



NVIDIA DRIVE OS 6.0 TensorRT 8.4.11

API Reference

NVIDIA DRIVE OS 6.0.4

TensorRT API Reference

8.4.11

June 2022

Table 1 Revision history

Date	Summary of change
June 2, 2022	Initial draft
June 2 - June 21, 2022	Review
June 22, 2022	Approval review

Contents

1	Standard and Safe	1
2	TensorRT	3
3	Deprecated List	5
4	Namespace Index	11
4.1	Namespace List	11
5	Hierarchical Index	13
5.1	Class Hierarchy	13
6	Class Index	17
6.1	Class List	17
7	File Index	23
7.1	File List	23
8	Namespace Documentation	25
8.1	nvcaffeparser1 Namespace Reference	25
8.1.1	Detailed Description	25
8.1.2	Function Documentation	25
8.1.2.1	createCaffeParser()	26
8.1.2.2	shutdownProtobufLibrary()	26
8.2	nvinfer1 Namespace Reference	26
8.2.1	Detailed Description	35
8.2.2	Typedef Documentation	35
8.2.2.1	AllocatorFlags	35
8.2.2.2	AsciiChar	35
8.2.2.3	BuilderFlags	35
8.2.2.4	char_t	36
8.2.2.5	Dims	36
8.2.2.6	NetworkDefinitionCreationFlags	36
8.2.2.7	PluginFormat	36
8.2.2.8	QuantizationFlags	37
8.2.2.9	TacticSources	37
8.2.2.10	TensorFormats	37
8.2.3	Enumeration Type Documentation	37
8.2.3.1	ActivationType	37
8.2.3.2	AllocatorFlag	38
8.2.3.3	BuilderFlag	38
8.2.3.4	CalibrationAlgoType	39
8.2.3.5	DataType	40

8.2.3.6	DeviceType	40
8.2.3.7	DimensionOperation	40
8.2.3.8	ElementWiseOperation	41
8.2.3.9	EngineCapability	42
8.2.3.10	ErrorCode	43
8.2.3.11	FillOperation	44
8.2.3.12	GatherMode	44
8.2.3.13	LayerInformationFormat	45
8.2.3.14	LayerType	45
8.2.3.15	LoopOutput	46
8.2.3.16	MatrixOperation	47
8.2.3.17	MemoryPoolType	47
8.2.3.18	NetworkDefinitionCreationFlag	48
8.2.3.19	OptProfileSelector	48
8.2.3.20	PaddingMode	49
8.2.3.21	PluginFieldType	52
8.2.3.22	PluginVersion	52
8.2.3.23	PoolingType	53
8.2.3.24	ProfilingVerbosity	53
8.2.3.25	QuantizationFlag	53
8.2.3.26	ReduceOperation	54
8.2.3.27	ResizeCoordinateTransformation	54
8.2.3.28	ResizeMode	55
8.2.3.29	ResizeRoundMode	55
8.2.3.30	ResizeSelector	56
8.2.3.31	RNNDirection	56
8.2.3.32	RNNGateType	57
8.2.3.33	RNNInputMode	57
8.2.3.34	RNNOperation	58
8.2.3.35	ScaleMode	59
8.2.3.36	ScatterMode	59
8.2.3.37	SliceMode	59
8.2.3.38	TacticSource	60
8.2.3.39	TensorFormat	60
8.2.3.40	TensorLocation	62
8.2.3.41	TopKOperation	62
8.2.3.42	TripLimit	63
8.2.3.43	UnaryOperation	63
8.2.3.44	WeightsRole	64
8.2.4	Function Documentation	64
8.2.4.1	EnumMax()	64
8.2.4.2	EnumMax< BuilderFlag >()	65
8.2.4.3	EnumMax< CalibrationAlgoType >()	65
8.2.4.4	EnumMax< DeviceType >()	65
8.2.4.5	EnumMax< DimensionOperation >()	65
8.2.4.6	EnumMax< FillOperation >()	66
8.2.4.7	EnumMax< GatherMode >()	66
8.2.4.8	EnumMax< LayerInformationFormat >()	66
8.2.4.9	EnumMax< LayerType >()	66
8.2.4.10	EnumMax< LoopOutput >()	67
8.2.4.11	EnumMax< MatrixOperation >()	67
8.2.4.12	EnumMax< MemoryPoolType >()	67
8.2.4.13	EnumMax< NetworkDefinitionCreationFlag >()	67
8.2.4.14	EnumMax< OptProfileSelector >()	68

