLABatchSize() const noexcept
10029     {
10030         return mImpl->getMaxDLABatchSize();
10031     }
10032
10036     int32_t getNbDLACores() const noexcept
10037     {
10038         return mImpl->getNbDLACores();
10039     }
10040
10052     void setGpuAllocator(IGpuAllocator* allocator) noexcept
10053     {
10054         mImpl->setGpuAllocator(allocator);
10055     }
10056
10062     nvinfer1::IBuilderConfig* createBuilderConfig() noexcept
10063     {
10064         return mImpl->createBuilderConfig();
10065     }
10066
10077     TRT_DEPRECATED nvinfer1::ICudaEngine* buildEngineWithConfig(
10078         INetworkDefinition& network, IBuilderConfig& config) noexcept
10079     {
10080         return mImpl->buildEngineWithConfig(network, config);
10081     }
10082
10095     nvinfer1::INetworkDefinition* createNetworkV2(NetworkDefinitionCreationFlags flags) noexcept
10096     {
10097         return mImpl->createNetworkV2(flags);
10098     }
10099

```

```

10109     nvinferl::IOptimizationProfile* createOptimizationProfile() noexcept
10110     {
10111         return mImpl->createOptimizationProfile();
10112     }
10113
10125     //
10128     void setErrorRecorder(IErrorRecorder* recorder) noexcept
10129     {
10130         mImpl->setErrorRecorder(recorder);
10131     }
10132
10143     IErrorRecorder* getErrorRecorder() const noexcept
10144     {
10145         return mImpl->getErrorRecorder();
10146     }
10147
10151     void reset() noexcept
10152     {
10153         mImpl->reset();
10154     }
10155
10159     bool platformHasTf32() const noexcept
10160     {
10161         return mImpl->platformHasTf32();
10162     }
10163
10178     nvinferl::IHostMemory* buildSerializedNetwork(INetworkDefinition& network, IBuilderConfig& config)
10179     noexcept
10180     {
10181         return mImpl->buildSerializedNetwork(network, config);
10182     }
10202     bool isNetworkSupported(INetworkDefinition const& network, IBuilderConfig const& config) const
10203     noexcept
10204     {
10205         return mImpl->isNetworkSupported(network, config);
10206     }
10212     ILogger* getLogger() const noexcept
10213     {
10214         return mImpl->getLogger();
10215     }
10216
10226     bool setMaxThreads(int32_t maxThreads) noexcept
10227     {
10228         return mImpl->setMaxThreads(maxThreads);
10229     }
10230
10240     int32_t getMaxThreads() const noexcept
10241     {
10242         return mImpl->getMaxThreads();
10243     }
10244
10250     IPluginRegistry& getPluginRegistry() noexcept
10251     {
10252         return mImpl->getPluginRegistry();
10253     }
10254
10255 protected:
10256     apiv::VBuilder* mImpl;
10257 };
10258
10259 } // namespace nvinferl
10260
10265 extern "C" TENSORRTAPI void* createInferBuilder.INTERNAL(void* logger, int32_t version) noexcept;
10266
10267 namespace nvinferl
10268 {
10269     namespace
10270     {
10271
10279         inline IBuilder* createInferBuilder(ILogger& logger) noexcept
10280         {
10281             return static_cast<IBuilder*>(createInferBuilder.INTERNAL(&logger, NV_TENSORRT_VERSION));
10282         }
10283
10284     } // namespace
10285
10298     extern "C" TENSORRTAPI nvinferl::IPluginRegistry* getBuilderPluginRegistry(
10299         nvinferl::EngineCapability capability) noexcept;
10300

```

```

10301 namespace safe
10302 {
10304 class IPluginRegistry;
10305 } // namespace safe
10306
10314 extern "C" TENSORRTAPI nvinferl::safe::IPluginRegistry* getBuilderSafePluginRegistry(
10315     nvinferl::EngineCapability capability) noexcept;
10316
10317 } // namespace nvinferl
10318
10319 #endif // NV_INFER_H

```

10.5 NvInferConsistency.h File Reference

```

#include "NvInferConsistencyImpl.h"
#include "NvInferRuntimeBase.h"
#include "NvInferRuntimePlugin.h"

```

Classes

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

Namespaces

- namespace [nvinferl](#)
The TensorRT API version 1 namespace.
- namespace [nvinferl::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) 1993-2023 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 "NvInferRuntimeBase.h"
18 #include "NvInferRuntimePlugin.h"
19
20
21
22
23
24 namespace nvinfer1
25 {
26
27 namespace consistency
28 {
29
30
31
32
33
34
35
36
37 class IConsistencyChecker
38 {
39 public:
40     //
41     bool validate() const noexcept
42     {
43         return mImpl->validate();
44     }
45
46     virtual ~IConsistencyChecker() = default;
47
48 protected:
49     apiv::VConsistencyChecker* mImpl;
50     IConsistencyChecker() = default;
51     IConsistencyChecker(IConsistencyChecker const& other) = delete;
52     IConsistencyChecker& operator=(IConsistencyChecker const& other) = delete;
53     IConsistencyChecker(IConsistencyChecker&& other) = delete;
54     IConsistencyChecker& operator=(IConsistencyChecker&& other) = delete;
55 };
56
57 class IPluginChecker : public IPluginCreator
58 {
59 public:
60     virtual bool validate(char const* name, void const* serialData, size_t serialLength, PluginTensorDesc
61         const* in,
62         size_t nbInputs, PluginTensorDesc const* out, size_t nbOutputs, int64_t workspaceSize) const noexcept
63         = 0;
64
65     IPluginChecker() = default;
66     virtual ~IPluginChecker() override = default;
67
68 protected:
69     IPluginChecker(IPluginChecker const&) = default;
70     IPluginChecker(IPluginChecker&&) = default;
71     IPluginChecker& operator=(IPluginChecker const&) &= default;
72     IPluginChecker& operator=(IPluginChecker&&) &= default;
73 };
74
75 } // namespace consistency
76
77 } // namespace nvinfer1
78
79 extern "C" TENSORRTAPI void* createConsistencyChecker_INTERNAL(void* logger, void const* blob, size_t size,
80     int32_t version);
81
82
83 namespace nvinfer1
84 {
85
86 namespace consistency
87 {

```



```

125 namespace // anonymous
126 {
127
128 inline IConsistencyChecker* createConsistencyChecker(ILogger& logger, void const* blob, size_t size)
129 {
130     return static_cast<IConsistencyChecker*>(
131         createConsistencyChecker_INTERNAL(&logger, blob, size, NV_TENSORRT_VERSION));
132 }
133
134 } // namespace
135
136 } // namespace consistency
137
138 } // namespace nvinfer1
139
140 #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) 1993-2023 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
30 namespace nvinfer1
31 {
32     class Dims2 : public Dims
33     {
34     public:
35         Dims2()
36             : Dims2(0, 0)
37         {
38         }
39
40         Dims2(int32_t d0, int32_t d1)
41         {
42             nbDims = 2;
43             d[0] = d0;
44             d[1] = d1;
45             for (int32_t i{nbDims}; i < Dims::MAX_DIMS; ++i)
46             {
47                 d[i] = 0;
48             }
49         }
50     };
51
52     class DimsHW : public Dims2
53     {
54     public:
55         DimsHW()
56             : Dims2()
57         {
58         }
59
60         DimsHW(int32_t height, int32_t width)
61             : Dims2(height, width)
62         {
63         }
64
65         int32_t& h()
66         {
67             return d[0];
68         }
69
70         int32_t h() const
71         {
72             return d[0];
73         }
74
75         int32_t& w()
76         {
77             return d[1];
78         }
79
80         int32_t w() const
81         {
82             return d[1];
83         }
84     };
85
86     class Dims3 : public Dims2
87     {
88     public:
89         Dims3()
90             : Dims2()
91         {
92         }
93
94         Dims3(int32_t d0, int32_t d1, int32_t d2)
95             : Dims2(d0, d1)
96         {
97             d[2] = d2;
98             for (int32_t i{nbDims}; i < Dims::MAX_DIMS; ++i)
99             {
100                 d[i] = 0;
101             }
102         }
103
104         int32_t& d0()
105         {
106             return d[0];
107         }
108
109         int32_t d0() const
110         {
111             return d[0];
112         }
113
114         int32_t& d1()
115         {
116             return d[1];
117         }
118
119         int32_t d1() const
120         {
121             return d[1];
122         }
123
124         int32_t& d2()
125         {
126             return d[2];
127         }
128
129         int32_t d2() const
130         {
131             return d[2];
132         }
133     };
134
135     class Dims4 : public Dims3
136     {
137     public:
138         Dims4()
139             : Dims3()
140         {
141         }
142
143         Dims4(int32_t d0, int32_t d1, int32_t d2, int32_t d3)
144             : Dims3(d0, d1, d2)
145         {
146             d[3] = d3;
147             for (int32_t i{nbDims}; i < Dims::MAX_DIMS; ++i)
148             {
149                 d[i] = 0;
150             }
151         }
152
153         int32_t& d0()
154         {
155             return d[0];
156         }
157
158         int32_t d0() const
159         {
160             return d[0];
161         }
162
163         int32_t& d1()
164         {
165             return d[1];
166         }
167
168         int32_t d1() const
169         {
170             return d[1];
171         }
172
173         int32_t& d2()
174         {
175             return d[2];
176         }
177
178         int32_t d2() const
179         {
180             return d[2];
181         }
182
183         int32_t& d3()
184         {
185             return d[3];
186         }
187
188         int32_t d3() const
189         {
190             return d[3];
191         }
192     };
193
194     class Dims5 : public Dims4
195     {
196     public:
197         Dims5()
198             : Dims4()
199         {
200         }
201
202         Dims5(int32_t d0, int32_t d1, int32_t d2, int32_t d3, int32_t d4)
203             : Dims4(d0, d1, d2, d3)
204         {
205             d[4] = d4;
206             for (int32_t i{nbDims}; i < Dims::MAX_DIMS; ++i)
207             {
208                 d[i] = 0;
209             }
210         }
211
212         int32_t& d0()
213         {
214             return d[0];
215         }
216
217         int32_t d0() const
218         {
219             return d[0];
220         }
221
222         int32_t& d1()
223         {
224             return d[1];
225         }
226
227         int32_t d1() const
228         {
229             return d[1];
230         }
231
232         int32_t& d2()
233         {
234             return d[2];
235         }
236
237         int32_t d2() const
238         {
239             return d[2];
240         }
241
242         int32_t& d3()
243         {
244             return d[3];
245         }
246
247         int32_t d3() const
248         {
249             return d[3];
250         }
251
252         int32_t& d4()
253         {
254             return d[4];
255         }
256
257         int32_t d4() const
258         {
259             return d[4];
260         }
261     };
262
263     class Dims6 : public Dims5
264     {
265     public:
266         Dims6()
267             : Dims5()
268         {
269         }
270
271         Dims6(int32_t d0, int32_t d1, int32_t d2, int32_t d3, int32_t d4, int32_t d5)
272             : Dims5(d0, d1, d2, d3, d4)
273         {
274             d[5] = d5;
275             for (int32_t i{nbDims}; i < Dims::MAX_DIMS; ++i)
276             {
277                 d[i] = 0;
278             }
279         }
280
281         int32_t& d0()
282         {
283             return d[0];
284         }
285
286         int32_t d0() const
287         {
288             return d[0];
289         }
290
291         int32_t& d1()
292         {
293             return d[1];
294         }
295
296         int32_t d1() const
297         {
298             return d[1];
299         }
300
301         int32_t& d2()
302         {
303             return d[2];
304         }
305
306         int32_t d2() const
307         {
308             return d[2];
309         }
310
311         int32_t& d3()
312         {
313             return d[3];
314         }
315
316         int32_t d3() const
317         {
318             return d[3];
319         }
320
321         int32_t& d4()
322         {
323             return d[4];
324         }
325
326         int32_t d4() const
327         {
328             return d[4];
329         }
330
331         int32_t& d5()
332         {
333             return d[5];
334         }
335
336         int32_t d5() const
337         {
338             return d[5];
339         }
340     };
341
342     class Dims7 : public Dims6
343     {
344     public:
345         Dims7()
346             : Dims6()
347         {
348         }
349
350         Dims7(int32_t d0, int32_t d1, int32_t d2, int32_t d3, int32_t d4, int32_t d5, int32_t d6)
351             : Dims6(d0, d1, d2, d3, d4, d5)
352         {
353             d[6] = d6;
354             for (int32_t i{nbDims}; i < Dims::MAX_DIMS; ++i)
355             {
356                 d[i] = 0;
357             }
358         }
359
360         int32_t& d0()
361         {
362             return d[0];
363         }
364
365         int32_t d0() const
366         {
367             return d[0];
368         }
369
370         int32_t& d1()
371         {
372             return d[1];
373         }
374
375         int32_t d1() const
376         {
377             return d[1];
378         }
379
380         int32_t& d2()
381         {
382             return d[2];
383         }
384
385         int32_t d2() const
386         {
387             return d[2];
388         }
389
390         int32_t& d3()
391         {
392             return d[3];
393         }
394
395         int32_t d3() const
396         {
397             return d[3];
398         }
399
400         int32_t& d4()
401         {
402             return d[4];
403         }
404
405         int32_t d4() const
406         {
407             return d[4];
408         }
409
410         int32_t& d5()
411         {
412             return d[5];
413         }
414
415         int32_t d5() const
416         {
417             return d[5];
418         }
419
420         int32_t& d6()
421         {
422             return d[6];
423         }
424
425         int32_t d6() const
426         {
427             return d[6];
428         }
429     };
430
431     class Dims8 : public Dims7
432     {
433     public:
434         Dims8()
435             : Dims7()
436         {
437         }
438
439         Dims8(int32_t d0, int32_t d1, int32_t d2, int32_t d3, int32_t d4, int32_t d5, int32_t d6, int32_t d7)
440             : Dims7(d0, d1, d2, d3, d4, d5, d6)
441         {
442             d[7] = d7;
443             for (int32_t i{nbDims}; i < Dims::MAX_DIMS; ++i)
444             {
445                 d[i] = 0;
446             }
447         }
448
449         int32_t& d0()
450         {
451             return d[0];
452         }
453
454         int32_t d0() const
455         {
456             return d[0];
457         }
458
459         int32_t& d1()
460         {
461             return d[1];
462         }
463
464         int32_t d1() const
465         {
466             return d[1];
467         }
468
469         int32_t& d2()
470         {
471             return d[2];
472         }
473
474         int32_t d2() const
475         {
476             return d[2];
477         }
478
479         int32_t& d3()
480         {
481             return d[3];
482         }
483
484         int32_t d3() const
485         {
486             return d[3];
487         }
488
489         int32_t& d4()
490         {
491             return d[4];
492         }
493
494         int32_t d4() const
495         {
496             return d[4];
497         }
498
499         int32_t& d5()
500         {
501             return d[5];
502         }
503
504         int32_t d5() const
505         {
506             return d[5];
507         }
508
509         int32_t& d6()
510         {
511             return d[6];
512         }
513
514         int32_t d6() const
515         {
516             return d[6];
517         }
518
519         int32_t& d7()
520         {
521             return d[7];
522         }
523
524         int32_t d7() const
525         {
526             return d[7];
527         }
528     };
529
530     class Dims9 : public Dims8
531     {
532     public:
533         Dims9()
534             : Dims8()
535         {
536         }
537
538         Dims9(int32_t d0, int32_t d1, int32_t d2, int32_t d3, int32_t d4, int32_t d5, int32_t d6, int32_t d7, int32_t d8)
539             : Dims8(d0, d1, d2, d3, d4, d5, d6, d7)
540         {
541             d[8] = d8;
542             for (int32_t i{nbDims}; i < Dims::MAX_DIMS; ++i)
543             {
544                 d[i] = 0;
545             }
546         }
547
548         int32_t& d0()
549         {
550             return d[0];
551         }
552
553         int32_t d0() const
554         {
555             return d[0];
556         }
557
558         int32_t& d1()
559         {
560             return d[1];
561         }
562
563         int32_t d1() const
564         {
565             return d[1];
566         }
567
568         int32_t& d2()
569         {
570             return d[2];
571         }
572
573         int32_t d2() const
574         {
575             return d[2];
576         }
577
578         int32_t& d3()
579         {
580             return d[3];
581         }
582
583         int32_t d3() const
584         {
585             return d[3];
586         }
587
588         int32_t& d4()
589         {
590             return d[4];
591         }
592
593         int32_t d4() const
594         {
595             return d[4];
596         }
597
598         int32_t& d5()
599         {
600             return d[5];
601         }
602
603         int32_t d5() const
604         {
605             return d[5];
606         }
607
608         int32_t& d6()
609         {
610             return d[6];
611         }
612
613         int32_t d6() const
614         {
615             return d[6];
616         }
617
618         int32_t& d7()
619         {
620             return d[7];
621         }
622
623         int32_t d7() const
624         {
625             return d[7];
626         }
627
628         int32_t& d8()
629         {
630             return d[8];
631         }
632
633         int32_t d8() const
634         {
635             return d[8];
636         }
637     };
638
639     class Dims10 : public Dims9
640     {
641     public:
642         Dims10()
643             : Dims9()
644         {
645         }
646
647         Dims10(int32_t d0, int32_t d1, int32_t d2, int32_t d3, int32_t d4, int32_t d5, int32_t d6, int32_t d7, int32_t d8, int32_t d9)
648             : Dims9(d0, d1, d2, d3, d4, d5, d6, d7, d8)
649         {
650             d[9] = d9;
651             for (int32_t i{nbDims}; i < Dims::MAX_DIMS; ++i)
652             {
653                 d[i] = 0;
654             }
655         }
656
657         int32_t& d0()
658         {
659             return d[0];
660         }
661
662         int32_t d0() const
663         {
664             return d[0];
665         }
666
667         int32_t& d1()
668         {
669             return d[1];
670         }
671
672         int32_t d1() const
673         {
674             return d[1];
675         }
676
677         int32_t& d2()
678         {
679             return d[2];
680         }
681
682         int32_t d2() const
683         {
684             return d[2];
685         }
686
687         int32_t& d3()
688         {
689             return d[3];
690         }
691
692         int32_t d3() const
693         {
694             return d[3];
695         }
696
697         int32_t& d4()
698         {
699             return d[4];
700         }
701
702         int32_t d4() const
703         {
704             return d[4];
705         }
706
707         int32_t& d5()
708         {
709             return d[5];
710         }
711
712         int32_t d5() const
713         {
714             return d[5];
715         }
716
717         int32_t& d6()
718         {
719             return d[6];
720         }
721
722         int32_t d6() const
723         {
724             return d[6];
725         }
726
727         int32_t& d7()
728         {
729             return d[7];
730         }
731
732         int32_t d7() const
733         {
734             return d[7];
735         }
736
737         int32_t& d8()
738         {
739             return d[8];
740         }
741
742         int32_t d8() const
743         {
744             return d[8];
745         }
746
747         int32_t& d9()
748         {
749             return d[9];
750         }
751
752         int32_t d9() const
753         {
754             return d[9];
755         }
756     };
757
758     class Dims11 : public Dims10
759     {
760     public:
761         Dims11()
762             : Dims10()
763         {
764         }
765
766         Dims11(int32_t d0, int32_t d1, int32_t d2, int32_t d3, int32_t d4, int32_t d5, int32_t d6, int32_t d7, int32_t d8, int32_t d9, int32_t d10)
767             : Dims10(d0, d1, d2, d3, d4, d5, d6, d7, d8, d9)
768         {
769             d[10] = d10;
770             for (int32_t i{nbDims}; i < Dims::MAX_DIMS; ++i)
771             {
772                 d[i] = 0;
773             }
774         }
775
776         int32_t& d0()
777         {
778             return d[0];
779         }
780
781         int32_t d0() const
782         {
783             return d[0];
784         }
785
786         int32_t& d1()
787         {
788             return d[1];
789         }
790
791         int32_t d1() const
792         {
793             return d[1];
794         }
795
796         int32_t& d2()
797         {
798             return d[2];
799         }
800
801         int32_t d2() const
802         {
803             return d[2];
804         }
805
806         int32_t& d3()
807         {
808             return d[3];
809         }
810
811         int32_t d3() const
812         {
813             return d[3];
814         }
815
816         int32_t& d4()
817         {
818             return d[4];
819         }
820
821         int32_t d4() const
822         {
823             return d[4];
824         }
825
826         int32_t& d5()
827         {
828             return d[5];
829         }
830
831         int32_t d5() const
832         {
833             return d[5];
834         }
835
836         int32_t& d6()
837         {
838             return d[6];
839         }
840
841         int32_t d6() const
842         {
843             return d[6];
844         }
845
846         int32_t& d7()
847         {
848             return d[7];
849         }
850
851         int32_t d7() const
852         {
853             return d[7];
854         }
855
856         int32_t& d8()
857         {
858             return d[8];
859         }
860
861         int32_t d8() const
862         {
863             return d[8];
864         }
865
866         int32_t& d9()
867         {
868             return d[9];
869         }
870
871         int32_t d9() const
872         {
873             return d[9];
874         }
875
876         int32_t& d10()
877         {
878             return d[10];
879         }
880
881         int32_t d10() const
882         {
883             return d[10];
884         }
885     };
886
887     class Dims12 : public Dims11
888     {
889     public:
890         Dims12()
891             : Dims11()
892         {
893         }
894
895         Dims12(int32_t d0, int32_t d1, int32_t d2, int32_t d3, int32_t d4, int32_t d5, int32_t d6, int32_t d7, int32_t d8, int32_t d9, int32_t d10, int32_t d11)
896             : Dims11(d0, d1, d2, d3, d4, d5, d6, d7, d8, d9, d10)
897         {
898             d[11] = d11;
899             for (int32_t i{nbDims}; i < Dims::MAX_DIMS; ++i)
900             {
901                 d[i] = 0;
902             }
903         }
904
905         int32_t& d0()
906         {
907             return d[0];
908         }
909
910         int32_t d0() const
911         {
912             return d[0];
913         }
914
915         int32_t& d1()
916         {
917             return d[1];
918         }
919
920         int32_t d1() const
921         {
922             return d[1];
923         }
924
925         int32_t& d2()
926         {
927             return d[2];
928         }
929
930         int32_t d2() const
931         {
932             return d[2];
933         }
934
935         int32_t& d3()
936         {
937             return d[3];
938         }
939
940         int32_t d3() const
941         {
942             return d[3];
943         }
944
945         int32_t& d4()
946         {
947             return d[4];
948         }
949
950         int32_t d4() const
951         {
952             return d[4];
953         }
954
955         int32_t& d5()
956         {
957             return d[5];
958         }
959
960         int32_t d5() const
961         {
962             return d[5];
963         }
964
965         int32_t& d6()
966         {
967             return d[6];
968         }
969
970         int32_t d6() const
971         {
972             return d[6];
973         }
974
975         int32_t& d7()
976         {
977             return d[7];
978         }
979
980         int32_t d7() const
981         {
982             return d[7];
983         }
984
985         int32_t& d8()
986         {
987             return d[8];
988         }
989
990         int32_t d8() const
991         {
992             return d[8];
993         }
994
995         int32_t& d9()
996         {
997             return d[9];
998         }
999
1000        int32_t d9() const
1001        {
1002            return d[9];
1003        }
1004
1005        int32_t& d10()
1006        {
1007            return d[10];
1008        }
1009
1010        int32_t d10() const
1011        {
1012            return d[10];
1013        }
1014
1015        int32_t& d11()
1016        {
1017            return d[11];
1018        }
1019
1020        int32_t d11() const
1021        {
1022            return d[11];
1023        }
1024    };
1025
1026     class Dims13 : public Dims12
1027     {
1028     public:
1029         Dims13()
1030             : Dims12()
1031         {
1032         }
1033
1034         Dims13(int32_t d0, int32_t d1, int32_t d2, int32_t d3, int32_t d4, int32_t d5, int32_t d6, int32_t d7, int32_t d8, int32_t d9, int32_t d10, int32_t d11, int32_t d12)
1035             : Dims12(d0, d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11)
1036         {
1037             d[12] = d12;
1038             for (int32_t i{nbDims}; i < Dims::MAX_DIMS; ++i)
1039             {
1040                 d[i] = 0;
1041             }
1042         }
1043
1044         int32_t& d0()
1045         {
1046             return d[0];
1047         }
1048
1049         int32_t d0() const
1050         {
1051             return d[0];
1052         }
1053
1054         int32_t& d1()
1055         {
1056             return d[1];
1057         }
1058
1059         int32_t d1() const
1060         {
1061             return d[1];
1062         }
1063
1064         int32_t& d2()
1065         {
1066             return d[2];
1067         }
1068
1069         int32_t d2() const
1070         {
1071             return d[2];
1072         }
1073
1074         int32_t& d3()
1075         {
1076             return d[3];
1077         }
1078
1079         int32_t d3() const
1080         {
1081             return d[3];
1082         }
1083
1084         int32_t& d4()
1085         {
1086             return d[4];
1087         }
1088
1089         int32_t d4() const
1090         {
1091             return d[4];
1092         }
1093
1094         int32_t& d5()
1095         {
1096             return d[5];
1097         }
1098
1099         int32_t d5() const
1100         {
1101             return d[5];
1102         }
1103
1104         int32_t& d6()
1105         {
1106             return d[6];
1107         }
1108
1109         int32_t d6() const
1110         {
1111             return d[6];
1112         }
1113
1114         int32_t& d7()
1115         {
1116             return d[7];
1117         }
1118
1119         int32_t d7() const
1120         {
1121             return d[7];
1122         }
1123
1124         int32_t& d8()
1125         {
1126             return d[8];
1127         }
1128
1129         int32_t d8() const
1130         {
1131             return d[8];
1132         }
1133
1134         int32_t& d9()
1135         {
1136             return d[9];
1137         }
1138
1139         int32_t d9() const
1140         {
1141             return d[9];
1142         }
1143
1144         int32_t& d10()
1145         {
1146             return d[10];
1147         }
1148
1149         int32_t d10() const
1150         {
1151             return d[10];
1152         }
1153
1154         int32_t& d11()
1155         {
1156             return d[11];
1157         }
1158
1159         int32_t d11() const
1160         {
1161             return d[11];
1162         }
1163
1164         int32_t& d12()
1165         {
1166             return d[12];
1167         }
1168
1169         int32_t d12() const
1170         {
1171             return d[12];
1172         }
1173     };
1174
1175     class Dims14 : public Dims13
1176     {
1177     public:
1178         Dims14()
1179             : Dims13()
1180         {
1181         }
1182
1183         Dims14(int32_t d0, int32_t d1, int32_t d2, int32_t d3, int32_t d4, int32_t d5, int32_t d6, int32_t d7, int32_t d8, int32_t d9, int32_t d10, int32_t d11, int32_t d12, int32_t d13)
1184             : Dims13(d0, d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12)
1185         {
1186             d[13] = d13;
1187             for (int32_t i{nbDims}; i < Dims::MAX_DIMS; ++i)
1188             {
1189                 d[i] = 0;
1190             }
1191         }
1192
1193         int32_t& d0()
1194         {
1195             return d[0];
1196         }
1197
1198         int32_t d0() const
1199         {
1200             return d[0];
1201         }
1202
1203         int32_t& d1()
1204         {
1205             return d[1];
1206         }
1207
1208         int32_t d1() const
1209         {
1210             return d[1];
1211         }
1212
1213         int32_t& d2()
1214         {
1215             return d[2];
1216         }
1217
1218         int32_t d2() const
1219         {
1220             return d[2];
1221         }
1222
1223         int32_t& d3()
1224         {
1225             return d[3];
1226         }
1227
1228         int32_t d3() const
1229         {
1230             return d[3];
1231         }
1232
1233         int32_t& d4()
1234         {
1235             return d[4];
1236         }
1237
1238         int32_t d4() const
1239         {
1240             return d[4];
1241         }
1242
1243         int32_t& d5()
1244         {
1245             return d[5];
1246         }
1247
1248         int32_t d5() const
1249         {
1250             return d[5];
1251         }
1252
1253         int32_t& d6()
1254         {
1255             return d[6];
1256         }
1257
1258         int32_t d6() const
1259         {
1260             return d[6];
1261         }
1262
1263         int32_t& d7()
1264         {
1265             return d[7];
1266         }
1267
1268         int32_t d7() const
1269         {
1270             return d[7];
1271         }
1272
1273         int32_t& d8()
1274         {
1275             return d[8];
1276         }
1277
1278         int32_t d8() const
1279         {
1280             return d[8];
1281         }
1282
1283         int32_t& d9()
1284         {
1285             return d[9];
1286         }
1287
1288         int32_t d9() const
1289         {
1290             return d[9];
1291         }
1292
1293         int32_t& d10()
1294         {
1295             return d[10];
1296         }
1297
1298         int32_t d10() const
1299         {
1300             return d[10];
1301         }
1302
1303         int32_t& d11()
1304         {
1305             return d[11];
1306         }
1307
1308         int32_t d11() const
1309         {
1310             return d[11];
1311         }
1312
1313         int32_t& d12()
1314         {
1315             return d[12];
1316         }
1317
1318         int32_t d12() const
1319         {
1320             return d[12];
1321         }
1322
1323         int32_t& d13()
1324         {
1325             return d[13];
1326         }
1327
1328         int32_t d13() const
1329         {
1330             return d[13];
1331         }
1332     };
1333
1334     class Dims15 : public Dims14
1335     {
1336     public:
1337         Dims15()
1338             : Dims
```

```

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 Grid↔AnchorGenerator 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 (<code>acrossSpatial</code> is true) or nearby spatial locations (within channel in which case <code>acrossSpatial</code> 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) 1993–2023 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) 1993-2023 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
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  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.

- class [nvinfer1::ILoggerFinder](#)

A virtual base class to find a logger. Allows a plugin to find an instance of a logger if it needs to emit a log message. A pointer to an instance of this class is passed to a plugin shared library on initialization when that plugin is serialized as part of a version-compatible plan. See the plugin chapter in the developer guide for details.

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) 1993-2023 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 #include "NvInferImpl.h"
19 #include "NvInferRuntimeCommon.h"
20
21 namespace nvinfer1
22 {
23     class IExecutionContext;
24     class ICudaEngine;
25     class IPluginFactory;
26     class IEngineInspector;
27
28     class INoCopy
29     {
30     protected:
31         INoCopy() = default;
32         virtual ~INoCopy() = default;
33         INoCopy(INoCopy const& other) = delete;
34         INoCopy& operator=(INoCopy const& other) = delete;
35         INoCopy(INoCopy&& other) = delete;
36         INoCopy& operator=(INoCopy&& other) = delete;
37     };
38
39     enum class EngineCapability : int32_t
40     {
41         kSTANDARD = 0,
42
43         kDEFAULT TRT_DEPRECATED_ENUM = kSTANDARD,
44
45         kSAFETY = 1,
46
47         kSAFE_GPU TRT_DEPRECATED_ENUM = kSAFETY,
48
49         kDLA_STANDALONE = 2,
50
51         kSAFE_DLA TRT_DEPRECATED_ENUM = kDLA_STANDALONE,
52     };
53
54     namespace impl
55     {
56         template <T>
57         struct EnumMaxImpl<EngineCapability>
58         {
59             static constexpr int32_t kVALUE = 3;
60         };
61     } // namespace impl
62 }
```

```

110
125 class Weights
126 {
127 public:
128     DataType type;
129     void const* values;
130     int64_t count;
131 };
132
133 class IHostMemory : public INoCopy
134 {
135 public:
136     virtual ~IHostMemory() noexcept = default;
137
138     void* data() const noexcept
139     {
140         return mImpl->data();
141     }
142
143     std::size_t size() const noexcept
144     {
145         return mImpl->size();
146     }
147
148     DataType type() const noexcept
149     {
150         return mImpl->type();
151     }
152
153     TRT_DEPRECATED void destroy() noexcept
154     {
155         delete this;
156     }
157
158 protected:
159     apiv::VHostMemory* mImpl;
160 };
161
162 enum class DimensionOperation : int32_t
163 {
164     kSUM = 0,
165     kPROD = 1,
166     kMAX = 2,
167     kMIN = 3,
168     kSUB = 4,
169     kEQUAL = 5,
170     kLESS = 6,
171     kFLOOR_DIV = 7,
172     kCEIL_DIV = 8
173 };
174
175 template <>
176 constexpr inline int32_t EnumMax<DimensionOperation>() noexcept
177 {
178     return 9;
179 }
180
181 enum class TensorLocation : int32_t
182 {
183     kDEVICE = 0,
184     kHOST = 1,
185 };
186
187 namespace impl
188 {
189     template <>
190     struct EnumMaxImpl<TensorLocation>
191     {
192         static constexpr int32_t kVALUE = 2;
193     };
194 } // namespace impl
195
196 class IDimensionExpr : public INoCopy
197 {
198 public:
199     bool isConstant() const noexcept
200     {
201         return mImpl->isConstant();
202     }
203
204     int32_t getConstantValue() const noexcept
205     {

```

```

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:
284     IDimensionExpr const* constant(int32_t value) noexcept
285     {
286         return mImpl->constant(value);
287     }
288
289     IDimensionExpr const* operation(
290         DimensionOperation op, IDimensionExpr const& first, IDimensionExpr const& second) noexcept
291     {
292         return mImpl->operation(op, first, second);
293     }
294
295 protected:
296     apiv::VExprBuilder* mImpl;
297     virtual ~IExprBuilder() noexcept = default;
298 };
299
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
331 class IPluginV2DynamicExt : public nvinfer1::IPluginV2Ext
332 {
333 public:
334     IPluginV2DynamicExt* clone() const noexcept override = 0;
335
336     virtual DimsExprs getOutputDimensions(
337         int32_t outputIndex, DimsExprs const* inputs, int32_t nbInputs, IExprBuilder& exprBuilder) noexcept =
338         0;
339
340     static constexpr int32_t kFORMAT_COMBINATION_LIMIT = 100;
341
342     virtual bool supportsFormatCombination(
343         int32_t pos, PluginTensorDesc const* inOut, int32_t nbInputs, int32_t nbOutputs) noexcept = 0;
344
345     virtual void configurePlugin(DynamicPluginTensorDesc const* in, int32_t nbInputs,
346         DynamicPluginTensorDesc const* out, int32_t nbOutputs) noexcept = 0;
347
348     virtual size_t getWorkspaceSize(PluginTensorDesc const* inputs, int32_t nbInputs, PluginTensorDesc const*
349         outputs,
350         int32_t nbOutputs) const noexcept = 0;
351
352     virtual int32_t enqueue(PluginTensorDesc const* inputDesc, PluginTensorDesc const* outputDesc,
353         void const* const* inputs, void* const* outputs, void* workspace, cudaStream_t stream) noexcept = 0;
354
355 protected:
356     int32_t getTensorRTVersion() const noexcept override
357     {
358         return (static_cast<int32_t>(PluginVersion::kV2_DYNAMICEXT) << 24 | (NV_TENSORRT_VERSION & 0xFFFFF));
359     }
360
361     virtual ~IPluginV2DynamicExt() noexcept {}
362
363 private:
364     // Following are obsolete base class methods, and must not be implemented or used.
365
366     void configurePlugin(Dims const*, int32_t, Dims const*, int32_t, DataType const*, DataType const*, bool
367         const*,
368         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 }

```

```

637 }
638
644 using TempfileControlFlags = uint32_t;
645
653 class IRuntime : public INoCopy
654 {
655 public:
656     virtual ~IRuntime() noexcept = default;
657
673     TRT_DEPRECATED nvinfer1::ICudaEngine* deserializeCudaEngine(
674         void const* blob, std::size_t size, IPluginFactory* pluginFactory) noexcept
675     {
676         return mImpl->deserializeCudaEngine(blob, size, nullptr);
677     }
678
689     void setDLACore(int32_t dlaCore) noexcept
690     {
691         mImpl->setDLACore(dlaCore);
692     }
693
698     int32_t getDLACore() const noexcept
699     {
700         return mImpl->getDLACore();
701     }
702
706     int32_t getNbDLACores() const noexcept
707     {
708         return mImpl->getNbDLACores();
709     }
710
718     TRT_DEPRECATED void destroy() noexcept
719     {
720         delete this;
721     }
722
732     void setGpuAllocator(IGpuAllocator* allocator) noexcept
733     {
734         mImpl->setGpuAllocator(allocator);
735     }
736
748     //
751     void setErrorRecorder(IErrorRecorder* recorder) noexcept
752     {
753         mImpl->setErrorRecorder(recorder);
754     }
755
766     IErrorRecorder* getErrorRecorder() const noexcept
767     {
768         return mImpl->getErrorRecorder();
769     }
770
781     ICudaEngine* deserializeCudaEngine(void const* blob, std::size_t size) noexcept
782     {
783         return mImpl->deserializeCudaEngine(blob, size, nullptr);
784     }
785
791     ILogger* getLogger() const noexcept
792     {
793         return mImpl->getLogger();
794     }
795
805     bool setMaxThreads(int32_t maxThreads) noexcept
806     {
807         return mImpl->setMaxThreads(maxThreads);
808     }
809
819     int32_t getMaxThreads() const noexcept
820     {
821         return mImpl->getMaxThreads();
822     }
823
854     void setTemporaryDirectory(char const* path) noexcept
855     {
856         return mImpl->setTemporaryDirectory(path);
857     }
858
865     char const* getTemporaryDirectory() const noexcept
866     {
867         return mImpl->getTemporaryDirectory();
868     }
869

```

```

881     void setTempfileControlFlags(TempfileControlFlags flags) noexcept
882     {
883         return mImpl->setTempfileControlFlags(flags);
884     }
885
886     TempfileControlFlags getTempfileControlFlags() const noexcept
887     {
888         return mImpl->getTempfileControlFlags();
889     }
890
891     IPluginRegistry& getPluginRegistry() noexcept
892     {
893         return mImpl->getPluginRegistry();
894     }
895
896     IRuntime* loadRuntime(char const* path) noexcept
897     {
898         return mImpl->loadRuntime(path);
899     }
900
901     void setEngineHostCodeAllowed(bool allowed) noexcept
902     {
903         return mImpl->setEngineHostCodeAllowed(allowed);
904     }
905
906     bool getEngineHostCodeAllowed() const noexcept
907     {
908         return mImpl->getEngineHostCodeAllowed();
909     }
910
911 protected:
912     apiv::VRuntime* mImpl;
913 };
914
915 class IRefitter : public INoCopy
916 {
917 public:
918     virtual ~IRefitter() noexcept = default;
919
920     bool setWeights(char const* layerName, WeightsRole role, Weights weights) noexcept
921     {
922         return mImpl->setWeights(layerName, role, weights);
923     }
924
925     bool refitCudaEngine() noexcept
926     {
927         return mImpl->refitCudaEngine();
928     }
929
930     int32_t getMissing(int32_t size, char const** layerNames, WeightsRole* roles) noexcept
931     {
932         return mImpl->getMissing(size, layerNames, roles);
933     }
934
935     int32_t getAll(int32_t size, char const** layerNames, WeightsRole* roles) noexcept
936     {
937         return mImpl->getAll(size, layerNames, roles);
938     }
939
940     TRT_DEPRECATED void destroy() noexcept
941     {
942         delete this;
943     }
944
945     bool setDynamicRange(char const* tensorName, float min, float max) noexcept
946     {
947         return mImpl->setDynamicRange(tensorName, min, max);
948     }
949
950     float getDynamicRangeMin(char const* tensorName) const noexcept
951     {
952         return mImpl->getDynamicRangeMin(tensorName);
953     }
954
955     float getDynamicRangeMax(char const* tensorName) const noexcept
956     {
957         return mImpl->getDynamicRangeMax(tensorName);
958     }
959
960     int32_t getTensorsWithDynamicRange(int32_t size, char const** tensorNames) const noexcept
961     {

```

```

1106         return mImpl->getTensorsWithDynamicRange(size, tensorNames);
1107     }
1108
1109     //
1110     void setErrorRecorder(IErrorRecorder* recorder) noexcept
1111     {
1112         mImpl->setErrorRecorder(recorder);
1113     }
1114
1115     IErrorRecorder* getErrorRecorder() const noexcept
1116     {
1117         return mImpl->getErrorRecorder();
1118     }
1119
1120     bool setNamedWeights(char const* name, Weights weights) noexcept
1121     {
1122         return mImpl->setNamedWeights(name, weights);
1123     }
1124
1125     int32_t getMissingWeights(int32_t size, char const** weightsNames) noexcept
1126     {
1127         return mImpl->getMissingWeights(size, weightsNames);
1128     }
1129
1130     int32_t getAllWeights(int32_t size, char const** weightsNames) noexcept
1131     {
1132         return mImpl->getAllWeights(size, weightsNames);
1133     }
1134
1135     ILogger* getLogger() const noexcept
1136     {
1137         return mImpl->getLogger();
1138     }
1139
1140     bool setMaxThreads(int32_t maxThreads) noexcept
1141     {
1142         return mImpl->setMaxThreads(maxThreads);
1143     }
1144
1145     int32_t getMaxThreads() const noexcept
1146     {
1147         return mImpl->getMaxThreads();
1148     }
1149
1150 protected:
1151     apiv::VRefitter* mImpl;
1152 };
1153
1154 enum class OptProfileSelector : int32_t
1155 {
1156     kMIN = 0,
1157     kOPT = 1,
1158     kMAX = 2
1159 };
1160
1161 template <>
1162 constexpr inline int32_t EnumMax<OptProfileSelector>() noexcept
1163 {
1164     return 3;
1165 }
1166
1167 class IOptimizationProfile : public INoCopy
1168 {
1169 public:
1170     bool setDimensions(char const* inputName, OptProfileSelector select, Dims dims) noexcept
1171     {
1172         return mImpl->setDimensions(inputName, select, dims);
1173     }
1174
1175     Dims getDimensions(char const* inputName, OptProfileSelector select) const noexcept
1176     {
1177         return mImpl->getDimensions(inputName, select);
1178     }
1179
1180     bool setShapeValues(
1181         char const* inputName, OptProfileSelector select, int32_t const* values, int32_t nbValues) noexcept
1182     {
1183         return mImpl->setShapeValues(inputName, select, values, nbValues);
1184     }
1185
1186     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,
1481     kCUBLAS_LT = 1,
1486     kCUDNN = 2,
1487
1491     kEDGE_MASK_CONVOLUTIONS = 3,
1492
1495     kJIT_CONVOLUTIONS = 4,
1496 };
1497
1498 template <>
1499 constexpr inline int32_t EnumMax<TacticSource>() noexcept
1500 {
1501     return 5;
1502 }
1503
1510 using TacticSources = uint32_t;
1511
1521 enum class ProfilingVerbosity : int32_t
1522 {
1523     kLAYER_NAMES_ONLY = 0,
1524     kNONE = 1,
1525     kDETAILED = 2,
1526
1528     kDEFAULT TRT_DEPRECATED_ENUM = kLAYER_NAMES_ONLY,
1530     kVERBOSE TRT_DEPRECATED_ENUM = kDETAILED
1531 };
1532
1534 template <>
1535 constexpr inline int32_t EnumMax<ProfilingVerbosity>() noexcept
1536 {
1537     return 3;
1538 }
1539
1547 class ICudaEngine : public INoCopy
1548 {
1549 public:
1550     virtual ~ICudaEngine() noexcept = default;
1551
1564     TRT_DEPRECATED int32_t getNbBindings() const noexcept
1565     {
1566         return mImpl->getNbBindings();
1567     }
1568
1592     TRT_DEPRECATED int32_t getBindingIndex(char const* name) const noexcept
1593     {
1594         return mImpl->getBindingIndex(name);
1595     }
1596
1615     TRT_DEPRECATED char const* getBindingName(int32_t bindingIndex) const noexcept
1616     {