8.2.4.15	EnumMax< ProfilingVerbosity >()	68
8.2.4.16	EnumMax< QuantizationFlag >()	68
8.2.4.17	EnumMax< ReduceOperation >()	68
8.2.4.18	EnumMax< RNNDirection >()	69
8.2.4.19	EnumMax< RNNGateType >()	69
8.2.4.20	EnumMax< RNNInputMode >()	69
8.2.4.21	EnumMax< RNNOperation >()	69
8.2.4.22	EnumMax< ScaleMode >()	70
8.2.4.23	EnumMax< ScatterMode >()	70
8.2.4.24	EnumMax< SliceMode >()	70
8.2.4.25	EnumMax< TacticSource >()	70
8.2.4.26	EnumMax< TopKOperation >()	71
8.2.4.27	EnumMax< TripLimit >()	71
8.2.4.28	EnumMax< UnaryOperation >()	71
8.2.4.29	EnumMax< WeightsRole >()	71
8.2.4.30	getBuilderPluginRegistry()	72
8.3	nvinfer1::consistency Namespace Reference	72
8.4	nvinfer1::impl Namespace Reference	72
8.5	nvinfer1::plugin Namespace Reference	73
8.5.1	Enumeration Type Documentation	73
8.5.1.1	CodeTypeSSD	73
8.6	nvinfer1::safe Namespace Reference	74
8.6.1	Detailed Description	74
8.6.2	Function Documentation	74
8.6.2.1	createInferRuntime()	75
8.6.2.2	getSafePluginRegistry()	75
8.7	nvinfer1::utils Namespace Reference	75
8.7.1	Function Documentation	76
8.7.1.1	reorderSubBuffers()	76
8.7.1.2	reshapeWeights()	77
8.7.1.3	transposeSubBuffers()	78
8.8	nvonnxparser Namespace Reference	78
8.8.1	Detailed Description	79
8.8.2	Enumeration Type Documentation	79
8.8.2.1	ErrorCode	79
8.8.3	Function Documentation	80
8.8.3.1	createONNXConfig()	80
8.8.3.2	EnumMax()	80
8.8.3.3	EnumMax< ErrorCode >()	80
8.9	nvuffparser Namespace Reference	80
8.9.1	Detailed Description	81
8.9.2	Enumeration Type Documentation	81
8.9.2.1	FieldType	81
8.9.2.2	UffInputOrder	81
8.9.3	Function Documentation	82
8.9.3.1	createUffParser()	82
8.9.3.2	shutdownProtobufLibrary()	82
9	Class Documentation	83
9.1	nvinfer1::plugin::DetectionOutputParameters Struct Reference	83
9.1.1	Detailed Description	83
9.1.2	Member Data Documentation	84
9.1.2.1	backgroundLabelId	84
9.1.2.2	codeType	84