```



```

1617         return mImpl->getBindingName(bindingIndex);
1618     }
1619
1630     TRT_DEPRECATED bool bindingIsInput(int32_t bindingIndex) const noexcept
1631     {
1632         return mImpl->bindingIsInput(bindingIndex);
1633     }
1634
1657     TRT_DEPRECATED Dims getBindingDimensions(int32_t bindingIndex) const noexcept
1658     {
1659         return mImpl->getBindingDimensions(bindingIndex);
1660     }
1661
1672     Dims getTensorShape(char const* tensorName) const noexcept
1673     {
1674         return mImpl->getTensorShape(tensorName);
1675     }
1676
1687     TRT_DEPRECATED DataType getBindingDataType(int32_t bindingIndex) const noexcept
1688     {
1689         return mImpl->getBindingDataType(bindingIndex);
1690     }
1691
1702     DataType getTensorDataType(char const* tensorName) const noexcept
1703     {
1704         return mImpl->getTensorDataType(tensorName);
1705     }
1706
1718     TRT_DEPRECATED int32_t getMaxBatchSize() const noexcept
1719     {
1720         return mImpl->getMaxBatchSize();
1721     }
1722
1732     int32_t getNbLayers() const noexcept
1733     {
1734         return mImpl->getNbLayers();
1735     }
1736
1746     IHostMemory* serialize() const noexcept
1747     {
1748         return mImpl->serialize();
1749     }
1750
1763     IExecutionContext* createExecutionContext() noexcept
1764     {
1765         return mImpl->createExecutionContext();
1766     }
1767
1775     TRT_DEPRECATED void destroy() noexcept
1776     {
1777         delete this;
1778     }
1779
1793     TRT_DEPRECATED TensorLocation getLocation(int32_t bindingIndex) const noexcept
1794     {
1795         return mImpl->getLocation(bindingIndex);
1796     }
1797
1810     TensorLocation getTensorLocation(char const* tensorName) const noexcept
1811     {
1812         return mImpl->getTensorLocation(tensorName);
1813     }
1814
1830     bool isShapeInferenceIO(char const* tensorName) const noexcept
1831     {
1832         return mImpl->isShapeInferenceIO(tensorName);
1833     }
1834
1844     TensorIOMode getTensorIOMode(char const* tensorName) const noexcept
1845     {
1846         return mImpl->getTensorIOMode(tensorName);
1847     }
1848
1853     IExecutionContext* createExecutionContextWithoutDeviceMemory() noexcept
1854     {
1855         return mImpl->createExecutionContextWithoutDeviceMemory();
1856     }
1857
1863     size_t getDeviceMemorySize() const noexcept
1864     {
1865         return mImpl->getDeviceMemorySize();

```

```

1866     }
1867
1873     bool isRefittable() const noexcept
1874     {
1875         return mImpl->isRefittable();
1876     }
1877
1890     TRT_DEPRECATED int32_t getBindingBytesPerComponent(int32_t bindingIndex) const noexcept
1891     {
1892         return mImpl->getBindingBytesPerComponent(bindingIndex);
1893     }
1894
1911     int32_t getTensorBytesPerComponent(char const* tensorName) const noexcept
1912     {
1913         return mImpl->getTensorBytesPerComponent(tensorName);
1914     }
1915
1929     int32_t getTensorBytesPerComponent(char const* tensorName, int32_t profileIndex) const noexcept
1930     {
1931         return mImpl->getTensorBytesPerComponentV2(tensorName, profileIndex);
1932     }
1933
1945     TRT_DEPRECATED int32_t getBindingComponentsPerElement(int32_t bindingIndex) const noexcept
1946     {
1947         return mImpl->getBindingComponentsPerElement(bindingIndex);
1948     }
1949
1966     int32_t getTensorComponentsPerElement(char const* tensorName) const noexcept
1967     {
1968         return mImpl->getTensorComponentsPerElement(tensorName);
1969     }
1970
1984     int32_t getTensorComponentsPerElement(char const* tensorName, int32_t profileIndex) const noexcept
1985     {
1986         return mImpl->getTensorComponentsPerElementV2(tensorName, profileIndex);
1987     }
1988
1998     TRT_DEPRECATED TensorFormat getBindingFormat(int32_t bindingIndex) const noexcept
1999     {
2000         return mImpl->getBindingFormat(bindingIndex);
2001     }
2002
2013     TensorFormat getTensorFormat(char const* tensorName) const noexcept
2014     {
2015         return mImpl->getTensorFormat(tensorName);
2016     }
2017
2027     TensorFormat getTensorFormat(char const* tensorName, int32_t profileIndex) const noexcept
2028     {
2029         return mImpl->getTensorFormatV2(tensorName, profileIndex);
2030     }
2031
2051     TRT_DEPRECATED char const* getBindingFormatDesc(int32_t bindingIndex) const noexcept
2052     {
2053         return mImpl->getBindingFormatDesc(bindingIndex);
2054     }
2055
2075     char const* getTensorFormatDesc(char const* tensorName) const noexcept
2076     {
2077         return mImpl->getTensorFormatDesc(tensorName);
2078     }
2079
2098     char const* getTensorFormatDesc(char const* tensorName, int32_t profileIndex) const noexcept
2099     {
2100         return mImpl->getTensorFormatDescV2(tensorName, profileIndex);
2101     }
2102
2114     TRT_DEPRECATED int32_t getBindingVectorizedDim(int32_t bindingIndex) const noexcept
2115     {
2116         return mImpl->getBindingVectorizedDim(bindingIndex);
2117     }
2118
2131     int32_t getTensorVectorizedDim(char const* tensorName) const noexcept
2132     {
2133         return mImpl->getTensorVectorizedDim(tensorName);
2134     }
2135
2147     int32_t getTensorVectorizedDim(char const* tensorName, int32_t profileIndex) const noexcept
2148     {
2149         return mImpl->getTensorVectorizedDimV2(tensorName, profileIndex);
2150     }

```

```

2151
2162 char const* getName() const noexcept
2163 {
2164     return mImpl->getName();
2165 }
2166
2173 int32_t getNbOptimizationProfiles() const noexcept
2174 {
2175     return mImpl->getNbOptimizationProfiles();
2176 }
2177
2205 TRT_DEPRECATED Dims getProfileDimensions(
2206     int32_t bindingIndex, int32_t profileIndex, OptProfileSelector select) const noexcept
2207 {
2208     return mImpl->getProfileDimensions(bindingIndex, profileIndex, select);
2209 }
2210
2226 Dims getProfileShape(char const* tensorName, int32_t profileIndex, OptProfileSelector select) const
noexcept
2227 {
2228     return mImpl->getProfileShape(tensorName, profileIndex, select);
2229 }
2230
2255 TRT_DEPRECATED int32_t const* getProfileShapeValues(
2256     int32_t profileIndex, int32_t inputIndex, OptProfileSelector select) const noexcept
2257 {
2258     return mImpl->getProfileShapeValues(profileIndex, inputIndex, select);
2259 }
2260
2294 TRT_DEPRECATED bool isShapeBinding(int32_t bindingIndex) const noexcept
2295 {
2296     return mImpl->isShapeBinding(bindingIndex);
2297 }
2298
2311 TRT_DEPRECATED bool isExecutionBinding(int32_t bindingIndex) const noexcept
2312 {
2313     return mImpl->isExecutionBinding(bindingIndex);
2314 }
2315
2326 EngineCapability getEngineCapability() const noexcept
2327 {
2328     return mImpl->getEngineCapability();
2329 }
2330
2341 //
2344 void setErrorRecorder(IErrorRecorder* recorder) noexcept
2345 {
2346     return mImpl->setErrorRecorder(recorder);
2347 }
2348
2359 IErrorRecorder* getErrorRecorder() const noexcept
2360 {
2361     return mImpl->getErrorRecorder();
2362 }
2363
2378 bool hasImplicitBatchDimension() const noexcept
2379 {
2380     return mImpl->hasImplicitBatchDimension();
2381 }
2382
2393 TacticSources getTacticSources() const noexcept
2394 {
2395     return mImpl->getTacticSources();
2396 }
2397
2404 ProfilingVerbosity getProfilingVerbosity() const noexcept
2405 {
2406     return mImpl->getProfilingVerbosity();
2407 }
2408
2414 IEngineInspector* createEngineInspector() const noexcept
2415 {
2416     return mImpl->createEngineInspector();
2417 }
2418
2427 int32_t getNbIOTensors() const noexcept
2428 {
2429     return mImpl->getNbIOTensors();
2430 }
2431
2439 char const* getIOTensorName(int32_t index) const noexcept

```

```

2440     {
2441         return mImpl->getIOTensorName(index);
2442     }
2443
2444     HardwareCompatibilityLevel getHardwareCompatibilityLevel() const noexcept
2445     {
2446         return mImpl->getHardwareCompatibilityLevel();
2447     }
2448
2449     int32_t getNbAuxStreams() const noexcept
2450     {
2451         return mImpl->getNbAuxStreams();
2452     }
2453
2454 protected:
2455     apiv::VCudaEngine* mImpl;
2456 };
2457
2458 class IOutputAllocator
2459 {
2460 public:
2461     virtual int32_t getInterfaceVersion() const noexcept
2462     {
2463         return 1;
2464     }
2465
2466     virtual void* reallocateOutput(char const* tensorName, void* currentMemory, uint64_t size, uint64_t
alignment) noexcept = 0;
2467
2468     virtual void notifyShape(char const* tensorName, Dims const& dims) noexcept = 0;
2469     virtual ~IOutputAllocator() = default;
2470 };
2471
2472 class IExecutionContext : public INoCopy
2473 {
2474 public:
2475     virtual ~IExecutionContext() noexcept = default;
2476
2477     TRT_DEPRECATED bool execute(int32_t batchSize, void* const* bindings) noexcept
2478     {
2479         return mImpl->execute(batchSize, bindings);
2480     }
2481
2482     TRT_DEPRECATED bool enqueue(
int32_t batchSize, void* const* bindings, cudaStream_t stream, cudaEvent_t* inputConsumed) noexcept
2483     {
2484         return mImpl->enqueue(batchSize, bindings, stream, inputConsumed);
2485     }
2486
2487     void setDebugSync(bool sync) noexcept
2488     {
2489         mImpl->setDebugSync(sync);
2490     }
2491
2492     bool getDebugSync() const noexcept
2493     {
2494         return mImpl->getDebugSync();
2495     }
2496
2497     void setProfiler(IProfiler* profiler) noexcept
2498     {
2499         mImpl->setProfiler(profiler);
2500     }
2501
2502     IProfiler* getProfiler() const noexcept
2503     {
2504         return mImpl->getProfiler();
2505     }
2506
2507     ICudaEngine const& getEngine() const noexcept
2508     {
2509         return mImpl->getEngine();
2510     }
2511
2512     TRT_DEPRECATED void destroy() noexcept
2513     {
2514         delete this;
2515     }
2516
2517     void setName(char const* name) noexcept

```

```

2682     {
2683         mImpl->setName(name);
2684     }
2685
2691     char const* getName() const noexcept
2692     {
2693         return mImpl->getName();
2694     }
2695
2708     void setDeviceMemory(void* memory) noexcept
2709     {
2710         mImpl->setDeviceMemory(memory);
2711     }
2712
2731     TRT_DEPRECATED Dims getStrides(int32_t bindingIndex) const noexcept
2732     {
2733         return mImpl->getStrides(bindingIndex);
2734     }
2735
2752     Dims getTensorStrides(char const* tensorName) const noexcept
2753     {
2754         return mImpl->getTensorStrides(tensorName);
2755     }
2756
2757 public:
2789     TRT_DEPRECATED
2790     bool setOptimizationProfile(int32_t profileIndex) noexcept
2791     {
2792         return mImpl->setOptimizationProfile(profileIndex);
2793     }
2794
2804     int32_t getOptimizationProfile() const noexcept
2805     {
2806         return mImpl->getOptimizationProfile();
2807     }
2808
2843     TRT_DEPRECATED bool setBindingDimensions(int32_t bindingIndex, Dims dimensions) noexcept
2844     {
2845         return mImpl->setBindingDimensions(bindingIndex, dimensions);
2846     }
2847
2861     bool setInputShape(char const* tensorName, Dims const& dims) noexcept
2862     {
2863         return mImpl->setInputShape(tensorName, dims);
2864     }
2865
2894     TRT_DEPRECATED Dims getBindingDimensions(int32_t bindingIndex) const noexcept
2895     {
2896         return mImpl->getBindingDimensions(bindingIndex);
2897     }
2898
2931     Dims getTensorShape(char const* tensorName) const noexcept
2932     {
2933         return mImpl->getTensorShape(tensorName);
2934     }
2935
2966     TRT_DEPRECATED bool setInputShapeBinding(int32_t bindingIndex, int32_t const* data) noexcept
2967     {
2968         return mImpl->setInputShapeBinding(bindingIndex, data);
2969     }
2970
2990     TRT_DEPRECATED bool getShapeBinding(int32_t bindingIndex, int32_t* data) const noexcept
2991     {
2992         return mImpl->getShapeBinding(bindingIndex, data);
2993     }
2994
3008     bool allInputDimensionsSpecified() const noexcept
3009     {
3010         return mImpl->allInputDimensionsSpecified();
3011     }
3012
3025     bool allInputShapesSpecified() const noexcept
3026     {
3027         return mImpl->allInputShapesSpecified();
3028     }
3029
3041     //
3044     void setErrorRecorder(IErrorRecorder* recorder) noexcept
3045     {
3046         mImpl->setErrorRecorder(recorder);
3047     }

```

```

3048
3059     IErrorRecorder* getErrorRecorder() const noexcept
3060     {
3061         return mImpl->getErrorRecorder();
3062     }
3063
3076     bool executeV2(void* const* bindings) noexcept
3077     {
3078         return mImpl->executeV2(bindings);
3079     }
3080
3106     TRT_DEPRECATED bool enqueueV2(void* const* bindings, cudaStream_t stream, cudaEvent_t* inputConsumed)
noexcept
3107     {
3108         return mImpl->enqueueV2(bindings, stream, inputConsumed);
3109     }
3110
3153     bool setOptimizationProfileAsync(int32_t profileIndex, cudaStream_t stream) noexcept
3154     {
3155         return mImpl->setOptimizationProfileAsync(profileIndex, stream);
3156     }
3157
3168     void setEnqueueEmitsProfile(bool enqueueEmitsProfile) noexcept
3169     {
3170         mImpl->setEnqueueEmitsProfile(enqueueEmitsProfile);
3171     }
3172
3179     bool getEnqueueEmitsProfile() const noexcept
3180     {
3181         return mImpl->getEnqueueEmitsProfile();
3182     }
3183
3208     bool reportToProfiler() const noexcept
3209     {
3210         return mImpl->reportToProfiler();
3211     }
3212
3250     bool setTensorAddress(char const* tensorName, void* data) noexcept
3251     {
3252         return mImpl->setTensorAddress(tensorName, data);
3253     }
3254
3267     void const* getTensorAddress(char const* tensorName) const noexcept
3268     {
3269         return mImpl->getTensorAddress(tensorName);
3270     }
3271
3289     bool setInputTensorAddress(char const* tensorName, void const* data) noexcept
3290     {
3291         return mImpl->setInputTensorAddress(tensorName, data);
3292     }
3293
3308     void* getOutputTensorAddress(char const* tensorName) const noexcept
3309     {
3310         return mImpl->getOutputTensorAddress(tensorName);
3311     }
3312
3341     int32_t inferShapes(int32_t nbMaxNames, char const** tensorNames) noexcept
3342     {
3343         return mImpl->inferShapes(nbMaxNames, tensorNames);
3344     }
3345
3357     bool setInputConsumedEvent(cudaEvent_t event) noexcept
3358     {
3359         return mImpl->setInputConsumedEvent(event);
3360     }
3361
3367     cudaEvent_t getInputConsumedEvent() const noexcept
3368     {
3369         return mImpl->getInputConsumedEvent();
3370     }
3371
3386     bool setOutputAllocator(char const* tensorName, IOutputAllocator* outputAllocator) noexcept
3387     {
3388         return mImpl->setOutputAllocator(tensorName, outputAllocator);
3389     }
3390
3399     IOutputAllocator* getOutputAllocator(char const* tensorName) const noexcept
3400     {
3401         return mImpl->getOutputAllocator(tensorName);
3402     }

```

```

3403
3417     int64_t getMaxOutputSize(char const* tensorName) const noexcept
3418     {
3419         return mImpl->getMaxOutputSize(tensorName);
3420     }
3421
3437     bool setTemporaryStorageAllocator(IGpuAllocator* allocator) noexcept
3438     {
3439         return mImpl->setTemporaryStorageAllocator(allocator);
3440     }
3441
3447     IGPUAllocator* getTemporaryStorageAllocator() const noexcept
3448     {
3449         return mImpl->getTemporaryStorageAllocator();
3450     }
3451
3465     bool enqueueV3(cudaStream_t stream) noexcept
3466     {
3467         return mImpl->enqueueV3(stream);
3468     }
3469
3480     void setPersistentCacheLimit(size_t size) noexcept
3481     {
3482         mImpl->setPersistentCacheLimit(size);
3483     }
3484
3491     size_t getPersistentCacheLimit() const noexcept
3492     {
3493         return mImpl->getPersistentCacheLimit();
3494     }
3495
3515     bool setNvtxVerbosity(ProfilingVerbosity verbosity) noexcept
3516     {
3517         return mImpl->setNvtxVerbosity(verbosity);
3518     }
3519
3527     ProfilingVerbosity getNvtxVerbosity() const noexcept
3528     {
3529         return mImpl->getNvtxVerbosity();
3530     }
3531
3558     void setAuxStreams(cudaStream_t* auxStreams, int32_t nbStreams) noexcept
3559     {
3560         mImpl->setAuxStreams(auxStreams, nbStreams);
3561     }
3562
3563 protected:
3564     apiv::VExecutionContext* mImpl;
3565 }; // class IExecutionContext
3566
3574 enum class LayerInformationFormat : int32_t
3575 {
3576     kONELINE = 0,
3577     kJSON = 1,
3578 };
3579
3582 template <>
3583 constexpr inline int32_t EnumMax<LayerInformationFormat>() noexcept
3584 {
3585     return 2;
3586 }
3587
3603 class IEngineInspector : public INoCopy
3604 {
3605 public:
3606     virtual ~IEngineInspector() noexcept = default;
3607
3620     bool setExecutionContext(IExecutionContext const* context) noexcept
3621     {
3622         return mImpl->setExecutionContext(context);
3623     }
3624
3632     IExecutionContext const* getExecutionContext() const noexcept
3633     {
3634         return mImpl->getExecutionContext();
3635     }
3636
3657     char const* getLayerInformation(int32_t layerIndex, LayerInformationFormat format) const noexcept
3658     {
3659         return mImpl->getLayerInformation(layerIndex, format);
3660     }

```

```

3661
3680     char const* getEngineInformation(LayerInformationFormat format) const noexcept
3681     {
3682         return mImpl->getEngineInformation(format);
3683     }
3684
3696     //
3699     void setErrorRecorder(IErrorRecorder* recorder) noexcept
3700     {
3701         mImpl->setErrorRecorder(recorder);
3702     }
3703
3714     IErrorRecorder* getErrorRecorder() const noexcept
3715     {
3716         return mImpl->getErrorRecorder();
3717     }
3718
3719 protected:
3720     apiv::VEngineInspector* mImpl;
3721 }; // class IEngineInspector
3722
3723 } // namespace nvinfer1
3724
3729 extern "C" TENSORRTAPI void* createInferRuntime_INTERNAL(void* logger, int32_t version) noexcept;
3730
3735 extern "C" TENSORRTAPI void* createInferRefitter_INTERNAL(void* engine, void* logger, int32_t version)
3736     noexcept;
3737
3740 extern "C" TENSORRTAPI nvinfer1::IPluginRegistry* getPluginRegistry() noexcept;
3741
3747 extern "C" TENSORRTAPI nvinfer1::ILogger* getLogger() noexcept;
3748
3749 namespace nvinfer1
3750 {
3751     namespace // unnamed namespace avoids linkage surprises when linking objects built with different versions
3752         of this
3753         // header.
3754     {
3755         inline IRuntime* createInferRuntime(ILogger& logger) noexcept
3756         {
3757             return static_cast<IRuntime*>(createInferRuntime_INTERNAL(&logger, NV_TENSORRT_VERSION));
3758         }
3759
3769         inline IRefitter* createInferRefitter(ICudaEngine& engine, ILogger& logger) noexcept
3770         {
3771             return static_cast<IRefitter*>(createInferRefitter_INTERNAL(&engine, &logger, NV_TENSORRT_VERSION));
3772         }
3773
3774     } // namespace
3775
3787     template <typename T>
3788     class PluginRegistrar
3789     {
3790     public:
3791         PluginRegistrar()
3792         {
3793             getPluginRegistry()->registerCreator(instance, "");
3794         }
3795
3796     private:
3797         T instance{};
3798     };
3799
3800 } // namespace nvinfer1
3801
3802 #define REGISTER_TENSORRT_PLUGIN(name)
3803     \
3804     static nvinfer1::PluginRegistrar<name> pluginRegistrar##name {}
3805
3806 namespace nvinfer1
3807 {
3816     class ILoggerFinder
3817     {
3818     public:
3826         virtual ILogger* findLogger() = 0;
3827
3828     protected:
3829         virtual ~ILoggerFinder() = default;
3830     };
3831
3832 } // namespace nvinfer1

```



```

3833
3834 #endif // NV_INFER_RUNTIME_H

```

10.15 NvInferRuntimeBase.h File Reference

```

#include "NvInferVersion.h"
#include <cstdint>
#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::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::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::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 file contains common definitions, data structures and interfaces shared between the standard and safe runtime.

Warning

Do not directly include this file. Instead include either [NvInferRuntime.h](#) (for the standard runtime) or [NvInferSafeRuntime.h](#) (for the safety runtime).

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 NvInferRuntimeBase.h

[Go to the documentation of this file.](#)

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2023 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_BASE_H
14 #define NV_INFER_RUNTIME_BASE_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
65
66
67
68 extern "C"
69 {
70     struct cublasContext;
71     struct cudnnContext;
72 }
73
74 #define NV_TENSORRT_VERSION nvInfer1::kNV_TENSORRT_VERSION_IMPL
75 namespace nvInfer1
76 {
77     static constexpr int32_t kNV_TENSORRT_VERSION_IMPL
78         = (NV_TENSORRT_MAJOR * 1000) + (NV_TENSORRT_MINOR * 100) + NV_TENSORRT_PATCH;
79
80     using char_t = char;
81
82     using AsciiChar = char_t;
83
84     class IErrorRecorder;
85     class IGpuAllocator;
86
87     namespace impl
88     {
89     template <typename T>
90     struct EnumMaxImpl;
91     } // namespace impl
92
93     template <typename T>
94     constexpr int32_t EnumMax() noexcept
95     {
96         return impl::EnumMaxImpl<T>::kVALUE;
97     }
98
99     enum class DataType : int32_t
100     {
101         kFLOAT = 0,
102         kHALF = 1,
103         kINT8 = 2,
104         kINT32 = 3,
105         kBOOL = 4,
106         kUINT8 = 5,
107         kFP8 = 6
108     };
109
110     namespace impl
111     {
112     template <>
113     struct EnumMaxImpl<DataType>
114     {
115         static constexpr int32_t kVALUE = 7;
116     };
117     } // namespace impl
118
119     class Dims32
120     {
121     public:
122         static constexpr int32_t MAX_DIMS{8};
123         int32_t nbDims;
124         int32_t d[MAX_DIMS];
125     };
126
127     using Dims = Dims32;
128
129     enum class TensorFormat : int32_t
130     {
131         kLINEAR = 0,

```

```

224     kCHW2 = 1,
225
232     kHWC8 = 2,
233
249     kCHW4 = 3,
250
261     kCHW16 = 4,
262
272     kCHW32 = 5,
273
280     kDHWC8 = 6,
281
288     kCDHW32 = 7,
289
292     kHWC = 8,
293
302     kDLA.LINEAR = 9,
303
317     kDLA.HWC4 = 10,
318
325     kHWC16 = 11,
326
329     kDHWC = 12
330 };
331
332 namespace impl
333 {
334     template <>
335     struct EnumMaxImpl<TensorFormat>
336     {
337         static constexpr int32_t kVALUE = 13;
338     };
339 } // namespace impl
340
341 enum class AllocatorFlag : int32_t
342 {
343     kRESIZABLE = 0,
344 };
345
346 namespace impl
347 {
348     template <>
349     struct EnumMaxImpl<AllocatorFlag>
350     {
351         static constexpr int32_t kVALUE = 1;
352     };
353 } // namespace impl
354
355 using AllocatorFlags = uint32_t;
356
357 class IGpuAllocator
358 {
359 public:
360     virtual void* allocate(uint64_t const size, uint64_t const alignment, AllocatorFlags const flags)
361         noexcept = 0;
362
363     TRT_DEPRECATED virtual void free(void* const memory) noexcept = 0;
364
365     virtual ~IGpuAllocator() = default;
366     IGpuAllocator() = default;
367
368     virtual void* reallocate(void* /*baseAddr*/, uint64_t /*alignment*/, uint64_t /*newSize*/) noexcept
369     {
370         return nullptr;
371     }
372
373     virtual bool deallocate(void* const memory) noexcept
374     {
375         this->free(memory);
376         return true;
377     }
378
379 protected:
380     // @cond SuppressDoxyWarnings
381     IGpuAllocator(IGpuAllocator const&) = default;
382     IGpuAllocator(IGpuAllocator&&) = default;
383     IGpuAllocator& operator=(IGpuAllocator const&) &= default;
384     IGpuAllocator& operator=(IGpuAllocator&&) &= default;
385     // @endcond
386 };
387
388
389
390
391
392
393
394
395
396
397
398
399
400

```

```

513 class ILogger
514 {
515 public:
521     enum class Severity : int32_t
522     {
524         kINTERNAL_ERROR = 0,
526         kERROR = 1,
528         kWARNING = 2,
530         kINFO = 3,
532         kVERBOSE = 4,
533     };
534
553     virtual void log(Severity severity, AsciiChar const* msg) noexcept = 0;
554
555     ILogger() = default;
556     virtual ~ILogger() = default;
557
558 protected:
559 // @cond SuppressDoxyWarnings
560     ILogger(ILogger const&) = default;
561     ILogger(ILogger&&) = default;
562     ILogger& operator=(ILogger const&) &= default;
563     ILogger& operator=(ILogger&&) &= default;
564 // @endcond
565 };
566
567 namespace impl
568 {
569     template <>
571     struct EnumMaxImpl<ILogger::Severity>
572     {
574         static constexpr int32_t kVALUE = 5;
575     };
576 } // namespace impl
577
583 enum class ErrorCode : int32_t
584 {
588     kSUCCESS = 0,
589
593     kUNSPECIFIED_ERROR = 1,
594
599     kINTERNAL_ERROR = 2,
600
605     kINVALID_ARGUMENT = 3,
606
614     kINVALID_CONFIG = 4,
615
621     kFAILED_ALLOCATION = 5,
622
627     kFAILED_INITIALIZATION = 6,
628
635     kFAILED_EXECUTION = 7,
636
644     kFAILED_COMPUTATION = 8,
645
658     kINVALID_STATE = 9,
659
670     kUNSUPPORTED_STATE = 10,
671 };
672 };
673
674 namespace impl
675 {
677     template <>
678     struct EnumMaxImpl<ErrorCode>
679     {
681         static constexpr int32_t kVALUE = 11;
682     };
683 } // namespace impl
684
712 class IErrorRecorder
713 {
714 public:
718     using ErrorDesc = char const*;
719
723     static constexpr size_t kMAX_DESC_LENGTH{127U};
724
728     using RefCount = int32_t;
729
730     IErrorRecorder() = default;
731     virtual ~IErrorRecorder() noexcept = default;

```

```

732
733 // Public API used to retrieve information from the error recorder.
734
758 virtual int32_t getNbErrors() const noexcept = 0;
759
778 virtual ErrorCode getErrorCode(int32_t errorIdx) const noexcept = 0;
779
801 virtual ErrorDesc getErrorDesc(int32_t errorIdx) const noexcept = 0;
802
817 virtual bool hasOverflowed() const noexcept = 0;
818
833 virtual void clear() noexcept = 0;
834
835 // API used by TensorRT to report Error information to the application.
836
863 virtual bool reportError(ErrorCode val, ErrorDesc desc) noexcept = 0;
864
881 virtual RefCount incRefCount() noexcept = 0;
882
899 virtual RefCount decRefCount() noexcept = 0;
900
901 protected:
902 // @cond SuppressDoxyWarnings
903 IErrorRecorder(IErrorRecorder const&) = default;
904 IErrorRecorder(IErrorRecorder&&) = default;
905 IErrorRecorder& operator=(IErrorRecorder const&) &= default;
906 IErrorRecorder& operator=(IErrorRecorder&&) &= default;
907 // @endcond
908 }; // class IErrorRecorder
909
915 enum class TensorIOMode : int32_t
916 {
917     kNONE = 0,
918
919     kINPUT = 1,
920
921     kOUTPUT = 2
922 };
923
924 namespace impl
925 {
926     template <>
927     struct EnumMaxImpl<TensorIOMode>
928     {
929         // Declaration of kVALUE that represents maximum number of elements in TensorIOMode enum
930         static constexpr int32_t kVALUE = 3;
931     };
932 } // namespace impl
933 } // namespace nvinfer1
934
935 extern "C" TENSORRTAPI int32_t getInferLibVersion() noexcept;
936
937 #endif // NV_INFER_RUNTIME_BASE_H

```

10.17 NvInferRuntimeCommon.h File Reference

```

#include "NvInferRuntimeBase.h"
#include "NvInferRuntimePlugin.h"

```

Classes

- class [nvinfer1::IPluginRegistry](#)

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

Namespaces

- namespace [nvinfer1](#)

The TensorRT API version 1 namespace.

10.17.1 Detailed Description

This file provides the [nvinfer1::IPluginRegistry](#) interface, which will be moved to the [NvInferRuntime.h](#) header in TensorRT 9.0.

Warning

This file will be removed in TensorRT 9.0.

Do not directly include this file. Instead include [NvInferRuntime.h](#)

10.18 NvInferRuntimeCommon.h

[Go to the documentation of this file.](#)

```

1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2023 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 "NvInferRuntimeBase.h"
17 #include "NvInferRuntimePlugin.h"
18
19 namespace nvinfer1
20 {
21
22     class IPluginRegistry
23     {
24     public:
25         using PluginLibraryHandle = void*;
26         virtual bool registerCreator(IPluginCreator& creator, AsciiChar const* const pluginNamespace) noexcept =
27             0;
28
29         virtual IPluginCreator* const* getPluginCreatorList(int32_t* const numCreators) const noexcept = 0;
30
31         virtual IPluginCreator* getPluginCreator(AsciiChar const* const pluginName, AsciiChar const* const
32             pluginVersion,
33             AsciiChar const* const pluginNamespace = "") noexcept
34             = 0;
35
36         // @cond SuppressDoxyWarnings
37         IPluginRegistry() = default;
38         IPluginRegistry(IPluginRegistry const&) = delete;
39         IPluginRegistry(IPluginRegistry&&) = delete;
40         IPluginRegistry& operator=(IPluginRegistry const&) &= delete;
41         IPluginRegistry& operator=(IPluginRegistry&&) &= delete;
42         // @endcond
43
44     protected:
45         virtual ~IPluginRegistry() noexcept = default;
46
47     public:
48         //

```

```

120     virtual void setErrorRecorder(IErrorRecorder* const recorder) noexcept = 0;
121
122     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
123
124     virtual bool deregisterCreator(IPluginCreator const& creator) noexcept = 0;
125
126     virtual bool isParentSearchEnabled() const = 0;
127
128     virtual void setParentSearchEnabled(bool const enabled) = 0;
129
130     virtual PluginLibraryHandle loadLibrary(AsciiChar const* pluginPath) noexcept = 0;
131
132     virtual void deregisterLibrary(PluginLibraryHandle handle) noexcept = 0;
133 };
134
135 } // namespace nvinfer1
136
137 #endif /* NV_INFER_RUNTIME_COMMON_H */

```

10.19 NvInferRuntimePlugin.h File Reference

```
#include "NvInferRuntimeBase.h"
```

Classes

- struct [nvinfer1::PluginTensorDesc](#)
Fields that a plugin might see for an input or output.
- class [nvinfer1::IPluginV2](#)
Plugin class for user-implemented layers.
- class [nvinfer1::IPluginV2Ext](#)
Plugin class for user-implemented layers.
- class [nvinfer1::IPluginV2IOExt](#)
Plugin class for user-implemented layers.
- class [nvinfer1::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.

Namespaces

- namespace [nvinfer1](#)
The TensorRT API version 1 namespace.

Typedefs

- using [nvinfer1::PluginFormat](#) = TensorFormat
PluginFormat is reserved for backward compatibility.

Enumerations

- 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 }

The possible field types for custom layer.

10.19.1 Detailed Description

This file contains common definitions, data structures and interfaces that relate to plugins and are shared between the standard and safe runtime.

Warning

Do not directly include this file. Instead include either [NvInferRuntime.h](#) (for the standard runtime) or [NvInferSafeRuntime.h](#) (for the safety runtime).

10.20 NvInferRuntimePlugin.h

[Go to the documentation of this file.](#)

```

1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2023 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_PLUGIN_H
14 #define NV_INFER_RUNTIME_PLUGIN_H
15
16 #include "NvInferRuntimeBase.h"
17
18
19 namespace nvinfer1
20 {
21
22     using PluginFormat = TensorFormat;
23
24     struct PluginTensorDesc
25     {
26         Dims dims;
27         DataType type;
28         TensorFormat format;
29         float scale;
30     };
31
32     enum class PluginVersion : uint8_t
33     {
34         kV2 = 0,
35         kV2_EXT = 1,
36         kV2_IOEXT = 2,
37         kV2_DYNAMICEXT = 3,
38     };
39
40     class TRT_DEPRECATED IPluginV2
41     {

```

```

99 public:
112     virtual int32_t getTensorRTVersion() const noexcept
113     {
114         return NV_TENSORRT_VERSION;
115     }
116
129     virtual AsciiChar const* getPluginType() const noexcept = 0;
130
143     virtual AsciiChar const* getPluginVersion() const noexcept = 0;
144
158     virtual int32_t getNbOutputs() const noexcept = 0;
159
183     virtual Dims getOutputDimensions(int32_t index, Dims const* inputs, int32_t nbInputDims) noexcept = 0;
184
207     virtual bool supportsFormat(DataType type, PluginFormat format) const noexcept = 0;
208
241     virtual void configureWithFormat(Dims const* inputDims, int32_t nbInputs, Dims const* outputDims, int32_t
nbOutputs,
242         DataType type, PluginFormat format, int32_t maxBatchSize) noexcept
243         = 0;
244
256     virtual int32_t initialize() noexcept = 0;
257
270     virtual void terminate() noexcept = 0;
271
288     virtual size_t getWorkspaceSize(int32_t maxBatchSize) const noexcept = 0;
289
311     virtual int32_t enqueue(int32_t batchSize, void const* const* inputs, void* const* outputs, void*
workspace,
312         cudaStream_t stream) noexcept
313         = 0;
314
325     virtual size_t getSerializationSize() const noexcept = 0;
326
340     virtual void serialize(void* buffer) const noexcept = 0;
341
350     virtual void destroy() noexcept = 0;
351
369     virtual IPluginV2* clone() const noexcept = 0;
370
385     virtual void setPluginNamespace(AsciiChar const* pluginNamespace) noexcept = 0;
386
398     virtual AsciiChar const* getPluginNamespace() const noexcept = 0;
399
400     // @cond SuppressDoxyWarnings
401     IPluginV2() = default;
402     virtual ~IPluginV2() noexcept = default;
403 // @endcond
404
405 protected:
406 // @cond SuppressDoxyWarnings
407     IPluginV2(IPluginV2 const&) = default;
408     IPluginV2(IPluginV2&&) = default;
409     IPluginV2& operator=(IPluginV2 const&) &= default;
410     IPluginV2& operator=(IPluginV2&&) &= default;
411 // @endcond
412 };
413
427 class TRT_DEPRECATED IPluginV2Ext : public IPluginV2
428 {
429 public:
453     virtual nvinfer1::DataType getOutputDataType(
454         int32_t index, nvinfer1::DataType const* inputTypes, int32_t nbInputs) const noexcept
455         = 0;
456
474     virtual bool isOutputBroadcastAcrossBatch(
475         int32_t outputIndex, bool const* inputIsBroadcasted, int32_t nbInputs) const noexcept
476         = 0;
477
500     virtual bool canBroadcastInputAcrossBatch(int32_t inputIndex) const noexcept = 0;
501
539     virtual void configurePlugin(Dims const* inputDims, int32_t nbInputs, Dims const* outputDims, int32_t
nbOutputs,
540         DataType const* inputTypes, DataType const* outputTypes, bool const* inputIsBroadcast,
541         bool const* outputIsBroadcast, PluginFormat floatFormat, int32_t maxBatchSize) noexcept
542         = 0;
543
544     IPluginV2Ext() = default;
545     ~IPluginV2Ext() override = default;
546
572     virtual void attachToContext(

```

```

573     cudnnContext* /*cudnn*/, cublasContext* /*cublas*/, IGpuAllocator* /*allocator*/) noexcept
574     {
575     }
576
590     virtual void detachFromContext() noexcept {}
591
606     IPluginV2Ext* clone() const noexcept override = 0;
607
608 protected:
609     // @cond SuppressDoxyWarnings
610     IPluginV2Ext(IPluginV2Ext const&) = default;
611     IPluginV2Ext(IPluginV2Ext&&) = default;
612     IPluginV2Ext& operator=(IPluginV2Ext const&) &= default;
613     IPluginV2Ext& operator=(IPluginV2Ext&&) &= default;
614 // @endcond
615
630     int32_t getTensorRTVersion() const noexcept override
631     {
632         return static_cast<int32_t>((static_cast<uint32_t>(PluginVersion::kV2_EXT) << 24U)
633             | (static_cast<uint32_t>(NV_TENSORRT_VERSION) & 0xFFFFFU));
634     }
635
642     void configureWithFormat(Dims const* /*inputDims*/, int32_t /*nbInputs*/, Dims const* /*outputDims*/,
643         int32_t /*nbOutputs*/, DataType /*type*/, PluginFormat /*format*/, int32_t /*maxBatchSize*/) noexcept
        override
644     {
645     }
646 };
647
657 class IPluginV2IOExt : public IPluginV2Ext
658 {
659 public:
677     virtual void configurePlugin(
678         PluginTensorDesc const* in, int32_t nbInput, PluginTensorDesc const* out, int32_t nbOutput) noexcept
679         = 0;
680
718     virtual bool supportsFormatCombination(
719         int32_t pos, PluginTensorDesc const* inOut, int32_t nbInputs, int32_t nbOutputs) const noexcept
720         = 0;
721
722     // @cond SuppressDoxyWarnings
723     IPluginV2IOExt() = default;
724     ~IPluginV2IOExt() override = default;
725 // @endcond
726
727 protected:
728 // @cond SuppressDoxyWarnings
729     IPluginV2IOExt(IPluginV2IOExt const&) = default;
730     IPluginV2IOExt(IPluginV2IOExt&&) = default;
731     IPluginV2IOExt& operator=(IPluginV2IOExt const&) &= default;
732     IPluginV2IOExt& operator=(IPluginV2IOExt&&) &= default;
733 // @endcond
734
746     int32_t getTensorRTVersion() const noexcept override
747     {
748         return static_cast<int32_t>((static_cast<uint32_t>(PluginVersion::kV2_IOEXT) << 24U)
749             | (static_cast<uint32_t>(NV_TENSORRT_VERSION) & 0xFFFFFU));
750     }
751
752 private:
753     // Following are obsolete base class methods, and must not be implemented or used.
754
755     void configurePlugin(Dims const*, int32_t, Dims const*, int32_t, DataType const*, DataType const*, bool
756         const*, bool const*, PluginFormat, int32_t) noexcept final
757     {
758     }
759
760     bool supportsFormat(DataType, PluginFormat) const noexcept final
761     {
762         return false;
763     }
764 };
765
770
771 enum class PluginFieldType : int32_t
772 {
773     kFLOAT16 = 0,
774     kFLOAT32 = 1,
775     kFLOAT64 = 2,
776     kINT8 = 3,

```

```

782     kINT16 = 4,
784     kINT32 = 5,
786     kCHAR = 6,
788     kDIMS = 7,
790     kUNKNOWN = 8
791 };
792
793 class PluginField
794 {
795 public:
796     AsciiChar const* name;
797     void const* data;
798     PluginFieldType type;
799     int32_t length;
800
801     PluginField(AsciiChar const* const name_ = nullptr, void const* const data_ = nullptr,
802                 PluginFieldType const type_ = PluginFieldType::kUNKNOWN, int32_t const length_ = 0) noexcept
803         : name(name_)
804         , data(data_)
805         , type(type_)
806         , length(length_)
807     {
808     }
809 };
810
811 struct PluginFieldCollection
812 {
813     int32_t nbFields;
814     PluginField const* fields;
815 };
816
817 class IPluginCreator
818 {
819 public:
820     virtual int32_t getTensorRTVersion() const noexcept
821     {
822         return NV_TENSORRT_VERSION;
823     }
824
825     virtual AsciiChar const* getPluginName() const noexcept = 0;
826
827     virtual AsciiChar const* getPluginVersion() const noexcept = 0;
828
829     virtual PluginFieldCollection const* getFieldNames() noexcept = 0;
830
831     virtual IPluginV2* createPlugin(AsciiChar const* name, PluginFieldCollection const* fc) noexcept = 0;
832
833     virtual IPluginV2* deserializePlugin(AsciiChar const* name, void const* serialData, size_t serialLength)
834         noexcept
835         = 0;
836
837     virtual void setPluginNamespace(AsciiChar const* pluginNamespace) noexcept = 0;
838
839     virtual AsciiChar const* getPluginNamespace() const noexcept = 0;
840
841     IPluginCreator() = default;
842     virtual ~IPluginCreator() = default;
843
844 protected:
845     // @cond SuppressDoxyWarnings
846     IPluginCreator(IPluginCreator const&) = default;
847     IPluginCreator(IPluginCreator&&) = default;
848     IPluginCreator& operator=(IPluginCreator const&) &= default;
849     IPluginCreator& operator=(IPluginCreator&&) &= default;
850     // @endcond
851 };
852
853 } // namespace nvinfer1
854 #endif // NV_INFER_RUNTIME_PLUGIN_H

```

10.21 NvInferSafeRuntime.h File Reference

```

#include "NvInferRuntimeBase.h"
#include "NvInferRuntimePlugin.h"

```

```
#include <cstddef>
#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::RuntimeErrorInformation](#)
Space to record information about runtime errors.
- class [nvinfer1::safe::IExecutionContext](#)
Functionally safe context for executing inference using an engine.
- class [nvinfer1::safe::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 must also have a corresponding [IPluginCreator](#) implementation.
- 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 {}`

Enumerations

- enum class [nvinfer1::safe::RuntimeErrorType](#) : uint64_t {
 [nvinfer1::safe::kNAN_CONSUMED](#) = 1ULL << 0 , [nvinfer1::safe::kINF_CONSUMED](#) = 1ULL << 1 ,
 [nvinfer1::safe::kGATHER_OOB](#) = 1ULL << 2 , [nvinfer1::safe::kSCATTER_OOB](#) = 1ULL << 3 ,
 [nvinfer1::safe::kSCATTER_RACE](#) = 1ULL << 4 }

Functions

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

10.21.1 Detailed Description

The functionality in this file is only supported in NVIDIA Drive(R) products.

10.21.2 Macro Definition Documentation

10.21.2.1 REGISTER_SAFE_TENSORRT_PLUGIN

```
#define REGISTER_SAFE_TENSORRT_PLUGIN(  
    name )    static nvinfer1::safe::PluginRegistrar<name> pluginRegistrar##name {}
```

10.22 NvInferSafeRuntime.h

[Go to the documentation of this file.](#)

```
1 /*  
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2023 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 "NvInferRuntimeBase.h"  
17 #include "NvInferRuntimePlugin.h"  
18 #include <cstdlib>  
19 #include <stdint>  
20  
25  
31 namespace nvinfer1  
32 {  
38     namespace safe  
39     {  
41         class ICudaEngine;  
43         class IExecutionContext;  
44  
45         class IRuntime  
46         {  
47         public:  
81             virtual ICudaEngine* deserializeCudaEngine(void const* const blob, std::size_t const size) noexcept = 0;  
82  
93             virtual void setGpuAllocator(IGpuAllocator* const allocator) noexcept = 0;  
94  
112            virtual void setErrorRecorder(IErrorRecorder* const recorder) noexcept = 0;  
113  
129            virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;  
130  
131            IRuntime() = default;  
132            virtual ~IRuntime() noexcept = default;  
133            IRuntime(IRuntime const&) = delete;  
134            IRuntime(IRuntime&&) = delete;  
135            IRuntime& operator=(IRuntime const&) & = delete;  
136            IRuntime& operator=(IRuntime&&) & = delete;  
137        };  
138
```



```

146 class ICudaEngine
147 {
148 public:
162     TRT_DEPRECATED virtual std::int32_t getNbBindings() const noexcept = 0;
163
186     TRT_DEPRECATED virtual std::int32_t getBindingIndex(AsciiChar const* const name) const noexcept = 0;
187
207     TRT_DEPRECATED virtual AsciiChar const* getBindingName(std::int32_t const bindingIndex) const noexcept =
    0;
208
225     TRT_DEPRECATED virtual bool bindingIsInput(std::int32_t const bindingIndex) const noexcept = 0;
226
242     TRT_DEPRECATED virtual Dims getBindingDimensions(std::int32_t const bindingIndex) const noexcept = 0;
243
259     TRT_DEPRECATED virtual DataType getBindingDataType(std::int32_t const bindingIndex) const noexcept = 0;
260
278     virtual IExecutionContext* createExecutionContext() noexcept = 0;
279
298     virtual IExecutionContext* createExecutionContextWithoutDeviceMemory() noexcept = 0;
299
313     virtual size_t getDeviceMemorySize() const noexcept = 0;
314
334     TRT_DEPRECATED virtual std::int32_t getBindingBytesPerComponent(std::int32_t const bindingIndex) const
    noexcept = 0;
335
355     TRT_DEPRECATED virtual std::int32_t getBindingComponentsPerElement(std::int32_t const bindingIndex) const
    noexcept = 0;
356
373     TRT_DEPRECATED virtual TensorFormat getBindingFormat(std::int32_t const bindingIndex) const noexcept = 0;
374
393     TRT_DEPRECATED virtual std::int32_t getBindingVectorizedDim(std::int32_t const bindingIndex) const
    noexcept = 0;
394
410     virtual AsciiChar const* getName() const noexcept = 0;
411
428     virtual void setErrorRecorder(IErrorRecorder* const recorder) noexcept = 0;
429
446     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
447
448     ICudaEngine() = default;
449     virtual ~ICudaEngine() noexcept = default;
450     ICudaEngine(ICudaEngine const&) = delete;
451     ICudaEngine(ICudaEngine&&) = delete;
452     ICudaEngine& operator=(ICudaEngine const&) &= delete;
453     ICudaEngine& operator=(ICudaEngine&&) &= delete;
454
471     virtual Dims getTensorShape(AsciiChar const* const tensorName) const noexcept = 0;
472
490     virtual DataType getTensorDataType(AsciiChar const* const tensorName) const noexcept = 0;
491
510     virtual TensorIOMode getTensorIOMode(AsciiChar const* const tensorName) const noexcept = 0;
511
533     virtual std::int32_t getTensorBytesPerComponent(AsciiChar const* const tensorName) const noexcept = 0;
534
555     virtual std::int32_t getTensorComponentsPerElement(AsciiChar const* const tensorName) const noexcept =
    0;
556
574     virtual TensorFormat getTensorFormat(AsciiChar const* const tensorName) const noexcept = 0;
575
596     virtual std::int32_t getTensorVectorizedDim(AsciiChar const* const tensorName) const noexcept = 0;
597
609     virtual std::int32_t getNbIOTensors() const noexcept = 0;
610
629     virtual AsciiChar const* getIOTensorName(std::int32_t const index) const noexcept = 0;
630 };
631
641 struct RuntimeErrorInformation
642 {
644     uint64_t bitMask;
645 };
646
647 enum class RuntimeErrorType : uint64_t
648 {
650     kNAN_CONSUMED = 1ULL << 0,
652     kINF_CONSUMED = 1ULL << 1,
654     kGATHER_OOB = 1ULL << 2,
656     kSCATTER_OOB = 1ULL << 3,
658     kSCATTER_RACE = 1ULL << 4,
659 };
660
674 class IExecutionContext

```

```

675 {
676 public:
686     virtual ICudaEngine const& getEngine() const noexcept = 0;
687
703     virtual void setName(AsciiChar const* const name) noexcept = 0;
704
717     virtual AsciiChar const* getName() const noexcept = 0;
718
740     virtual void setDeviceMemory(void* const memory) noexcept = 0;
741
758     TRT_DEPRECATED virtual Dims getStrides(std::int32_t const bindingIndex) const noexcept = 0;
759
777     virtual void setErrorRecorder(IErrorRecorder* const recorder) noexcept = 0;
778
794     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
795
828     TRT_DEPRECATED virtual bool enqueueV2(
829         void* const* bindings, cudaStream_t const stream, cudaEvent_t const* const inputConsumed)
noexcept = 0;
830
831     IExecutionContext() = default;
832     virtual ~IExecutionContext() noexcept = default;
833     IExecutionContext(IExecutionContext const&) = delete;
834     IExecutionContext(IExecutionContext&&) = delete;
835     IExecutionContext& operator=(IExecutionContext const&) &= delete;
836     IExecutionContext& operator=(IExecutionContext&&) &= delete;
837
858     virtual void setErrorBuffer(RuntimeErrorInformation* const buffer) noexcept = 0;
859
871     virtual RuntimeErrorInformation* getErrorBuffer() const noexcept = 0;
872
894     virtual Dims getTensorStrides(AsciiChar const* const tensorName) const noexcept = 0;
895
924     virtual bool setInputTensorAddress(AsciiChar const* const tensorName, void const* const data) noexcept =
0;
925
953     virtual bool setOutputTensorAddress(AsciiChar const* const tensorName, void* const data) noexcept = 0;
954
969     virtual bool setInputConsumedEvent(cudaEvent_t const event) noexcept = 0;
970
981     virtual cudaEvent_t getInputConsumedEvent() const noexcept = 0;
982
1001     virtual void const* getInputTensorAddress(AsciiChar const* const tensorName) const noexcept = 0;
1002
1021     virtual void* getOutputTensorAddress(AsciiChar const* const tensorName) const noexcept = 0;
1022
1042     virtual bool enqueueV3(cudaStream_t const stream) noexcept = 0;
1043 };
1044
1062
1063 class IPluginRegistry
1064 {
1065 public:
1089     virtual bool registerCreator(IPluginCreator& creator, AsciiChar const* const pluginNamespace) noexcept
= 0;
1090
1104     virtual IPluginCreator* const* getPluginCreatorList(int32_t* const numCreators) const noexcept = 0;
1105
1128     virtual IPluginCreator* getPluginCreator(AsciiChar const* const pluginName, AsciiChar const* const
pluginVersion,
1129         AsciiChar const* const pluginNamespace = "") noexcept = 0;
1130
1147     virtual void setErrorRecorder(IErrorRecorder* const recorder) noexcept = 0;
1148
1166     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
1167
1184     virtual bool deregisterCreator(IPluginCreator const& creator) noexcept = 0;
1185
1186     // @cond SuppressDoxyWarnings
1187     IPluginRegistry() = default;
1188     IPluginRegistry(IPluginRegistry const&) = delete;
1189     IPluginRegistry(IPluginRegistry&&) = delete;
1190     IPluginRegistry& operator=(IPluginRegistry const&) &= delete;
1191     IPluginRegistry& operator=(IPluginRegistry&&) &= delete;
1192     // @endcond
1193
1194 protected:
1195     virtual ~IPluginRegistry() noexcept = default;
1196 };
1197
1212 IRuntime* createInferRuntime(ILogger& logger) noexcept;

```

```

1213
1221 extern "C" TENSORRTAPI IPluginRegistry* getSafePluginRegistry() noexcept;
1222
1234 template <typename T>
1235 class PluginRegistrar
1236 {
1237 public:
1238     PluginRegistrar()
1239     {
1240         getSafePluginRegistry()->registerCreator(instance, "");
1241     }
1242
1243 private:
1244     T instance{};
1245 };
1246
1247 } // namespace safe
1248
1249 } // namespace nvinfer1
1250
1251 #define REGISTER_SAFE_TENSORRT_PLUGIN(name)
1252     \
1253     static nvinfer1::safe::PluginRegistrar<name> pluginRegistrar##name {}
1254 #endif // NV_INFER_SAFE_RUNTIME_H

```

10.23 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 13`
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 13`
Shared object library patch version number.
- `#define NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS 0`
An early access release.
- `#define NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE 1`
A release candidate.
- `#define NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY 2`
A final release.
- `#define NV_TENSORRT_RELEASE_TYPE NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY`
TensorRT release type.

10.23.1 Detailed Description

Defines the TensorRT version

10.23.2 Macro Definition Documentation

10.23.2.1 NV_TENSORRT_BUILD

```
#define NV_TENSORRT_BUILD 0
```

TensorRT build number.

10.23.2.2 NV_TENSORRT_LWS_MAJOR

```
#define NV_TENSORRT_LWS_MAJOR 0
```

TensorRT LWS major version.

10.23.2.3 NV_TENSORRT_LWS_MINOR

```
#define NV_TENSORRT_LWS_MINOR 0
```

TensorRT LWS minor version.

10.23.2.4 NV_TENSORRT_LWS_PATCH

```
#define NV_TENSORRT_LWS_PATCH 0
```

TensorRT LWS patch version.

10.23.2.5 NV_TENSORRT_MAJOR

```
#define NV_TENSORRT_MAJOR 8
```

TensorRT major version.

10.23.2.6 NV_TENSORRT_MINOR

```
#define NV_TENSORRT_MINOR 6
```

TensorRT minor version.

10.23.2.7 NV_TENSORRT_PATCH

```
#define NV_TENSORRT_PATCH 13
```

TensorRT patch version.

10.23.2.8 NV_TENSORRT_RELEASE_TYPE

```
#define NV_TENSORRT_RELEASE_TYPE NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY
```

TensorRT release type.

10.23.2.9 NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS

```
#define NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS 0
```

An early access release.

10.23.2.10 NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY

```
#define NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY 2
```

A final release.

10.23.2.11 NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE

```
#define NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE 1
```

A release candidate.

10.23.2.12 NV_TENSORRT_SONAME_MAJOR

```
#define NV_TENSORRT_SONAME_MAJOR 8
```

Shared object library major version number.

10.23.2.13 NV_TENSORRT_SONAME_MINOR

```
#define NV_TENSORRT_SONAME_MINOR 6
```

Shared object library minor version number.

10.23.2.14 NV_TENSORRT_SONAME_PATCH

```
#define NV_TENSORRT_SONAME_PATCH 13
```

Shared object library patch version number.

10.24 NvInferVersion.h

[Go to the documentation of this file.](#)

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2023 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_VERSION_H
14 #define NV_INFER_VERSION_H
15
16 #define NV_TENSORRT_MAJOR 8
17 #define NV_TENSORRT_MINOR 6
18 #define NV_TENSORRT_PATCH 13
19 #define NV_TENSORRT_BUILD 0
20
```

```

26 #define NV_TENSORRT_LWS_MAJOR 0
27 #define NV_TENSORRT_LWS_MINOR 0
28 #define NV_TENSORRT_LWS_PATCH 0
29
30 // This #define is deprecated in TensorRT 8.6 and will be removed in 10.0. Use NV_TENSORRT_MAJOR.
31 #define NV_TENSORRT_SONAME_MAJOR 8
32 // This #define is deprecated in TensorRT 8.6 and will be removed in 10.0. Use NV_TENSORRT_MINOR.
33 #define NV_TENSORRT_SONAME_MINOR 6
34 // This #define is deprecated in TensorRT 8.6 and will be removed in 10.0. Use NV_TENSORRT_PATCH.
35 #define NV_TENSORRT_SONAME_PATCH 13
36
37 #define NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS 0
38 #define NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE 1
39 #define NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY 2
40
41 #define NV_TENSORRT_RELEASE_TYPE NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY
42
43 #endif // NV_INFER_VERSION_H

```

10.25 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 ()`

10.25.1 Detailed Description

This is the API file for the Configuration Manager for ONNX Parser for Nvidia TensorRT.

10.26 NvOnnxConfig.h

[Go to the documentation of this file.](#)

```

1  /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2023 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     virtual void setModelDtype(const nvinfer1::DataType) noexcept = 0;
48     virtual nvinfer1::DataType getModelDtype() const noexcept = 0;
49     virtual char const* getModelFileName() const noexcept = 0;
50
51     virtual void setModelFileName(char const* onnxFilename) noexcept = 0;
52     virtual Verbosity getVerbosityLevel() const noexcept = 0;
53
54     virtual void addVerbosity() noexcept = 0;
55
56     virtual void reduceVerbosity() noexcept = 0;
57
58     virtual void setVerbosityLevel(Verbosity) noexcept = 0;
59
60     virtual char const* getTextFileName() const noexcept = 0;
61
62     virtual void setTextFileName(char const* textFileName) noexcept = 0;
63
64     virtual char const* getFullTextFileName() const noexcept = 0;
65
66     virtual void setFullTextFileName(char const* fullTextFileName) noexcept = 0;
67
68     virtual bool getPrintLayerInfo() const noexcept = 0;
69
70     virtual void setPrintLayerInfo(bool) noexcept = 0;
71
72     TRT_DEPRECATED virtual void destroy() noexcept = 0;
73
74 }; // class IOnnxConfig
75
76 TENSORRTAPI IOnnxConfig* createONNXConfig();
77
78 } // namespace nvonnxparser
79
80 #endif

```

10.27 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 }
 - 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 different possible supported input order.*
- 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.

10.27.1 Detailed Description

This is the API for the UFF Parser

10.27.2 Macro Definition Documentation

10.27.2.1 UFF_REQUIRED_VERSION_MAJOR

```
#define UFF_REQUIRED_VERSION_MAJOR 0
```

10.27.2.2 UFF_REQUIRED_VERSION_MINOR

```
#define UFF_REQUIRED_VERSION_MINOR 6
```

10.27.2.3 UFF_REQUIRED_VERSION_PATCH

```
#define UFF_REQUIRED_VERSION_PATCH 9
```

10.28 NvUffParser.h

[Go to the documentation of this file.](#)

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2023 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 }
```

```

62
72 class TENSORRTAPI FieldMap
73 {
74 public:
75     char const* name{};
76     void const* data{};
77     FieldType type{FieldType::kUNKNOWN};
78     int32_t length{1};
79
82     TRT_DEPRECATED FieldMap(char const* name, void const* data, FieldType const type, int32_t length = 1);
83
85     FieldMap() = default;
86 };
87
88 struct FieldCollection
89 {
90     int32_t nbFields;
91     FieldMap const* fields;
92 };
93
101 class IUffParser
102 {
103 public:
111     virtual bool registerInput(char const* inputName, nvinfer1::Dims inputDims, UffInputOrder inputOrder)
112         noexcept = 0;
118     virtual bool registerOutput(char const* outputName) noexcept = 0;
119
127     virtual bool parse(char const* file, nvinfer1::INetworkDefinition& network,
128         nvinfer1::DataType weightsType = nvinfer1::DataType::kFLOAT) noexcept = 0;
129
138     virtual bool parseBuffer(char const* buffer, std::size_t size, nvinfer1::INetworkDefinition& network,
139         nvinfer1::DataType weightsType = nvinfer1::DataType::kFLOAT) noexcept = 0;
140
144     TRT_DEPRECATED virtual void destroy() noexcept = 0;
145
149     virtual int32_t getUffRequiredVersionMajor() noexcept = 0;
150
154     virtual int32_t getUffRequiredVersionMinor() noexcept = 0;
155
159     virtual int32_t getUffRequiredVersionPatch() noexcept = 0;
160
164     virtual void setPluginNamespace(char const* libNamespace) noexcept = 0;
165
166     virtual ~IUffParser() noexcept = default;
167
168 public:
180     //
183     virtual void setErrorRecorder(nvinfer1::IErrorRecorder* recorder) noexcept = 0;
184
195     virtual nvinfer1::IErrorRecorder* getErrorRecorder() const noexcept = 0;
196 };
197
208 TENSORRTAPI IUffParser* createUffParser() noexcept;
209
215 TENSORRTAPI void shutdownProtobufLibrary(void) noexcept;
216
217 } // namespace nvuffparser
218
223 extern "C" TENSORRTAPI void* createNvUffParser.INTERNAL() noexcept;
224
225 #endif /* !NV_UFF_PARSER_H */

```

10.29 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.*

10.29.1 Detailed Description

This file includes various utility functions

10.30 NvUtils.h

[Go to the documentation of this file.](#)

```

1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2023 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 namespace nvinfer1
19 {
20     namespace utils
21     {
22
23         TRT_DEPRECATED TENSORRTAPI bool reshapeWeights(
24             Weights const& input, int32_t const* shape, int32_t const* shapeOrder, void* data, int32_t nbDims)
25             noexcept;
26
27         TRT_DEPRECATED TENSORRTAPI bool reorderSubBuffers(
28             void* input, int32_t const* order, int32_t num, int32_t size) noexcept;
29
30         TRT_DEPRECATED TENSORRTAPI bool transposeSubBuffers(
31             void* input, DataType type, int32_t num, int32_t height, int32_t width) noexcept;
32     }
33 } // namespace nvinfer1
34
35 #endif // NV_UTILS_H

```

10.31 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.
- using [nvonnxparser::OnnxParserFlags](#) = `uint32_t`
Represents one or more [OnnxParserFlag](#) values using binary OR operations, e.g., `1U << OnnxParserFlag::kNATIVE\_INSTANCENORM`.

Enumerations

- enum class [nvonnxparser::ErrorCode](#) : `int` {
[nvonnxparser::kSUCCESS](#) = 0, [nvonnxparser::kINTERNAL_ERROR](#) = 1, [nvonnxparser::kMEM_ALLOC_FAILED](#) = 2, [nvonnxparser::kMODEL_DESERIALIZE_FAILED](#) = 3, [nvonnxparser::kINVALID_VALUE](#) = 4, [nvonnxparser::kINVALID_GRAPH](#) = 5, [nvonnxparser::kINVALID_NODE](#) = 6, [nvonnxparser::kUNSUPPORTED_GRAPH](#) = 7, [nvonnxparser::kUNSUPPORTED_NODE](#) = 8 }
The type of error that the parser may return.
- enum class [nvonnxparser::OnnxParserFlag](#) : `int32_t` { [nvonnxparser::kNATIVE_INSTANCENORM](#) = 0 }

Functions

- `template<typename T>`
`constexpr int32_t nvonnxparser::EnumMax ()`
- `template<> constexpr int32_t nvonnxparser::EnumMax< ErrorCode > ()`
- `template<> constexpr int32_t nvonnxparser::EnumMax< OnnxParserFlag > ()`
- [TENSORRTAPI](#) `void * createNvOnnxParser\_INTERNAL (void *network, void *logger, int version)`
- [TENSORRTAPI](#) `int getNvOnnxParserVersion ()`

10.31.1 Detailed Description

This is the API for the ONNX Parser

10.31.2 Macro Definition Documentation

10.31.2.1 NV_ONNX_PARSER_MAJOR

```
#define NV_ONNX_PARSER_MAJOR 0
```

10.31.2.2 NV_ONNX_PARSER_MINOR

```
#define NV_ONNX_PARSER_MINOR 1
```

10.31.2.3 NV_ONNX_PARSER_PATCH

```
#define NV_ONNX_PARSER_PATCH 0
```

10.31.3 Typedef Documentation

10.31.3.1 SubGraph_t

[SubGraph_t](#)

The data structure containing the parsing capability of a set of nodes in an ONNX graph.

10.31.3.2 SubGraphCollection_t

[SubGraphCollection_t](#)

The data structure containing all SubGraph_t partitioned out of an ONNX graph.

10.31.4 Function Documentation

10.31.4.1 createNvOnnxParser_INTERNAL()

```
TENSORRTAPI void * createNvOnnxParser_INTERNAL (
    void * network,
    void * logger,
    int version )
```

10.31.4.2 getNvOnnxParserVersion()

```
TENSORRTAPI int getNvOnnxParserVersion ( )
```

10.32 NvOnnxParser.h

[Go to the documentation of this file.](#)

```
1 /*
2  * Copyright (c) 1993-2023, 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
34
35
36 #define NV_ONNX_PARSER_MAJOR 0
37 #define NV_ONNX_PARSER_MINOR 1
38 #define NV_ONNX_PARSER_PATCH 0
39
40 static constexpr int32_t NV_ONNX_PARSER_VERSION
41     = ((NV_ONNX_PARSER_MAJOR * 10000) + (NV_ONNX_PARSER_MINOR * 100) + NV_ONNX_PARSER_PATCH);
42
43
44
45
46
47
48
49 typedef std::pair<std::vector<size_t>, bool> SubGraph_t;
50
51
52
53
54
55
56
57 typedef std::vector<SubGraph_t> SubGraphCollection_t;
58
59
60
61
62
63
64 namespace nvonnxparser
```

```

65 {
66
67 template <typename T>
68 constexpr inline int32_t EnumMax();
69
70 enum class ErrorCode : int
71 {
72     kSUCCESS = 0,
73     kINTERNAL_ERROR = 1,
74     kMEM_ALLOC_FAILED = 2,
75     kMODEL_DESERIALIZE_FAILED = 3,
76     kINVALID_VALUE = 4,
77     kINVALID_GRAPH = 5,
78     kINVALID_NODE = 6,
79     kUNSUPPORTED_GRAPH = 7,
80     kUNSUPPORTED_NODE = 8
81 };
82
83 template <>
84 constexpr inline int32_t EnumMax<ErrorCode>()
85 {
86     return 9;
87 }
88
89 using OnnxParserFlags = uint32_t;
90
91 enum class OnnxParserFlag : int32_t
92 {
93     kNATIVE_INSTANCENORM = 0
94 };
95
96 template <>
97 constexpr inline int32_t EnumMax<OnnxParserFlag>()
98 {
99     return 1;
100 }
101
102 class IParserError
103 {
104 public:
105     virtual ErrorCode code() const = 0;
106     virtual const char* desc() const = 0;
107     virtual const char* file() const = 0;
108     virtual int line() const = 0;
109     virtual const char* func() const = 0;
110     virtual int node() const = 0;
111 protected:
112     virtual ~IParserError() {}
113 };
114
115 class IParser
116 {
117 public:
118     virtual bool parse(
119         void const* serialized_onnx_model, size_t serialized_onnx_model_size, const char* model_path = nullptr)
120         = 0;
121
122     virtual bool parseFromFile(const char* onnx_model_file, int verbosity) = 0;
123
124     virtual bool supportsModel(void const* serialized_onnx_model, size_t serialized_onnx_model_size,
125         SubGraphCollection_t& sub_graph_collection, const char* model_path = nullptr)
126         = 0;
127
128     virtual bool parseWithWeightDescriptors(void const* serialized_onnx_model, size_t
129         serialized_onnx_model_size) = 0;
130
131     virtual bool supportsOperator(const char* op_name) const = 0;
132
133     TRT_DEPRECATED virtual void destroy() = 0;
134
135     virtual int getNbErrors() const = 0;
136
137     virtual IParserError const* getError(int index) const = 0;
138
139     virtual void clearErrors() = 0;
140
141     virtual ~IParser() noexcept = default;
142
143     virtual char const* const* getUsedVCPluginLibraries(int64_t& nb_plugin_libs) const noexcept = 0;
144
145 };

```