9.1.2.3	confidenceThreshold	84
9.1.2.4	confSigmoid	85
9.1.2.5	inputOrder	85
9.1.2.6	isBatchAgnostic	85
9.1.2.7	isNormalized	85
9.1.2.8	keepTopK	85
9.1.2.9	nmsThreshold	85
9.1.2.10	numClasses	85
9.1.2.11	shareLocation	86
9.1.2.12	topK	86
9.1.2.13	varianceEncodedInTarget	86
9.2	Dims Class Reference	86
9.2.1	Detailed Description	86
9.3	nvinfer1::Dims2 Class Reference	87
9.3.1	Detailed Description	87
9.3.2	Constructor & Destructor Documentation	87
9.3.2.1	Dims2() [1/2]	87
9.3.2.2	Dims2() [2/2]	87
9.4	nvinfer1::Dims3 Class Reference	88
9.4.1	Detailed Description	88
9.4.2	Constructor & Destructor Documentation	88
9.4.2.1	Dims3() [1/2]	89
9.4.2.2	Dims3() [2/2]	89
9.5	nvinfer1::Dims32 Class Reference	89
9.5.1	Member Data Documentation	90
9.5.1.1	d	90
9.5.1.2	MAX_DIMS	90
9.5.1.3	nbDims	90
9.6	nvinfer1::Dims4 Class Reference	90
9.6.1	Detailed Description	91
9.6.2	Constructor & Destructor Documentation	91
9.6.2.1	Dims4() [1/2]	91
9.6.2.2	Dims4() [2/2]	91
9.7	nvinfer1::DimsExprs Class Reference	92
9.7.1	Detailed Description	92
9.7.2	Member Data Documentation	92
9.7.2.1	d	92
9.7.2.2	nbDims	92
9.8	nvinfer1::DimsHW Class Reference	93
9.8.1	Detailed Description	93
9.8.2	Constructor & Destructor Documentation	93
9.8.2.1	DimsHW() [1/2]	94
9.8.2.2	DimsHW() [2/2]	94
9.8.3	Member Function Documentation	94
9.8.3.1	h() [1/2]	94
9.8.3.2	h() [2/2]	94
9.8.3.3	w() [1/2]	95
9.8.3.4	w() [2/2]	95
9.9	nvinfer1::DynamicPluginTensorDesc Class Reference	95
9.9.1	Detailed Description	95
9.9.2	Member Data Documentation	96
9.9.2.1	desc	96
9.9.2.2	max	96
9.9.2.3	min	96

9.10	nvinfer1::impl::EnumMaxImpl< T > Struct Template Reference	96
9.10.1	Detailed Description	96
9.11	nvinfer1::impl::EnumMaxImpl< ActivationType > Struct Reference	97
9.11.1	Detailed Description	97
9.11.2	Member Data Documentation	97
9.11.2.1	kVALUE	97
9.12	nvinfer1::impl::EnumMaxImpl< AllocatorFlag > Struct Reference	97
9.12.1	Detailed Description	98
9.12.2	Member Data Documentation	98
9.12.2.1	kVALUE	98
9.13	nvinfer1::impl::EnumMaxImpl< DataType > Struct Reference	98
9.13.1	Detailed Description	98
9.13.2	Member Data Documentation	98
9.13.2.1	kVALUE	99
9.14	nvinfer1::impl::EnumMaxImpl< ElementWiseOperation > Struct Reference	99
9.14.1	Detailed Description	99
9.14.2	Member Data Documentation	99
9.14.2.1	kVALUE	99
9.15	nvinfer1::impl::EnumMaxImpl< EngineCapability > Struct Reference	100
9.15.1	Detailed Description	100
9.15.2	Member Data Documentation	100
9.15.2.1	kVALUE	100
9.16	nvinfer1::impl::EnumMaxImpl< ErrorCode > Struct Reference	100
9.16.1	Detailed Description	101
9.16.2	Member Data Documentation	101
9.16.2.1	kVALUE	101
9.17	nvinfer1::impl::EnumMaxImpl< ILogger::Severity > Struct Reference	101
9.17.1	Detailed Description	101
9.17.2	Member Data Documentation	102
9.17.2.1	kVALUE	102
9.18	nvinfer1::impl::EnumMaxImpl< PaddingMode > Struct Reference	102
9.18.1	Detailed Description	102
9.18.2	Member Data Documentation	102
9.18.2.1	kVALUE	102
9.19	nvinfer1::impl::EnumMaxImpl< PoolingType > Struct Reference	103
9.19.1	Detailed Description	103
9.19.2	Member Data Documentation	103
9.19.2.1	kVALUE	103
9.20	nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation > Struct Reference	103
9.20.1	Detailed Description	104
9.20.2	Member Data Documentation	104
9.20.2.1	kVALUE	104
9.21	nvinfer1::impl::EnumMaxImpl< ResizeMode > Struct Reference	104
9.21.1	Detailed Description	104
9.21.2	Member Data Documentation	104
9.21.2.1	kVALUE	105
9.22	nvinfer1::impl::EnumMaxImpl< ResizeRoundMode > Struct Reference	105
9.22.1	Detailed Description	105
9.22.2	Member Data Documentation	105
9.22.2.1	kVALUE	105
9.23	nvinfer1::impl::EnumMaxImpl< ResizeSelector > Struct Reference	105
9.23.1	Detailed Description	106
9.23.2	Member Data Documentation	106
9.23.2.1	kVALUE	106