```

301     virtual void setFlags(OnnxParserFlags onnxParserFlags) noexcept = 0;
302
310     virtual OnnxParserFlags getFlags() const noexcept = 0;
311
319     virtual void clearFlag(OnnxParserFlag onnxParserFlag) noexcept = 0;
320
328     virtual void setFlag(OnnxParserFlag onnxParserFlag) noexcept = 0;
329
337     virtual bool getFlag(OnnxParserFlag onnxParserFlag) const noexcept = 0;
338 };
339
340 } // namespace nvonnxparser
341
342 extern "C" TENSORRTAPI void* createNvOnnxParser_INTERNAL(void* network, void* logger, int version);
343 extern "C" TENSORRTAPI int getNvOnnxParserVersion();
344
345 namespace nvonnxparser
346 {
347
348     namespace
349     {
350
366     inline IParser* createParser(nvinfer1::INetworkDefinition& network, nvinfer1::ILogger& logger)
367     {
368         return static_cast<IParser*>(createNvOnnxParser_INTERNAL(&network, &logger, NV_ONNX_PARSER_VERSION));
369     }
370
371 } // namespace
372
373 } // namespace nvonnxparser
374
375 #endif // NV_ONNX_PARSER_H

```

Index

- ~IActivationLayer
 - nvinfer1::IActivationLayer, [130](#)
- ~IAlgorithm
 - nvinfer1::IAlgorithm, [134](#)
- ~IAlgorithmContext
 - nvinfer1::IAlgorithmContext, [137](#)
- ~IAlgorithmIOInfo
 - nvinfer1::IAlgorithmIOInfo, [139](#)
- ~IAlgorithmSelector
 - nvinfer1::IAlgorithmSelector, [141](#)
- ~IAlgorithmVariant
 - nvinfer1::IAlgorithmVariant, [144](#)
- ~IAssertionLayer
 - nvinfer1::IAssertionLayer, [146](#)
- ~IBinaryProtoBlob
 - nvcaffeparser1::IBinaryProtoBlob, [148](#)
- ~IBlobNameToTensor
 - nvcaffeparser1::IBlobNameToTensor, [149](#)
- ~IBuilder
 - nvinfer1::IBuilder, [152](#)
- ~IBuilderConfig
 - nvinfer1::IBuilderConfig, [163](#)
- ~ICaffeParser
 - nvcaffeparser1::ICaffeParser, [186](#)
- ~ICastLayer
 - nvinfer1::ICastLayer, [191](#)
- ~IConcatenationLayer
 - nvinfer1::IConcatenationLayer, [193](#)
- ~IConditionLayer
 - nvinfer1::IConditionLayer, [195](#)
- ~IConsistencyChecker
 - nvinfer1::consistency::IConsistencyChecker, [196](#)
- ~IConstantLayer
 - nvinfer1::IConstantLayer, [199](#)
- ~IConvolutionLayer
 - nvinfer1::IConvolutionLayer, [202](#)
- ~ICudaEngine
 - nvinfer1::ICudaEngine, [216](#)
 - nvinfer1::safe::ICudaEngine, [244](#)
- ~IDeconvolutionLayer
 - nvinfer1::IDeconvolutionLayer, [263](#)
- ~IDequantizeLayer
 - nvinfer1::IDequantizeLayer, [274](#)
- ~IDimensionExpr
 - nvinfer1::IDimensionExpr, [276](#)
- ~IEinsumLayer
 - nvinfer1::IEinsumLayer, [278](#)
- ~IElementWiseLayer
 - nvinfer1::IElementWiseLayer, [280](#)
- ~IEngineInspector
 - nvinfer1::IEngineInspector, [282](#)
- ~IErrorRecorder
 - nvinfer1::IErrorRecorder, [288](#)
- ~IExecutionContext
 - nvinfer1::IExecutionContext, [296](#)
 - nvinfer1::safe::IExecutionContext, [325](#)
- ~IExprBuilder
 - nvinfer1::IExprBuilder, [339](#)
- ~IFillLayer
 - nvinfer1::IFillLayer, [341](#)
- ~IFullyConnectedLayer
 - nvinfer1::IFullyConnectedLayer, [348](#)
- ~IGatherLayer
 - nvinfer1::IGatherLayer, [353](#)
- ~IGpuAllocator
 - nvinfer1::IGpuAllocator, [356](#)
- ~IGridSampleLayer
 - nvinfer1::IGridSampleLayer, [361](#)
- ~IHostMemory
 - nvinfer1::IHostMemory, [365](#)
- ~IIdentityLayer
 - nvinfer1::IIdentityLayer, [367](#)
- ~IIIfConditional
 - nvinfer1::IIIfConditional, [369](#)
- ~IIIfConditionalBoundaryLayer
 - nvinfer1::IIIfConditionalBoundaryLayer, [372](#)
- ~IIIfConditionalInputLayer
 - nvinfer1::IIIfConditionalInputLayer, [373](#)
- ~IIIfConditionalOutputLayer
 - nvinfer1::IIIfConditionalOutputLayer, [374](#)
- ~IInt8Calibrator
 - nvinfer1::IInt8Calibrator, [376](#)
- ~IInt8EntropyCalibrator
 - nvinfer1::IInt8EntropyCalibrator, [379](#)
- ~IInt8EntropyCalibrator2
 - nvinfer1::IInt8EntropyCalibrator2, [380](#)
- ~IInt8LegacyCalibrator
 - nvinfer1::IInt8LegacyCalibrator, [381](#)
- ~IInt8MinMaxCalibrator
 - nvinfer1::IInt8MinMaxCalibrator, [383](#)

- ~IteratorLayer
 - nvinfer1::IteratorLayer, [385](#)
- ~ILRNLayr
 - nvinfer1::ILRNLayr, [408](#)
- ~ILayer
 - nvinfer1::ILayer, [388](#)
- ~ILogger
 - nvinfer1::ILogger, [397](#)
- ~ILoggerFinder
 - nvinfer1::ILoggerFinder, [399](#)
- ~ILoop
 - nvinfer1::ILoop, [400](#)
- ~ILoopBoundaryLayer
 - nvinfer1::ILoopBoundaryLayer, [403](#)
- ~ILoopOutputLayer
 - nvinfer1::ILoopOutputLayer, [405](#)
- ~IMatrixMultiplyLayer
 - nvinfer1::IMatrixMultiplyLayer, [413](#)
- ~INMSLayer
 - nvinfer1::INMSLayer, [461](#)
- ~INetworkDefinition
 - nvinfer1::INetworkDefinition, [418](#)
- ~INoCopy
 - nvinfer1::INoCopy, [465](#)
- ~INonZeroLayer
 - nvinfer1::INonZeroLayer, [467](#)
- ~INormalizationLayer
 - nvinfer1::INormalizationLayer, [468](#)
- ~IONnxConfig
 - nvonnxparser::IONnxConfig, [475](#)
- ~IOptimizationProfile
 - nvinfer1::IOptimizationProfile, [482](#)
- ~IOutputAllocator
 - nvinfer1::IOutputAllocator, [487](#)
- ~IPaddingLayer
 - nvinfer1::IPaddingLayer, [490](#)
- ~IParametricReLULayer
 - nvinfer1::IParametricReLULayer, [494](#)
- ~IParser
 - nvonnxparser::IParser, [496](#)
- ~IParserError
 - nvonnxparser::IParserError, [502](#)
- ~IPluginChecker
 - nvinfer1::consistency::IPluginChecker, [505](#)
- ~IPluginCreator
 - nvinfer1::IPluginCreator, [507](#)
- ~IPluginFactoryV2
 - nvcaffeparser1::IPluginFactoryV2, [512](#)
- ~IPluginRegistry
 - nvinfer1::IPluginRegistry, [515](#)
 - nvinfer1::safe::IPluginRegistry, [520](#)
- ~IPluginV2DynamicExt
 - nvinfer1::IPluginV2DynamicExt, [538](#)
- ~IPluginV2Ext
 - nvinfer1::IPluginV2Ext, [544](#)
- ~IPluginV2Layer
 - nvinfer1::IPluginV2Layer, [555](#)
- ~IPoolingLayer
 - nvinfer1::IPoolingLayer, [558](#)
- ~IProfiler
 - nvinfer1::IProfiler, [566](#)
- ~IQuantizeLayer
 - nvinfer1::IQuantizeLayer, [569](#)
- ~IRNNv2Layer
 - nvinfer1::IRNNv2Layer, [600](#)
- ~IRaggedSoftMaxLayer
 - nvinfer1::IRaggedSoftMaxLayer, [571](#)
- ~IRecurrenceLayer
 - nvinfer1::IRecurrenceLayer, [573](#)
- ~IReduceLayer
 - nvinfer1::IReduceLayer, [575](#)
- ~IRefitter
 - nvinfer1::IRefitter, [578](#)
- ~IResizeLayer
 - nvinfer1::IResizeLayer, [588](#)
- ~IReverseSequenceLayer
 - nvinfer1::IReverseSequenceLayer, [597](#)
- ~IRuntime
 - nvinfer1::IRuntime, [609](#)
 - nvinfer1::safe::IRuntime, [618](#)
- ~IScaleLayer
 - nvinfer1::IScaleLayer, [624](#)
- ~IScatterLayer
 - nvinfer1::IScatterLayer, [629](#)
- ~ISelectLayer
 - nvinfer1::ISelectLayer, [631](#)
- ~IShapeLayer
 - nvinfer1::IShapeLayer, [632](#)
- ~IShuffleLayer
 - nvinfer1::IShuffleLayer, [634](#)
- ~ISliceLayer
 - nvinfer1::ISliceLayer, [640](#)
- ~ISoftMaxLayer
 - nvinfer1::ISoftMaxLayer, [645](#)
- ~ITensor
 - nvinfer1::ITensor, [648](#)
- ~ITimingCache
 - nvinfer1::ITimingCache, [659](#)
- ~ITopKLayer
 - nvinfer1::ITopKLayer, [662](#)
- ~ITripLimitLayer
 - nvinfer1::ITripLimitLayer, [665](#)
- ~IUffParser
 - nvuffparser::IUffParser, [666](#)
- ~IUnaryLayer
 - nvinfer1::IUnaryLayer, [671](#)
- ActivationType

- [nvinfer1](#), [43](#)
- [addActivation](#)
 - [nvinfer1::INetworkDefinition](#), [419](#)
- [addAssertion](#)
 - [nvinfer1::INetworkDefinition](#), [419](#)
- [addCast](#)
 - [nvinfer1::INetworkDefinition](#), [420](#)
- [addConcatenation](#)
 - [nvinfer1::INetworkDefinition](#), [420](#)
- [addConstant](#)
 - [nvinfer1::INetworkDefinition](#), [421](#)
- [addConvolution](#)
 - [nvinfer1::INetworkDefinition](#), [421](#)
- [addConvolutionNd](#)
 - [nvinfer1::INetworkDefinition](#), [422](#)
- [addDeconvolution](#)
 - [nvinfer1::INetworkDefinition](#), [423](#)
- [addDeconvolutionNd](#)
 - [nvinfer1::INetworkDefinition](#), [424](#)
- [addDequantize](#)
 - [nvinfer1::INetworkDefinition](#), [425](#)
- [addEinsum](#)
 - [nvinfer1::INetworkDefinition](#), [425](#)
- [addElementWise](#)
 - [nvinfer1::INetworkDefinition](#), [426](#)
- [addFill](#)
 - [nvinfer1::INetworkDefinition](#), [426](#)
- [addFullyConnected](#)
 - [nvinfer1::INetworkDefinition](#), [427](#)
- [addGather](#)
 - [nvinfer1::INetworkDefinition](#), [428](#)
- [addGatherV2](#)
 - [nvinfer1::INetworkDefinition](#), [428](#)
- [addGridSample](#)
 - [nvinfer1::INetworkDefinition](#), [429](#)
- [addIdentity](#)
 - [nvinfer1::INetworkDefinition](#), [429](#)
- [addIfConditional](#)
 - [nvinfer1::INetworkDefinition](#), [430](#)
- [addInput](#)
 - [nvinfer1::IIfConditional](#), [369](#)
 - [nvinfer1::INetworkDefinition](#), [430](#)
- [addIterator](#)
 - [nvinfer1::ILoop](#), [401](#)
- [addLoop](#)
 - [nvinfer1::INetworkDefinition](#), [431](#)
- [addLoopOutput](#)
 - [nvinfer1::ILoop](#), [401](#)
- [addLRN](#)
 - [nvinfer1::INetworkDefinition](#), [431](#)
- [addMatrixMultiply](#)
 - [nvinfer1::INetworkDefinition](#), [432](#)
- [addNMS](#)
 - [nvinfer1::INetworkDefinition](#), [433](#)
- [addNonZero](#)
 - [nvinfer1::INetworkDefinition](#), [433](#)
- [addNormalization](#)
 - [nvinfer1::INetworkDefinition](#), [434](#)
- [addOneHot](#)
 - [nvinfer1::INetworkDefinition](#), [434](#)
- [addOptimizationProfile](#)
 - [nvinfer1::IBuilderConfig](#), [163](#)
- [addOutput](#)
 - [nvinfer1::IIfConditional](#), [369](#)
- [addPadding](#)
 - [nvinfer1::INetworkDefinition](#), [435](#)
- [addPaddingNd](#)
 - [nvinfer1::INetworkDefinition](#), [436](#)
- [addParametricReLU](#)
 - [nvinfer1::INetworkDefinition](#), [436](#)
- [addPluginV2](#)
 - [nvinfer1::INetworkDefinition](#), [437](#)
- [addPooling](#)
 - [nvinfer1::INetworkDefinition](#), [437](#)
- [addPoolingNd](#)
 - [nvinfer1::INetworkDefinition](#), [438](#)
- [addQuantize](#)
 - [nvinfer1::INetworkDefinition](#), [439](#)
- [addRaggedSoftMax](#)
 - [nvinfer1::INetworkDefinition](#), [439](#)
- [addRecurrence](#)
 - [nvinfer1::ILoop](#), [401](#)
- [addReduce](#)
 - [nvinfer1::INetworkDefinition](#), [440](#)
- [addResize](#)
 - [nvinfer1::INetworkDefinition](#), [441](#)
- [addReverseSequence](#)
 - [nvinfer1::INetworkDefinition](#), [441](#)
- [addRNNv2](#)
 - [nvinfer1::INetworkDefinition](#), [442](#)
- [addScale](#)
 - [nvinfer1::INetworkDefinition](#), [443](#)
- [addScaleNd](#)
 - [nvinfer1::INetworkDefinition](#), [444](#)
- [addScatter](#)
 - [nvinfer1::INetworkDefinition](#), [445](#)
- [addSelect](#)
 - [nvinfer1::INetworkDefinition](#), [446](#)
- [addShape](#)
 - [nvinfer1::INetworkDefinition](#), [447](#)
- [addShuffle](#)
 - [nvinfer1::INetworkDefinition](#), [447](#)
- [addSlice](#)
 - [nvinfer1::INetworkDefinition](#), [448](#)
- [addSoftMax](#)
 - [nvinfer1::INetworkDefinition](#), [448](#)
- [addTopK](#)
 - [nvinfer1::INetworkDefinition](#), [449](#)

- addTripLimit
 - nvinfer1::ILoop, [401](#)
- addUnary
 - nvinfer1::INetworkDefinition, [450](#)
- addVerbosity
 - nvonnxparser::IOnnxConfig, [475](#)
- allInputDimensionsSpecified
 - nvinfer1::IExecutionContext, [296](#)
- allInputShapesSpecified
 - nvinfer1::IExecutionContext, [297](#)
- allocate
 - nvinfer1::IGpuAllocator, [357](#)
- AllocatorFlag
 - nvinfer1, [43](#)
- AllocatorFlags
 - nvinfer1, [40](#)
- anchorsRatioCount
 - nvinfer1::plugin::RPROIParams, [687](#)
- anchorsScaleCount
 - nvinfer1::plugin::RPROIParams, [687](#)
- AsciiChar
 - nvinfer1, [40](#)
- aspectRatios
 - nvinfer1::plugin::GridAnchorParameters, [128](#)
 - nvinfer1::plugin::PriorBoxParameters, [682](#)
- attachToContext
 - nvinfer1::IPluginV2Ext, [545](#)
- backgroundLabelId
 - nvinfer1::plugin::DetectionOutputParameters, [98](#)
 - nvinfer1::plugin::NMSParameters, [673](#)
- bindingIsInput
 - nvinfer1::ICudaEngine, [217](#)
 - nvinfer1::safe::ICudaEngine, [244](#)
- bitMask
 - nvinfer1::safe::RuntimeErrorInformation, [689](#)
- BoundingBoxFormat
 - nvinfer1, [44](#)
- buildEngineWithConfig
 - nvinfer1::IBuilder, [152](#)
- BuilderFlag
 - nvinfer1, [44](#)
- BuilderFlags
 - nvinfer1, [40](#)
- buildSerializedNetwork
 - nvinfer1::IBuilder, [152](#)
- CalibrationAlgoType
 - nvinfer1, [46](#)
- canBroadcastInputAcrossBatch
 - nvinfer1::IPluginV2Ext, [545](#)
- canRunOnDLA
 - nvinfer1::IBuilderConfig, [164](#)
- CENTER_SIZE
 - nvinfer1::plugin, [85](#)
- char_t
 - nvinfer1, [40](#)
- child
 - nvinfer1::plugin::softmaxTree, [690](#)
- classes
 - nvinfer1::plugin::RegionParameters, [686](#)
- clear
 - nvinfer1::IErrorRecorder, [288](#)
- clearErrors
 - nvonnxparser::IParser, [496](#)
- clearFlag
 - nvinfer1::IBuilderConfig, [164](#)
 - nvonnxparser::IParser, [496](#)
- clearQuantizationFlag
 - nvinfer1::IBuilderConfig, [164](#)
- clip
 - nvinfer1::plugin::PriorBoxParameters, [682](#)
- clone
 - nvinfer1::IPluginV2, [526](#)
 - nvinfer1::IPluginV2DynamicExt, [539](#)
 - nvinfer1::IPluginV2Ext, [546](#)
- code
 - nvonnxparser::IParserError, [503](#)
- codeType
 - nvinfer1::plugin::DetectionOutputParameters, [98](#)
- CodeTypeSSD
 - nvinfer1::plugin, [84](#)
- combine
 - nvinfer1::ITimingCache, [659](#)
- confidenceThreshold
 - nvinfer1::plugin::DetectionOutputParameters, [98](#)
- configurePlugin
 - nvinfer1::IPluginV2DynamicExt, [539](#)
 - nvinfer1::IPluginV2Ext, [547](#)
 - nvinfer1::IPluginV2IOExt, [552](#)
- configureWithFormat
 - nvinfer1::IPluginV2, [527](#)
 - nvinfer1::IPluginV2Ext, [548](#)
- confSigmoid
 - nvinfer1::plugin::DetectionOutputParameters, [98](#)
- constant
 - nvinfer1::IExprBuilder, [339](#)
- coords
 - nvinfer1::plugin::RegionParameters, [686](#)
- CORNER
 - nvinfer1::plugin, [85](#)
- CORNER_SIZE
 - nvinfer1::plugin, [85](#)
- count
 - nvinfer1::Weights, [692](#)
- createAnchorGeneratorPlugin
 - NvInferPlugin.h, [753](#)
- createBatchedNMSPlugin
 - NvInferPlugin.h, [754](#)

- createBuilderConfig
 - nvinfer1::IBuilder, [153](#)
- createCaffeParser
 - nvcaffeparser1, [29](#)
- createConsistencyChecker_INTERNAL
 - NvInferConsistency.h, [748](#)
- createEngineInspector
 - nvinfer1::ICudaEngine, [217](#)
- createExecutionContext
 - nvinfer1::ICudaEngine, [217](#)
 - nvinfer1::safe::ICudaEngine, [245](#)
- createExecutionContextWithoutDeviceMemory
 - nvinfer1::ICudaEngine, [218](#)
 - nvinfer1::safe::ICudaEngine, [245](#)
- createInferRuntime
 - nvinfer1::safe, [86](#)
- createInstanceNormalizationPlugin
 - NvInferPlugin.h, [754](#)
- createNetworkV2
 - nvinfer1::IBuilder, [153](#)
- createNMSPlugin
 - NvInferPlugin.h, [755](#)
- createNormalizePlugin
 - NvInferPlugin.h, [755](#)
- createNvOnnxParser_INTERNAL
 - NvOnnxParser.h, [812](#)
- createONNXConfig
 - nvonnxparser, [92](#)
- createOptimizationProfile
 - nvinfer1::IBuilder, [154](#)
- createPlugin
 - nvcaffeparser1::IPluginFactoryV2, [512](#)
 - nvinfer1::IPluginCreator, [508](#)
- createPriorBoxPlugin
 - NvInferPlugin.h, [756](#)
- createRegionPlugin
 - NvInferPlugin.h, [756](#)
- createReorgPlugin
 - NvInferPlugin.h, [756](#)
- createRPNROIPlugin
 - NvInferPlugin.h, [757](#)
- createSplitPlugin
 - NvInferPlugin.h, [757](#)
- createTimingCache
 - nvinfer1::IBuilderConfig, [164](#)
- createUffParser
 - nvuffparser, [95](#)
- d
 - nvinfer1::Dims32, [104](#)
 - nvinfer1::DimsExprs, [106](#)
- data
 - nvinfer1::IHostMemory, [365](#)
 - nvinfer1::plugin::Quadruple, [685](#)
 - nvinfer1::PluginField, [676](#)
 - nvuffparser::FieldMap, [126](#)
- DataType
 - nvinfer1, [46](#)
- deallocate
 - nvinfer1::IGpuAllocator, [358](#)
- decRefCount
 - nvinfer1::IErrorRecorder, [288](#)
- deregisterCreator
 - nvinfer1::IPluginRegistry, [515](#)
 - nvinfer1::safe::IPluginRegistry, [521](#)
- deregisterLibrary
 - nvinfer1::IPluginRegistry, [515](#)
- desc
 - nvinfer1::DynamicPluginTensorDesc, [110](#)
 - nvonnxparser::IParserError, [503](#)
- deserializeCudaEngine
 - nvinfer1::IRuntime, [609](#)
 - nvinfer1::safe::IRuntime, [619](#)
- deserializePlugin
 - nvinfer1::IPluginCreator, [508](#)
- destroy
 - nvcaffeparser1::IBinaryProtoBlob, [148](#)
 - nvcaffeparser1::ICaffeParser, [186](#)
 - nvinfer1::IBuilder, [154](#)
 - nvinfer1::IBuilderConfig, [165](#)
 - nvinfer1::ICudaEngine, [218](#)
 - nvinfer1::IExecutionContext, [297](#)
 - nvinfer1::IHostMemory, [365](#)
 - nvinfer1::INetworkDefinition, [450](#)
 - nvinfer1::IPluginV2, [528](#)
 - nvinfer1::IRefitter, [578](#)
 - nvinfer1::IRuntime, [610](#)
 - nvonnxparser::IONnxConfig, [475](#)
 - nvonnxparser::IParser, [496](#)
 - nvuffparser::IUffParser, [667](#)
- detachFromContext
 - nvinfer1::IPluginV2Ext, [548](#)
- DeviceType
 - nvinfer1, [47](#)
- DimensionOperation
 - nvinfer1, [47](#)
- Dims, [100](#)
 - nvinfer1, [40](#)
- dims
 - nvinfer1::PluginTensorDesc, [680](#)
- Dims2
 - nvinfer1::Dims2, [101](#)
- Dims3
 - nvinfer1::Dims3, [102](#), [103](#)
- Dims4
 - nvinfer1::Dims4, [105](#)
- DimsHW
 - nvinfer1::DimsHW, [108](#)

- dynamicRangeIsSet
 - nvinfer1::ITensor, [649](#)
- ElementWiseOperation
 - nvinfer1, [48](#)
- EngineCapability
 - nvinfer1, [49](#)
- enqueue
 - nvinfer1::IExecutionContext, [298](#)
 - nvinfer1::IPluginV2, [528](#)
 - nvinfer1::IPluginV2DynamicExt, [540](#)
- enqueueV2
 - nvinfer1::IExecutionContext, [298](#)
 - nvinfer1::safe::IExecutionContext, [325](#)
- enqueueV3
 - nvinfer1::IExecutionContext, [299](#)
 - nvinfer1::safe::IExecutionContext, [326](#)
- EnumMax
 - nvinfer1, [74](#)
 - nvonnxparser, [92](#)
- EnumMax< BoundingBoxFormat >
 - nvinfer1, [75](#)
- EnumMax< BuilderFlag >
 - nvinfer1, [75](#)
- EnumMax< CalibrationAlgoType >
 - nvinfer1, [75](#)
- EnumMax< DeviceType >
 - nvinfer1, [75](#)
- EnumMax< DimensionOperation >
 - nvinfer1, [76](#)
- EnumMax< ErrorCode >
 - nvonnxparser, [92](#)
- EnumMax< FillOperation >
 - nvinfer1, [76](#)
- EnumMax< GatherMode >
 - nvinfer1, [76](#)
- EnumMax< LayerInformationFormat >
 - nvinfer1, [76](#)
- EnumMax< LayerType >
 - nvinfer1, [77](#)
- EnumMax< LoopOutput >
 - nvinfer1, [77](#)
- EnumMax< MatrixOperation >
 - nvinfer1, [77](#)
- EnumMax< MemoryPoolType >
 - nvinfer1, [77](#)
- EnumMax< NetworkDefinitionCreationFlag >
 - nvinfer1, [78](#)
- EnumMax< OnnxParserFlag >
 - nvonnxparser, [92](#)
- EnumMax< OptProfileSelector >
 - nvinfer1, [78](#)
- EnumMax< ProfilingVerbosity >
 - nvinfer1, [78](#)
- EnumMax< QuantizationFlag >
 - nvinfer1, [78](#)
- EnumMax< ReduceOperation >
 - nvinfer1, [79](#)
- EnumMax< RNNDirection >
 - nvinfer1, [79](#)
- EnumMax< RNNGateType >
 - nvinfer1, [79](#)
- EnumMax< RNNInputMode >
 - nvinfer1, [79](#)
- EnumMax< RNNOperation >
 - nvinfer1, [80](#)
- EnumMax< SampleMode >
 - nvinfer1, [80](#)
- EnumMax< ScaleMode >
 - nvinfer1, [80](#)
- EnumMax< ScatterMode >
 - nvinfer1, [80](#)
- EnumMax< TacticSource >
 - nvinfer1, [81](#)
- EnumMax< TempfileControlFlag >
 - nvinfer1, [81](#)
- EnumMax< TopKOperation >
 - nvinfer1, [81](#)
- EnumMax< TripLimit >
 - nvinfer1, [81](#)
- EnumMax< UnaryOperation >
 - nvinfer1, [82](#)
- EnumMax< WeightsRole >
 - nvinfer1, [82](#)
- ErrorCode
 - nvinfer1, [49](#)
 - nvonnxparser, [91](#)
- ErrorDesc
 - nvinfer1::IErrorRecorder, [287](#)
- execute
 - nvinfer1::IExecutionContext, [300](#)
- executeV2
 - nvinfer1::IExecutionContext, [300](#)
- featureStride
 - nvinfer1::plugin::RPROIParams, [687](#)
- FieldMap
 - nvuffparser::FieldMap, [125](#), [126](#)
- fields
 - nvinfer1::PluginFieldCollection, [677](#)
 - nvuffparser::FieldCollection, [124](#)
- FieldType
 - nvuffparser, [94](#)
- file
 - nvonnxparser::IParserError, [503](#)
- FillOperation
 - nvinfer1, [51](#)
- find

- nvcaffeparser1::IBlobNameToTensor, [150](#)
- findLogger
 - nvinfer1::ILoggerFinder, [399](#)
- flip
 - nvinfer1::plugin::PriorBoxParameters, [682](#)
- format
 - nvinfer1::PluginTensorDesc, [680](#)
- free
 - nvinfer1::IGpuAllocator, [358](#)
- func
 - nvonnxparser::IParserError, [503](#)
- GatherMode
 - nvinfer1, [51](#)
- getActivationType
 - nvinfer1::IActivationLayer, [130](#)
- getAlgorithm
 - nvinfer1::IInt8Calibrator, [376](#)
 - nvinfer1::IInt8EntropyCalibrator, [379](#)
 - nvinfer1::IInt8EntropyCalibrator2, [380](#)
 - nvinfer1::IInt8LegacyCalibrator, [381](#)
 - nvinfer1::IInt8MinMaxCalibrator, [384](#)
- getAlgorithmIOInfo
 - nvinfer1::IAlgorithm, [134](#)
- getAlgorithmIOInfoByIndex
 - nvinfer1::IAlgorithm, [134](#)
- getAlgorithmSelector
 - nvinfer1::IBuilderConfig, [165](#)
- getAlgorithmVariant
 - nvinfer1::IAlgorithm, [135](#)
- getAlignCorners
 - nvinfer1::IGridSampleLayer, [362](#)
 - nvinfer1::IResizeLayer, [589](#)
- getAll
 - nvinfer1::IRefitter, [579](#)
- getAllowedFormats
 - nvinfer1::ITensor, [649](#)
- getAllWeights
 - nvinfer1::IRefitter, [579](#)
- getAlpha
 - nvinfer1::IActivationLayer, [130](#)
 - nvinfer1::IFillLayer, [342](#)
 - nvinfer1::ILRNLayer, [409](#)
- getAverageCountExcludesPadding
 - nvinfer1::IPoolingLayer, [558](#)
- getAvgTimingIterations
 - nvinfer1::IBuilderConfig, [166](#)
- getAxes
 - nvinfer1::INormalizationLayer, [469](#)
 - nvinfer1::ISoftMaxLayer, [645](#)
- getAxis
 - nvinfer1::IConcatenationLayer, [193](#)
 - nvinfer1::IDequantizeLayer, [274](#)
 - nvinfer1::IIteratorLayer, [385](#)
 - nvinfer1::ILoopOutputLayer, [406](#)
 - nvinfer1::IOneHotLayer, [473](#)
 - nvinfer1::IQuantizeLayer, [570](#)
 - nvinfer1::IScatterLayer, [629](#)
- getBatch
 - nvinfer1::IInt8Calibrator, [376](#)
- getBatchAxis
 - nvinfer1::IReverseSequenceLayer, [597](#)
- getBatchSize
 - nvinfer1::IInt8Calibrator, [377](#)
- getBeta
 - nvinfer1::IActivationLayer, [131](#)
 - nvinfer1::IFillLayer, [342](#)
 - nvinfer1::ILRNLayer, [409](#)
- getBiasForGate
 - nvinfer1::IRNNv2Layer, [600](#)
- getBiasWeights
 - nvinfer1::IConvolutionLayer, [203](#)
 - nvinfer1::IDeconvolutionLayer, [263](#)
 - nvinfer1::IFullyConnectedLayer, [348](#)
- getBindingBytesPerComponent
 - nvinfer1::ICudaEngine, [218](#)
 - nvinfer1::safe::ICudaEngine, [246](#)
- getBindingComponentsPerElement
 - nvinfer1::ICudaEngine, [219](#)
 - nvinfer1::safe::ICudaEngine, [247](#)
- getBindingDataType
 - nvinfer1::ICudaEngine, [219](#)
 - nvinfer1::safe::ICudaEngine, [247](#)
- getBindingDimensions
 - nvinfer1::ICudaEngine, [220](#)
 - nvinfer1::IExecutionContext, [301](#)
 - nvinfer1::safe::ICudaEngine, [248](#)
- getBindingFormat
 - nvinfer1::ICudaEngine, [220](#)
 - nvinfer1::safe::ICudaEngine, [249](#)
- getBindingFormatDesc
 - nvinfer1::ICudaEngine, [221](#)
- getBindingIndex
 - nvinfer1::ICudaEngine, [221](#)
 - nvinfer1::safe::ICudaEngine, [249](#)
- getBindingName
 - nvinfer1::ICudaEngine, [222](#)
 - nvinfer1::safe::ICudaEngine, [250](#)
- getBindingVectorizedDim
 - nvinfer1::ICudaEngine, [223](#)
 - nvinfer1::safe::ICudaEngine, [251](#)
- getBlendFactor
 - nvinfer1::IPoolingLayer, [558](#)
- getBoundingBoxFormat
 - nvinfer1::INMSLayer, [461](#)
- getBroadcastAcrossBatch
 - nvinfer1::ITensor, [649](#)
- getBuilder

- nvinfer1::INetworkDefinition, 451
- getBuilderOptimizationLevel
 - nvinfer1::IBuilderConfig, 166
- getBuilderPluginRegistry
 - nvinfer1, 82
- getBuilderSafePluginRegistry
 - nvinfer1, 82
- getCalibrationProfile
 - nvinfer1::IBuilderConfig, 166
- getCellState
 - nvinfer1::IRNNv2Layer, 601
- getChannelAxis
 - nvinfer1::IScaleLayer, 624
- getComponentsPerElement
 - nvinfer1::IAlgorithmIOInfo, 139
- getComputePrecision
 - nvinfer1::INormalizationLayer, 469
- getConditional
 - nvinfer1::IIfConditionalBoundaryLayer, 372
- getConstantValue
 - nvinfer1::IDimensionExpr, 276
- getCoordinateTransformation
 - nvinfer1::IResizeLayer, 589
- getCubicCoeff
 - nvinfer1::IResizeLayer, 589
- getData
 - nvcaffeparser1::IBinaryProtoBlob, 148
- getDataLength
 - nvinfer1::IRNNv2Layer, 601
- getDataType
 - nvcaffeparser1::IBinaryProtoBlob, 148
 - nvinfer1::IAlgorithmIOInfo, 139
- getDebugSync
 - nvinfer1::IExecutionContext, 302
- getDefaultDeviceType
 - nvinfer1::IBuilderConfig, 166
- getDeviceMemorySize
 - nvinfer1::ICudaEngine, 223
 - nvinfer1::safe::ICudaEngine, 252
- getDeviceType
 - nvinfer1::IBuilderConfig, 167
- getDilation
 - nvinfer1::IConvolutionLayer, 203
- getDilationNd
 - nvinfer1::IConvolutionLayer, 203
 - nvinfer1::IDeconvolutionLayer, 263
- getDimensionName
 - nvinfer1::ITensor, 649
- getDimensions
 - nvcaffeparser1::IBinaryProtoBlob, 148
 - nvinfer1::IAlgorithmContext, 137
 - nvinfer1::IConstantLayer, 199
 - nvinfer1::IFillLayer, 342
 - nvinfer1::IOptimizationProfile, 482
 - nvinfer1::ITensor, 650
- getDirection
 - nvinfer1::IRNNv2Layer, 601
- getDLACore
 - nvinfer1::IBuilderConfig, 167
 - nvinfer1::IRuntime, 610
- getDynamicRangeMax
 - nvinfer1::IRefitter, 580
 - nvinfer1::ITensor, 650
- getDynamicRangeMin
 - nvinfer1::IRefitter, 580
 - nvinfer1::ITensor, 650
- getEngine
 - nvinfer1::IExecutionContext, 302
 - nvinfer1::safe::IExecutionContext, 327
- getEngineCapability
 - nvinfer1::IBuilderConfig, 167
 - nvinfer1::ICudaEngine, 223
- getEngineHostCodeAllowed
 - nvinfer1::IRuntime, 611
- getEngineInformation
 - nvinfer1::IEngineInspector, 283
- getEnqueueEmitsProfile
 - nvinfer1::IExecutionContext, 302
- getEpsilon
 - nvinfer1::INormalizationLayer, 469
- getEquation
 - nvinfer1::IEinsumLayer, 278
- getError
 - nvonnxparser::IParser, 497
- getErrorBuffer
 - nvinfer1::safe::IExecutionContext, 327
- getErrorCode
 - nvinfer1::IErrorRecorder, 289
- getErrorDesc
 - nvinfer1::IErrorRecorder, 290
- getErrorRecorder
 - nvcaffeparser1::ICaffeParser, 186
 - nvinfer1::IBuilder, 154
 - nvinfer1::ICudaEngine, 224
 - nvinfer1::IEngineInspector, 283
 - nvinfer1::IExecutionContext, 303
 - nvinfer1::INetworkDefinition, 451
 - nvinfer1::IPluginRegistry, 516
 - nvinfer1::IRefitter, 580
 - nvinfer1::IRuntime, 611
 - nvinfer1::safe::ICudaEngine, 252
 - nvinfer1::safe::IExecutionContext, 328
 - nvinfer1::safe::IPluginRegistry, 521
 - nvinfer1::safe::IRuntime, 620
 - nvuffparser::IUffParser, 667
- getExcludeOutside
 - nvinfer1::IResizeLayer, 589
- getExecutionContext

- nvinfer1::IEngineInspector, [284](#)
- getExtraMemoryTarget
 - nvinfer1::IOptimizationProfile, [483](#)
- getFieldNames
 - nvinfer1::IPluginCreator, [509](#)
- getFirstTranspose
 - nvinfer1::IShuffleLayer, [634](#)
- getFlag
 - nvinfer1::IBuilderConfig, [167](#)
 - nvonnxparser::IParser, [497](#)
- getFlags
 - nvinfer1::IBuilderConfig, [168](#)
 - nvonnxparser::IParser, [497](#)
- getFullTextFileName
 - nvonnxparser::IOnnxConfig, [476](#)
- getGatherAxis
 - nvinfer1::IGatherLayer, [354](#)
- getHardwareCompatibilityLevel
 - nvinfer1::IBuilderConfig, [168](#)
 - nvinfer1::ICudaEngine, [224](#)
- getHiddenSize
 - nvinfer1::IRNNv2Layer, [601](#)
- getHiddenState
 - nvinfer1::IRNNv2Layer, [602](#)
- getImplementation
 - nvinfer1::IAlgorithmVariant, [144](#)
- getInferLibVersion
 - NvInferRuntimeBase.h, [785](#)
- getInput
 - nvinfer1::ILayer, [389](#)
 - nvinfer1::INetworkDefinition, [451](#)
- getInputConsumedEvent
 - nvinfer1::IExecutionContext, [303](#)
 - nvinfer1::safe::IExecutionContext, [328](#)
- getInputMode
 - nvinfer1::IRNNv2Layer, [602](#)
- getInputTensorAddress
 - nvinfer1::safe::IExecutionContext, [329](#)
- getInt8Calibrator
 - nvinfer1::IBuilderConfig, [168](#)
- getInterfaceVersion
 - nvinfer1::IOutputAllocator, [487](#)
- getInterpolationMode
 - nvinfer1::IGridSampleLayer, [362](#)
- getIOTensorName
 - nvinfer1::ICudaEngine, [224](#)
 - nvinfer1::safe::ICudaEngine, [252](#)
- getK
 - nvinfer1::ILRNLayer, [409](#)
 - nvinfer1::ITopKLayer, [662](#)
- getKeepDimensions
 - nvinfer1::IReduceLayer, [575](#)
- getKernelSize
 - nvinfer1::IConvolutionLayer, [203](#)
 - nvinfer1::IDeconvolutionLayer, [263](#)
- getKernelSizeNd
 - nvinfer1::IConvolutionLayer, [204](#)
 - nvinfer1::IDeconvolutionLayer, [264](#)
- getKernelWeights
 - nvinfer1::IConvolutionLayer, [204](#)
 - nvinfer1::IDeconvolutionLayer, [264](#)
 - nvinfer1::IFullyConnectedLayer, [348](#)
- getLayer
 - nvinfer1::INetworkDefinition, [452](#)
- getLayerCount
 - nvinfer1::IRNNv2Layer, [602](#)
- getLayerInformation
 - nvinfer1::IEngineInspector, [284](#)
- getLocation
 - nvinfer1::ICudaEngine, [225](#)
 - nvinfer1::ITensor, [651](#)
- getLogger
 - nvinfer1::IBuilder, [155](#)
 - nvinfer1::IRefitter, [581](#)
 - nvinfer1::IRuntime, [611](#)
 - NvInferRuntime.h, [765](#)
- getLoop
 - nvinfer1::ILoopBoundaryLayer, [404](#)
- getLoopOutput
 - nvinfer1::ILoopOutputLayer, [406](#)
- getMaxAuxStreams
 - nvinfer1::IBuilderConfig, [169](#)
- getMaxBatchSize
 - nvinfer1::IBuilder, [155](#)
 - nvinfer1::ICudaEngine, [225](#)
- getMaxDLABatchSize
 - nvinfer1::IBuilder, [155](#)
- getMaxOutputSize
 - nvinfer1::IExecutionContext, [303](#)
- getMaxSeqLength
 - nvinfer1::IRNNv2Layer, [602](#)
- getMaxThreads
 - nvinfer1::IBuilder, [156](#)
 - nvinfer1::IRefitter, [581](#)
 - nvinfer1::IRuntime, [612](#)
- getMaxWorkspaceSize
 - nvinfer1::IBuilderConfig, [169](#)
- getMemoryPoolLimit
 - nvinfer1::IBuilderConfig, [169](#)
- getMessage
 - nvinfer1::IAssertionLayer, [146](#)
- getMetadata
 - nvinfer1::ILayer, [389](#)
- getMinTimingIterations
 - nvinfer1::IBuilderConfig, [170](#)
- getMissing
 - nvinfer1::IRefitter, [581](#)
- getMissingWeights

- nvinfer1::IRefitter, 582
- getMode
 - nvinfer1::IGatherLayer, 354
 - nvinfer1::IScaleLayer, 624
 - nvinfer1::IScatterLayer, 629
 - nvinfer1::ISliceLayer, 640
- getModelDtype
 - nvonnxparser::IOnnxConfig, 476
- getModelFileName
 - nvonnxparser::IOnnxConfig, 476
- getName
 - nvinfer1::IAlgorithmContext, 137
 - nvinfer1::ICudaEngine, 226
 - nvinfer1::IExecutionContext, 304
 - nvinfer1::IIfConditional, 370
 - nvinfer1::ILayer, 389
 - nvinfer1::ILoop, 402
 - nvinfer1::INetworkDefinition, 452
 - nvinfer1::ITensor, 651
 - nvinfer1::safe::ICudaEngine, 253
 - nvinfer1::safe::IExecutionContext, 330
- getNbAuxStreams
 - nvinfer1::ICudaEngine, 226
- getNbBindings
 - nvinfer1::ICudaEngine, 226
 - nvinfer1::safe::ICudaEngine, 253
- getNbDLACores
 - nvinfer1::IBuilder, 156
 - nvinfer1::IRuntime, 612
- getNbElementWiseDims
 - nvinfer1::IGatherLayer, 354
- getNbErrors
 - nvinfer1::IErrorRecorder, 290
 - nvonnxparser::IParser, 497
- getNbGroups
 - nvinfer1::IConvolutionLayer, 204
 - nvinfer1::IDeconvolutionLayer, 264
 - nvinfer1::INormalizationLayer, 469
- getNbInputs
 - nvinfer1::IAlgorithmContext, 137
 - nvinfer1::ILayer, 389
 - nvinfer1::INetworkDefinition, 453
- getNbIOTensors
 - nvinfer1::ICudaEngine, 227
 - nvinfer1::safe::ICudaEngine, 254
- getNbLayers
 - nvinfer1::ICudaEngine, 227
 - nvinfer1::INetworkDefinition, 453
- getNbOptimizationProfiles
 - nvinfer1::IBuilderConfig, 170
 - nvinfer1::ICudaEngine, 227
- getNbOutputChannels
 - nvinfer1::IFullyConnectedLayer, 348
- getNbOutputMaps
 - nvinfer1::IConvolutionLayer, 204
 - nvinfer1::IDeconvolutionLayer, 264
- getNbOutputs
 - nvinfer1::IAlgorithmContext, 137
 - nvinfer1::ILayer, 390
 - nvinfer1::INetworkDefinition, 453
 - nvinfer1::IPluginV2, 529
- getNbPluginsToSerialize
 - nvinfer1::IBuilderConfig, 170
- getNbShapeValues
 - nvinfer1::IOptimizationProfile, 483
- getNearestRounding
 - nvinfer1::IResizeLayer, 590
- getNvOnnxParserVersion
 - NvOnnxParser.h, 812
- getNvtxVerbosity
 - nvinfer1::IExecutionContext, 304
- getOperation
 - nvinfer1::IElementWiseLayer, 280
 - nvinfer1::IFillLayer, 343
 - nvinfer1::IMatrixMultiplyLayer, 413
 - nvinfer1::IReduceLayer, 575
 - nvinfer1::IRNNv2Layer, 602
 - nvinfer1::ITopKLayer, 662
 - nvinfer1::IUnaryLayer, 671
- getOptimizationProfile
 - nvinfer1::IExecutionContext, 304
- getOutput
 - nvinfer1::ILayer, 390
 - nvinfer1::INetworkDefinition, 454
- getOutputAllocator
 - nvinfer1::IExecutionContext, 305
- getOutputDataType
 - nvinfer1::IPluginV2Ext, 549
- getOutputDimensions
 - nvinfer1::IPluginV2, 529
 - nvinfer1::IPluginV2DynamicExt, 541
 - nvinfer1::IResizeLayer, 590
- getOutputTensorAddress
 - nvinfer1::IExecutionContext, 305
 - nvinfer1::safe::IExecutionContext, 330
- getOutputType
 - nvinfer1::ILayer, 390
- getPadding
 - nvinfer1::IConvolutionLayer, 205
 - nvinfer1::IDeconvolutionLayer, 265
 - nvinfer1::IPoolingLayer, 558
- getPaddingMode
 - nvinfer1::IConvolutionLayer, 205
 - nvinfer1::IDeconvolutionLayer, 265
 - nvinfer1::IPoolingLayer, 559
- getPaddingNd
 - nvinfer1::IConvolutionLayer, 205
 - nvinfer1::IDeconvolutionLayer, 265

- nvinfer1::IPoolingLayer, 559
- getPersistentCacheLimit
 - nvinfer1::IExecutionContext, 306
- getPlugin
 - nvinfer1::IPluginV2Layer, 555
- getPluginCreator
 - nvinfer1::IPluginRegistry, 516
 - nvinfer1::safe::IPluginRegistry, 522
- getPluginCreatorList
 - nvinfer1::IPluginRegistry, 517
 - nvinfer1::safe::IPluginRegistry, 523
- getPluginName
 - nvinfer1::IPluginCreator, 509
- getPluginNamespace
 - nvinfer1::IPluginCreator, 510
 - nvinfer1::IPluginV2, 531
- getPluginRegistry
 - nvinfer1::IBuilder, 156
 - nvinfer1::IRuntime, 612
 - NvInferRuntime.h, 765
- getPluginToSerialize
 - nvinfer1::IBuilderConfig, 171
- getPluginType
 - nvinfer1::IPluginV2, 531
- getPluginVersion
 - nvinfer1::IPluginCreator, 510
 - nvinfer1::IPluginV2, 532
- getPoolingType
 - nvinfer1::IPoolingLayer, 559
- getPostPadding
 - nvinfer1::IConvolutionLayer, 206
 - nvinfer1::IDeconvolutionLayer, 266
 - nvinfer1::IPaddingLayer, 490
 - nvinfer1::IPoolingLayer, 560
- getPostPaddingNd
 - nvinfer1::IPaddingLayer, 490
- getPower
 - nvinfer1::IScaleLayer, 624
- getPrecision
 - nvinfer1::ILayer, 391
- getPrePadding
 - nvinfer1::IConvolutionLayer, 206
 - nvinfer1::IDeconvolutionLayer, 266
 - nvinfer1::IPaddingLayer, 491
 - nvinfer1::IPoolingLayer, 560
- getPrePaddingNd
 - nvinfer1::IPaddingLayer, 491
- getPreviewFeature
 - nvinfer1::IBuilderConfig, 171
- getPrintLayerInfo
 - nvonnxparser::IOnnxConfig, 477
- getProfileDimensions
 - nvinfer1::ICudaEngine, 228
- getProfiler
 - nvinfer1::IExecutionContext, 306
- getProfileShape
 - nvinfer1::ICudaEngine, 229
- getProfileShapeValues
 - nvinfer1::ICudaEngine, 229
- getProfileStream
 - nvinfer1::IBuilderConfig, 172
- getProfilingVerbosity
 - nvinfer1::IBuilderConfig, 172
 - nvinfer1::ICudaEngine, 230
- getQuantile
 - nvinfer1::IInt8LegacyCalibrator, 381
- getQuantizationFlag
 - nvinfer1::IBuilderConfig, 172
- getQuantizationFlags
 - nvinfer1::IBuilderConfig, 172
- getReduceAxes
 - nvinfer1::IReduceLayer, 575
 - nvinfer1::ITopKLayer, 662
- getRegressionCutoff
 - nvinfer1::IInt8LegacyCalibrator, 382
- getReshapeDimensions
 - nvinfer1::IShuffleLayer, 634
- getResizeMode
 - nvinfer1::IResizeLayer, 590
- getReverse
 - nvinfer1::IIteratorLayer, 385
- getSafePluginRegistry
 - nvinfer1::safe, 87
- getSampleMode
 - nvinfer1::IGridSampleLayer, 362
- getScale
 - nvinfer1::IScaleLayer, 625
- getScales
 - nvinfer1::IResizeLayer, 590
- getSecondTranspose
 - nvinfer1::IShuffleLayer, 635
- getSelectorForSinglePixel
 - nvinfer1::IResizeLayer, 591
- getSequenceAxis
 - nvinfer1::IReverseSequenceLayer, 597
- getSequenceLengths
 - nvinfer1::IRNNv2Layer, 603
- getSerializationSize
 - nvinfer1::IPluginV2, 532
- getShapeBinding
 - nvinfer1::IExecutionContext, 306
- getShapeValues
 - nvinfer1::IOptimizationProfile, 483
- getShift
 - nvinfer1::IScaleLayer, 625
- getSize
 - nvinfer1::ISliceLayer, 640
- getStart

- nvinfer1::ISliceLayer, 641
- getStride
 - nvinfer1::IConvolutionLayer, 206
 - nvinfer1::IDeconvolutionLayer, 266
 - nvinfer1::IPoolingLayer, 560
 - nvinfer1::ISliceLayer, 641
- getStrideNd
 - nvinfer1::IConvolutionLayer, 206
 - nvinfer1::IDeconvolutionLayer, 266
 - nvinfer1::IPoolingLayer, 560
- getStrides
 - nvinfer1::IAlgorithmIOInfo, 140
 - nvinfer1::IExecutionContext, 307
 - nvinfer1::safe::IExecutionContext, 331
- getTactic
 - nvinfer1::IAlgorithmVariant, 144
- getTacticSources
 - nvinfer1::IBuilderConfig, 173
 - nvinfer1::ICudaEngine, 230
- getTempfileControlFlags
 - nvinfer1::IRuntime, 612
- getTemporaryDirectory
 - nvinfer1::IRuntime, 613
- getTemporaryStorageAllocator
 - nvinfer1::IExecutionContext, 307
- getTensorAddress
 - nvinfer1::IExecutionContext, 308
- getTensorBytesPerComponent
 - nvinfer1::ICudaEngine, 230, 232
 - nvinfer1::safe::ICudaEngine, 254
- getTensorComponentsPerElement
 - nvinfer1::ICudaEngine, 232, 233
 - nvinfer1::safe::ICudaEngine, 255
- getTensorDataType
 - nvinfer1::ICudaEngine, 234
 - nvinfer1::safe::ICudaEngine, 256
- getTensorFormat
 - nvinfer1::IAlgorithmIOInfo, 140
 - nvinfer1::ICudaEngine, 234
 - nvinfer1::safe::ICudaEngine, 257
- getTensorFormatDesc
 - nvinfer1::ICudaEngine, 235
- getTensorIOMode
 - nvinfer1::ICudaEngine, 236
 - nvinfer1::safe::ICudaEngine, 257
- getTensorLocation
 - nvinfer1::ICudaEngine, 236
- getTensorRTVersion
 - nvinfer1::IPluginCreator, 510
 - nvinfer1::IPluginV2, 533
 - nvinfer1::IPluginV2DynamicExt, 541
 - nvinfer1::IPluginV2Ext, 550
 - nvinfer1::IPluginV2IOExt, 553
- getTensorShape
 - nvinfer1::ICudaEngine, 237
 - nvinfer1::IExecutionContext, 308
 - nvinfer1::safe::ICudaEngine, 258
- getTensorStrides
 - nvinfer1::IExecutionContext, 309
 - nvinfer1::safe::IExecutionContext, 332
- getTensorsWithDynamicRange
 - nvinfer1::IRefitter, 582
- getTensorVectorizedDim
 - nvinfer1::ICudaEngine, 237, 238
 - nvinfer1::safe::ICudaEngine, 259
- getTextFileName
 - nvonnxparser::IOnnxConfig, 477
- getTimingCache
 - nvinfer1::IBuilderConfig, 173
- getTimingMSec
 - nvinfer1::IAlgorithm, 135
- getTopKBoxLimit
 - nvinfer1::INMSLayer, 461
- getToType
 - nvinfer1::ICastLayer, 191
- getTripLimit
 - nvinfer1::ITripLimitLayer, 665
- getType
 - nvinfer1::ILayer, 391
 - nvinfer1::ITensor, 651
- getUffRequiredVersionMajor
 - nvuffparser::IUffParser, 667
- getUffRequiredVersionMinor
 - nvuffparser::IUffParser, 667
- getUffRequiredVersionPatch
 - nvuffparser::IUffParser, 668
- getUsedVCPluginLibraries
 - nvonnxparser::IParser, 498
- getVectorizedDim
 - nvinfer1::IAlgorithmIOInfo, 140
- getVerbosityLevel
 - nvonnxparser::IOnnxConfig, 477
- getWeights
 - nvinfer1::IConstantLayer, 199
- getWeightsForGate
 - nvinfer1::IRNNv2Layer, 603
- getWindowSize
 - nvinfer1::ILRNLayer, 409
 - nvinfer1::IPoolingLayer, 561
- getWindowSizeNd
 - nvinfer1::IPoolingLayer, 561
- getWorkspaceSize
 - nvinfer1::IAlgorithm, 135
 - nvinfer1::IPluginV2, 533
 - nvinfer1::IPluginV2DynamicExt, 541
- getZeroIsPlaceholder
 - nvinfer1::IShuffleLayer, 635
- group

- nvinfer1::plugin::softmaxTree, 690
- groupOffset
 - nvinfer1::plugin::softmaxTree, 690
- groups
 - nvinfer1::plugin::softmaxTree, 690
- groupSize
 - nvinfer1::plugin::softmaxTree, 690
- H
 - nvinfer1::plugin::GridAnchorParameters, 128
- h
 - nvinfer1::DimsHW, 108
- HardwareCompatibilityLevel
 - nvinfer1, 52
- hasExplicitPrecision
 - nvinfer1::INetworkDefinition, 454
- hasImplicitBatchDimension
 - nvinfer1::ICudaEngine, 238
 - nvinfer1::INetworkDefinition, 455
- hasOverflowed
 - nvinfer1::IErrorRecorder, 291
- IConsistencyChecker
 - nvinfer1::consistency::IConsistencyChecker, 196, 197
- ICudaEngine
 - nvinfer1::safe::ICudaEngine, 243, 244
- IErrorRecorder
 - nvinfer1::IErrorRecorder, 287
- IExecutionContext
 - nvinfer1::safe::IExecutionContext, 324, 325
- IGpuAllocator
 - nvinfer1::IGpuAllocator, 357
- ILogger
 - nvinfer1::ILogger, 397
- imgH
 - nvinfer1::plugin::PriorBoxParameters, 682
- imgW
 - nvinfer1::plugin::PriorBoxParameters, 683
- incRefCount
 - nvinfer1::IErrorRecorder, 291
- inferShapes
 - nvinfer1::IExecutionContext, 310
- initialize
 - nvinfer1::IPluginV2, 534
- initLibNvInferPlugins
 - NvInferPlugin.h, 758
- INoCopy
 - nvinfer1::INoCopy, 465
- INonZero, 466
- inputOrder
 - nvinfer1::plugin::DetectionOutputParameters, 99
- InterpolationMode
 - nvinfer1, 52
- iouThreshold
 - nvinfer1::plugin::NMSParameters, 673
 - nvinfer1::plugin::RPROIParams, 687
- IPluginChecker
 - nvinfer1::consistency::IPluginChecker, 505
- IPluginCreator
 - nvinfer1::IPluginCreator, 507
- IPluginV2Ext
 - nvinfer1::IPluginV2Ext, 544
- IRuntime
 - nvinfer1::safe::IRuntime, 618, 619
- isBatchAgnostic
 - nvinfer1::plugin::DetectionOutputParameters, 99
- isConstant
 - nvinfer1::IDimensionExpr, 276
- isDeviceTypeSet
 - nvinfer1::IBuilderConfig, 173
- isExecutionBinding
 - nvinfer1::ICudaEngine, 239
- isExecutionTensor
 - nvinfer1::ITensor, 652
- isNetworkInput
 - nvinfer1::ITensor, 652
- isNetworkOutput
 - nvinfer1::ITensor, 652
- isNetworkSupported
 - nvinfer1::IBuilder, 156
- isNormalized
 - nvinfer1::plugin::DetectionOutputParameters, 99
 - nvinfer1::plugin::NMSParameters, 673
- isOutputBroadcastAcrossBatch
 - nvinfer1::IPluginV2Ext, 550
- isParentSearchEnabled
 - nvinfer1::IPluginRegistry, 517
- isPluginV2
 - nvcaffeparser1::IPluginFactoryV2, 513
- isRefittable
 - nvinfer1::ICudaEngine, 239
- isShapeBinding
 - nvinfer1::ICudaEngine, 239
- isShapeInferenceIO
 - nvinfer1::ICudaEngine, 240
- isShapeTensor
 - nvinfer1::ITensor, 653
- isValid
 - nvinfer1::IOptimizationProfile, 484
- kABS
 - nvinfer1, 73
- kACOS
 - nvinfer1, 74
- kACOSH
 - nvinfer1, 74
- kACTIVATION
 - nvinfer1, 53

- kALIGN_CORNERS
 - nvinfer1, [64](#)
- kALLOW_IN_MEMORY_FILES
 - nvinfer1, [70](#)
- kALLOW_TEMPORARY_FILES
 - nvinfer1, [70](#)
- kAMPERE_PLUS
 - nvinfer1, [52](#)
- kAND
 - nvinfer1, [48](#)
- kANY
 - nvinfer1, [74](#)
- kASIN
 - nvinfer1, [74](#)
- kASINH
 - nvinfer1, [74](#)
- kASSERTION
 - nvinfer1, [54](#)
- kASYMMETRIC
 - nvinfer1, [64](#)
- kATAN
 - nvinfer1, [74](#)
- kATANH
 - nvinfer1, [74](#)
- kAVERAGE
 - nvinfer1, [61](#)
- kAVG
 - nvinfer1, [63](#)
- kBIAS
 - nvinfer1, [74](#)
- kBIDIRECTION
 - nvinfer1, [65](#)
- kBOOL
 - nvinfer1, [47](#)
- kCAFFE_ROUND_DOWN
 - nvinfer1, [60](#)
- kCAFFE_ROUND_UP
 - nvinfer1, [60](#)
- kCALIBRATE_BEFORE_FUSION
 - nvinfer1, [63](#)
- kCAST
 - nvinfer1, [54](#)
- kCDHW32
 - nvinfer1, [71](#)
- kCEIL
 - nvinfer1, [64](#), [74](#)
- kCEIL_DIV
 - nvinfer1, [48](#)
- kCELL
 - nvinfer1, [65](#)
- kCENTER_SIZES
 - nvinfer1, [44](#)
- kCHANNEL
 - nvinfer1, [68](#)
- kCHAR
 - nvinfer1, [60](#)
 - nvuffparser, [95](#)
- kCHW16
 - nvinfer1, [71](#)
- kCHW2
 - nvinfer1, [70](#)
- kCHW32
 - nvinfer1, [71](#)
- kCHW4
 - nvinfer1, [71](#)
- kCLAMP
 - nvinfer1, [68](#)
- kCLIP
 - nvinfer1, [43](#)
- kCONCATENATE
 - nvinfer1, [54](#)
- kCONCATENATION
 - nvinfer1, [53](#)
- kCONDITION
 - nvinfer1, [54](#)
- kCONDITIONAL_INPUT
 - nvinfer1, [54](#)
- kCONDITIONAL_OUTPUT
 - nvinfer1, [54](#)
- kCONSTANT
 - nvinfer1, [54](#), [74](#)
- kCONVOLUTION
 - nvinfer1, [53](#)
- kCORNER_PAIRS
 - nvinfer1, [44](#)
- kCOS
 - nvinfer1, [73](#)
- kCOSH
 - nvinfer1, [74](#)
- kCOUNT
 - nvinfer1, [73](#)
- kCUBIC
 - nvinfer1, [52](#)
- kCUBLAS
 - nvinfer1, [69](#)
- kCUBLAS_LT
 - nvinfer1, [69](#)
- kCUDNN
 - nvinfer1, [69](#)
- kDATATYPE
 - nvuffparser, [95](#)
- kDEBUG
 - nvinfer1, [44](#)
- kDECONVOLUTION
 - nvinfer1, [53](#)
- kDEFAULT
 - nvinfer1, [49](#), [52](#), [62](#), [67](#)
- kDEQUANTIZE

- nvinfer1, [54](#)
- kDETAILED
 - nvinfer1, [62](#)
- kDEVICE
 - nvinfer1, [72](#)
- kDHW
 - nvinfer1, [72](#)
- kDHW8
 - nvinfer1, [71](#)
- kDIMS
 - nvinfer1, [60](#)
 - nvuffparser, [95](#)
- kDIRECT_IO
 - nvinfer1, [45](#)
- kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805
 - nvinfer1, [61](#)
- kDISABLE_TIMING_CACHE
 - nvinfer1, [45](#)
- kDIV
 - nvinfer1, [48](#)
- kDLA
 - nvinfer1, [47](#)
- kDLA_GLOBAL_DRAM
 - nvinfer1, [56](#)
- kDLA_HWC4
 - nvinfer1, [71](#)
- kDLA_LINEAR
 - nvinfer1, [71](#)
- kDLA_LOCAL_DRAM
 - nvinfer1, [56](#)
- kDLA_MANAGED_SRAM
 - nvinfer1, [55](#)
- kDLA_STANDALONE
 - nvinfer1, [49](#)
- kEDGE_MASK_CONVOLUTIONS
 - nvinfer1, [69](#)
- keepTopK
 - nvinfer1::plugin::DetectionOutputParameters, [99](#)
 - nvinfer1::plugin::NMSParameters, [673](#)
- KEINSUM
 - nvinfer1, [54](#)
- KELEMENT
 - nvinfer1, [52](#), [68](#)
- KELEMENTWISE
 - nvinfer1, [53](#), [68](#)
- kELU
 - nvinfer1, [43](#)
- kENABLE_TACTIC_HEURISTIC
 - nvinfer1, [45](#)
- kENTROPY_CALIBRATION
 - nvinfer1, [46](#)
- kENTROPY_CALIBRATION_2
 - nvinfer1, [46](#)
- KEQUAL
 - nvinfer1, [48](#)
- kERF
 - nvinfer1, [74](#)
- kERROR
 - nvinfer1::ILogger, [397](#)
- kEXCLUDE_LEAN_RUNTIME
 - nvinfer1, [46](#)
- kEXP
 - nvinfer1, [73](#)
- kEXPLICIT_BATCH
 - nvinfer1, [56](#)
- kEXPLICIT_PRECISION
 - nvinfer1, [56](#)
- kEXPLICIT_ROUND_DOWN
 - nvinfer1, [60](#)
- kEXPLICIT_ROUND_UP
 - nvinfer1, [60](#)
- kFAILED_ALLOCATION
 - nvinfer1, [50](#)
- kFAILED_COMPUTATION
 - nvinfer1, [50](#)
- kFAILED_EXECUTION
 - nvinfer1, [50](#)
- kFAILED_INITIALIZATION
 - nvinfer1, [50](#)
- kFASTER_DYNAMIC_SHAPES_0805
 - nvinfer1, [61](#)
- kFILL
 - nvinfer1, [54](#), [68](#)
- kFLOAT
 - nvinfer1, [46](#)
 - nvuffparser, [95](#)
- kFLOAT16
 - nvinfer1, [60](#)
- kFLOAT32
 - nvinfer1, [60](#)
- kFLOAT64
 - nvinfer1, [60](#)
- kFLOOR
 - nvinfer1, [64](#), [74](#)
- kFLOOR_DIV
 - nvinfer1, [48](#)
- kFORGET
 - nvinfer1, [65](#)
- kFORMAT_COMBINATION_LIMIT
 - nvinfer1::IPluginV2DynamicExt, [543](#)
- kFORMULA
 - nvinfer1, [65](#)
- kFP16
 - nvinfer1, [44](#)
- kFP8
 - nvinfer1, [46](#), [47](#)
- kFULLY_CONNECTED
 - nvinfer1, [53](#)

- kGATHER
 - nvinfer1, [53](#)
- kGATHER_OOB
 - nvinfer1::safe, [86](#)
- kGPU
 - nvinfer1, [47](#)
- kGPU_FALLBACK
 - nvinfer1, [44](#)
- kGREATER
 - nvinfer1, [48](#)
- kGRID_SAMPLE
 - nvinfer1, [54](#)
- kGRU
 - nvinfer1, [67](#)
- kHALF
 - nvinfer1, [46](#)
- kHALF_DOWN
 - nvinfer1, [64](#)
- kHALF_PIXEL
 - nvinfer1, [64](#)
- kHALF_UP
 - nvinfer1, [64](#)
- kHARD_SIGMOID
 - nvinfer1, [43](#)
- kHIDDEN
 - nvinfer1, [65](#)
- kHOST
 - nvinfer1, [72](#)
- kHWC
 - nvinfer1, [71](#)
- kHWC16
 - nvinfer1, [71](#)
- kHWC8
 - nvinfer1, [70](#)
- kIDENTITY
 - nvinfer1, [54](#)
- kINF_CONSUMED
 - nvinfer1::safe, [86](#)
- kINFO
 - nvinfer1::ILogger, [397](#)
- kINPUT
 - nvinfer1, [65](#), [72](#)
- kINT16
 - nvinfer1, [60](#)
- kINT32
 - nvinfer1, [47](#), [60](#)
 - nvuffparser, [95](#)
- kINT8
 - nvinfer1, [44](#), [47](#), [60](#)
- kINTERNAL_ERROR
 - nvinfer1, [50](#)
 - nvinfer1::ILogger, [397](#)
 - nvonnxparser, [91](#)
- kINVALID_ARGUMENT
 - nvinfer1, [50](#)
- kINVALID_CONFIG
 - nvinfer1, [50](#)
- kINVALID_GRAPH
 - nvonnxparser, [91](#)
- kINVALID_NODE
 - nvonnxparser, [91](#)
- kINVALID_STATE
 - nvinfer1, [50](#)
- kINVALID_VALUE
 - nvonnxparser, [91](#)
- kISINF
 - nvinfer1, [74](#)
- kITERATOR
 - nvinfer1, [54](#)
- kJIT_CONVOLUTIONS
 - nvinfer1, [69](#)
- kJSON
 - nvinfer1, [53](#)
- kKERNEL
 - nvinfer1, [74](#)
- kLAST_VALUE
 - nvinfer1, [54](#)
- kLAYER_NAMES_ONLY
 - nvinfer1, [62](#)
- kLEAKY_RELU
 - nvinfer1, [43](#)
- kLEGACY_CALIBRATION
 - nvinfer1, [46](#)
- kLESS
 - nvinfer1, [48](#)
- kLINEAR
 - nvinfer1, [52](#), [66](#), [70](#)
- kLinspace
 - nvinfer1, [51](#)
- kLOG
 - nvinfer1, [73](#)
- kLOOP_OUTPUT
 - nvinfer1, [54](#)
- kLRN
 - nvinfer1, [53](#)
- kLSTM
 - nvinfer1, [67](#)
- kMATRIX_MULTIPLY
 - nvinfer1, [53](#)
- kMAX
 - nvinfer1, [48](#), [57](#), [61](#), [63](#), [72](#)
- kMAX_AVERAGE_BLEND
 - nvinfer1, [61](#)
- kMAX_DESC_LENGTH
 - nvinfer1::IErrorRecorder, [293](#)
- kMEM_ALLOC_FAILED
 - nvonnxparser, [91](#)
- kMIN

- [nvinfer1](#), [48](#), [57](#), [63](#), [72](#)
- kMINMAX_CALIBRATION
 - [nvinfer1](#), [46](#)
- kMODEL_DESERIALIZE_FAILED
 - [nvonnxparser](#), [91](#)
- kNAN_CONSUMED
 - [nvinfer1::safe](#), [86](#)
- kNATIVE_INSTANCENORM
 - [nvonnxparser](#), [92](#)
- kNC
 - [nvuffparser](#), [95](#)
- kNCHW
 - [nvuffparser](#), [95](#)
- kND
 - [nvinfer1](#), [52](#), [68](#)
- kNEAREST
 - [nvinfer1](#), [52](#)
- kNEG
 - [nvinfer1](#), [73](#)
- kNHWC
 - [nvuffparser](#), [95](#)
- kNMS
 - [nvinfer1](#), [54](#)
- kNON_ZERO
 - [nvinfer1](#), [54](#)
- kNONE
 - [nvinfer1](#), [52](#), [55](#), [62](#), [72](#)
- kNORMALIZATION
 - [nvinfer1](#), [54](#)
- kNOT
 - [nvinfer1](#), [74](#)
- kOBEY_PRECISION_CONSTRAINTS
 - [nvinfer1](#), [45](#)
- kONE_HOT
 - [nvinfer1](#), [54](#)
- kONELINE
 - [nvinfer1](#), [53](#)
- kOPT
 - [nvinfer1](#), [57](#)
- kOR
 - [nvinfer1](#), [48](#)
- kOUTPUT
 - [nvinfer1](#), [65](#), [72](#)
- kPADDING
 - [nvinfer1](#), [53](#)
- kPARAMETRIC_RELU
 - [nvinfer1](#), [54](#)
- kPLUGIN
 - [nvinfer1](#), [53](#)
- kPLUGIN_V2
 - [nvinfer1](#), [54](#)
- kPOOLING
 - [nvinfer1](#), [53](#)
- kPOW
 - [nvinfer1](#), [48](#)
- kPREFER_PRECISION_CONSTRAINTS
 - [nvinfer1](#), [45](#)
- kPROD
 - [nvinfer1](#), [48](#), [63](#)
- kPROFILE_SHARING_0806
 - [nvinfer1](#), [62](#)
- kQUANTIZE
 - [nvinfer1](#), [54](#)
- kRAGGED_SOFTMAX
 - [nvinfer1](#), [53](#)
- kRANDOM_NORMAL
 - [nvinfer1](#), [51](#)
- kRANDOM_UNIFORM
 - [nvinfer1](#), [51](#)
- kRECIP
 - [nvinfer1](#), [73](#)
- kRECURRENCE
 - [nvinfer1](#), [54](#)
- kREDUCE
 - [nvinfer1](#), [53](#)
- kREFIT
 - [nvinfer1](#), [45](#)
- kREFLECT
 - [nvinfer1](#), [68](#)
- kREJECT_EMPTY_ALGORITHMS
 - [nvinfer1](#), [45](#)
- kRELU
 - [nvinfer1](#), [43](#), [67](#)
- kRESET
 - [nvinfer1](#), [65](#)
- kRESIZABLE
 - [nvinfer1](#), [44](#)
- kRESIZE
 - [nvinfer1](#), [54](#)
- kREVERSE
 - [nvinfer1](#), [54](#)
- kREVERSE_SEQUENCE
 - [nvinfer1](#), [54](#)
- kRNN_V2
 - [nvinfer1](#), [54](#)
- kROUND
 - [nvinfer1](#), [74](#)
- kSAFE_DLA
 - [nvinfer1](#), [49](#)
- kSAFE_GPU
 - [nvinfer1](#), [49](#)
- kSAFETY
 - [nvinfer1](#), [49](#)
- kSAFETY_SCOPE
 - [nvinfer1](#), [45](#)
- kSAME_LOWER
 - [nvinfer1](#), [60](#)
- kSAME_UPPER

- nvinfer1, 60
- kSCALE
 - nvinfer1, 53, 74
- kSCALED_TANH
 - nvinfer1, 43
- kSCATTER
 - nvinfer1, 54
- kSCATTER_OOB
 - nvinfer1::safe, 86
- kSCATTER_RACE
 - nvinfer1::safe, 86
- kSELECT
 - nvinfer1, 54
- kSELU
 - nvinfer1, 43
- kSHAPE
 - nvinfer1, 54
- kSHIFT
 - nvinfer1, 74
- kSHUFFLE
 - nvinfer1, 53
- kSIGMOID
 - nvinfer1, 43
- kSIGN
 - nvinfer1, 74
- kSIN
 - nvinfer1, 73
- kSINH
 - nvinfer1, 74
- kSKIP
 - nvinfer1, 66
- kSLICE
 - nvinfer1, 54
- kSOFTMAX
 - nvinfer1, 53
- kSOFTPLUS
 - nvinfer1, 43
- kSOFTSIGN
 - nvinfer1, 43
- kSPARSE_WEIGHTS
 - nvinfer1, 45
- kSQRT
 - nvinfer1, 73
- kSTANDARD
 - nvinfer1, 49
- kSTRICT_BOUNDS
 - nvinfer1, 67
- kSTRICT_TYPES
 - nvinfer1, 45
- kSUB
 - nvinfer1, 48
- kSUCCESS
 - nvinfer1, 50
 - nvonnxparser, 91
- kSUM
 - nvinfer1, 48, 63
- kTACTIC_DRAM
 - nvinfer1, 56
- kTAN
 - nvinfer1, 73
- kTANH
 - nvinfer1, 43, 67
- kTF32
 - nvinfer1, 45
- kTHRESHOLDED_RELU
 - nvinfer1, 43
- kTOPK
 - nvinfer1, 53
- kTRANPOSE
 - nvinfer1, 55
- kTRIP_LIMIT
 - nvinfer1, 54
- kUINT8
 - nvinfer1, 47
- kUNARY
 - nvinfer1, 53
- kUNIDIRECTION
 - nvinfer1, 65
- kUNIFORM
 - nvinfer1, 68
- kUNKNOWN
 - nvinfer1, 60
 - nvuffparser, 95
- kUNSPECIFIED_ERROR
 - nvinfer1, 50
- kUNSUPPORTED_GRAPH
 - nvonnxparser, 91
- kUNSUPPORTED_NODE
 - nvonnxparser, 91
- kUNSUPPORTED_STATE
 - nvinfer1, 51
- kUPDATE
 - nvinfer1, 65
- kUPPER
 - nvinfer1, 65
- kV2
 - nvinfer1, 60
- kV2_DYNAMICEXT
 - nvinfer1, 60
- kV2_EXT
 - nvinfer1, 60
- kV2_IOEXT
 - nvinfer1, 60
- kVALUE
 - nvinfer1::impl::EnumMaxImpl< ActivationType >, 111
 - nvinfer1::impl::EnumMaxImpl< AllocatorFlag >, 112

- `nvinfer1::impl::EnumMaxImpl< DataType >`, 113
 - `nvinfer1::impl::EnumMaxImpl< ElementWiseOperation >`, 113
 - `nvinfer1::impl::EnumMaxImpl< EngineCapability >`, 114
 - `nvinfer1::impl::EnumMaxImpl< ErrorCode >`, 115
 - `nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel >`, 116
 - `nvinfer1::impl::EnumMaxImpl< ILogger::Severity >`, 117
 - `nvinfer1::impl::EnumMaxImpl< InterpolationMode >`, 117
 - `nvinfer1::impl::EnumMaxImpl< PaddingMode >`, 118
 - `nvinfer1::impl::EnumMaxImpl< PoolingType >`, 119
 - `nvinfer1::impl::EnumMaxImpl< PreviewFeature >`, 119
 - `nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation >`, 120
 - `nvinfer1::impl::EnumMaxImpl< ResizeRoundMode >`, 121
 - `nvinfer1::impl::EnumMaxImpl< ResizeSelector >`, 121
 - `nvinfer1::impl::EnumMaxImpl< TensorFormat >`, 122
 - `nvinfer1::impl::EnumMaxImpl< TensorIOMode >`, 123
 - `nvinfer1::impl::EnumMaxImpl< TensorLocation >`, 124
- `kVECTOR`
 - `nvinfer1`, 55
- `kVERBOSE`
 - `nvinfer1`, 62
 - `nvinfer1::ILogger`, 397
- `kVERSION_COMPATIBLE`
 - `nvinfer1`, 45
- `kWARNING`
 - `nvinfer1::ILogger`, 397
- `kWHILE`
 - `nvinfer1`, 73
- `kWORKSPACE`
 - `nvinfer1`, 55
- `kWRAP`
 - `nvinfer1`, 68
- `kXOR`
 - `nvinfer1`, 48
- `LayerInformationFormat`
 - `nvinfer1`, 52
- `LayerType`
 - `nvinfer1`, 53
- `leaf`
 - `nvinfer1::plugin::softmaxTree`, 690
- `length`
 - `nvinfer1::PluginField`, 676
 - `nvuffparser::FieldMap`, 126
- `line`
 - `nvonnxparser::IParserError`, 503
- `loadLibrary`
 - `nvinfer1::IPluginRegistry`, 517
- `loadRuntime`
 - `nvinfer1::IRuntime`, 613
- `log`
 - `nvinfer1::ILogger`, 398
- `LoopOutput`
 - `nvinfer1`, 54
- `markOutput`
 - `nvinfer1::INetworkDefinition`, 455
- `markOutputForShapes`
 - `nvinfer1::INetworkDefinition`, 456
- `MatrixOperation`
 - `nvinfer1`, 55
- `max`
 - `nvinfer1::DynamicPluginTensorDesc`, 110
- `MAX_DIMS`
 - `nvinfer1::Dims32`, 104
- `maxSize`
 - `nvinfer1::plugin::GridAnchorParameters`, 128
 - `nvinfer1::plugin::PriorBoxParameters`, 683
- `mBoundary`
 - `nvinfer1::IfConditionalBoundaryLayer`, 372
 - `nvinfer1::ILoopBoundaryLayer`, 404
- `MemoryPoolType`
 - `nvinfer1`, 55
- `mImpl`
 - `nvinfer1::consistency::IConsistencyChecker`, 197
 - `nvinfer1::IActivationLayer`, 132
 - `nvinfer1::IAlgorithm`, 135
 - `nvinfer1::IAlgorithmContext`, 138
 - `nvinfer1::IAlgorithmIOInfo`, 141
 - `nvinfer1::IAlgorithmVariant`, 145
 - `nvinfer1::IAssertionLayer`, 147
 - `nvinfer1::IBuilder`, 160
 - `nvinfer1::IBuilderConfig`, 185
 - `nvinfer1::ICastLayer`, 192
 - `nvinfer1::IConcatenationLayer`, 194
 - `nvinfer1::IConditionLayer`, 195
 - `nvinfer1::IConstantLayer`, 200
 - `nvinfer1::IConvolutionLayer`, 213
 - `nvinfer1::ICudaEngine`, 241
 - `nvinfer1::IDeconvolutionLayer`, 272
 - `nvinfer1::IDequantizeLayer`, 274
 - `nvinfer1::IDimensionExpr`, 276
 - `nvinfer1::IEinsumLayer`, 279
 - `nvinfer1::IElementWiseLayer`, 281
 - `nvinfer1::IEngineInspector`, 286

- nvinfer1::IExecutionContext, [323](#)
 - nvinfer1::IExprBuilder, [340](#)
 - nvinfer1::IFillLayer, [346](#)
 - nvinfer1::IFullyConnectedLayer, [350](#)
 - nvinfer1::IGatherLayer, [356](#)
 - nvinfer1::IGridSampleLayer, [363](#)
 - nvinfer1::IHostMemory, [366](#)
 - nvinfer1::IIdentityLayer, [367](#)
 - nvinfer1::IIfConditional, [371](#)
 - nvinfer1::IIfConditionalInputLayer, [373](#)
 - nvinfer1::IIfConditionalOutputLayer, [375](#)
 - nvinfer1::IIteratorLayer, [386](#)
 - nvinfer1::ILoop, [403](#)
 - nvinfer1::ILoopOutputLayer, [407](#)
 - nvinfer1::ILRNLayer, [411](#)
 - nvinfer1::IMatrixMultiplyLayer, [414](#)
 - nvinfer1::INetworkDefinition, [459](#)
 - nvinfer1::INMSLayer, [463](#)
 - nvinfer1::INonZeroLayer, [467](#)
 - nvinfer1::INormalizationLayer, [471](#)
 - nvinfer1::IOneHotLayer, [473](#)
 - nvinfer1::IOptimizationProfile, [486](#)
 - nvinfer1::IPaddingLayer, [493](#)
 - nvinfer1::IParametricReLULayer, [494](#)
 - nvinfer1::IPluginV2Layer, [556](#)
 - nvinfer1::IPoolingLayer, [566](#)
 - nvinfer1::IQuantizeLayer, [570](#)
 - nvinfer1::IRaggedSoftMaxLayer, [572](#)
 - nvinfer1::IRecurrenceLayer, [573](#)
 - nvinfer1::IReduceLayer, [577](#)
 - nvinfer1::IRefitter, [586](#)
 - nvinfer1::IResizeLayer, [595](#)
 - nvinfer1::IReverseSequenceLayer, [598](#)
 - nvinfer1::IRNNv2Layer, [607](#)
 - nvinfer1::IRuntime, [617](#)
 - nvinfer1::IScaleLayer, [627](#)
 - nvinfer1::IScatterLayer, [630](#)
 - nvinfer1::ISelectLayer, [631](#)
 - nvinfer1::IShapeLayer, [632](#)
 - nvinfer1::IShuffleLayer, [638](#)
 - nvinfer1::ISliceLayer, [644](#)
 - nvinfer1::ISoftMaxLayer, [646](#)
 - nvinfer1::ITensor, [657](#)
 - nvinfer1::ITimingCache, [660](#)
 - nvinfer1::ITopKLayer, [664](#)
 - nvinfer1::ITripLimitLayer, [665](#)
 - nvinfer1::IUnaryLayer, [672](#)
- min
 - nvinfer1::DynamicPluginTensorDesc, [110](#)
- minBoxSize
 - nvinfer1::plugin::RPROIParams, [687](#)
- minSize
 - nvinfer1::plugin::GridAnchorParameters, [128](#)
 - nvinfer1::plugin::PriorBoxParameters, [683](#)
- mLayer
 - nvinfer1::ILayer, [396](#)
- n
 - nvinfer1::plugin::softmaxTree, [691](#)
- name
 - nvinfer1::plugin::softmaxTree, [691](#)
 - nvinfer1::PluginField, [676](#)
 - nvuffparser::FieldMap, [126](#)
- nbDims
 - nvinfer1::Dims32, [104](#)
 - nvinfer1::DimsExprs, [106](#)
- nbFields
 - nvinfer1::PluginFieldCollection, [677](#)
 - nvuffparser::FieldCollection, [124](#)
- NetworkDefinitionCreationFlag
 - nvinfer1, [56](#)
- NetworkDefinitionCreationFlags
 - nvinfer1, [41](#)
- nmsMaxOut
 - nvinfer1::plugin::RPROIParams, [688](#)
- nmsThreshold
 - nvinfer1::plugin::DetectionOutputParameters, [99](#)
- node
 - nvonnxparser::IParserError, [503](#)
- notifyShape
 - nvinfer1::IOutputAllocator, [488](#)
- num
 - nvinfer1::plugin::RegionParameters, [686](#)
- numAspectRatios
 - nvinfer1::plugin::GridAnchorParameters, [128](#)
 - nvinfer1::plugin::PriorBoxParameters, [683](#)
- numClasses
 - nvinfer1::plugin::DetectionOutputParameters, [99](#)
 - nvinfer1::plugin::NMSParameters, [673](#)
- numMaxSize
 - nvinfer1::plugin::PriorBoxParameters, [683](#)
- numMinSize
 - nvinfer1::plugin::PriorBoxParameters, [683](#)
- NV_ONNX_PARSER_MAJOR
 - NvOnnxParser.h, [811](#)
- NV_ONNX_PARSER_MINOR
 - NvOnnxParser.h, [811](#)
- NV_ONNX_PARSER_PATCH
 - NvOnnxParser.h, [811](#)
- NV_TENSORRT_BUILD
 - NvInferVersion.h, [801](#)
- NV_TENSORRT_LWS_MAJOR
 - NvInferVersion.h, [801](#)
- NV_TENSORRT_LWS_MINOR
 - NvInferVersion.h, [801](#)
- NV_TENSORRT_LWS_PATCH
 - NvInferVersion.h, [801](#)
- NV_TENSORRT_MAJOR

- NvInferVersion.h, 801
- NV_TENSORRT_MINOR
 - NvInferVersion.h, 802
- NV_TENSORRT_PATCH
 - NvInferVersion.h, 802
- NV_TENSORRT_RELEASE_TYPE
 - NvInferVersion.h, 802
- NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS
 - NvInferVersion.h, 802
- NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY
 - NvInferVersion.h, 802
- NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE
 - NvInferVersion.h, 802
- NV_TENSORRT_SONAME_MAJOR
 - NvInferVersion.h, 803
- NV_TENSORRT_SONAME_MINOR
 - NvInferVersion.h, 803
- NV_TENSORRT_SONAME_PATCH
 - NvInferVersion.h, 803
- NV_TENSORRT_VERSION
 - NvInferRuntimeBase.h, 784
- NvCaffeParser.h, 693, 694
- nvcaffeparser1, 29
 - createCaffeParser, 29
 - shutdownProtobufLibrary, 30
- nvcaffeparser1::IBinaryProtoBlob, 147
 - ~IBinaryProtoBlob, 148
 - destroy, 148
 - getData, 148
 - getDataType, 148
 - getDimensions, 148
- nvcaffeparser1::IBlobNameToTensor, 149
 - ~IBlobNameToTensor, 149
 - find, 150
- nvcaffeparser1::ICaffeParser, 185
 - ~ICaffeParser, 186
 - destroy, 186
 - getErrorRecorder, 186
 - parse, 187
 - parseBinaryProto, 187
 - parseBuffers, 188
 - setErrorRecorder, 189
 - setPluginFactoryV2, 189
 - setPluginNamespace, 189
 - setProtobufBufferSize, 190
- nvcaffeparser1::IPluginFactoryV2, 512
 - ~IPluginFactoryV2, 512
 - createPlugin, 512
 - isPluginV2, 513
- NvInfer.h, 695, 702
- nvinfer1, 30
 - ActivationType, 43
 - AllocatorFlag, 43
 - AllocatorFlags, 40
 - AsciiChar, 40
 - BoundingBoxFormat, 44
 - BuilderFlag, 44
 - BuilderFlags, 40
 - CalibrationAlgoType, 46
 - char_t, 40
 - DataType, 46
 - DeviceType, 47
 - DimensionOperation, 47
 - Dims, 40
 - ElementWiseOperation, 48
 - EngineCapability, 49
 - EnumMax, 74
 - EnumMax< BoundingBoxFormat >, 75
 - EnumMax< BuilderFlag >, 75
 - EnumMax< CalibrationAlgoType >, 75
 - EnumMax< DeviceType >, 75
 - EnumMax< DimensionOperation >, 76
 - EnumMax< FillOperation >, 76
 - EnumMax< GatherMode >, 76
 - EnumMax< LayerInformationFormat >, 76
 - EnumMax< LayerType >, 77
 - EnumMax< LoopOutput >, 77
 - EnumMax< MatrixOperation >, 77
 - EnumMax< MemoryPoolType >, 77
 - EnumMax< NetworkDefinitionCreationFlag >, 78
 - EnumMax< OptProfileSelector >, 78
 - EnumMax< ProfilingVerbosity >, 78
 - EnumMax< QuantizationFlag >, 78
 - EnumMax< ReduceOperation >, 79
 - EnumMax< RNNDirection >, 79
 - EnumMax< RNNGateType >, 79
 - EnumMax< RNNInputMode >, 79
 - EnumMax< RNNOperation >, 80
 - EnumMax< SampleMode >, 80
 - EnumMax< ScaleMode >, 80
 - EnumMax< ScatterMode >, 80
 - EnumMax< TacticSource >, 81
 - EnumMax< TempfileControlFlag >, 81
 - EnumMax< TopKOperation >, 81
 - EnumMax< TripLimit >, 81
 - EnumMax< UnaryOperation >, 82
 - EnumMax< WeightsRole >, 82
 - ErrorCode, 49
 - FillOperation, 51
 - GatherMode, 51
 - getBuilderPluginRegistry, 82
 - getBuilderSafePluginRegistry, 82
 - HardwareCompatibilityLevel, 52
 - InterpolationMode, 52
 - kABS, 73
 - kACOS, 74
 - kACOSH, 74
 - kACTIVATION, 53

- kALIGN_CORNERS, 64
- kALLOW_IN_MEMORY_FILES, 70
- kALLOW_TEMPORARY_FILES, 70
- kAMPERE.PLUS, 52
- kAND, 48
- kANY, 74
- kASIN, 74
- kASINH, 74
- kASSERTION, 54
- kASYMMETRIC, 64
- kATAN, 74
- kATANH, 74
- kAVERAGE, 61
- kAVG, 63
- kBIAS, 74
- kBIDIRECTION, 65
- kBOOL, 47
- kCAFFE_ROUND_DOWN, 60
- kCAFFE_ROUND_UP, 60
- kCALIBRATE_BEFORE_FUSION, 63
- kCAST, 54
- kCDHW32, 71
- kCEIL, 64, 74
- kCEIL_DIV, 48
- kCELL, 65
- kCENTER_SIZES, 44
- kCHANNEL, 68
- kCHAR, 60
- kCHW16, 71
- kCHW2, 70
- kCHW32, 71
- kCHW4, 71
- kCLAMP, 68
- kCLIP, 43
- kCONCATENATE, 54
- kCONCATENATION, 53
- kCONDITION, 54
- kCONDITIONAL_INPUT, 54
- kCONDITIONAL_OUTPUT, 54
- kCONSTANT, 54, 74
- kCONVOLUTION, 53
- kCORNER_PAIRS, 44
- kCOS, 73
- kCOSH, 74
- kCOUNT, 73
- kCUBIC, 52
- kCUBLAS, 69
- kCUBLAS_LT, 69
- kCUDNN, 69
- kDEBUG, 44
- kDECONVOLUTION, 53
- kDEFAULT, 49, 52, 62, 67
- kDEQUANTIZE, 54
- kDETAILED, 62
- kDEVICE, 72
- kDHW, 72
- kDHW8, 71
- kDIMS, 60
- kDIRECT_IO, 45
- kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805, 61
- kDISABLE_TIMING_CACHE, 45
- kDIV, 48
- kDLA, 47
- kDLA_GLOBAL_DRAM, 56
- kDLA_HWC4, 71
- kDLA_LINEAR, 71
- kDLA_LOCAL_DRAM, 56
- kDLA_MANAGED_SRAM, 55
- kDLA_STANDALONE, 49
- kEDGE_MASK_CONVOLUTIONS, 69
- kEINSUM, 54
- kELEMENT, 52, 68
- kELEMENTWISE, 53, 68
- kELU, 43
- kENABLE_TACTIC_HEURISTIC, 45
- kENTROPY_CALIBRATION, 46
- kENTROPY_CALIBRATION_2, 46
- kEQUAL, 48
- kERF, 74
- kEXCLUDE_LEAN_RUNTIME, 46
- kEXP, 73
- kEXPLICIT_BATCH, 56
- kEXPLICIT_PRECISION, 56
- kEXPLICIT_ROUND_DOWN, 60
- kEXPLICIT_ROUND_UP, 60
- kFAILED_ALLOCATION, 50
- kFAILED_COMPUTATION, 50
- kFAILED_EXECUTION, 50
- kFAILED_INITIALIZATION, 50
- kFASTER_DYNAMIC_SHAPES_0805, 61
- kFILL, 54, 68
- kFLOAT, 46
- kFLOAT16, 60
- kFLOAT32, 60
- kFLOAT64, 60
- kFLOOR, 64, 74
- kFLOOR_DIV, 48
- kFORGET, 65
- kFORMULA, 65
- kFP16, 44
- kFP8, 46, 47
- kFULLY_CONNECTED, 53
- kGATHER, 53
- kGPU, 47
- kGPU_FALLBACK, 44
- kGREATER, 48
- kGRID_SAMPLE, 54

kGRU, 67
kHALF, 46
kHALF_DOWN, 64
kHALF_PIXEL, 64
kHALF_UP, 64
kHARD_SIGMOID, 43
kHIDDEN, 65
kHOST, 72
kHWC, 71
kHWC16, 71
kHWC8, 70
kIDENTITY, 54
kINPUT, 65, 72
kINT16, 60
kINT32, 47, 60
kINT8, 44, 47, 60
kINTERNAL_ERROR, 50
kINVALID_ARGUMENT, 50
kINVALID_CONFIG, 50
kINVALID_STATE, 50
kISINF, 74
kITERATOR, 54
kJIT_CONVOLUTIONS, 69
kJSON, 53
kKERNEL, 74
kLAST_VALUE, 54
kLAYER_NAMES_ONLY, 62
kLEAKY_RELU, 43
kLEGACY_CALIBRATION, 46
kLESS, 48
kLINEAR, 52, 66, 70
kLinspace, 51
kLOG, 73
kLOOP_OUTPUT, 54
kLRN, 53
kLSTM, 67
kMATRIX_MULTIPLY, 53
kMAX, 48, 57, 61, 63, 72
kMAX_AVERAGE_BLEND, 61
kMIN, 48, 57, 63, 72
kMINMAX_CALIBRATION, 46
kND, 52, 68
kNEAREST, 52
kNEG, 73
kNMS, 54
kNON_ZERO, 54
kNONE, 52, 55, 62, 72
kNORMALIZATION, 54
kNOT, 74
kOBEY_PRECISION_CONSTRAINTS, 45
kONE_HOT, 54
kONELINE, 53
kOPT, 57
kOR, 48
kOUTPUT, 65, 72
kPADDING, 53
kPARAMETRIC_RELU, 54
kPLUGIN, 53
kPLUGIN_V2, 54
kPOOLING, 53
kPOW, 48
kPREFER_PRECISION_CONSTRAINTS, 45
kPROD, 48, 63
kPROFILE_SHARING_0806, 62
kQUANTIZE, 54
kRAGGED_SOFTMAX, 53
kRANDOM_NORMAL, 51
kRANDOM_UNIFORM, 51
kRECIP, 73
kRECURRENCE, 54
kREDUCE, 53
kREFIT, 45
kREFLECT, 68
kREJECT_EMPTY_ALGORITHMS, 45
kRELU, 43, 67
kRESET, 65
kRESIZABLE, 44
kRESIZE, 54
kREVERSE, 54
kREVERSE_SEQUENCE, 54
kRNN_V2, 54
kROUND, 74
kSAFE_DLA, 49
kSAFE_GPU, 49
kSAFETY, 49
kSAFETY_SCOPE, 45
kSAME_LOWER, 60
kSAME_UPPER, 60
kSCALE, 53, 74
kSCALED_TANH, 43
kSCATTER, 54
kSELECT, 54
kSELU, 43
kSHAPE, 54
kSHIFT, 74
kSHUFFLE, 53
kSIGMOID, 43
kSIGN, 74
kSIN, 73
kSINH, 74
kSKIP, 66
kSLICE, 54
kSOFTMAX, 53
kSOFTPLUS, 43
kSOFTSIGN, 43
kSPARSE_WEIGHTS, 45
kSQRT, 73
kSTANDARD, 49

- kSTRICT_BOUNDS, 67
- kSTRICT_TYPES, 45
- kSUB, 48
- kSUCCESS, 50
- kSUM, 48, 63
- KTACTIC_DRAM, 56
- KTAN, 73
- KTANH, 43, 67
- kTF32, 45
- kTHRESHOLDED_RELU, 43
- kTOPK, 53
- kTRANSPPOSE, 55
- kTRIP_LIMIT, 54
- kUINT8, 47
- kUNARY, 53
- kUNIDIRECTION, 65
- kUNIFORM, 68
- kUNKNOWN, 60
- kUNSPECIFIED_ERROR, 50
- kUNSUPPORTED_STATE, 51
- kUPDATE, 65
- kUPPER, 65
- kV2, 60
- kV2_DYNAMICEXT, 60
- kV2_EXT, 60
- kV2_IOEXT, 60
- kVECTOR, 55
- kVERBOSE, 62
- kVERSION_COMPATIBLE, 45
- kWHILE, 73
- kWORKSPACE, 55
- kWRAP, 68
- kXOR, 48
- LayerInformationFormat, 52
- LayerType, 53
- LoopOutput, 54
- MatrixOperation, 55
- MemoryPoolType, 55
- NetworkDefinitionCreationFlag, 56
- NetworkDefinitionCreationFlags, 41
- OptProfileSelector, 56
- PaddingMode, 57
- PluginFieldType, 60
- PluginFormat, 41
- PluginVersion, 60
- PoolingType, 60
- PreviewFeature, 61
- ProfilingVerbosity, 62
- QuantizationFlag, 62
- QuantizationFlags, 41
- ReduceOperation, 63
- ResizeCoordinateTransformation, 63
- ResizeMode, 41
- ResizeRoundMode, 64
- ResizeSelector, 64
- RNNDirection, 65
- RNNGateType, 65
- RNNInputMode, 66
- RNNOperation, 66
- SampleMode, 67
- ScaleMode, 68
- ScatterMode, 68
- SliceMode, 42
- TacticSource, 69
- TacticSources, 42
- TempfileControlFlag, 69
- TempfileControlFlags, 42
- TensorFormat, 70
- TensorFormats, 42
- TensorIOMode, 72
- TensorLocation, 72
- TopKOperation, 72
- TripLimit, 73
- UnaryOperation, 73
- WeightsRole, 74
- nvInfer1::consistency, 83
- nvInfer1::consistency::IConsistencyChecker, 195
 - ~IConsistencyChecker, 196
 - IConsistencyChecker, 196, 197
 - mImpl, 197
 - operator=, 197
 - validate, 197
- nvInfer1::consistency::IPluginChecker, 504
 - ~IPluginChecker, 505
 - IPluginChecker, 505
 - operator=, 505
 - validate, 506
- nvInfer1::Dims2, 101
 - Dims2, 101
- nvInfer1::Dims3, 102
 - Dims3, 102, 103
- nvInfer1::Dims32, 103
 - d, 104
 - MAX_DIMS, 104
 - nbDims, 104
- nvInfer1::Dims4, 105
 - Dims4, 105
- nvInfer1::DimsExprs, 106
 - d, 106
 - nbDims, 106
- nvInfer1::DimsHW, 107
 - DimsHW, 108
 - h, 108
 - w, 109
- nvInfer1::DynamicPluginTensorDesc, 109
 - desc, 110
 - max, 110
 - min, 110

- `nvInfer1::IActivationLayer`, 129
 - `~IActivationLayer`, 130
 - `getActivationType`, 130
 - `getAlpha`, 130
 - `getBeta`, 131
 - `mImpl`, 132
 - `setActivationType`, 131
 - `setAlpha`, 131
 - `setBeta`, 132
- `nvInfer1::IAlgorithm`, 133
 - `~IAlgorithm`, 134
 - `getAlgorithmIOInfo`, 134
 - `getAlgorithmIOInfoByIndex`, 134
 - `getAlgorithmVariant`, 135
 - `getTimingMSec`, 135
 - `getWorkspaceSize`, 135
 - `mImpl`, 135
- `nvInfer1::IAlgorithmContext`, 136
 - `~IAlgorithmContext`, 137
 - `getDimensions`, 137
 - `getName`, 137
 - `getNbInputs`, 137
 - `getNbOutputs`, 137
 - `mImpl`, 138
- `nvInfer1::IAlgorithmIOInfo`, 138
 - `~IAlgorithmIOInfo`, 139
 - `getComponentsPerElement`, 139
 - `getDataType`, 139
 - `getStrides`, 140
 - `getTensorFormat`, 140
 - `getVectorizedDim`, 140
 - `mImpl`, 141
- `nvInfer1::IAlgorithmSelector`, 141
 - `~IAlgorithmSelector`, 141
 - `reportAlgorithms`, 142
 - `selectAlgorithms`, 142
- `nvInfer1::IAlgorithmVariant`, 143
 - `~IAlgorithmVariant`, 144
 - `getImplementation`, 144
 - `getTactic`, 144
 - `mImpl`, 145
- `nvInfer1::IAssertionLayer`, 145
 - `~IAssertionLayer`, 146
 - `getMessage`, 146
 - `mImpl`, 147
 - `setMessage`, 146
- `nvInfer1::IBuilder`, 150
 - `~IBuilder`, 152
 - `buildEngineWithConfig`, 152
 - `buildSerializedNetwork`, 152
 - `createBuilderConfig`, 153
 - `createNetworkV2`, 153
 - `createOptimizationProfile`, 154
 - `destroy`, 154
 - `getErrorRecorder`, 154
 - `getLogger`, 155
 - `getMaxBatchSize`, 155
 - `getMaxDLABatchSize`, 155
 - `getMaxThreads`, 156
 - `getNbDLACores`, 156
 - `getPluginRegistry`, 156
 - `isNetworkSupported`, 156
 - `mImpl`, 160
 - `platformHasFastFp16`, 157
 - `platformHasFastInt8`, 157
 - `platformHasTf32`, 157