9.24	nvinfer1::impl::EnumMaxImpl< TensorFormat > Struct Reference	106
9.24.1	Detailed Description	106
9.24.2	Member Data Documentation	107
9.24.2.1	kVALUE	107
9.25	nvinfer1::impl::EnumMaxImpl< TensorLocation > Struct Reference	107
9.25.1	Detailed Description	107
9.25.2	Member Data Documentation	107
9.25.2.1	kVALUE	107
9.26	nvuffparser::FieldCollection Struct Reference	108
9.26.1	Member Data Documentation	108
9.26.1.1	fields	108
9.26.1.2	nbFields	108
9.27	nvuffparser::FieldMap Class Reference	108
9.27.1	Detailed Description	109
9.27.2	Constructor & Destructor Documentation	109
9.27.2.1	FieldMap()	109
9.27.3	Member Data Documentation	109
9.27.3.1	data	109
9.27.3.2	length	109
9.27.3.3	name	109
9.27.3.4	type	110
9.28	nvinfer1::safe::FloatingPointErrorInformation Struct Reference	110
9.28.1	Detailed Description	110
9.28.2	Member Data Documentation	110
9.28.2.1	nbInfErrors	110
9.28.2.2	nbNanErrors	111
9.29	nvinfer1::plugin::GridAnchorParameters Struct Reference	111
9.29.1	Detailed Description	111
9.29.2	Member Data Documentation	112
9.29.2.1	aspectRatios	112
9.29.2.2	H	112
9.29.2.3	maxSize	112
9.29.2.4	minSize	112
9.29.2.5	numAspectRatios	112
9.29.2.6	variance	112
9.29.2.7	W	113
9.30	nvinfer1::IActivationLayer Class Reference	113
9.30.1	Detailed Description	114
9.30.2	Constructor & Destructor Documentation	114
9.30.2.1	~IActivationLayer()	114
9.30.3	Member Function Documentation	114
9.30.3.1	getActivationType()	114
9.30.3.2	getAlpha()	115
9.30.3.3	getBeta()	115
9.30.3.4	setActivationType()	115
9.30.3.5	setAlpha()	116
9.30.3.6	setBeta()	116
9.30.4	Member Data Documentation	116
9.30.4.1	mImpl	116
9.31	nvinfer1::IAlgorithm Class Reference	117
9.31.1	Detailed Description	117
9.31.2	Constructor & Destructor Documentation	118
9.31.2.1	~IAlgorithm()	118
9.31.3	Member Function Documentation	118

9.31.3.1	getAlgorithmIOInfo()	118
9.31.3.2	getAlgorithmIOInfoByIndex()	118
9.31.3.3	getAlgorithmVariant()	119
9.31.3.4	getTimingMSec()	119
9.31.3.5	getWorkspaceSize()	119
9.31.4	Member Data Documentation	119
9.31.4.1	mImpl	119
9.32	nvinfer1::IAAlgorithmContext Class Reference	120
9.32.1	Detailed Description	120
9.32.2	Constructor & Destructor Documentation	121
9.32.2.1	~IAAlgorithmContext()	121
9.32.3	Member Function Documentation	121
9.32.3.1	getDimensions()	121
9.32.3.2	getName()	121
9.32.3.3	getNbInputs()	121
9.32.3.4	getNbOutputs()	122
9.32.4	Member Data Documentation	122
9.32.4.1	mImpl	122
9.33	nvinfer1::IAAlgorithmIOInfo Class Reference	122
9.33.1	Detailed Description	123
9.33.2	Constructor & Destructor Documentation	123
9.33.2.1	~IAAlgorithmIOInfo()	123
9.33.3	Member Function Documentation	123
9.33.3.1	getDataType()	123
9.33.3.2	getStrides()	124
9.33.3.3	getTensorFormat()	124
9.33.4	Member Data Documentation	124
9.33.4.1	mImpl	124
9.34	nvinfer1::IAAlgorithmSelector Class Reference	124
9.34.1	Detailed Description	125
9.34.2	Constructor & Destructor Documentation	125
9.34.2.1	~IAAlgorithmSelector()	125
9.34.3	Member Function Documentation	125
9.34.3.1	reportAlgorithms()	125
9.34.3.2	selectAlgorithms()	126
9.35	nvinfer1::IAAlgorithmVariant Class Reference	126
9.35.1	Detailed Description	127
9.35.2	Constructor & Destructor Documentation	127
9.35.2.1	~IAAlgorithmVariant()	127
9.35.3	Member Function Documentation	127
9.35.3.1	getImplementation()	128
9.35.3.2	getTactic()	128
9.35.4	Member Data Documentation	128
9.35.4.1	mImpl	128
9.36	nvinfer1::IAAssertionLayer Class Reference	128
9.36.1	Detailed Description	129
9.36.2	Constructor & Destructor Documentation	129
9.36.2.1	~IAAssertionLayer()	129
9.36.3	Member Function Documentation	129
9.36.3.1	getMessage()	130
9.36.3.2	setMessage()	130
9.36.4	Member Data Documentation	130
9.36.4.1	mImpl	130
9.37	nvcaffeparser1::IBinaryProtoBlob Class Reference	130

9.37.1	Detailed Description	131
9.37.2	Constructor & Destructor Documentation	131
9.37.2.1	~IBinaryProtoBlob()	131
9.37.3	Member Function Documentation	131
9.37.3.1	destroy()	131
9.37.3.2	getData()	132
9.37.3.3	getDataType()	132
9.37.3.4	getDimensions()	132
9.38	nvcaffeparser1::IBlobNameToTensor Class Reference	132
9.38.1	Detailed Description	132
9.38.2	Constructor & Destructor Documentation	133
9.38.2.1	~IBlobNameToTensor()	133
9.38.3	Member Function Documentation	133
9.38.3.1	find()	133
9.39	nvinfer1::IBuilder Class Reference	133
9.39.1	Detailed Description	135
9.39.2	Constructor & Destructor Documentation	135
9.39.2.1	~IBuilder()	135
9.39.3	Member Function Documentation	135
9.39.3.1	buildEngineWithConfig()	136
9.39.3.2	buildSerializedNetwork()	136
9.39.3.3	createBuilderConfig()	137
9.39.3.4	createNetworkV2()	137
9.39.3.5	createOptimizationProfile()	137
9.39.3.6	destroy()	138
9.39.3.7	getErrorRecorder()	138
9.39.3.8	getLogger()	138
9.39.3.9	getMaxBatchSize()	139
9.39.3.10	getMaxDLABatchSize()	139
9.39.3.11	getMaxThreads()	139
9.39.3.12	getNbDLACores()	140
9.39.3.13	isNetworkSupported()	140
9.39.3.14	platformHasFastFp16()	140
9.39.3.15	platformHasFastInt8()	141
9.39.3.16	platformHasTf32()	141
9.39.3.17	reset()	141
9.39.3.18	setErrorRecorder()	141
9.39.3.19	setGpuAllocator()	142
9.39.3.20	setMaxBatchSize()	142
9.39.3.21	setMaxThreads()	142
9.39.4	Member Data Documentation	143
9.39.4.1	mImpl	143
9.40	nvinfer1::IBuilderConfig Class Reference	143
9.40.1	Detailed Description	146
9.40.2	Constructor & Destructor Documentation	146
9.40.2.1	~IBuilderConfig()	146
9.40.3	Member Function Documentation	146
9.40.3.1	addOptimizationProfile()	146
9.40.3.2	canRunOnDLA()	147
9.40.3.3	clearFlag()	147
9.40.3.4	clearQuantizationFlag()	147
9.40.3.5	createTimingCache()	148
9.40.3.6	destroy()	148
9.40.3.7	getAlgorithmSelector()	149