 - `reset`, 157
 - `setErrorRecorder`, 158
 - `setGpuAllocator`, 158
 - `setMaxBatchSize`, 159
 - `setMaxThreads`, 159
- `nvInfer1::IBuilderConfig`, 160
 - `~IBuilderConfig`, 163
 - `addOptimizationProfile`, 163
 - `canRunOnDLA`, 164
 - `clearFlag`, 164
 - `clearQuantizationFlag`, 164
 - `createTimingCache`, 164
 - `destroy`, 165
 - `getAlgorithmSelector`, 165
 - `getAvgTimingIterations`, 166
 - `getBuilderOptimizationLevel`, 166
 - `getCalibrationProfile`, 166
 - `getDefaultDeviceType`, 166
 - `getDeviceType`, 167
 - `getDLACore`, 167
 - `getEngineCapability`, 167
 - `getFlag`, 167
 - `getFlags`, 168
 - `getHardwareCompatibilityLevel`, 168
 - `getInt8Calibrator`, 168
 - `getMaxAuxStreams`, 169
 - `getMaxWorkspaceSize`, 169
 - `getMemoryPoolLimit`, 169
 - `getMinTimingIterations`, 170
 - `getNbOptimizationProfiles`, 170
 - `getNbPluginsToSerialize`, 170
 - `getPluginToSerialize`, 171
 - `getPreviewFeature`, 171
 - `getProfileStream`, 172
 - `getProfilingVerbosity`, 172
 - `getQuantizationFlag`, 172
 - `getQuantizationFlags`, 172
 - `getTacticSources`, 173
 - `getTimingCache`, 173
 - `isDeviceTypeSet`, 173
 - `mImpl`, 185
 - `reset`, 174

- resetDeviceType, 174
- setAlgorithmSelector, 174
- setAvgTimingIterations, 175
- setBuilderOptimizationLevel, 175
- setCalibrationProfile, 175
- setDefaultDeviceType, 176
- setDeviceType, 176
- setDLACore, 177
- setEngineCapability, 177
- setFlag, 177
- setFlags, 178
- setHardwareCompatibilityLevel, 178
- setInt8Calibrator, 179
- setMaxAuxStreams, 179
- setMaxWorkspaceSize, 180
- setMemoryPoolLimit, 180
- setMinTimingIterations, 181
- setPluginsToSerialize, 181
- setPreviewFeature, 182
- setProfileStream, 182
- setProfilingVerbosity, 182
- setQuantizationFlag, 183
- setQuantizationFlags, 183
- setTacticSources, 183
- setTimingCache, 184
- nvinfer1::ICastLayer, 190
 - ~ICastLayer, 191
 - getToType, 191
 - mImpl, 192
 - setToType, 191
- nvinfer1::IConcatenationLayer, 192
 - ~IConcatenationLayer, 193
 - getAxis, 193
 - mImpl, 194
 - setAxis, 193
- nvinfer1::IConditionLayer, 194
 - ~IConditionLayer, 195
 - mImpl, 195
- nvinfer1::IConstantLayer, 198
 - ~IConstantLayer, 199
 - getDimensions, 199
 - getWeights, 199
 - mImpl, 200
 - setDimensions, 199
 - setWeights, 200
- nvinfer1::IConvolutionLayer, 200
 - ~IConvolutionLayer, 202
 - getBiasWeights, 203
 - getDilation, 203
 - getDilationNd, 203
 - getKernelSize, 203
 - getKernelSizeNd, 204
 - getKernelWeights, 204
 - getNbGroups, 204
 - getNbOutputMaps, 204
 - getPadding, 205
 - getPaddingMode, 205
 - getPaddingNd, 205
 - getPostPadding, 206
 - getPrePadding, 206
 - getStride, 206
 - getStrideNd, 206
 - mImpl, 213
 - setBiasWeights, 207
 - setDilation, 207
 - setDilationNd, 207
 - setInput, 208
 - setKernelSize, 208
 - setKernelSizeNd, 209
 - setKernelWeights, 209
 - setNbGroups, 209
 - setNbOutputMaps, 210
 - setPadding, 210
 - setPaddingMode, 211
 - setPaddingNd, 211
 - setPostPadding, 211
 - setPrePadding, 212
 - setStride, 212
 - setStrideNd, 212
- nvinfer1::ICudaEngine, 213
 - ~ICudaEngine, 216
 - bindingIsInput, 217
 - createEngineInspector, 217
 - createExecutionContext, 217
 - createExecutionContextWithoutDeviceMemory, 218
 - destroy, 218
 - getBindingBytesPerComponent, 218
 - getBindingComponentsPerElement, 219
 - getBindingDataType, 219
 - getBindingDimensions, 220
 - getBindingFormat, 220
 - getBindingFormatDesc, 221
 - getBindingIndex, 221
 - getBindingName, 222
 - getBindingVectorizedDim, 223
 - getDeviceMemorySize, 223
 - getEngineCapability, 223
 - getErrorRecorder, 224
 - getHardwareCompatibilityLevel, 224
 - getIOTensorName, 224
 - getLocation, 225
 - getMaxBatchSize, 225
 - getName, 226
 - getNbAuxStreams, 226
 - getNbBindings, 226
 - getNbIOTensors, 227
 - getNbLayers, 227
 - getNbOptimizationProfiles, 227

- getProfileDimensions, 228
- getProfileShape, 229
- getProfileShapeValues, 229
- getProfilingVerbosity, 230
- getTacticSources, 230
- getTensorBytesPerComponent, 230, 232
- getTensorComponentsPerElement, 232, 233
- getTensorDataType, 234
- getTensorFormat, 234
- getTensorFormatDesc, 235
- getTensorIOMode, 236
- getTensorLocation, 236
- getTensorShape, 237
- getTensorVectorizedDim, 237, 238
- hasImplicitBatchDimension, 238
- isExecutionBinding, 239
- isRefittable, 239
- isShapeBinding, 239
- isShapeInferenceIO, 240
- mImpl, 241
- serialize, 240
- setErrorRecorder, 241
- nvinfer1::IDeconvolutionLayer, 261
 - ~IDeconvolutionLayer, 263
 - getBiasWeights, 263
 - getDilationNd, 263
 - getKernelSize, 263
 - getKernelSizeNd, 264
 - getKernelWeights, 264
 - getNbGroups, 264
 - getNbOutputMaps, 264
 - getPadding, 265
 - getPaddingMode, 265
 - getPaddingNd, 265
 - getPostPadding, 266
 - getPrePadding, 266
 - getStride, 266
 - getStrideNd, 266
 - mImpl, 272
 - setBiasWeights, 267
 - setDilationNd, 267
 - setInput, 267
 - setKernelSize, 268
 - setKernelSizeNd, 268
 - setKernelWeights, 268
 - setNbGroups, 269
 - setNbOutputMaps, 269
 - setPadding, 269
 - setPaddingMode, 270
 - setPaddingNd, 270
 - setPostPadding, 270
 - setPrePadding, 271
 - setStride, 271
 - setStrideNd, 271
- nvinfer1::IDequantizeLayer, 272
 - ~IDequantizeLayer, 274
 - getAxis, 274
 - mImpl, 274
 - setAxis, 274
- nvinfer1::IDimensionExpr, 275
 - ~IDimensionExpr, 276
 - getConstantValue, 276
 - isConstant, 276
 - mImpl, 276
- nvinfer1::IEinsumLayer, 277
 - ~IEinsumLayer, 278
 - getEquation, 278
 - mImpl, 279
 - setEquation, 278
- nvinfer1::IElementWiseLayer, 279
 - ~IElementWiseLayer, 280
 - getOperation, 280
 - mImpl, 281
 - setOperation, 280
- nvinfer1::IEngineInspector, 281
 - ~IEngineInspector, 282
 - getEngineInformation, 283
 - getErrorRecorder, 283
 - getExecutionContext, 284
 - getLayerInformation, 284
 - mImpl, 286
 - setErrorRecorder, 285
 - setExecutionContext, 285
- nvinfer1::IErrorRecorder, 286
 - ~IErrorRecorder, 288
 - clear, 288
 - decRefCount, 288
 - ErrorDesc, 287
 - getErrorCode, 289
 - getErrorDesc, 290
 - getNbErrors, 290
 - hasOverflowed, 291
 - IErrorRecorder, 287
 - incRefCount, 291
 - kMAX_DESC_LENGTH, 293
 - RefCount, 287
 - reportError, 292
- nvinfer1::IExecutionContext, 293
 - ~IExecutionContext, 296
 - allInputDimensionsSpecified, 296
 - allInputShapesSpecified, 297
 - destroy, 297
 - enqueue, 298
 - enqueueV2, 298
 - enqueueV3, 299
 - execute, 300
 - executeV2, 300
 - getBindingDimensions, 301

- getDebugSync, 302
- getEngine, 302
- getEnqueueEmitsProfile, 302
- getErrorRecorder, 303
- getInputConsumedEvent, 303
- getMaxOutputSize, 303
- getName, 304
- getNvtxVerbosity, 304
- getOptimizationProfile, 304
- getOutputAllocator, 305
- getOutputTensorAddress, 305
- getPersistentCacheLimit, 306
- getProfiler, 306
- getShapeBinding, 306
- getStrides, 307
- getTemporaryStorageAllocator, 307
- getTensorAddress, 308
- getTensorShape, 308
- getTensorStrides, 309
- inferShapes, 310
- mImpl, 323
- reportToProfiler, 310
- setAuxStreams, 311
- setBindingDimensions, 312
- setDebugSync, 313
- setDeviceMemory, 313
- setEnqueueEmitsProfile, 313
- setErrorRecorder, 314
- setInputConsumedEvent, 314
- setInputShape, 315
- setInputShapeBinding, 315
- setInputTensorAddress, 316
- setName, 317
- setNvtxVerbosity, 317
- setOptimizationProfile, 318
- setOptimizationProfileAsync, 319
- setOutputAllocator, 320
- setPersistentCacheLimit, 320
- setProfiler, 321
- setTemporaryStorageAllocator, 321
- setTensorAddress, 322
- nvinfer1::IExprBuilder, 338
 - ~IExprBuilder, 339
 - constant, 339
 - mImpl, 340
 - operation, 339
- nvinfer1::IFillLayer, 340
 - ~IFillLayer, 341
 - getAlpha, 342
 - getBeta, 342
 - getDimensions, 342
 - getOperation, 343
 - mImpl, 346
 - setAlpha, 343
 - setBeta, 344
 - setDimensions, 344
 - setInput, 344
 - setOperation, 345
- nvinfer1::IFullyConnectedLayer, 346
 - ~IFullyConnectedLayer, 348
 - getBiasWeights, 348
 - getKernelWeights, 348
 - getNbOutputChannels, 348
 - mImpl, 350
 - setBiasWeights, 349
 - setInput, 349
 - setKernelWeights, 350
 - setNbOutputChannels, 350
- nvinfer1::IGatherLayer, 351
 - ~IGatherLayer, 353
 - getGatherAxis, 354
 - getMode, 354
 - getNbElementWiseDims, 354
 - mImpl, 356
 - setGatherAxis, 354
 - setMode, 355
 - setNbElementWiseDims, 355
- nvinfer1::IGpuAllocator, 356
 - ~IGpuAllocator, 356
 - allocate, 357
 - deallocate, 358
 - free, 358
 - IGpuAllocator, 357
 - reallocate, 359
- nvinfer1::IGridSampleLayer, 360
 - ~IGridSampleLayer, 361
 - getAlignCorners, 362
 - getInterpolationMode, 362
 - getSampleMode, 362
 - mImpl, 363
 - setAlignCorners, 362
 - setInterpolationMode, 363
 - setSampleMode, 363
- nvinfer1::IHostMemory, 364
 - ~IHostMemory, 365
 - data, 365
 - destroy, 365
 - mImpl, 366
 - size, 365
 - type, 365
- nvinfer1::IIdentityLayer, 366
 - ~IIdentityLayer, 367
 - mImpl, 367
- nvinfer1::IIfConditional, 368
 - ~IIfConditional, 369
 - addInput, 369
 - addOutput, 369
 - getName, 370

- mImpl, 371
 - setCondition, 370
 - setName, 370
- nvinfer1::IfConditionalBoundaryLayer, 371
 - ~IfConditionalBoundaryLayer, 372
 - getConditional, 372
 - mBoundary, 372
- nvinfer1::IfConditionalInputLayer, 373
 - ~IfConditionalInputLayer, 373
 - mImpl, 373
- nvinfer1::IfConditionalOutputLayer, 374
 - ~IfConditionalOutputLayer, 374
 - mImpl, 375
- nvinfer1::Int8Calibrator, 375
 - ~Int8Calibrator, 376
 - getAlgorithm, 376
 - getBatch, 376
 - getBatchSize, 377
 - readCalibrationCache, 377
 - writeCalibrationCache, 378
- nvinfer1::Int8EntropyCalibrator, 378
 - ~Int8EntropyCalibrator, 379
 - getAlgorithm, 379
- nvinfer1::Int8EntropyCalibrator2, 379
 - ~Int8EntropyCalibrator2, 380
 - getAlgorithm, 380
- nvinfer1::Int8LegacyCalibrator, 380
 - ~Int8LegacyCalibrator, 381
 - getAlgorithm, 381
 - getQuantile, 381
 - getRegressionCutoff, 382
 - readHistogramCache, 382
 - writeHistogramCache, 382
- nvinfer1::Int8MinMaxCalibrator, 383
 - ~Int8MinMaxCalibrator, 383
 - getAlgorithm, 384
- nvinfer1::IteratorLayer, 384
 - ~IteratorLayer, 385
 - getAxis, 385
 - getReverse, 385
 - mImpl, 386
 - setAxis, 385
 - setReverse, 385
- nvinfer1::ILayer, 386
 - ~ILayer, 388
 - getInput, 389
 - getMetadata, 389
 - getName, 389
 - getNbInputs, 389
 - getNbOutputs, 390
 - getOutput, 390
 - getOutputType, 390
 - getPrecision, 391
 - getType, 391
- mLayer, 396
 - outputTypeIsSet, 391
 - precisionIsSet, 392
 - resetOutputType, 392
 - resetPrecision, 392
 - setInput, 393
 - setMetadata, 393
 - setName, 394
 - setOutputType, 394
 - setPrecision, 395
- nvinfer1::ILogger, 396
 - ~ILogger, 397
 - ILogger, 397
 - kERROR, 397
 - kINFO, 397
 - kINTERNAL_ERROR, 397
 - kVERBOSE, 397
 - kWARNING, 397
 - log, 398
 - Severity, 397
- nvinfer1::ILoggerFinder, 398
 - ~ILoggerFinder, 399
 - findLogger, 399
- nvinfer1::ILoop, 400
 - ~ILoop, 400
 - addIterator, 401
 - addLoopOutput, 401
 - addRecurrence, 401
 - addTripLimit, 401
 - getName, 402
 - mImpl, 403
 - setName, 402
- nvinfer1::ILoopBoundaryLayer, 403
 - ~ILoopBoundaryLayer, 403
 - getLoop, 404
 - mBoundary, 404
- nvinfer1::ILoopOutputLayer, 404
 - ~ILoopOutputLayer, 405
 - getAxis, 406
 - getLoopOutput, 406
 - mImpl, 407
 - setAxis, 406
 - setInput, 406
- nvinfer1::ILRNLayer, 407
 - ~ILRNLayer, 408
 - getAlpha, 409
 - getBeta, 409
 - getK, 409
 - getWindowSize, 409
 - mImpl, 411
 - setAlpha, 410
 - setBeta, 410
 - setK, 410
 - setWindowSize, 411

nvinfer1::IMatrixMultiplyLayer, [412](#)
 ~IMatrixMultiplyLayer, [413](#)
 getOperation, [413](#)
 mImpl, [414](#)
 setOperation, [413](#)
 nvinfer1::impl, [83](#)
 nvinfer1::impl::EnumMaxImpl< ActivationType >, [111](#)
 kVALUE, [111](#)
 nvinfer1::impl::EnumMaxImpl< AllocatorFlag >, [112](#)
 kVALUE, [112](#)
 nvinfer1::impl::EnumMaxImpl< DataType >, [112](#)
 kVALUE, [113](#)
 nvinfer1::impl::EnumMaxImpl< ElementWiseOperation
 >, [113](#)
 kVALUE, [113](#)
 nvinfer1::impl::EnumMaxImpl< EngineCapability >, [114](#)
 kVALUE, [114](#)
 nvinfer1::impl::EnumMaxImpl< ErrorCode >, [115](#)
 kVALUE, [115](#)
 nvinfer1::impl::EnumMaxImpl< HardwareCompatibil-
 ityLevel >, [115](#)
 kVALUE, [116](#)
 nvinfer1::impl::EnumMaxImpl< ILogger::Severity >, [116](#)
 kVALUE, [117](#)
 nvinfer1::impl::EnumMaxImpl< InterpolationMode >, [117](#)
 kVALUE, [117](#)
 nvinfer1::impl::EnumMaxImpl< PaddingMode >, [118](#)
 kVALUE, [118](#)
 nvinfer1::impl::EnumMaxImpl< PoolingType >, [118](#)
 kVALUE, [119](#)
 nvinfer1::impl::EnumMaxImpl< PreviewFeature >, [119](#)
 kVALUE, [119](#)
 nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTrans-
 formation >, [120](#)
 kVALUE, [120](#)
 nvinfer1::impl::EnumMaxImpl< ResizeRoundMode >, [120](#)
 kVALUE, [121](#)
 nvinfer1::impl::EnumMaxImpl< ResizeSelector >, [121](#)
 kVALUE, [121](#)
 nvinfer1::impl::EnumMaxImpl< T >, [111](#)
 nvinfer1::impl::EnumMaxImpl< TensorFormat >, [122](#)
 kVALUE, [122](#)
 nvinfer1::impl::EnumMaxImpl< TensorIOMode >, [123](#)
 kVALUE, [123](#)
 nvinfer1::impl::EnumMaxImpl< TensorLocation >, [123](#)
 kVALUE, [124](#)
 nvinfer1::INetworkDefinition, [414](#)
 ~INetworkDefinition, [418](#)
 addActivation, [419](#)
 addAssertion, [419](#)
 addCast, [420](#)
 addConcatenation, [420](#)
 addConstant, [421](#)
 addConvolution, [421](#)
 addConvolutionNd, [422](#)
 addDeconvolution, [423](#)
 addDeconvolutionNd, [424](#)
 addDequantize, [425](#)
 addEinsum, [425](#)
 addElementWise, [426](#)
 addFill, [426](#)
 addFullyConnected, [427](#)
 addGather, [428](#)
 addGatherV2, [428](#)
 addGridSample, [429](#)
 addIdentity, [429](#)
 addIfConditional, [430](#)
 addInput, [430](#)
 addLoop, [431](#)
 addLRN, [431](#)
 addMatrixMultiply, [432](#)
 addNMS, [433](#)
 addNonZero, [433](#)
 addNormalization, [434](#)
 addOneHot, [434](#)
 addPadding, [435](#)
 addPaddingNd, [436](#)
 addParametricReLU, [436](#)
 addPluginV2, [437](#)
 addPooling, [437](#)
 addPoolingNd, [438](#)
 addQuantize, [439](#)
 addRaggedSoftMax, [439](#)
 addReduce, [440](#)
 addResize, [441](#)
 addReverseSequence, [441](#)
 addRNNv2, [442](#)
 addScale, [443](#)
 addScaleNd, [444](#)
 addScatter, [445](#)
 addSelect, [446](#)
 addShape, [447](#)
 addShuffle, [447](#)
 addSlice, [448](#)
 addSoftMax, [448](#)
 addTopK, [449](#)
 addUnary, [450](#)
 destroy, [450](#)
 getBuilder, [451](#)
 getErrorRecorder, [451](#)
 getInput, [451](#)
 getLayer, [452](#)
 getName, [452](#)
 getNbInputs, [453](#)

- getNbLayers, [453](#)
- getNbOutputs, [453](#)
- getOutput, [454](#)
- hasExplicitPrecision, [454](#)
- hasImplicitBatchDimension, [455](#)
- markOutput, [455](#)
- markOutputForShapes, [456](#)
- mImpl, [459](#)
- removeTensor, [456](#)
- setErrorRecorder, [457](#)
- setName, [457](#)
- setWeightsName, [458](#)
- unmarkOutput, [458](#)
- unmarkOutputForShapes, [459](#)
- nvInfer1::INMSLayer, [459](#)
 - ~INMSLayer, [461](#)
 - getBoundingBoxFormat, [461](#)
 - getTopKBoxLimit, [461](#)
 - mImpl, [463](#)
 - setBoundingBoxFormat, [462](#)
 - setInput, [462](#)
 - setTopKBoxLimit, [463](#)
- nvInfer1::INoCopy, [464](#)
 - ~INoCopy, [465](#)
 - INoCopy, [465](#)
 - operator=, [465](#)
- nvInfer1::INonZeroLayer, [466](#)
 - ~INonZeroLayer, [467](#)
 - mImpl, [467](#)
- nvInfer1::INormalizationLayer, [467](#)
 - ~INormalizationLayer, [468](#)
 - getAxes, [469](#)
 - getComputePrecision, [469](#)
 - getEpsilon, [469](#)
 - getNbGroups, [469](#)
 - mImpl, [471](#)
 - setAxes, [470](#)
 - setComputePrecision, [470](#)
 - setEpsilon, [470](#)
 - setNbGroups, [471](#)
- nvInfer1::IOneHotLayer, [472](#)
 - getAxis, [473](#)
 - mImpl, [473](#)
 - setAxis, [473](#)
- nvInfer1::IOptimizationProfile, [481](#)
 - ~IOptimizationProfile, [482](#)
 - getDimensions, [482](#)
 - getExtraMemoryTarget, [483](#)
 - getNbShapeValues, [483](#)
 - getShapeValues, [483](#)
 - isValid, [484](#)
 - mImpl, [486](#)
 - setDimensions, [484](#)
 - setExtraMemoryTarget, [485](#)
 - setShapeValues, [485](#)
- nvInfer1::IOutputAllocator, [487](#)
 - ~IOutputAllocator, [487](#)
 - getInterfaceVersion, [487](#)
 - notifyShape, [488](#)
 - reallocateOutput, [488](#)
- nvInfer1::IPaddingLayer, [489](#)
 - ~IPaddingLayer, [490](#)
 - getPostPadding, [490](#)
 - getPostPaddingNd, [490](#)
 - getPrePadding, [491](#)
 - getPrePaddingNd, [491](#)
 - mImpl, [493](#)
 - setPostPadding, [491](#)
 - setPostPaddingNd, [492](#)
 - setPrePadding, [492](#)
 - setPrePaddingNd, [492](#)
- nvInfer1::IParametricReLULayer, [493](#)
 - ~IParametricReLULayer, [494](#)
 - mImpl, [494](#)
- nvInfer1::IPluginCreator, [506](#)
 - ~IPluginCreator, [507](#)
 - createPlugin, [508](#)
 - deserializePlugin, [508](#)
 - getFieldNames, [509](#)
 - getPluginName, [509](#)
 - getPluginNamespace, [510](#)
 - getPluginVersion, [510](#)
 - getTensorRTVersion, [510](#)
 - IPluginCreator, [507](#)
 - setPluginNamespace, [511](#)
- nvInfer1::IPluginRegistry, [513](#)
 - ~IPluginRegistry, [515](#)
 - deregisterCreator, [515](#)
 - deregisterLibrary, [515](#)
 - getErrorRecorder, [516](#)
 - getPluginCreator, [516](#)
 - getPluginCreatorList, [517](#)
 - isParentSearchEnabled, [517](#)
 - loadLibrary, [517](#)
 - PluginLibraryHandle, [515](#)
 - registerCreator, [518](#)
 - setErrorRecorder, [518](#)
 - setParentSearchEnabled, [519](#)
- nvInfer1::IPluginV2, [525](#)
 - clone, [526](#)
 - configureWithFormat, [527](#)
 - destroy, [528](#)
 - enqueue, [528](#)
 - getNbOutputs, [529](#)
 - getOutputDimensions, [529](#)
 - getPluginNamespace, [531](#)
 - getPluginType, [531](#)
 - getPluginVersion, [532](#)

- getSerializationSize, 532
 - getTensorRTVersion, 533
 - getWorkspaceSize, 533
 - initialize, 534
 - serialize, 534
 - setPluginNamespace, 535
 - supportsFormat, 535
 - terminate, 536
- nvInfer1::IPluginV2DynamicExt, 537
 - ~IPluginV2DynamicExt, 538
 - clone, 539
 - configurePlugin, 539
 - enqueue, 540
 - getOutputDimensions, 541
 - getTensorRTVersion, 541
 - getWorkspaceSize, 541
 - kFORMAT_COMBINATION_LIMIT, 543
 - supportsFormatCombination, 542
- nvInfer1::IPluginV2Ext, 543
 - ~IPluginV2Ext, 544
 - attachToContext, 545
 - canBroadcastInputAcrossBatch, 545
 - clone, 546
 - configurePlugin, 547
 - configureWithFormat, 548
 - detachFromContext, 548
 - getOutputDataType, 549
 - getTensorRTVersion, 550
 - IPluginV2Ext, 544
 - isOutputBroadcastAcrossBatch, 550
- nvInfer1::IPluginV2IOExt, 551
 - configurePlugin, 552
 - getTensorRTVersion, 553
 - supportsFormatCombination, 553
- nvInfer1::IPluginV2Layer, 554
 - ~IPluginV2Layer, 555
 - getPlugin, 555
 - mImpl, 556
- nvInfer1::IPoolingLayer, 556
 - ~IPoolingLayer, 558
 - getAverageCountExcludesPadding, 558
 - getBlendFactor, 558
 - getPadding, 558
 - getPaddingMode, 559
 - getPaddingNd, 559
 - getPoolingType, 559
 - getPostPadding, 560
 - getPrePadding, 560
 - getStride, 560
 - getStrideNd, 560
 - getWindowSize, 561
 - getWindowSizeNd, 561
 - mImpl, 566
 - setAverageCountExcludesPadding, 561
 - setBlendFactor, 562
 - setPadding, 562
 - setPaddingMode, 562
 - setPaddingNd, 563
 - setPoolingType, 563
 - setPostPadding, 563
 - setPrePadding, 564
 - setStride, 564
 - setStrideNd, 564
 - setWindowSize, 565
 - setWindowSizeNd, 565
- nvInfer1::IProfiler, 566
 - ~IProfiler, 566
 - reportLayerTime, 567
- nvInfer1::IQuantizeLayer, 568
 - ~IQuantizeLayer, 569
 - getAxis, 570
 - mImpl, 570
 - setAxis, 570
- nvInfer1::IRaggedSoftMaxLayer, 571
 - ~IRaggedSoftMaxLayer, 571
 - mImpl, 572
- nvInfer1::IRecurrenceLayer, 572
 - ~IRecurrenceLayer, 573
 - mImpl, 573
 - setInput, 573
- nvInfer1::IReduceLayer, 574
 - ~IReduceLayer, 575
 - getKeepDimensions, 575
 - getOperation, 575
 - getReduceAxes, 575
 - mImpl, 577
 - setKeepDimensions, 576
 - setOperation, 576
 - setReduceAxes, 576
- nvInfer1::IRefitter, 577
 - ~IRefitter, 578
 - destroy, 578
 - getAll, 579
 - getAllWeights, 579
 - getDynamicRangeMax, 580
 - getDynamicRangeMin, 580
 - getErrorRecorder, 580
 - getLogger, 581
 - getMaxThreads, 581
 - getMissing, 581
 - getMissingWeights, 582
 - getTensorsWithDynamicRange, 582
 - mImpl, 586
 - refitCudaEngine, 583
 - setDynamicRange, 583
 - setErrorRecorder, 584
 - setMaxThreads, 584
 - setNamedWeights, 584

- setWeights, 585
- nvInfer1::IResizeLayer, 586
 - ~IResizeLayer, 588
 - getAlignCorners, 589
 - getCoordinateTransformation, 589
 - getCubicCoeff, 589
 - getExcludeOutside, 589
 - getNearestRounding, 590
 - getOutputDimensions, 590
 - getResizeMode, 590
 - getScales, 590
 - getSelectorForSinglePixel, 591
 - mImpl, 595
 - setAlignCorners, 591
 - setCoordinateTransformation, 591
 - setCubicCoeff, 592
 - setExcludeOutside, 592
 - setInput, 592
 - setNearestRounding, 593
 - setOutputDimensions, 593
 - setResizeMode, 594
 - setScales, 594
 - setSelectorForSinglePixel, 595
- nvInfer1::IReverseSequenceLayer, 596
 - ~IReverseSequenceLayer, 597
 - getBatchAxis, 597
 - getSequenceAxis, 597
 - mImpl, 598
 - setBatchAxis, 597
 - setSequenceAxis, 598
- nvInfer1::IRNNv2Layer, 599
 - ~IRNNv2Layer, 600
 - getBiasForGate, 600
 - getCellState, 601
 - getDataLength, 601
 - getDirection, 601
 - getHiddenSize, 601
 - getHiddenState, 602
 - getInputMode, 602
 - getLayerCount, 602
 - getMaxSeqLength, 602
 - getOperation, 602
 - getSequenceLengths, 603
 - getWeightsForGate, 603
 - mImpl, 607
 - setBiasForGate, 603
 - setCellState, 604
 - setDirection, 604
 - setHiddenState, 604
 - setInputMode, 605
 - setOperation, 605
 - setSequenceLengths, 605
 - setWeightsForGate, 606
- nvInfer1::IRuntime, 607
 - ~IRuntime, 609
 - deserializeCudaEngine, 609
 - destroy, 610
 - getDLACore, 610
 - getEngineHostCodeAllowed, 611
 - getErrorRecorder, 611
 - getLogger, 611
 - getMaxThreads, 612
 - getNbDLACores, 612
 - getPluginRegistry, 612
 - getTempfileControlFlags, 612
 - getTemporaryDirectory, 613
 - loadRuntime, 613
 - mImpl, 617
 - setDLACore, 614
 - setEngineHostCodeAllowed, 614
 - setErrorRecorder, 614
 - setGpuAllocator, 615
 - setMaxThreads, 615
 - setTempfileControlFlags, 616
 - setTemporaryDirectory, 616
- nvInfer1::IScaleLayer, 622
 - ~IScaleLayer, 624
 - getChannelAxis, 624
 - getMode, 624
 - getPower, 624
 - getScale, 625
 - getShift, 625
 - mImpl, 627
 - setChannelAxis, 625
 - setMode, 625
 - setPower, 626
 - setScale, 626
 - setShift, 626
- nvInfer1::IScatterLayer, 627
 - ~IScatterLayer, 629
 - getAxis, 629
 - getMode, 629
 - mImpl, 630
 - setAxis, 629
 - setMode, 629
- nvInfer1::ISelectLayer, 630
 - ~ISelectLayer, 631
 - mImpl, 631
- nvInfer1::IShapeLayer, 631
 - ~IShapeLayer, 632
 - mImpl, 632
- nvInfer1::IShuffleLayer, 633
 - ~IShuffleLayer, 634
 - getFirstTranspose, 634
 - getReshapeDimensions, 634
 - getSecondTranspose, 635
 - getZeroIsPlaceholder, 635
 - mImpl, 638

- setFirstTranspose, 635
 - setInput, 636
 - setReshapeDimensions, 636
 - setSecondTranspose, 637
 - setZeroIsPlaceholder, 637
- nvinfer1::ISliceLayer, 638
 - ~ISliceLayer, 640
 - getMode, 640
 - getSize, 640
 - getStart, 641
 - getStride, 641
 - mImpl, 644
 - setInput, 641
 - setMode, 642
 - setSize, 642
 - setStart, 643
 - setStride, 643
- nvinfer1::ISoftMaxLayer, 644
 - ~ISoftMaxLayer, 645
 - getAxes, 645
 - mImpl, 646
 - setAxes, 645
- nvinfer1::ITensor, 646
 - ~ITensor, 648
 - dynamicRangeIsSet, 649
 - getAllowedFormats, 649
 - getBroadcastAcrossBatch, 649
 - getDimensionName, 649
 - getDimensions, 650
 - getDynamicRangeMax, 650
 - getDynamicRangeMin, 650
 - getLocation, 651
 - getName, 651
 - getType, 651
 - isExecutionTensor, 652
 - isNetworkInput, 652
 - isNetworkOutput, 652
 - isShapeTensor, 653
 - mImpl, 657
 - resetDynamicRange, 653
 - setAllowedFormats, 653
 - setBroadcastAcrossBatch, 654
 - setDimensionName, 655
 - setDimensions, 655
 - setDynamicRange, 656
 - setLocation, 656
 - setName, 656
 - setType, 657
- nvinfer1::ITimingCache, 658
 - ~ITimingCache, 659
 - combine, 659
 - mImpl, 660
 - reset, 659
 - serialize, 660
- nvinfer1::ITopKLayer, 661
 - ~ITopKLayer, 662
 - getK, 662
 - getOperation, 662
 - getReduceAxes, 662
 - mImpl, 664
 - setInput, 663
 - setK, 663
 - setOperation, 663
 - setReduceAxes, 664
- nvinfer1::ITripLimitLayer, 664
 - ~ITripLimitLayer, 665
 - getTripLimit, 665
 - mImpl, 665
- nvinfer1::IUnaryLayer, 670
 - ~IUnaryLayer, 671
 - getOperation, 671
 - mImpl, 672
 - setOperation, 671
- nvinfer1::Permutation, 674
 - order, 674
- nvinfer1::plugin, 84
 - CENTER_SIZE, 85
 - CodeTypeSSD, 84
 - CORNER, 85
 - CORNER_SIZE, 85
 - TF_CENTER, 85
- nvinfer1::plugin::DetectionOutputParameters, 97
 - backgroundLabelId, 98
 - codeType, 98
 - confidenceThreshold, 98
 - confSigmoid, 98
 - inputOrder, 99
 - isBatchAgnostic, 99
 - isNormalized, 99
 - keepTopK, 99
 - nmsThreshold, 99
 - numClasses, 99
 - shareLocation, 99
 - topK, 100
 - varianceEncodedInTarget, 100
- nvinfer1::plugin::GridAnchorParameters, 127
 - aspectRatios, 128
 - H, 128
 - maxSize, 128
 - minSize, 128
 - numAspectRatios, 128
 - variance, 128
 - W, 128
- nvinfer1::plugin::NMSPParameters, 672
 - backgroundLabelId, 673
 - iouThreshold, 673
 - isNormalized, 673
 - keepTopK, 673

- numClasses, 673
- scoreThreshold, 674
- shareLocation, 674
- topK, 674
- nvinfer1::plugin::PriorBoxParameters, 681
 - aspectRatios, 682
 - clip, 682
 - flip, 682
 - imgH, 682
 - imgW, 683
 - maxSize, 683
 - minSize, 683
 - numAspectRatios, 683
 - numMaxSize, 683
 - numMinSize, 683
 - offset, 683
 - stepH, 684
 - stepW, 684
 - variance, 684
- nvinfer1::plugin::Quadruple, 684
 - data, 685
- nvinfer1::plugin::RegionParameters, 685
 - classes, 686
 - coords, 686
 - num, 686
 - smTree, 686
- nvinfer1::plugin::RPROIParams, 686
 - anchorsRatioCount, 687
 - anchorsScaleCount, 687
 - featureStride, 687
 - iouThreshold, 687
 - minBoxSize, 687
 - nmsMaxOut, 688
 - poolingH, 688
 - poolingW, 688
 - preNmsTop, 688
 - spatialScale, 688
- nvinfer1::plugin::softmaxTree, 689
 - child, 690
 - group, 690
 - groupOffset, 690
 - groups, 690
 - groupSize, 690
 - leaf, 690
 - n, 691
 - name, 691
 - parent, 691
- nvinfer1::PluginField, 675
 - data, 676
 - length, 676
 - name, 676
 - PluginField, 675
 - type, 676
- nvinfer1::PluginFieldCollection, 677
 - fields, 677
 - nbFields, 677
- nvinfer1::PluginRegistrar< T >, 678
 - PluginRegistrar, 678
- nvinfer1::PluginTensorDesc, 679
 - dims, 680
 - format, 680
 - scale, 680
 - type, 680
- nvinfer1::safe, 85
 - createInferRuntime, 86
 - getSafePluginRegistry, 87
 - kGATHER_OOB, 86
 - kINF_CONSUMED, 86
 - kNAN_CONSUMED, 86
 - kSCATTER_OOB, 86
 - kSCATTER_RACE, 86
 - RuntimeErrorType, 86
- nvinfer1::safe::ICudaEngine, 242
 - ~ICudaEngine, 244
 - bindingIsInput, 244
 - createExecutionContext, 245
 - createExecutionContextWithoutDeviceMemory, 245
 - getBindingBytesPerComponent, 246
 - getBindingComponentsPerElement, 247
 - getBindingDataType, 247
 - getBindingDimensions, 248
 - getBindingFormat, 249
 - getBindingIndex, 249
 - getBindingName, 250
 - getBindingVectorizedDim, 251
 - getDeviceMemorySize, 252
 - getErrorRecorder, 252
 - getIOTensorName, 252
 - getName, 253
 - getNbBindings, 253
 - getNbIOTensors, 254
 - getTensorBytesPerComponent, 254
 - getTensorComponentsPerElement, 255
 - getTensorDataType, 256
 - getTensorFormat, 257
 - getTensorIOMode, 257
 - getTensorShape, 258
 - getTensorVectorizedDim, 259
 - ICudaEngine, 243, 244
 - operator=, 260
 - setErrorRecorder, 260
- nvinfer1::safe::IExecutionContext, 323
 - ~IExecutionContext, 325
 - enqueueV2, 325
 - enqueueV3, 326
 - getEngine, 327
 - getErrorBuffer, 327
 - getErrorRecorder, 328

- getInputConsumedEvent, 328
- getInputTensorAddress, 329
- getName, 330
- getOutputTensorAddress, 330
- getStrides, 331
- getTensorStrides, 332
- IEExecutionContext, 324, 325
- operator=, 332
- setDeviceMemory, 333
- setErrorBuffer, 333
- setErrorRecorder, 334
- setInputConsumedEvent, 335
- setInputTensorAddress, 335
- setName, 336
- setOutputTensorAddress, 337
- nvinfer1::safe::IPluginRegistry, 519
 - ~IPluginRegistry, 520
 - deregisterCreator, 521
 - getErrorRecorder, 521
 - getPluginCreator, 522
 - getPluginCreatorList, 523
 - registerCreator, 523
 - setErrorRecorder, 524
- nvinfer1::safe::IRuntime, 617
 - ~IRuntime, 618
 - deserializeCudaEngine, 619
 - getErrorRecorder, 620
 - IRuntime, 618, 619
 - operator=, 620, 621
 - setErrorRecorder, 621
 - setGpuAllocator, 621
- nvinfer1::safe::PluginRegistrar< T >, 678
 - PluginRegistrar, 679
- nvinfer1::safe::RuntimeErrorInformation, 688
 - bitMask, 689
- nvinfer1::utils, 87
 - reorderSubBuffers, 87
 - reshapeWeights, 88
 - transposeSubBuffers, 89
- nvinfer1::Weights, 691
 - count, 692
 - type, 692
 - values, 692
- NvInferConsistency.h, 748, 749
 - createConsistencyChecker_INTERNAL, 748
- NvInferLegacyDims.h, 750, 751
- NvInferPlugin.h, 752, 758
 - createAnchorGeneratorPlugin, 753
 - createBatchedNMSPlugin, 754
 - createInstanceNormalizationPlugin, 754
 - createNMSPlugin, 755
 - createNormalizePlugin, 755
 - createPriorBoxPlugin, 756
 - createRegionPlugin, 756
 - createReorgPlugin, 756
 - createRPNROIPlugin, 757
 - createSplitPlugin, 757
 - initLibNvInferPlugins, 758
- NvInferPluginUtils.h, 759, 760
- NvInferRuntime.h, 762, 766
 - getLogger, 765
 - getPluginRegistry, 765
 - REGISTER_TENSORRT_PLUGIN, 765
- NvInferRuntimeBase.h, 782, 785
 - getInferLibVersion, 785
 - NV_TENSORRT_VERSION, 784
 - TENSORRTAPI, 784
 - TRT_DEPRECATED, 784
 - TRT_DEPRECATED_API, 784
 - TRT_DEPRECATED_ENUM, 784
 - TRTNOEXCEPT, 784
- NvInferRuntimeCommon.h, 789, 790
- NvInferRuntimePlugin.h, 791, 792
- NvInferSafeRuntime.h, 795, 797
 - REGISTER_SAFE_TENSORRT_PLUGIN, 797
- NvInferVersion.h, 800, 803
 - NV_TENSORRT_BUILD, 801
 - NV_TENSORRT_LWS_MAJOR, 801
 - NV_TENSORRT_LWS_MINOR, 801
 - NV_TENSORRT_LWS_PATCH, 801
 - NV_TENSORRT_MAJOR, 801
 - NV_TENSORRT_MINOR, 802
 - NV_TENSORRT_PATCH, 802
 - NV_TENSORRT_RELEASE_TYPE, 802
 - NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS, 802
 - NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY, 802
 - NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE, 802
 - NV_TENSORRT_SONAME_MAJOR, 803
 - NV_TENSORRT_SONAME_MINOR, 803
 - NV_TENSORRT_SONAME_PATCH, 803
- NvOnnxConfig.h, 804, 805
- nvonnxparser, 90
 - createONNXConfig, 92
 - EnumMax, 92
 - EnumMax< ErrorCode >, 92
 - EnumMax< OnnxParserFlag >, 92
 - ErrorCode, 91
 - kINTERNAL_ERROR, 91
 - kINVALID_GRAPH, 91
 - kINVALID_NODE, 91
 - kINVALID_VALUE, 91
 - kMEM_ALLOC_FAILED, 91
 - kMODEL_DESERIALIZE_FAILED, 91
 - kNATIVE_INSTANCENORM, 92
 - kSUCCESS, 91

- kUNSUPPORTED_GRAPH, 91
- kUNSUPPORTED_NODE, 91
- OnnxParserFlag, 92
- OnnxParserFlags, 91
- NvOnnxParser.h, 809, 812
 - createNvOnnxParser_INTERNAL, 812
 - getNvOnnxParserVersion, 812
 - NV_ONNX_PARSER_MAJOR, 811
 - NV_ONNX_PARSER_MINOR, 811
 - NV_ONNX_PARSER_PATCH, 811
 - SubGraph_t, 811
 - SubGraphCollection_t, 811
- nvonnxparser::IOnnxConfig, 474
 - ~IOnnxConfig, 475
 - addVerbosity, 475
 - destroy, 475
 - getFullTextFileName, 476
 - getModelDtype, 476
 - getModelFileName, 476
 - getPrintLayerInfo, 477
 - getTextFileName, 477
 - getVerbosityLevel, 477
 - reduceVerbosity, 478
 - setFullTextFileName, 478
 - setModelDtype, 479
 - setModelFileName, 479
 - setPrintLayerInfo, 479
 - setTextFileName, 480
 - setVerbosityLevel, 480
 - Verbosity, 475
- nvonnxparser::IParser, 495
 - ~IParser, 496
 - clearErrors, 496
 - clearFlag, 496
 - destroy, 496
 - getError, 497
 - getFlag, 497
 - getFlags, 497
 - getNbErrors, 497
 - getUsedVCPluginLibraries, 498
 - parse, 498
 - parseFromFile, 499
 - parseWithWeightDescriptors, 499
 - setFlag, 500
 - setFlags, 500
 - supportsModel, 501
 - supportsOperator, 501
- nvonnxparser::IParserError, 502
 - ~IParserError, 502
 - code, 503
 - desc, 503
 - file, 503
 - func, 503
 - line, 503
 - node, 503
- nvuffparser, 93
 - createUffParser, 95
 - FieldType, 94
 - kCHAR, 95
 - kDATATYPE, 95
 - kDIMS, 95
 - kFLOAT, 95
 - kINT32, 95
 - kNC, 95
 - kNCHW, 95
 - kNHWC, 95
 - kUNKNOWN, 95
 - shutdownProtobufLibrary, 95
 - UffInputOrder, 95
- NvUffParser.h, 805, 807
 - UFF_REQUIRED_VERSION_MAJOR, 806
 - UFF_REQUIRED_VERSION_MINOR, 807
 - UFF_REQUIRED_VERSION_PATCH, 807
- nvuffparser::FieldCollection, 124
 - fields, 124
 - nbFields, 124
- nvuffparser::FieldMap, 125
 - data, 126
 - FieldMap, 125, 126
 - length, 126
 - name, 126
 - type, 126
- nvuffparser::IUffParser, 666
 - ~IUffParser, 666
 - destroy, 667
 - getErrorRecorder, 667
 - getUffRequiredVersionMajor, 667
 - getUffRequiredVersionMinor, 667
 - getUffRequiredVersionPatch, 668
 - parse, 668
 - parseBuffer, 668
 - registerInput, 669
 - registerOutput, 669
 - setErrorRecorder, 669
 - setPluginNamespace, 670
- NvUtils.h, 808, 809
- offset
 - nvinfer1::plugin::PriorBoxParameters, 683
- OnnxParserFlag
 - nvonnxparser, 92
- OnnxParserFlags
 - nvonnxparser, 91
- operation
 - nvinfer1::IExprBuilder, 339
- operator=
 - nvinfer1::consistency::IConsistencyChecker, 197
 - nvinfer1::consistency::IPluginChecker, 505

- nvinfer1::INoCopy, [465](#)
 - nvinfer1::safe::ICudaEngine, [260](#)
 - nvinfer1::safe::IExecutionContext, [332](#)
 - nvinfer1::safe::IRuntime, [620](#), [621](#)
- OptProfileSelector
 - nvinfer1, [56](#)
- order
 - nvinfer1::Permutation, [674](#)
- outputTypeIsSet
 - nvinfer1::ILayer, [391](#)
- PaddingMode
 - nvinfer1, [57](#)
- parent
 - nvinfer1::plugin::softmaxTree, [691](#)
- parse
 - nvcaffeparser1::ICaffeParser, [187](#)
 - nvonnxparser::IParser, [498](#)
 - nvuffparser::IUffParser, [668](#)
- parseBinaryProto
 - nvcaffeparser1::ICaffeParser, [187](#)
- parseBuffer
 - nvuffparser::IUffParser, [668](#)
- parseBuffers
 - nvcaffeparser1::ICaffeParser, [188](#)
- parseFromFile
 - nvonnxparser::IParser, [499](#)
- parseWithWeightDescriptors
 - nvonnxparser::IParser, [499](#)
- platformHasFastFp16
 - nvinfer1::IBuilder, [157](#)
- platformHasFastInt8
 - nvinfer1::IBuilder, [157](#)
- platformHasTf32
 - nvinfer1::IBuilder, [157](#)
- PluginField
 - nvinfer1::PluginField, [675](#)
- PluginFieldType
 - nvinfer1, [60](#)
- PluginFormat
 - nvinfer1, [41](#)
- PluginLibraryHandle
 - nvinfer1::IPluginRegistry, [515](#)
- PluginRegistrar
 - nvinfer1::PluginRegistrar< T >, [678](#)
 - nvinfer1::safe::PluginRegistrar< T >, [679](#)
- PluginVersion, [681](#)
 - nvinfer1, [60](#)
- poolingH
 - nvinfer1::plugin::RPROIParams, [688](#)
- PoolingType
 - nvinfer1, [60](#)
- poolingW
 - nvinfer1::plugin::RPROIParams, [688](#)
- precisionIsSet
 - nvinfer1::ILayer, [392](#)
- preNmsTop
 - nvinfer1::plugin::RPROIParams, [688](#)
- PreviewFeature
 - nvinfer1, [61](#)
- ProfilingVerbosity
 - nvinfer1, [62](#)
- QuantizationFlag
 - nvinfer1, [62](#)
- QuantizationFlags
 - nvinfer1, [41](#)
- readCalibrationCache
 - nvinfer1::IInt8Calibrator, [377](#)
- readHistogramCache
 - nvinfer1::IInt8LegacyCalibrator, [382](#)
- reallocate
 - nvinfer1::IGpuAllocator, [359](#)
- reallocateOutput
 - nvinfer1::IOutputAllocator, [488](#)
- ReduceOperation
 - nvinfer1, [63](#)
- reduceVerbosity
 - nvonnxparser::IOnnxConfig, [478](#)
- RefCount
 - nvinfer1::IErrorRecorder, [287](#)
- refitCudaEngine
 - nvinfer1::IRefitter, [583](#)
- REGISTER_SAFE_TENSORRT_PLUGIN
 - NvInferSafeRuntime.h, [797](#)
- REGISTER_TENSORRT_PLUGIN
 - NvInferRuntime.h, [765](#)
- registerCreator
 - nvinfer1::IPluginRegistry, [518](#)
 - nvinfer1::safe::IPluginRegistry, [523](#)
- registerInput
 - nvuffparser::IUffParser, [669](#)
- registerOutput
 - nvuffparser::IUffParser, [669](#)
- removeTensor
 - nvinfer1::INetworkDefinition, [456](#)
- reorderSubBuffers
 - nvinfer1::utils, [87](#)
- reportAlgorithms
 - nvinfer1::IAlgorithmSelector, [142](#)
- reportError
 - nvinfer1::IErrorRecorder, [292](#)
- reportLayerTime
 - nvinfer1::IProfiler, [567](#)
- reportToProfiler
 - nvinfer1::IExecutionContext, [310](#)
- reset
 - nvinfer1::IBuilder, [157](#)

- nvinfer1::IBuilderConfig, 174
 - nvinfer1::ITimingCache, 659
- resetDeviceType
 - nvinfer1::IBuilderConfig, 174
- resetDynamicRange
 - nvinfer1::ITensor, 653
- resetOutputType
 - nvinfer1::ILayer, 392
- resetPrecision
 - nvinfer1::ILayer, 392
- reshapeWeights
 - nvinfer1::utils, 88
- ResizeCoordinateTransformation
 - nvinfer1, 63
- ResizeMode
 - nvinfer1, 41
- ResizeRoundMode
 - nvinfer1, 64
- ResizeSelector
 - nvinfer1, 64
- RNNDirection
 - nvinfer1, 65
- RNNGateType
 - nvinfer1, 65
- RNNInputMode
 - nvinfer1, 66
- RNNOperation
 - nvinfer1, 66
- RuntimeErrorType
 - nvinfer1::safe, 86
- SampleMode
 - nvinfer1, 67
- scale
 - nvinfer1::PluginTensorDesc, 680
- ScaleMode
 - nvinfer1, 68
- ScatterMode
 - nvinfer1, 68
- scoreThreshold
 - nvinfer1::plugin::NMSPParameters, 674
- selectAlgorithms
 - nvinfer1::IAlgorithmSelector, 142
- serialize
 - nvinfer1::ICudaEngine, 240
 - nvinfer1::IPluginV2, 534
 - nvinfer1::ITimingCache, 660
- setActivationType
 - nvinfer1::IActivationLayer, 131
- setAlgorithmSelector
 - nvinfer1::IBuilderConfig, 174
- setAlignCorners
 - nvinfer1::IGridSampleLayer, 362
 - nvinfer1::IResizeLayer, 591
- setAllowedFormats
 - nvinfer1::ITensor, 653
- setAlpha
 - nvinfer1::IActivationLayer, 131
 - nvinfer1::IFillLayer, 343
 - nvinfer1::ILRNLayer, 410
- setAuxStreams
 - nvinfer1::IExecutionContext, 311
- setAverageCountExcludesPadding
 - nvinfer1::IPoolingLayer, 561
- setAvgTimingIterations
 - nvinfer1::IBuilderConfig, 175
- setAxes
 - nvinfer1::INormalizationLayer, 470
 - nvinfer1::ISoftMaxLayer, 645
- setAxis
 - nvinfer1::IConcatenationLayer, 193
 - nvinfer1::IDequantizeLayer, 274
 - nvinfer1::IIteratorLayer, 385
 - nvinfer1::ILoopOutputLayer, 406
 - nvinfer1::IOneHotLayer, 473
 - nvinfer1::IQuantizeLayer, 570
 - nvinfer1::IScatterLayer, 629
- setBatchAxis
 - nvinfer1::IReverseSequenceLayer, 597
- setBeta
 - nvinfer1::IActivationLayer, 132
 - nvinfer1::IFillLayer, 344
 - nvinfer1::ILRNLayer, 410
- setBiasForGate
 - nvinfer1::IRNNv2Layer, 603
- setBiasWeights
 - nvinfer1::IConvolutionLayer, 207
 - nvinfer1::IDeconvolutionLayer, 267
 - nvinfer1::IFullyConnectedLayer, 349
- setBindingDimensions
 - nvinfer1::IExecutionContext, 312
- setBlendFactor
 - nvinfer1::IPoolingLayer, 562
- setBoundingBoxFormat
 - nvinfer1::INMSLayer, 462
- setBroadcastAcrossBatch
 - nvinfer1::ITensor, 654
- setBuilderOptimizationLevel
 - nvinfer1::IBuilderConfig, 175
- setCalibrationProfile
 - nvinfer1::IBuilderConfig, 175
- setCellState
 - nvinfer1::IRNNv2Layer, 604
- setChannelAxis
 - nvinfer1::IScaleLayer, 625
- setComputePrecision
 - nvinfer1::INormalizationLayer, 470
- setCondition

- nvinfer1::IfConditional, 370
- setCoordinateTransformation
 - nvinfer1::IResizeLayer, 591
- setCubicCoeff
 - nvinfer1::IResizeLayer, 592
- setDebugSync
 - nvinfer1::IExecutionContext, 313
- setDefaultDeviceType
 - nvinfer1::IBuilderConfig, 176
- setDeviceMemory
 - nvinfer1::IExecutionContext, 313
 - nvinfer1::safe::IExecutionContext, 333
- setDeviceType
 - nvinfer1::IBuilderConfig, 176
- setDilation
 - nvinfer1::IConvolutionLayer, 207
- setDilationNd
 - nvinfer1::IConvolutionLayer, 207
 - nvinfer1::IDeconvolutionLayer, 267
- setDimensionName
 - nvinfer1::ITensor, 655
- setDimensions
 - nvinfer1::IConstantLayer, 199
 - nvinfer1::IFillLayer, 344
 - nvinfer1::IOptimizationProfile, 484
 - nvinfer1::ITensor, 655
- setDirection
 - nvinfer1::IRNNv2Layer, 604
- setDLACore
 - nvinfer1::IBuilderConfig, 177
 - nvinfer1::IRuntime, 614
- setDynamicRange
 - nvinfer1::IRefitter, 583
 - nvinfer1::ITensor, 656
- setEngineCapability
 - nvinfer1::IBuilderConfig, 177
- setEngineHostCodeAllowed
 - nvinfer1::IRuntime, 614
- setEnqueueEmitsProfile
 - nvinfer1::IExecutionContext, 313
- setEpsilon
 - nvinfer1::INormalizationLayer, 470
- setEquation
 - nvinfer1::IEinsumLayer, 278
- setErrorBuffer
 - nvinfer1::safe::IExecutionContext, 333
- setErrorRecorder
 - nvcaffeparser1::ICaffeParser, 189
 - nvinfer1::IBuilder, 158
 - nvinfer1::ICudaEngine, 241
 - nvinfer1::IEngineInspector, 285
 - nvinfer1::IExecutionContext, 314
 - nvinfer1::INetworkDefinition, 457
 - nvinfer1::IPluginRegistry, 518
 - nvinfer1::IRefitter, 584
 - nvinfer1::IRuntime, 614
 - nvinfer1::safe::ICudaEngine, 260
 - nvinfer1::safe::IExecutionContext, 334
 - nvinfer1::safe::IPluginRegistry, 524
 - nvinfer1::safe::IRuntime, 621
 - nvuffparser::IUffParser, 669
- setExcludeOutside
 - nvinfer1::IResizeLayer, 592
- setExecutionContext
 - nvinfer1::IEngineInspector, 285
- setExtraMemoryTarget
 - nvinfer1::IOptimizationProfile, 485
- setFirstTranspose
 - nvinfer1::IShuffleLayer, 635
- setFlag
 - nvinfer1::IBuilderConfig, 177
 - nvonnxparser::IParser, 500
- setFlags
 - nvinfer1::IBuilderConfig, 178
 - nvonnxparser::IParser, 500
- setFullTextFileName
 - nvonnxparser::IOnnxConfig, 478
- setGatherAxis
 - nvinfer1::IGatherLayer, 354
- setGpuAllocator
 - nvinfer1::IBuilder, 158
 - nvinfer1::IRuntime, 615
 - nvinfer1::safe::IRuntime, 621
- setHardwareCompatibilityLevel
 - nvinfer1::IBuilderConfig, 178
- setHiddenState
 - nvinfer1::IRNNv2Layer, 604
- setInput
 - nvinfer1::IConvolutionLayer, 208
 - nvinfer1::IDeconvolutionLayer, 267
 - nvinfer1::IFillLayer, 344
 - nvinfer1::IFullyConnectedLayer, 349
 - nvinfer1::ILayer, 393
 - nvinfer1::ILoopOutputLayer, 406
 - nvinfer1::INMSLayer, 462
 - nvinfer1::IRecurrenceLayer, 573
 - nvinfer1::IResizeLayer, 592
 - nvinfer1::IShuffleLayer, 636
 - nvinfer1::ISliceLayer, 641
 - nvinfer1::ITopKLayer, 663
- setInputConsumedEvent
 - nvinfer1::IExecutionContext, 314
 - nvinfer1::safe::IExecutionContext, 335
- setInputMode
 - nvinfer1::IRNNv2Layer, 605
- setInputShape
 - nvinfer1::IExecutionContext, 315
- setInputShapeBinding

- [nvinfer1::IExecutionContext, 315](#)
- [setInputTensorAddress](#)
 - [nvinfer1::IExecutionContext, 316](#)
 - [nvinfer1::safe::IExecutionContext, 335](#)
- [setInt8Calibrator](#)
 - [nvinfer1::IBuilderConfig, 179](#)
- [setInterpolationMode](#)
 - [nvinfer1::IGridSampleLayer, 363](#)
- [setK](#)
 - [nvinfer1::ILRNLayer, 410](#)
 - [nvinfer1::ITopKLayer, 663](#)
- [setKeepDimensions](#)
 - [nvinfer1::IReduceLayer, 576](#)
- [setKernelSize](#)
 - [nvinfer1::IConvolutionLayer, 208](#)
 - [nvinfer1::IDeconvolutionLayer, 268](#)
- [setKernelSizeNd](#)
 - [nvinfer1::IConvolutionLayer, 209](#)
 - [nvinfer1::IDeconvolutionLayer, 268](#)
- [setKernelWeights](#)
 - [nvinfer1::IConvolutionLayer, 209](#)
 - [nvinfer1::IDeconvolutionLayer, 268](#)
 - [nvinfer1::IFullyConnectedLayer, 350](#)
- [setLocation](#)
 - [nvinfer1::ITensor, 656](#)
- [setMaxAuxStreams](#)
 - [nvinfer1::IBuilderConfig, 179](#)
- [setMaxBatchSize](#)
 - [nvinfer1::IBuilder, 159](#)
- [setMaxThreads](#)
 - [nvinfer1::IBuilder, 159](#)
 - [nvinfer1::IRefitter, 584](#)
 - [nvinfer1::IRuntime, 615](#)
- [setMaxWorkspaceSize](#)
 - [nvinfer1::IBuilderConfig, 180](#)
- [setMemoryPoolLimit](#)
 - [nvinfer1::IBuilderConfig, 180](#)
- [setMessage](#)
 - [nvinfer1::IAssertionLayer, 146](#)
- [setMetadata](#)
 - [nvinfer1::ILayer, 393](#)
- [setMinTimingIterations](#)
 - [nvinfer1::IBuilderConfig, 181](#)
- [setMode](#)
 - [nvinfer1::IGatherLayer, 355](#)
 - [nvinfer1::IScaleLayer, 625](#)
 - [nvinfer1::IScatterLayer, 629](#)
 - [nvinfer1::ISliceLayer, 642](#)
- [setModelDtype](#)
 - [nvonnxparser::IOnnxConfig, 479](#)
- [setModelFileName](#)
 - [nvonnxparser::IOnnxConfig, 479](#)
- [setName](#)
 - [nvinfer1::IExecutionContext, 317](#)
 - [nvinfer1::IIfConditional, 370](#)
 - [nvinfer1::ILayer, 394](#)
 - [nvinfer1::ILoop, 402](#)
 - [nvinfer1::INetworkDefinition, 457](#)
 - [nvinfer1::ITensor, 656](#)
 - [nvinfer1::safe::IExecutionContext, 336](#)
- [setNamedWeights](#)
 - [nvinfer1::IRefitter, 584](#)
- [setNbElementWiseDims](#)
 - [nvinfer1::IGatherLayer, 355](#)
- [setNbGroups](#)
 - [nvinfer1::IConvolutionLayer, 209](#)
 - [nvinfer1::IDeconvolutionLayer, 269](#)
 - [nvinfer1::INormalizationLayer, 471](#)
- [setNbOutputChannels](#)
 - [nvinfer1::IFullyConnectedLayer, 350](#)
- [setNbOutputMaps](#)
 - [nvinfer1::IConvolutionLayer, 210](#)
 - [nvinfer1::IDeconvolutionLayer, 269](#)
- [setNearestRounding](#)
 - [nvinfer1::IResizeLayer, 593](#)
- [setNvtxVerbosity](#)
 - [nvinfer1::IExecutionContext, 317](#)
- [setOperation](#)
 - [nvinfer1::IElementWiseLayer, 280](#)
 - [nvinfer1::IFillLayer, 345](#)
 - [nvinfer1::IMatrixMultiplyLayer, 413](#)
 - [nvinfer1::IReduceLayer, 576](#)
 - [nvinfer1::IRNNv2Layer, 605](#)
 - [nvinfer1::ITopKLayer, 663](#)
 - [nvinfer1::IUnaryLayer, 671](#)
- [setOptimizationProfile](#)
 - [nvinfer1::IExecutionContext, 318](#)
- [setOptimizationProfileAsync](#)
 - [nvinfer1::IExecutionContext, 319](#)
- [setOutputAllocator](#)
 - [nvinfer1::IExecutionContext, 320](#)
- [setOutputDimensions](#)
 - [nvinfer1::IResizeLayer, 593](#)
- [setOutputTensorAddress](#)
 - [nvinfer1::safe::IExecutionContext, 337](#)
- [setOutputType](#)
 - [nvinfer1::ILayer, 394](#)
- [setPadding](#)
 - [nvinfer1::IConvolutionLayer, 210](#)
 - [nvinfer1::IDeconvolutionLayer, 269](#)
 - [nvinfer1::IPoolingLayer, 562](#)
- [setPaddingMode](#)
 - [nvinfer1::IConvolutionLayer, 211](#)
 - [nvinfer1::IDeconvolutionLayer, 270](#)
 - [nvinfer1::IPoolingLayer, 562](#)
- [setPaddingNd](#)
 - [nvinfer1::IConvolutionLayer, 211](#)
 - [nvinfer1::IDeconvolutionLayer, 270](#)

- nvinfer1::IPoolingLayer, [563](#)
- setParentSearchEnabled
 - nvinfer1::IPluginRegistry, [519](#)
- setPersistentCacheLimit
 - nvinfer1::IExecutionContext, [320](#)
- setPluginFactoryV2
 - nvcaffeparser1::ICaffeParser, [189](#)
- setPluginNamespace
 - nvcaffeparser1::ICaffeParser, [189](#)
 - nvinfer1::IPluginCreator, [511](#)
 - nvinfer1::IPluginV2, [535](#)
 - nvuffparser::IUffParser, [670](#)
- setPluginsToSerialize
 - nvinfer1::IBuilderConfig, [181](#)
- setPoolingType
 - nvinfer1::IPoolingLayer, [563](#)
- setPostPadding
 - nvinfer1::IConvolutionLayer, [211](#)
 - nvinfer1::IDeconvolutionLayer, [270](#)
 - nvinfer1::IPaddingLayer, [491](#)
 - nvinfer1::IPoolingLayer, [563](#)
- setPostPaddingNd
 - nvinfer1::IPaddingLayer, [492](#)
- setPower
 - nvinfer1::IScaleLayer, [626](#)
- setPrecision
 - nvinfer1::ILayer, [395](#)
- setPrePadding
 - nvinfer1::IConvolutionLayer, [212](#)
 - nvinfer1::IDeconvolutionLayer, [271](#)
 - nvinfer1::IPaddingLayer, [492](#)
 - nvinfer1::IPoolingLayer, [564](#)
- setPrePaddingNd
 - nvinfer1::IPaddingLayer, [492](#)
- setPreviewFeature
 - nvinfer1::IBuilderConfig, [182](#)
- setPrintLayerInfo
 - nvonnxparser::IOnnxConfig, [479](#)
- setProfiler
 - nvinfer1::IExecutionContext, [321](#)
- setProfileStream
 - nvinfer1::IBuilderConfig, [182](#)
- setProfilingVerbosity
 - nvinfer1::IBuilderConfig, [182](#)
- setProtobufBufferSize
 - nvcaffeparser1::ICaffeParser, [190](#)
- setQuantizationFlag
 - nvinfer1::IBuilderConfig, [183](#)
- setQuantizationFlags
 - nvinfer1::IBuilderConfig, [183](#)
- setReduceAxes
 - nvinfer1::IReduceLayer, [576](#)
 - nvinfer1::ITopKLayer, [664](#)
- setReshapeDimensions
 - nvinfer1::IShuffleLayer, [636](#)
- setResizeMode
 - nvinfer1::IResizeLayer, [594](#)
- setReverse
 - nvinfer1::IIteratorLayer, [385](#)
- setSampleMode
 - nvinfer1::IGridSampleLayer, [363](#)
- setScale
 - nvinfer1::IScaleLayer, [626](#)
- setScales
 - nvinfer1::IResizeLayer, [594](#)
- setSecondTranspose
 - nvinfer1::IShuffleLayer, [637](#)
- setSelectorForSinglePixel
 - nvinfer1::IResizeLayer, [595](#)
- setSequenceAxis
 - nvinfer1::IReverseSequenceLayer, [598](#)
- setSequenceLengths
 - nvinfer1::IRNNv2Layer, [605](#)
- setShapeValues
 - nvinfer1::IOptimizationProfile, [485](#)
- setShift
 - nvinfer1::IScaleLayer, [626](#)
- setSize
 - nvinfer1::ISliceLayer, [642](#)
- setStart
 - nvinfer1::ISliceLayer, [643](#)
- setStride
 - nvinfer1::IConvolutionLayer, [212](#)
 - nvinfer1::IDeconvolutionLayer, [271](#)
 - nvinfer1::IPoolingLayer, [564](#)
 - nvinfer1::ISliceLayer, [643](#)
- setStrideNd
 - nvinfer1::IConvolutionLayer, [212](#)
 - nvinfer1::IDeconvolutionLayer, [271](#)
 - nvinfer1::IPoolingLayer, [564](#)
- setTacticSources
 - nvinfer1::IBuilderConfig, [183](#)
- setTempfileControlFlags
 - nvinfer1::IRuntime, [616](#)
- setTemporaryDirectory
 - nvinfer1::IRuntime, [616](#)
- setTemporaryStorageAllocator
 - nvinfer1::IExecutionContext, [321](#)
- setTensorAddress
 - nvinfer1::IExecutionContext, [322](#)
- setTextFileName
 - nvonnxparser::IOnnxConfig, [480](#)
- setTimingCache
 - nvinfer1::IBuilderConfig, [184](#)
- setTopKBoxLimit
 - nvinfer1::INMSLayer, [463](#)
- setToType
 - nvinfer1::ICastLayer, [191](#)

- setType
 - nvinfer1::ITensor, [657](#)
- setVerbosityLevel
 - nvonnxparser::IOnnxConfig, [480](#)
- setWeights
 - nvinfer1::IConstantLayer, [200](#)
 - nvinfer1::IRefitter, [585](#)
- setWeightsForGate
 - nvinfer1::IRNNv2Layer, [606](#)
- setWeightsName
 - nvinfer1::INetworkDefinition, [458](#)
- setWindowSize
 - nvinfer1::ILRNLayer, [411](#)
 - nvinfer1::IPoolingLayer, [565](#)
- setWindowSizeNd
 - nvinfer1::IPoolingLayer, [565](#)
- setZeroIsPlaceholder
 - nvinfer1::IShuffleLayer, [637](#)
- Severity
 - nvinfer1::ILogger, [397](#)
- shareLocation
 - nvinfer1::plugin::DetectionOutputParameters, [99](#)
 - nvinfer1::plugin::NMSPParameters, [674](#)
- shutdownProtobufLibrary
 - nvcaffeparser1, [30](#)
 - nvuffparser, [95](#)
- size
 - nvinfer1::IHostMemory, [365](#)
- SliceMode
 - nvinfer1, [42](#)
- smTree
 - nvinfer1::plugin::RegionParameters, [686](#)
- spatialScale
 - nvinfer1::plugin::RPROIParams, [688](#)
- stepH
 - nvinfer1::plugin::PriorBoxParameters, [684](#)
- stepW
 - nvinfer1::plugin::PriorBoxParameters, [684](#)
- SubGraph_t
 - NvOnnxParser.h, [811](#)
- SubGraphCollection_t
 - NvOnnxParser.h, [811](#)
- supportsFormat
 - nvinfer1::IPluginV2, [535](#)
- supportsFormatCombination
 - nvinfer1::IPluginV2DynamicExt, [542](#)
 - nvinfer1::IPluginV2IOExt, [553](#)
- supportsModel
 - nvonnxparser::IParser, [501](#)
- supportsOperator
 - nvonnxparser::IParser, [501](#)
- TacticSource
 - nvinfer1, [69](#)
- TacticSources
 - nvinfer1, [42](#)
- TempfileControlFlag
 - nvinfer1, [69](#)
- TempfileControlFlags
 - nvinfer1, [42](#)
- TensorFormat
 - nvinfer1, [70](#)
- TensorFormats
 - nvinfer1, [42](#)
- TensorIOMode
 - nvinfer1, [72](#)
- TensorLocation
 - nvinfer1, [72](#)
- TENSORRTAPI
 - NvInferRuntimeBase.h, [784](#)
- terminate
 - nvinfer1::IPluginV2, [536](#)
- TF_CENTER
 - nvinfer1::plugin, [85](#)
- topK
 - nvinfer1::plugin::DetectionOutputParameters, [100](#)
 - nvinfer1::plugin::NMSPParameters, [674](#)
- TopKOperation
 - nvinfer1, [72](#)
- transposeSubBuffers
 - nvinfer1::utils, [89](#)
- TripLimit
 - nvinfer1, [73](#)
- TRT_DEPRECATED
 - NvInferRuntimeBase.h, [784](#)
- TRT_DEPRECATED_API
 - NvInferRuntimeBase.h, [784](#)
- TRT_DEPRECATED_ENUM
 - NvInferRuntimeBase.h, [784](#)
- TRTNOEXCEPT
 - NvInferRuntimeBase.h, [784](#)
- type
 - nvinfer1::IHostMemory, [365](#)
 - nvinfer1::PluginField, [676](#)
 - nvinfer1::PluginTensorDesc, [680](#)
 - nvinfer1::Weights, [692](#)
 - nvuffparser::FieldMap, [126](#)
- UFF_REQUIRED_VERSION_MAJOR
 - NvUffParser.h, [806](#)
- UFF_REQUIRED_VERSION_MINOR
 - NvUffParser.h, [807](#)
- UFF_REQUIRED_VERSION_PATCH
 - NvUffParser.h, [807](#)
- UffInputOrder
 - nvuffparser, [95](#)
- UnaryOperation
 - nvinfer1, [73](#)

- unmarkOutput
 - [nvinfer1::INetworkDefinition, 458](#)
- unmarkOutputForShapes
 - [nvinfer1::INetworkDefinition, 459](#)
- validate
 - [nvinfer1::consistency::IConsistencyChecker, 197](#)
 - [nvinfer1::consistency::IPluginChecker, 506](#)
- values
 - [nvinfer1::Weights, 692](#)
- variance
 - [nvinfer1::plugin::GridAnchorParameters, 128](#)
 - [nvinfer1::plugin::PriorBoxParameters, 684](#)
- varianceEncodedInTarget
 - [nvinfer1::plugin::DetectionOutputParameters, 100](#)
- Verbosity
 - [nvonnxparser::IOnnxConfig, 475](#)
- W
 - [nvinfer1::plugin::GridAnchorParameters, 128](#)
- w
 - [nvinfer1::DimsHW, 109](#)
- WeightsRole
 - [nvinfer1, 74](#)
- writeCalibrationCache
 - [nvinfer1::IInt8Calibrator, 378](#)
- writeHistogramCache
 - [nvinfer1::IInt8LegacyCalibrator, 382](#)

NVIDIA TensorRT Standard Python API Documentation

Release 8.6.13

NVIDIA

Jan 16, 2024

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	7
3.3.1	Volume	7
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	21
4.5.1	NetworkDefinitionCreationFlag	21
4.5.2	Builder	21
4.6	ICudaEngine	23
4.7	IExecutionContext	31
4.8	Runtime	38
4.9	Refitter	40
4.10	IErrorRecorder	42
4.11	ITimingCache	44

4.12	GPU Allocator	45
4.12.1	AllocatorFlag	45
4.12.2	IGpuAllocator	45
4.13	EngineInspector	46
5	Network	49
5.1	INetworkDefinition	49
5.2	Layer Base Classes	63
5.2.1	ITensor	63
5.2.2	ILayer	66
5.3	Layers	68
5.3.1	PaddingMode	68
5.3.2	IConvolutionLayer	69
5.3.3	IFullyConnectedLayer	70
5.3.4	IGridSampleLayer	70
5.3.5	IActivationLayer	71
5.3.6	IPoolingLayer	72
5.3.7	ILRNLayer	73
5.3.8	IScaleLayer	73
5.3.9	ISoftMaxLayer	74
5.3.10	IConcatenationLayer	75
5.3.11	IDeconvolutionLayer	75
5.3.12	IElementWiseLayer	76
5.3.13	IGatherLayer	76
5.3.14	RNN Layers	77
5.3.14.1	IRNNv2Layer	79
5.3.15	IPluginV2Layer	81
5.3.16	IUnaryLayer	81
5.3.17	IReduceLayer	82
5.3.18	IPaddingLayer	82
5.3.19	IParametricReLULayer	83
5.3.20	ISelectLayer	83
5.3.21	IShuffleLayer	83
5.3.22	ISliceLayer	84
5.3.23	IShapeLayer	85
5.3.24	ITopKLayer	86
5.3.25	IMatrixMultiplyLayer	86
5.3.26	IRaggedSoftMaxLayer	87
5.3.27	IIdentityLayer	87
5.3.28	IConstantLayer	87
5.3.29	IResizeLayer	88
5.3.30	ILoop	89
5.3.30.1	ILoopBoundaryLayer	90
5.3.30.1.1	ITripLimitLayer	90
5.3.30.1.2	IRecurrenceLayer	91
5.3.30.1.3	IIteratorLayer	91
5.3.30.1.4	ILoopOutputLayer	91
5.3.31	IFillLayer	92
5.3.32	IQuantizeLayer	93
5.3.33	IDequantizeLayer	93
5.3.34	IScatterLayer	94
5.3.35	IIfConditional	94
5.3.36	IConditionalLayer	95
5.3.37	IIfConditionalOutputLayer	95

5.3.38	IIfConditionalInputLayer	95
5.3.39	IEinsumLayer	95
5.3.40	IAssertionLayer	96
5.3.41	IOneHotLayer	96
5.3.42	INonZeroLayer	96
5.3.43	INMSLayer	96
5.3.44	IRReverseSequenceLayer	98
5.3.45	INormalizationLayer	98
6	Plugin	99
6.1	IPluginCreator	99
6.2	IPluginRegistry	101
7	Int8	103
7.1	IInt8Calibrator	103
7.2	IInt8LegacyCalibrator	105
7.3	IInt8EntropyCalibrator	106
7.4	IInt8EntropyCalibrator2	108
7.5	IInt8MinMaxCalibrator	109
8	Algorithm Selector	111
9	UFF Parser	115
9.1	Fields	116
10	Caffe Parser	119
10.1	Plugins	120
11	Onnx Parser	121
12	UFF Converter	125
12.1	Conversion Tools	125
12.1.1	Tensorflow Modelstream to UFF	125
12.1.2	Tensorflow Frozen Protobuf Model to UFF	126
13	UFF Operators	127
13.1	Input	127
13.1.1	Supported Datatypes	127
13.2	Identity	127
13.2.1	Inputs	127
13.2.2	Supported Datatypes	127
13.3	Const	127
13.3.1	Supported Datatypes	128
13.4	Conv	128
13.4.1	Inputs	128
13.4.2	Attributes	128
13.4.3	Supported Datatypes	128
13.5	ConvTranspose	128
13.5.1	Inputs	128
13.5.2	Attributes	129
13.5.3	Supported Datatypes	129
13.6	Pool	129
13.6.1	Inputs	129
13.6.2	Attributes	129
13.6.3	Supported Datatypes	129

13.7	FullyConnected	129
13.7.1	Inputs	129
13.7.2	Supported Datatypes	130
13.8	LRN	130
13.8.1	Inputs	130
13.8.2	Attributes	130
13.8.3	Supported Datatypes	130
13.9	Binary	130
13.9.1	Inputs	130
13.9.2	Attributes	130
13.9.3	Supported Datatypes	131
13.10	Unary	131
13.10.1	Inputs	131
13.10.2	Attributes	131
13.10.3	Supported Datatypes	131
13.11	Reshape	131
13.11.1	Inputs	131
13.11.2	Supported Datatypes	132
13.12	ExpandDims	132
13.12.1	Inputs	132
13.12.2	Attributes	132
13.12.3	Supported Datatypes	132
13.13	ArgMax	132
13.13.1	Inputs	132
13.13.2	Attributes	132
13.13.3	Supported Datatypes	133
13.14	ArgMin	133
13.14.1	Inputs	133
13.14.2	Attributes	133
13.14.3	Supported Datatypes	133
13.15	Transpose	133
13.15.1	Inputs	133
13.15.2	Attributes	133
13.15.3	Supported Datatypes	133
13.16	Reduce	134
13.16.1	Inputs	134
13.16.2	Attributes	134
13.16.3	Supported Datatypes	134
13.17	Concat	134
13.17.1	Inputs	134
13.17.2	Attributes	134
13.17.3	Supported Datatypes	135
13.18	MarkOutput	135
13.18.1	Inputs	135
13.18.2	Supported Datatypes	135
13.19	Activation	135
13.19.1	Inputs	135
13.19.2	Attributes	135
13.19.3	Supported Datatypes	135
13.20	Softmax	135
13.20.1	Inputs	136
13.20.2	Attributes	136
13.20.3	Supported Datatypes	136
13.21	BatchNorm	136

13.21.1	Inputs	136
13.21.2	Attributes	136
13.21.3	Supported Datatypes	136
13.22	Shape	136
13.22.1	Inputs	137
13.22.2	Supported Datatypes	137
13.23	StridedSlice	137
13.23.1	Inputs	137
13.23.2	Attributes	137
13.23.3	Supported Datatypes	137
13.24	Stack	137
13.24.1	Inputs	138
13.24.2	Attributes	138
13.24.3	Supported Datatypes	138
13.25	Squeeze	138
13.26	Flatten	138
13.26.1	Inputs	138
13.26.2	Supported Datatypes	138
13.27	Pad	138
13.27.1	Inputs	138
13.27.2	Supported Datatypes	139
13.28	Gather	139
13.28.1	Inputs	139
13.28.2	Supported Datatypes	139
13.29	GatherV2	139
13.29.1	Inputs	139
13.29.2	Attributes	139
13.29.3	Supported Datatypes	139
14	Graph Surgeon	141
14.1	Node Creation	141
14.2	Static Graph	142
14.3	Dynamic Graph (Inherits from StaticGraph)	144
Index		147

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 `itemsize` `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}`.

FP8 : Signed 8-bit floating point with 1 sign bit, 4 exponent bits, 3 mantissa bits, and exponent-bias 7.

Warning: fp8 is not supported yet and will result in an error or undefined behavior.

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`) $\rightarrow$ `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](#) . If the engine is built with profiling verbosity set to NONE, the `layerName` is the decimal index of the layer.
- **ms** – The time in milliseconds to execute the layer.

class `tensorrt.Profiler`(*self*: `tensorrt.tensorrt.Profiler`) → 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 CUBLAS tactic source will cause the cuBLAS handle passed to plugins in attachToContext to be null. **NOTE:** Setting CUBLAS tactic source takes no effect for core library if PreviewFeature.DISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805 is on.

CUBLAS_LT : Enables CUBLAS_LT tactics. Enabled by default. **NOTE:** Setting CUBLAS_LT tactic source takes no effect for core library if PreviewFeature.DISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805 is on.

CUDNN : Enables cuDNN tactics. Enabled by default. **NOTE:** Disabling CUDNN tactic source will cause the cuDNN handle passed to plugins in attachToContext to be null. **NOTE:** Setting CUDNN tactic source takes no effect for core library if PreviewFeature.DISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805 is on.

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 : [DEPRECATED] 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 : [DEPRECATED - will be removed in TensorRT 9.0] 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. The default value for this flag is on. Turning it off is deprecated.

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 on.

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.

`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.

`tensorrt.HardwareCompatibilityLevel`

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.

Members:

NONE : Do not require hardware compatibility with GPU architectures other than that of the GPU on which the engine was built.

AMPERE_PLUS : Require that the engine is compatible with Ampere and newer GPUs. This will limit the 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. This option will disable cuDNN, cuBLAS, and cuBLAS LT as tactic sources.

`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** – *Int8Calibrator* 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 3. Valid values include integers from 0 to the maximum optimization level, which is currently 5. Setting it to be greater than the maximum level results in identical behavior to the maximum level.
- **hardware_compatibility_level** – Hardware compatibility allows an engine compatible with GPU architectures other than that of the GPU on which the engine was built.
- **plugins_to_serialize** – The plugin libraries to be serialized with forward-compatible engines.
- **max_aux_streams** – The maximum number of auxiliary streams that TRT is allowed to use. If the network contains operators that can run in parallel, TRT can execute them using auxiliary streams in addition to the one provided to the `IExecutionContext::enqueueV3()` call. The default maximum number of auxiliary streams is determined by the heuristics in TensorRT on whether enabling multi-stream would improve the performance. This behavior can be overridden by calling this API to set the maximum number of auxiliary streams explicitly. Set this to 0 to enforce single-stream inference. The resulting engine may use fewer auxiliary streams than the maximum if the network does not contain enough parallelism or if TensorRT determines that using more auxiliary streams does not help improve the performance. Allowing more auxiliary streams does not always give better performance since there will be synchronizations overhead between streams. Using CUDA graphs at runtime can help reduce the overhead caused by cross-stream synchronizations. Using more auxiliary leads to more memory usage at runtime since some activation memory blocks will not be able to be reused.

__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.
- **num_aux_streams** – Read-only. The number of auxiliary streams used by this engine, which will be less than or equal to the maximum allowed number of auxiliary streams by setting `builder_config.max_aux_streams` when the engine is built.

__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.

`IEExecutionContext.execute_async_v2()` and `IEExecutionContext.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(*args, **kwargs)

Overloaded function.

1. `get_tensor_bytes_per_component(self: tensorrt.tensorrt.ICudaEngine, name: str) -> int`

Return the number of bytes per component of an element.

The vector component size is returned if `get_tensor_vectorized_dim()` != -1.

arg name The tensor name.

2. `get_tensor_bytes_per_component(self: tensorrt.tensorrt.ICudaEngine, name: str, profile_index: int) -> int`

Return the number of bytes per component of an element.

The vector component size is returned if `get_tensor_vectorized_dim()` != -1.

arg name The tensor name.

get_tensor_components_per_element(*args, **kwargs)

Overloaded function.

1. `get_tensor_components_per_element(self: tensorrt.tensorrt.ICudaEngine, name: str) -> int`

Return the number of components included in one element.

The number of elements in the vectors is returned if `get_tensor_vectorized_dim()` != -1.

arg name The tensor name.

2. `get_tensor_components_per_element(self: tensorrt.tensorrt.ICudaEngine, name: str, profile_index: int) -> int`

Return the number of components included in one element.

The number of elements in the vectors is returned if `get_tensor_vectorized_dim() != -1`.

arg name The tensor name.

get_tensor_dtype(*self*: `tensorrt.tensorrt.ICudaEngine`, *name*: *str*) → `tensorrt.tensorrt.DataType`
Return the required data type for a buffer from its tensor name.

Parameters **name** – The tensor name.

get_tensor_format(**args*, ***kwargs*)
Overloaded function.

1. `get_tensor_format(self: tensorrt.tensorrt.ICudaEngine, name: str) -> tensorrt.tensorrt.TensorFormat`

Return the tensor format.

arg name The tensor name.

2. `get_tensor_format(self: tensorrt.tensorrt.ICudaEngine, name: str, profile_index: int) -> tensorrt.tensorrt.TensorFormat`

Return the tensor format.

arg name The tensor name.

get_tensor_format_desc(**args*, ***kwargs*)
Overloaded function.

1. `get_tensor_format_desc(self: tensorrt.tensorrt.ICudaEngine, name: str) -> str`

Return the human readable description of the tensor format.

The description includes the order, vectorization, data type, strides, etc. For example:

Example 1: 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”

arg name The tensor name.

2. `get_tensor_format_desc(self: tensorrt.tensorrt.ICudaEngine, name: str, profile_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”

arg name The tensor name.

get_tensor_location(*self*: [tensorrt.tensorrt.ICudaEngine](#), *name*: *str*) → [tensorrt.tensorrt.TensorLocation](#)
Determine whether an input or output tensor must be on GPU or CPU.

Parameters name – The tensor name.

get_tensor_mode(*self*: [tensorrt.tensorrt.ICudaEngine](#), *name*: *str*) → [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(*args, **kwargs)
Overloaded function.

1. **get_tensor_vectorized_dim**(*self*: [tensorrt.tensorrt.ICudaEngine](#), *name*: *str*) -> *int*

Return the dimension index that the buffer is vectorized.

Specifically -1 is returned if scalars per vector is 1.

arg name The tensor name.

2. **get_tensor_vectorized_dim**(*self*: [tensorrt.tensorrt.ICudaEngine](#), *name*: *str*, *profile_index*: *int*) -> *int*

Return the dimension index that the buffer is vectorized.

Specifically -1 is returned if scalars per vector is 1.

arg name The tensor name.

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 [IExecutionContext.execute_v2\(\)](#) or [IExecutionContext.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** – [IErrorRecorder](#) Application-implemented error reporting interface for TensorRT objects.

- **enqueue_emits_profile** – bool Whether enqueue emits layer timing to the profiler. The default value is True. If set to False, enqueue will be asynchronous if there is a profiler attached. An extra method `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 en-queued.

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_aux_streams(*self*: `tensorrt.tensorrt.IExecutionContext`, *aux_streams*: `List[int]`) → None

Set the auxiliary streams that TensorRT should launch kernels on in the next `execute_async_v3()` call.

If set, TensorRT will launch the kernels that are supposed to run on the auxiliary streams using the streams provided by the user with this API. If this API is not called before the `execute_async_v3()` call, then TensorRT will use the auxiliary streams created by TensorRT internally.

TensorRT will always insert event synchronizations between the main stream provided via `execute_async_v3()` call and

- At the beginning of the `execute_async_v3()` call, TensorRT will make sure that all the auxiliary streams wait on the activities on the main stream.
- At the end of the `execute_async_v3()` call, TensorRT will make sure that the main stream wait on the activities on all the auxiliary streams.

The provided auxiliary streams must not be the default stream and must all be different to avoid deadlocks.

Parameters `aux_streams` – A list of cuda streams. If the length of the list is greater than `engine.num_aux_streams`, then only the first “`engine.num_aux_streams`” streams will be used. If the length is less than `engine.num_aux_streams`, such as an empty list, then TensorRT will use the provided streams for the first few auxiliary streams, and will create additional streams internally for the rest of the auxiliary streams.

set_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

tensorrt.TempfileControlFlag

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

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

These should be treated as bit offsets, e.g. in order to allow in-memory files for a given `IRuntime`:

```
runtime.tempfile_control_flags |= (1 << int(TempfileControlFlag.ALLOW_IN_MEMORY_
↪FILES))
```

Members:

`ALLOW_IN_MEMORY_FILES` : Allow creating and loading files in-memory (or unnamed files).

`ALLOW_TEMPORARY_FILES` : Allow creating and loading named files in a temporary directory on the filesystem.

class tensorrt.Runtime(*self*: `tensorrt.tensorrt.Runtime`, *logger*: `tensorrt.tensorrt.ILogger`) → None
Allows a serialized `ICudaEngine` to be deserialized.

Variables

- **error_recorder** – `IErrRecorder` Application-implemented error reporting interface for TensorRT objects.

- **gpu_allocator** – *IGpuAllocator* The GPU allocator to be used by the *Runtime*. All GPU memory acquired will use this allocator. If set to None, the default allocator will be used (Default: cudaMalloc/cudaFree).
- **DLA_core** – int The DLA core that the engine executes on. Must be between 0 and N-1 where N is the number of available DLA cores.
- **num_DLA_cores** – int The number of DLA engines available to this builder.
- **logger** – *ILogger* The logger provided when creating the refitter.
- **max_threads** – int The maximum thread that can be used by the *Runtime*.
- **temporary_directory** – str The temporary directory to use when loading executable code for engines. If set to None (the default), TensorRT will attempt to find a suitable directory for use using platform-specific heuristics: - On UNIX/Linux platforms, TensorRT will first try the TMPDIR environment variable, then fall back to /tmp - On Windows, TensorRT will try the TEMP environment variable.
- **tempfile_control_flags** – int Flags which control whether TensorRT is allowed to create in-memory or temporary files. See *TempfileControlFlag* for details.
- **engine_host_code_allowed** – bool Whether this runtime is allowed to deserialize engines that contain host executable code (Default: False).

Parameters **logger** – The logger to use.

__del__(*self: tensorrt.tensorrt.Runtime*) → None

__exit__(*exc_type, exc_value, traceback*)

Context managers are deprecated and have no effect. Objects are automatically freed when the reference count reaches 0.

__init__(*self: tensorrt.tensorrt.Runtime, logger: tensorrt.tensorrt.ILogger*) → None

Parameters **logger** – The logger to use.

deserialize_cuda_engine(*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.

load_runtime(*self: tensorrt.tensorrt.Runtime, path: str*) → *tensorrt.tensorrt.Runtime*

Load IRuntime from the file.

This method loads a runtime library from a shared library file. The runtime can then be used to execute a plan file built with BuilderFlag.VERSION_COMPATIBLE and BuilderFlag.EXCLUDE_LEAN_RUNTIME both set and built with the same version of TensorRT as the loaded runtime library.

Variables **path** – Path to the runtime lean library.

Returns The *IRuntime*, or None if it could not be loaded.

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** – *ILoggerRecorder* 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 increases until clear() is called.

Returns Returns the number of errors detected, or 0 if there are no errors.

report_error(*self*: [tensorrt.tensorrt.IErrorRecorder](#), *arg0*: [tensorrt.tensorrt.ErrorCodeTRT](#), *arg1*: *str*) → *bool*

Clear the error stack on the error recorder.

Report an error to the user that has a given value and human readable description. The function returns false if processing can continue, which implies that the reported error is not fatal. This does not guarantee that processing continues, but provides a hint to TensorRT.

Parameters

- **val** – The error code enum that is being reported.
- **desc** – The description of the error.

Returns True if the error is determined to be fatal and processing of the current function must end.

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.

To implement a custom allocator, ensure that you explicitly instantiate the base class in `__init__()` :

```
class MyAllocator(trt.IGpuAllocator):
    def __init__(self):
        trt.IGpuAllocator.__init__(self)
    ...
```

`__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.

reallocate(*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` = `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.

property builder

The builder from which this `INetworkDefinition` was created.

See `IBuilder` for more information.

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 product of the next dimension times the next stride. the strides are the same as for a C array with dimensions [N][C][H][W].

CHW2 : Two wide channel vectorized row major format.

This format is bound to FP16. 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 $[N][C][H][\text{roundUp}(W, 64/\text{elementSize})]$ where `elementSize` is 2 for FP16 and 1 for Int8, with the tensor coordinates (n, c, h, w) mapping to array subscript $[n][c][h][w]$.

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 $[N][H][\text{roundUp}(W, 32/C'/\text{elementSize})][C']$ 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 $[n][h][w][c]$.

DHWC : Non-vectorized channel-last format. This format is bound to FP32. It is only available for dimensions ≥ 4 .

class `tensorrt.ITensor`

A tensor in an [*INetworkDefinition*](#).

Variables

- **name** – `str` The tensor name. For a network input, the name is assigned by the application. For tensors which are layer outputs, a default name is assigned consisting of the layer name followed by the index of the output in brackets.

- **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 dynamic 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.

Ival metadata `str` The per-layer metadata.

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, SOFTPLUS, 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:

$$output = (input * scale + shift)^{power}$$

The coefficients can be applied on a per-tensor, per-channel, or per-element basis.

Note If the number of weights is 0, then a default value is used for shift, power, and scale. The default shift is 0, the default power is 1, and the default scale is 1.

The output size is the same as the input size.

Note The input tensor for this layer is required to have a minimum of 3 dimensions.

Variables

- **mode** – *ScaleMode* The scale mode.
- **shift** – *Weights* The shift value.
- **scale** – *Weights* The scale value.
- **power** – *Weights* The power value.
- **channel_axis** – int The channel axis.

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 `tensorrt.IConcatenationLayer`

A concatenation layer in an *INetworkDefinition*.

The output channel size is the sum of the channel sizes of the inputs. The other output sizes are the same as the other input sizes, which must all match.

Variables `axis` – `int` The axis along which concatenation occurs. The default axis is the number of tensor dimensions minus three, or zero if the tensor has fewer than three dimensions. For example, for a tensor with dimensions NCHW, it is C. For implicit batch mode, the number of tensor dimensions does NOT include the implicit batch dimension.

5.3.11 IDEconvolutionLayer

class `tensorrt.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 IIifConditionalOutputLayer

class `tensorrt.IIifConditionalOutputLayer`

Describes kinds of if-conditional outputs.

5.3.38 IIifConditionalInputLayer

class `tensorrt.IIifConditionalInputLayer`

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 `trt.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 `trt.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 $\text{Mean}(X, \text{axes})$ is a reduction over a set of axes, and $\text{Variance}(X) = \text{Mean}((X - \text{Mean}(X, \text{axes}))^2, \text{axes})$.

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.

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* [DEPRECATED] `TensorFormat` of the input/output of algorithm. This is deprecated since the strides, data type, and vectorization information is sufficient to uniquely identify tensor formats.
- **dtype** – *DataType* `DataType` of the input/output of algorithm.
- **strides** – *Dims* strides of the input/output tensor of algorithm.
- **vectorized_dim** – `int` the index of the vectorized dimension or -1 for non-vectorized formats.
- **components_per_element** – `int` the number of components per element. This is always 1 for non-vectorized formats.

`__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** – `IAlgorithmVariant`& 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 `IAlgorithmIOInfo`&

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
```

(continues on next page)

(continued from previous page)

```
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** – *ErrorRecorder* 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 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()`

```
clear_flag(self: tensorrt.tensorrt.OnnxParser, flag: nvonnxparser::OnnxParserFlag) → None
```

Clears the parser flag from the enabled flags.

Parameters `flag` – The flag to clear.

```
get_error(self: tensorrt.tensorrt.OnnxParser, index: int) → nvonnxparser::IParserError
```

Get an error that occurred during prior calls to `parse()`

Parameters `index` – Index of the error

```
get_flag(self: tensorrt.tensorrt.OnnxParser, flag: nvonnxparser::OnnxParserFlag) → bool
```

Check if a build mode flag is set.

Parameters `flag` – The flag to check.

Returns A *bool* indicating whether the flag is set.

get_used_vc_plugin_libraries(*self*: [tensorrt.tensorrt.OnnxParser](#)) → List[str]

Query the plugin libraries needed to implement operations used by the parser in a version-compatible engine.

This provides a list of plugin libraries on the filesystem needed to implement operations in the parsed network. If you are building a version-compatible engine using this network, provide this list to `IBuilderConfig.set_plugins_to_serialize()` to serialize these plugins along with the version-compatible engine, or, if you want to ship these plugin libraries externally to the engine, ensure that `IPluginRegistry.load_library()` is used to load these libraries in the appropriate runtime before deserializing the corresponding engine.

Returns List[str] List of plugin libraries found by the parser.

Raises `RuntimeError` if an internal error occurred when trying to fetch the list of plugin libraries.

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

set_flag(*self*: [tensorrt.tensorrt.OnnxParser](#), *flag*: [nvonnxparser::OnnxParserFlag](#)) → None

Add the input parser flag to the already enabled flags.

Parameters **flag** – The flag to set.

supports_model(*self*: [tensorrt.tensorrt.OnnxParser](#), *model*: *buffer*, *path*: *str = None*) → Tuple[bool, List[Tuple[List[int], bool]]]

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), 22
`__del__()` (*tensorrt.CaffeParser* method), 119
`__del__()` (*tensorrt.IBuilderConfig* method), 17
`__del__()` (*tensorrt.ICudaEngine* method), 24
`__del__()` (*tensorrt.IExecutionContext* method), 33
`__del__()` (*tensorrt.IHostMemory* method), 8
`__del__()` (*tensorrt.INetworkDefinition* method), 49
`__del__()` (*tensorrt.OnnxParser* method), 121
`__del__()` (*tensorrt.Refitter* method), 40
`__del__()` (*tensorrt.Runtime* method), 39
`__del__()` (*tensorrt.UffParser* method), 115
`__exit__()` (*tensorrt.Builder* method), 22
`__exit__()` (*tensorrt.CaffeParser* method), 119
`__exit__()` (*tensorrt.IBuilderConfig* method), 17
`__exit__()` (*tensorrt.ICudaEngine* method), 24
`__exit__()` (*tensorrt.IExecutionContext* method), 33
`__exit__()` (*tensorrt.IHostMemory* method), 8
`__exit__()` (*tensorrt.INetworkDefinition* method), 49
`__exit__()` (*tensorrt.OnnxParser* method), 121
`__exit__()` (*tensorrt.Refitter* method), 40
`__exit__()` (*tensorrt.Runtime* method), 39
`__exit__()` (*tensorrt.UffParser* method), 115
`__getitem__()` (*tensorrt.ICudaEngine* method), 24
`__getitem__()` (*tensorrt.INetworkDefinition* method), 49
`__init__()` (*tensorrt.Builder* method), 22
`__init__()` (*tensorrt.CaffeParser* method), 119
`__init__()` (*tensorrt.EngineInspector* method), 46
`__init__()` (*tensorrt.IAlgorithm* method), 112
`__init__()` (*tensorrt.IAlgorithmContext* method), 111
`__init__()` (*tensorrt.IAlgorithmIOInfo* method), 111
`__init__()` (*tensorrt.IAlgorithmSelector* method), 112
`__init__()` (*tensorrt.IAlgorithmVariant* method), 111
`__init__()` (*tensorrt.IBuilderConfig* method), 17
`__init__()` (*tensorrt.ICudaEngine* method), 25
`__init__()` (*tensorrt.IExecutionContext* method), 33
`__init__()` (*tensorrt.IGpuAllocator* method), 45
`__init__()` (*tensorrt.IHostMemory* method), 8
`__init__()` (*tensorrt.INetworkDefinition* method), 49
`__init__()` (*tensorrt.IOutputAllocator* method), 31
`__init__()` (*tensorrt.OnnxParser* method), 121

`__init__()` (*tensorrt.Refitter* method), 40
`__init__()` (*tensorrt.Runtime* method), 39
`__init__()` (*tensorrt.UffParser* method), 115
`__len__()` (*tensorrt.ICudaEngine* method), 25
`__len__()` (*tensorrt.INetworkDefinition* method), 49

A

ActivationType (in module *tensorrt*), 71
add_activation() (*tensorrt.INetworkDefinition* method), 49
add_assertion() (*tensorrt.INetworkDefinition* method), 50
add_cast() (*tensorrt.INetworkDefinition* method), 50
add_concatenation() (*tensorrt.INetworkDefinition* method), 50
add_constant() (*tensorrt.INetworkDefinition* method), 50
add_convolution() (*tensorrt.INetworkDefinition* method), 50
add_convolution_nd() (*tensorrt.INetworkDefinition* method), 51
add_deconvolution() (*tensorrt.INetworkDefinition* method), 51
add_deconvolution_nd() (*tensorrt.INetworkDefinition* method), 51
add_dequantize() (*tensorrt.INetworkDefinition* method), 52
add_einsum() (*tensorrt.INetworkDefinition* method), 52
add_elementwise() (*tensorrt.INetworkDefinition* method), 52
add_fill() (*tensorrt.INetworkDefinition* method), 52
add_fully_connected() (*tensorrt.INetworkDefinition* method), 52
add_gather() (*tensorrt.INetworkDefinition* method), 53
add_gather_v2() (*tensorrt.INetworkDefinition* method), 53
add_grid_sample() (*tensorrt.INetworkDefinition* method), 53
add_identity() (*tensorrt.INetworkDefinition* method), 53
add_if_conditional() (*tensorrt.INetworkDefinition* method), 53

`add_input()` (*tensorrt.IIfConditional method*), 94
`add_input()` (*tensorrt.INetworkDefinition method*), 54
`add_iterator()` (*tensorrt.ILoop method*), 89
`add_loop()` (*tensorrt.INetworkDefinition method*), 54
`add_loop_output()` (*tensorrt.ILoop method*), 89
`add_lrn()` (*tensorrt.INetworkDefinition method*), 54
`add_matrix_multiply()` (*tensorrt.INetworkDefinition method*), 54
`add_rms()` (*tensorrt.INetworkDefinition method*), 54
`add_non_zero()` (*tensorrt.INetworkDefinition method*), 55
`add_normalization()` (*tensorrt.INetworkDefinition method*), 55
`add_one_hot()` (*tensorrt.INetworkDefinition method*), 55
`add_optimization_profile()` (*tensorrt.IBuilderConfig method*), 17
`add_output()` (*tensorrt.IIfConditional method*), 94
`add_padding()` (*tensorrt.INetworkDefinition method*), 55
`add_padding_nd()` (*tensorrt.INetworkDefinition method*), 56
`add_parametric_relu()` (*tensorrt.INetworkDefinition method*), 56
`add_plugin_v2()` (*tensorrt.INetworkDefinition method*), 56
`add_pooling()` (*tensorrt.INetworkDefinition method*), 56
`add_pooling_nd()` (*tensorrt.INetworkDefinition method*), 56
`add_quantize()` (*tensorrt.INetworkDefinition method*), 57
`add_ragged_softmax()` (*tensorrt.INetworkDefinition method*), 57
`add_recurrence()` (*tensorrt.ILoop method*), 90
`add_reduce()` (*tensorrt.INetworkDefinition method*), 57
`add_resize()` (*tensorrt.INetworkDefinition method*), 57
`add_reverse_sequence()` (*tensorrt.INetworkDefinition method*), 58
`add_rnn_v2()` (*tensorrt.INetworkDefinition method*), 58
`add_scale()` (*tensorrt.INetworkDefinition method*), 59
`add_scale_nd()` (*tensorrt.INetworkDefinition method*), 59
`add_scatter()` (*tensorrt.INetworkDefinition method*), 60
`add_select()` (*tensorrt.INetworkDefinition method*), 60
`add_shape()` (*tensorrt.INetworkDefinition method*), 60
`add_shuffle()` (*tensorrt.INetworkDefinition method*), 60
`add_slice()` (*tensorrt.INetworkDefinition method*), 60
`add_softmax()` (*tensorrt.INetworkDefinition method*), 61
`add_topk()` (*tensorrt.INetworkDefinition method*), 61
`add_trip_limit()` (*tensorrt.ILoop method*), 90

`add_unary()` (*tensorrt.INetworkDefinition method*), 61
`allocate()` (*tensorrt.IGpuAllocator method*), 45
`AllocatorFlag` (in module *tensorrt*), 45
`append()` (*graphsurgeon.DynamicGraph method*), 144
`append()` (*tensorrt.PluginFieldCollection method*), 100
`as_graph_def()` (*graphsurgeon.StaticGraph method*), 142

B

`binding_is_input()` (*tensorrt.ICudaEngine method*), 25
`build_engine()` (*tensorrt.Builder method*), 22
`build_serialized_network()` (*tensorrt.Builder method*), 22
`Builder` (class in *tensorrt*), 21
`builder` (*tensorrt.INetworkDefinition property*), 61
`BuilderFlag` (in module *tensorrt*), 14

C

`CaffeParser` (class in *tensorrt*), 119
`CalibrationAlgoType` (in module *tensorrt*), 103
`can_run_on_DLA()` (*tensorrt.IBuilderConfig method*), 18
`clear()` (*tensorrt.IErrorRecorder method*), 43
`clear()` (*tensorrt.PluginFieldCollection method*), 100
`clear_errors()` (*tensorrt.OnnxParser method*), 121
`clear_flag()` (*tensorrt.IBuilderConfig method*), 18
`clear_flag()` (*tensorrt.OnnxParser method*), 121
`clear_quantization_flag()` (*tensorrt.IBuilderConfig method*), 18
`code()` (*tensorrt.ParserError method*), 123
`collapse_namespaces()` (*graphsurgeon.DynamicGraph method*), 144
`combine()` (*tensorrt.ITimingCache method*), 44
`create_builder_config()` (*tensorrt.Builder method*), 22
`create_engine_inspector()` (*tensorrt.ICudaEngine method*), 25
`create_execution_context()` (*tensorrt.ICudaEngine method*), 25
`create_execution_context_without_device_memory()` (*tensorrt.ICudaEngine method*), 25
`create_network()` (*tensorrt.Builder method*), 22
`create_node()` (in module *graphsurgeon*), 141
`create_optimization_profile()` (*tensorrt.Builder method*), 23
`create_plugin()` (*tensorrt.ICaffePluginFactoryV2 method*), 120
`create_plugin()` (*tensorrt.IPluginCreator method*), 100
`create_plugin_node()` (in module *graphsurgeon*), 141
`create_timing_cache()` (*tensorrt.IBuilderConfig method*), 18

D

`DataType` (in module `tensorrt`), 5
`deallocate()` (`tensorrt.IGpuAllocator` method), 45
`deregister_creator()` (`tensorrt.IPluginRegistry` method), 101
`deregister_library()` (`tensorrt.IPluginRegistry` method), 101
`desc()` (`tensorrt.ParserError` method), 123
`deserialize_cuda_engine()` (`tensorrt.Runtime` method), 39
`deserialize_plugin()` (`tensorrt.IPluginCreator` method), 100
`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`), 144

E

`ElementWiseOperation` (in module `tensorrt`), 76
`EngineCapability` (in module `tensorrt`), 13
`EngineInspector` (class in `tensorrt`), 46
`ErrorCode` (in module `tensorrt`), 122
`ErrorCodeTRT` (in module `tensorrt`), 42
`execute()` (`tensorrt.IExecutionContext` method), 33
`execute_async()` (`tensorrt.IExecutionContext` method), 33
`execute_async_v2()` (`tensorrt.IExecutionContext` method), 34
`execute_async_v3()` (`tensorrt.IExecutionContext` method), 34
`execute_v2()` (`tensorrt.IExecutionContext` method), 34
`extend()` (`graphsurgeon.DynamicGraph` method), 144
`extend()` (`tensorrt.PluginFieldCollection` method), 100

F

`FieldCollection` (class in `tensorrt`), 117
`FieldMap` (class in `tensorrt`), 116
`FieldType` (in module `tensorrt`), 116
`file()` (`tensorrt.ParserError` method), 123
`FillOperation` (in module `tensorrt`), 92
`find()` (`tensorrt.IBlobNameToTensor` method), 119
`find_node_chains_by_op()` (`graphsurgeon.StaticGraph` method), 142
`find_node_inputs()` (`graphsurgeon.StaticGraph` method), 142
`find_node_inputs_by_name()` (`graphsurgeon.StaticGraph` method), 142
`find_node_inputs_by_op()` (`graphsurgeon.StaticGraph` method), 143
`find_nodes_by_name()` (`graphsurgeon.StaticGraph` method), 143

`find_nodes_by_op()` (`graphsurgeon.StaticGraph` method), 143
`find_nodes_by_path()` (`graphsurgeon.StaticGraph` method), 143
`forward_inputs()` (`graphsurgeon.DynamicGraph` method), 144
`free()` (`tensorrt.IGpuAllocator` method), 45
`from_tensorflow()` (in module `uff`), 125
`from_tensorflow_frozen_model()` (in module `uff`), 126
`func()` (`tensorrt.ParserError` method), 123

G

`get_algorithm()` (`tensorrt.IInt8Calibrator` method), 103
`get_algorithm()` (`tensorrt.IInt8EntropyCalibrator` method), 106
`get_algorithm()` (`tensorrt.IInt8EntropyCalibrator2` method), 108
`get_algorithm()` (`tensorrt.IInt8LegacyCalibrator` method), 105
`get_algorithm()` (`tensorrt.IInt8MinMaxCalibrator` method), 109
`get_algorithm_io_info()` (`tensorrt.IAlgorithm` method), 112
`get_all()` (`tensorrt.Refitter` method), 40
`get_all_weights()` (`tensorrt.Refitter` method), 40
`get_batch()` (`tensorrt.IInt8Calibrator` method), 103
`get_batch()` (`tensorrt.IInt8EntropyCalibrator` method), 106
`get_batch()` (`tensorrt.IInt8EntropyCalibrator2` method), 108
`get_batch()` (`tensorrt.IInt8LegacyCalibrator` method), 105
`get_batch()` (`tensorrt.IInt8MinMaxCalibrator` method), 109
`get_batch_size()` (`tensorrt.IInt8Calibrator` method), 104
`get_batch_size()` (`tensorrt.IInt8EntropyCalibrator` method), 107
`get_batch_size()` (`tensorrt.IInt8EntropyCalibrator2` method), 108
`get_batch_size()` (`tensorrt.IInt8LegacyCalibrator` method), 105
`get_batch_size()` (`tensorrt.IInt8MinMaxCalibrator` method), 110
`get_bias_for_gate()` (`tensorrt.IRNNv2Layer` method), 80
`get_binding_bytes_per_component()` (`tensorrt.ICudaEngine` method), 25
`get_binding_components_per_element()` (`tensorrt.ICudaEngine` method), 25
`get_binding_dtype()` (`tensorrt.ICudaEngine` method), 25

<code>get_binding_format()</code> (<i>tensorrt.ICudaEngine method</i>), 26	<code>get_plugin_registry()</code> (<i>tensorrt.Runtime method</i>), 39
<code>get_binding_format_desc()</code> (<i>tensorrt.ICudaEngine method</i>), 26	<code>get_preview_feature()</code> (<i>tensorrt.IBuilderConfig method</i>), 18
<code>get_binding_index()</code> (<i>tensorrt.ICudaEngine method</i>), 26	<code>get_profile_shape()</code> (<i>tensorrt.ICudaEngine method</i>), 27
<code>get_binding_name()</code> (<i>tensorrt.ICudaEngine method</i>), 26	<code>get_profile_shape_input()</code> (<i>tensorrt.ICudaEngine method</i>), 27
<code>get_binding_shape()</code> (<i>tensorrt.ICudaEngine method</i>), 26	<code>get_quantization_flag()</code> (<i>tensorrt.IBuilderConfig method</i>), 19
<code>get_binding_shape()</code> (<i>tensorrt.IExecutionContext method</i>), 34	<code>get_shape()</code> (<i>tensorrt.IAlgorithmContext method</i>), 111
<code>get_binding_vectorized_dim()</code> (<i>tensorrt.ICudaEngine method</i>), 27	<code>get_shape()</code> (<i>tensorrt.IExecutionContext method</i>), 35
<code>get_builder_plugin_registry()</code> (<i>in module tensorrt</i>), 102	<code>get_shape()</code> (<i>tensorrt.IOptimizationProfile method</i>), 11
<code>get_calibration_profile()</code> (<i>tensorrt.IBuilderConfig method</i>), 18	<code>get_shape_input()</code> (<i>tensorrt.IOptimizationProfile method</i>), 11
<code>get_device_type()</code> (<i>tensorrt.IBuilderConfig method</i>), 18	<code>get_strides()</code> (<i>tensorrt.IExecutionContext method</i>), 35
<code>get_dimension_name()</code> (<i>tensorrt.ITensor method</i>), 65	<code>get_tactic_sources()</code> (<i>tensorrt.IBuilderConfig method</i>), 19
<code>get_dynamic_range()</code> (<i>tensorrt.Refitter method</i>), 40	<code>get_tensor_address()</code> (<i>tensorrt.IExecutionContext method</i>), 35
<code>get_engine_information()</code> (<i>tensorrt.EngineInspector method</i>), 46	<code>get_tensor_bytes_per_component()</code> (<i>tensorrt.ICudaEngine method</i>), 28
<code>get_error()</code> (<i>tensorrt.OnnxParser method</i>), 121	<code>get_tensor_components_per_element()</code> (<i>tensorrt.ICudaEngine method</i>), 28
<code>get_error_code()</code> (<i>tensorrt.IErrorRecorder method</i>), 43	<code>get_tensor_dtype()</code> (<i>tensorrt.ICudaEngine method</i>), 29
<code>get_error_desc()</code> (<i>tensorrt.IErrorRecorder method</i>), 43	<code>get_tensor_format()</code> (<i>tensorrt.ICudaEngine method</i>), 29
<code>get_flag()</code> (<i>tensorrt.IBuilderConfig method</i>), 18	<code>get_tensor_format_desc()</code> (<i>tensorrt.ICudaEngine method</i>), 29
<code>get_flag()</code> (<i>tensorrt.OnnxParser method</i>), 121	<code>get_tensor_location()</code> (<i>tensorrt.ICudaEngine method</i>), 30
<code>get_input()</code> (<i>tensorrt.ILayer method</i>), 67	<code>get_tensor_mode()</code> (<i>tensorrt.ICudaEngine method</i>), 30
<code>get_input()</code> (<i>tensorrt.INetworkDefinition method</i>), 61	<code>get_tensor_name()</code> (<i>tensorrt.ICudaEngine method</i>), 30
<code>get_input_consumed_event()</code> (<i>tensorrt.IExecutionContext method</i>), 35	<code>get_tensor_profile_shape()</code> (<i>tensorrt.ICudaEngine method</i>), 30
<code>get_layer()</code> (<i>tensorrt.INetworkDefinition method</i>), 62	<code>get_tensor_shape()</code> (<i>tensorrt.ICudaEngine method</i>), 30
<code>get_layer_information()</code> (<i>tensorrt.EngineInspector method</i>), 46	<code>get_tensor_shape()</code> (<i>tensorrt.IExecutionContext method</i>), 35
<code>get_location()</code> (<i>tensorrt.ICudaEngine method</i>), 27	<code>get_tensor_strides()</code> (<i>tensorrt.IExecutionContext method</i>), 35
<code>get_max_output_size()</code> (<i>tensorrt.IExecutionContext method</i>), 35	<code>get_tensor_vectorized_dim()</code> (<i>tensorrt.ICudaEngine method</i>), 30
<code>get_memory_pool_limit()</code> (<i>tensorrt.IBuilderConfig method</i>), 18	<code>get_tensors_with_dynamic_range()</code> (<i>tensorrt.Refitter method</i>), 41
<code>get_missing()</code> (<i>tensorrt.Refitter method</i>), 40	<code>get_timing_cache()</code> (<i>tensorrt.IBuilderConfig method</i>), 19
<code>get_missing_weights()</code> (<i>tensorrt.Refitter method</i>), 40	<code>get_used_vc_plugin_libraries()</code> (<i>tensorrt.OnnxParser method</i>), 121
<code>get_output()</code> (<i>tensorrt.ILayer method</i>), 67	<code>get_weights_for_gate()</code> (<i>tensorrt.IRNNv2Layer</i>
<code>get_output()</code> (<i>tensorrt.INetworkDefinition method</i>), 62	
<code>get_output_allocator()</code> (<i>tensorrt.IExecutionContext method</i>), 35	
<code>get_output_type()</code> (<i>tensorrt.ILayer method</i>), 68	
<code>get_plugin_creator()</code> (<i>tensorrt.IPluginRegistry method</i>), 101	
<code>get_plugin_registry()</code> (<i>in module tensorrt</i>), 102	
<code>get_plugin_registry()</code> (<i>tensorrt.Builder method</i>), 23	

- method), 80
 graph_inputs (graphsurgeon.StaticGraph attribute), 142
 graph_outputs (graphsurgeon.StaticGraph attribute), 142
- ## H
- HardwareCompatibilityLevel (in module tensorrt), 16
 has_overflowed() (tensorrt.IErrorRecorder method), 43
- ## I
- IActivationLayer (class in tensorrt), 71
 IAlgorithm (class in tensorrt), 112
 IAlgorithmContext (class in tensorrt), 111
 IAlgorithmIOInfo (class in tensorrt), 111
 IAlgorithmSelector (class in tensorrt), 112
 IAlgorithmVariant (class in tensorrt), 111
 IAssertionLayer (class in tensorrt), 96
 IBlobNameToTensor (class in tensorrt), 119
 IBuilderConfig (class in tensorrt), 16
 ICaffePluginFactoryV2 (class in tensorrt), 120
 IConcatenationLayer (class in tensorrt), 75
 IConditionLayer (class in tensorrt), 95
 IConstantLayer (class in tensorrt), 87
 IConvolutionLayer (class in tensorrt), 69
 ICudaEngine (class in tensorrt), 23
 IDEconvolutionLayer (class in tensorrt), 75
 IDEquantizeLayer (class in tensorrt), 93
 IEinsumLayer (class in tensorrt), 95
 IElementWiseLayer (class in tensorrt), 76
 IErrorRecorder (class in tensorrt), 43
 IExecutionContext (class in tensorrt), 32
 IFillLayer (class in tensorrt), 92
 IFullyConnectedLayer (class in tensorrt), 70
 IGatherLayer (class in tensorrt), 76
 IGPUAllocator (class in tensorrt), 45
 IGridSampleLayer (class in tensorrt), 71
 IHostMemory (class in tensorrt), 8
 IIdentityLayer (class in tensorrt), 87
 IIIfConditional (class in tensorrt), 94
 IIIfConditionalInputLayer (class in tensorrt), 95
 IIIfConditionalOutputLayer (class in tensorrt), 95
 IInt8Calibrator (class in tensorrt), 103
 IInt8EntropyCalibrator (class in tensorrt), 106
 IInt8EntropyCalibrator2 (class in tensorrt), 108
 IInt8LegacyCalibrator (class in tensorrt), 105
 IInt8MinMaxCalibrator (class in tensorrt), 109
 IIteratorLayer (class in tensorrt), 91
 ILayer (class in tensorrt), 67
 ILogger (class in tensorrt), 9
 ILogger.Severity (class in tensorrt), 9
 ILoop (class in tensorrt), 89
 ILoopBoundaryLayer (class in tensorrt), 90
 ILoopOutputLayer (class in tensorrt), 91
 ILRNLayer (class in tensorrt), 73
 IMatrixMultiplyLayer (class in tensorrt), 86
 INetworkDefinition (class in tensorrt), 49
 infer_shapes() (tensorrt.IExecutionContext method), 35
 init_libnvinfer_plugins() (in module tensorrt), 102
 INMSLayer (class in tensorrt), 96
 INonZeroLayer (class in tensorrt), 96
 INormalizationLayer (class in tensorrt), 98
 insert() (tensorrt.PluginFieldCollection method), 100
 InterpolationMode (in module tensorrt), 70
 IOneHotLayer (class in tensorrt), 96
 IOptimizationProfile (class in tensorrt), 11
 IOutputAllocator (class in tensorrt), 31
 IPaddingLayer (class in tensorrt), 82
 IParametricReLULayer (class in tensorrt), 83
 IPluginCreator (class in tensorrt), 100
 IPluginRegistry (class in tensorrt), 101
 IPluginV2Layer (class in tensorrt), 81
 IPoolingLayer (class in tensorrt), 72
 IProfiler (class in tensorrt), 10
 IQuantizeLayer (class in tensorrt), 93
 IRaggedSoftMaxLayer (class in tensorrt), 87
 IRecurrenceLayer (class in tensorrt), 91
 IReduceLayer (class in tensorrt), 82
 IResizeLayer (class in tensorrt), 88
 IReverseSequenceLayer (class in tensorrt), 98
 IRNNv2Layer (class in tensorrt), 79
 is_device_type_set() (tensorrt.IBuilderConfig method), 19
 is_execution_binding() (tensorrt.ICudaEngine method), 30
 is_network_supported() (tensorrt.Builder method), 23
 is_plugin_v2() (tensorrt.ICaffePluginFactoryV2 method), 120
 is_shape_binding() (tensorrt.ICudaEngine method), 30
 is_shape_inference_io() (tensorrt.ICudaEngine method), 31
 IScaleLayer (class in tensorrt), 73
 IScatterLayer (class in tensorrt), 94
 ISelectLayer (class in tensorrt), 83
 IShapeLayer (class in tensorrt), 85
 IShuffleLayer (class in tensorrt), 83
 ISliceLayer (class in tensorrt), 84
 ISoftMaxLayer (class in tensorrt), 74
 ITensor (class in tensorrt), 64
 ITimingCache (class in tensorrt), 44
 ITopKLayer (class in tensorrt), 86
 ITripLimitLayer (class in tensorrt), 90

`IUnaryLayer` (class in `tensorrt`), 82

L

`LayerType` (in module `tensorrt`), 66

`line()` (`tensorrt.ParserError` method), 123

`load_library()` (`tensorrt.IPluginRegistry` method), 101

`load_runtime()` (`tensorrt.Runtime` method), 39

`log()` (`tensorrt.ILogger` method), 9

`log()` (`tensorrt.Logger` method), 10

`Logger` (class in `tensorrt`), 10

`LoopOutput` (in module `tensorrt`), 91

M

`mark_output()` (`tensorrt.INetworkDefinition` method), 62

`mark_output_for_shapes()` (`tensorrt.INetworkDefinition` method), 62

`MatrixOperation` (in module `tensorrt`), 86

`MAX_DIMS` (`tensorrt.Dims` property), 7

`MemoryPoolType` (in module `tensorrt`), 15

N

`name` (`tensorrt.ILogger.Severity` property), 9

`NetworkDefinitionCreationFlag` (in module `tensorrt`), 21

`node()` (`tensorrt.ParserError` method), 123

`node_map` (`graphsurgeon.StaticGraph` attribute), 142

`node_outputs` (`graphsurgeon.StaticGraph` attribute), 142

`notify_shape()` (`tensorrt.IOutputAllocator` method), 31

`nptype()` (in module `tensorrt`), 5

`num_errors()` (`tensorrt.IErrorRecorder` method), 43

`numpy()` (`tensorrt.Weights` method), 6

O

`OnnxParser` (class in `tensorrt`), 121

`output_type_is_set()` (`tensorrt.ILayer` method), 68

P

`PaddingMode` (in module `tensorrt`), 68

`parse()` (`tensorrt.CaffeParser` method), 119

`parse()` (`tensorrt.OnnxParser` method), 122

`parse()` (`tensorrt.UffParser` method), 115

`parse_binary_proto()` (`tensorrt.CaffeParser` method), 120

`parse_buffer()` (`tensorrt.CaffeParser` method), 120

`parse_buffer()` (`tensorrt.UffParser` method), 115

`parse_from_file()` (`tensorrt.OnnxParser` method), 122

`parse_with_weight_descriptors()` (`tensorrt.OnnxParser` method), 122

`ParserError` (class in `tensorrt`), 123

`Permutation` (class in `tensorrt`), 83

`PluginField` (class in `tensorrt`), 99

`PluginFieldCollection` (class in `tensorrt`), 99

`PluginFieldType` (in module `tensorrt`), 99

`PoolingType` (in module `tensorrt`), 72

`pop()` (`tensorrt.PluginFieldCollection` method), 100

`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), 143

`read_calibration_cache()` (`tensorrt.Int8Calibrator` method), 104

`read_calibration_cache()` (`tensorrt.Int8EntropyCalibrator` method), 107

`read_calibration_cache()` (`tensorrt.Int8EntropyCalibrator2` method), 108

`read_calibration_cache()` (`tensorrt.Int8LegacyCalibrator` method), 105

`read_calibration_cache()` (`tensorrt.Int8MinMaxCalibrator` method), 110

`reallocate()` (`tensorrt.IGpuAllocator` method), 45

`reallocate_output()` (`tensorrt.IOutputAllocator` method), 32

`ReduceOperation` (in module `tensorrt`), 82

`refit_cuda_engine()` (`tensorrt.Refitter` method), 41

`Refitter` (class in `tensorrt`), 40

`register_creator()` (`tensorrt.IPluginRegistry` method), 102

`register_input()` (`tensorrt.UffParser` method), 116

`register_output()` (`tensorrt.UffParser` method), 116

`remove()` (`graphsurgeon.DynamicGraph` method), 145

`remove_tensor()` (`tensorrt.INetworkDefinition` method), 62

`report_algorithms()` (`tensorrt.IAlgorithmSelector` method), 112

`report_error()` (`tensorrt.IErrorRecorder` method), 44

`report_layer_time()` (`tensorrt.IProfiler` method), 10

`report_layer_time()` (`tensorrt.Profiler` method), 10

`report_to_profiler()` (`tensorrt.IExecutionContext` method), 36

`reset()` (`tensorrt.Builder` method), 23

`reset()` (`tensorrt.IBuilderConfig` method), 19

`reset()` (`tensorrt.ITimingCache` method), 44

`reset_device_type()` (`tensorrt.IBuilderConfig` method), 19

`reset_dynamic_range()` (`tensorrt.ITensor` method), 65

`reset_output_type()` (`tensorrt.ILayer` method), 68

reset_precision() (*tensorrt.ILayer method*), 68
 ResizeMode (*in module tensorrt*), 88
 RNNDirection (*in module tensorrt*), 78
 RNNGateType (*in module tensorrt*), 79
 RNNInputMode (*in module tensorrt*), 78
 RNNOperation (*in module tensorrt*), 77
 Runtime (*class in tensorrt*), 38

S

SampleMode (*in module tensorrt*), 70
 ScaleMode (*in module tensorrt*), 73
 select_algorithms() (*tensorrt.IAlgorithmSelector method*), 113
 serialize() (*tensorrt.ICudaEngine method*), 31
 serialize() (*tensorrt.ITimingCache method*), 44
 set_aux_streams() (*tensorrt.IExecutionContext method*), 36
 set_bias_for_gate() (*tensorrt.IRNNv2Layer method*), 80
 set_binding_shape() (*tensorrt.IExecutionContext method*), 36
 set_calibration_profile() (*tensorrt.IBuilderConfig method*), 19
 set_condition() (*tensorrt.IIfConditional method*), 94
 set_device_type() (*tensorrt.IBuilderConfig method*), 19
 set_dimension_name() (*tensorrt.ITensor method*), 65
 set_dynamic_range() (*tensorrt.ITensor method*), 65
 set_dynamic_range() (*tensorrt.Refit method*), 41
 set_flag() (*tensorrt.IBuilderConfig method*), 19
 set_flag() (*tensorrt.OnnxParser method*), 122
 set_input() (*tensorrt.IFillLayer method*), 92
 set_input() (*tensorrt.ILayer method*), 68
 set_input() (*tensorrt.ILoopOutputLayer method*), 92
 set_input() (*tensorrt.INMSLayer method*), 97
 set_input() (*tensorrt.IRecurrenceLayer method*), 91
 set_input() (*tensorrt.IResizeLayer method*), 89
 set_input() (*tensorrt.IShuffleLayer method*), 84
 set_input() (*tensorrt.ISliceLayer method*), 85
 set_input() (*tensorrt.ITopKLayer method*), 86
 set_input_consumed_event() (*tensorrt.IExecutionContext method*), 37
 set_input_shape() (*tensorrt.IExecutionContext method*), 37
 set_memory_pool_limit() (*tensorrt.IBuilderConfig method*), 20
 set_named_weights() (*tensorrt.Refit method*), 41
 set_optimization_profile_async() (*tensorrt.IExecutionContext method*), 37
 set_output_allocator() (*tensorrt.IExecutionContext method*), 37
 set_output_type() (*tensorrt.ILayer method*), 68
 set_preview_feature() (*tensorrt.IBuilderConfig method*), 20

set_quantization_flag() (*tensorrt.IBuilderConfig method*), 20
 set_shape() (*tensorrt.IOptimizationProfile method*), 11
 set_shape_input() (*tensorrt.IExecutionContext method*), 37
 set_shape_input() (*tensorrt.IOptimizationProfile method*), 12
 set_tactic_sources() (*tensorrt.IBuilderConfig method*), 20
 set_tensor_address() (*tensorrt.IExecutionContext method*), 38
 set_timing_cache() (*tensorrt.IBuilderConfig method*), 21
 set_weights() (*tensorrt.Refit method*), 41
 set_weights_for_gate() (*tensorrt.IRNNv2Layer method*), 80
 set_weights_name() (*tensorrt.INetworkDefinition method*), 62
 shutdown_protobuf_library() (*in module tensorrt*), 120
 SliceMode (*in module tensorrt*), 84
 StaticGraph (*class in graphsurgeon*), 142
 supports_model() (*tensorrt.OnnxParser method*), 122
 supports_operator() (*tensorrt.OnnxParser method*), 122

T

TacticSource (*in module tensorrt*), 13
 TempfileControlFlag (*in module tensorrt*), 38
 TensorFormat (*in module tensorrt*), 63
 TensorIOMode (*in module tensorrt*), 23
 TensorLocation (*in module tensorrt*), 63
 TopKOperation (*in module tensorrt*), 86
 TripLimit (*in module tensorrt*), 90

U

UffInputOrder (*in module tensorrt*), 115
 UffParser (*class in tensorrt*), 115
 UnaryOperation (*in module tensorrt*), 81
 unmark_output() (*tensorrt.INetworkDefinition method*), 62
 unmark_output_for_shapes() (*tensorrt.INetworkDefinition method*), 62

V

volume() (*in module tensorrt*), 7

W

Weights (*class in tensorrt*), 6
 WeightsRole (*in module tensorrt*), 6
 write() (*graphsurgeon.StaticGraph method*), 143
 write_calibration_cache() (*tensorrt.IInt8Calibrator method*), 104

`write_calibration_cache()` (*ten-*
sorrt.IInt8EntropyCalibrator method), [107](#)

`write_calibration_cache()` (*ten-*
sorrt.IInt8EntropyCalibrator2 method),
[109](#)

`write_calibration_cache()` (*ten-*
sorrt.IInt8LegacyCalibrator method), [106](#)

`write_calibration_cache()` (*ten-*
sorrt.IInt8MinMaxCalibrator method), [110](#)

`write_tensorboard()` (*graphsurgeon.StaticGraph*
method), [143](#)

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 NON-INFRINGEMENT, 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

© 2017-2024 NVIDIA Corporation & affiliates. All rights reserved.