9.40.3.8	getAvgTimingIterations()	149
9.40.3.9	getCalibrationProfile()	149
9.40.3.10	getDefaultDeviceType()	149
9.40.3.11	getDeviceType()	150
9.40.3.12	getDLACore()	150
9.40.3.13	getEngineCapability()	150
9.40.3.14	getFlag()	151
9.40.3.15	getFlags()	151
9.40.3.16	getInt8Calibrator()	151
9.40.3.17	getMaxWorkspaceSize()	152
9.40.3.18	getMemoryPoolLimit()	152
9.40.3.19	getMinTimingIterations()	153
9.40.3.20	getNbOptimizationProfiles()	153
9.40.3.21	getProfileStream()	153
9.40.3.22	getProfilingVerbosity()	154
9.40.3.23	getQuantizationFlag()	154
9.40.3.24	getQuantizationFlags()	154
9.40.3.25	getTacticSources()	155
9.40.3.26	getTimingCache()	155
9.40.3.27	isDeviceTypeSet()	155
9.40.3.28	reset()	156
9.40.3.29	resetDeviceType()	156
9.40.3.30	setAlgorithmSelector()	156
9.40.3.31	setAvgTimingIterations()	156
9.40.3.32	setCalibrationProfile()	157
9.40.3.33	setDefaultDeviceType()	157
9.40.3.34	setDeviceType()	157
9.40.3.35	setDLACore()	158
9.40.3.36	setEngineCapability()	158
9.40.3.37	setFlag()	159
9.40.3.38	setFlags()	159
9.40.3.39	setInt8Calibrator()	159
9.40.3.40	setMaxWorkspaceSize()	160
9.40.3.41	setMemoryPoolLimit()	160
9.40.3.42	setMinTimingIterations()	161
9.40.3.43	setProfileStream()	161
9.40.3.44	setProfilingVerbosity()	161
9.40.3.45	setQuantizationFlag()	162
9.40.3.46	setQuantizationFlags()	162
9.40.3.47	setTacticSources()	163
9.40.3.48	setTimingCache()	163
9.40.4	Member Data Documentation	164
9.40.4.1	mImpl	164
9.41	nvcaffeparser1::ICaffeParser Class Reference	164
9.41.1	Detailed Description	165
9.41.2	Constructor & Destructor Documentation	165
9.41.2.1	~ICaffeParser()	165
9.41.3	Member Function Documentation	165
9.41.3.1	destroy()	165
9.41.3.2	getErrorRecorder()	166
9.41.3.3	parse()	166
9.41.3.4	parseBinaryProto()	167
9.41.3.5	parseBuffers()	167
9.41.3.6	setErrorRecorder()	168

9.41.3.7	setPluginFactoryV2()	168
9.41.3.8	setPluginNamespace()	169
9.41.3.9	setProtobufBufferSize()	169
9.42	nvinfer1::IConcatenationLayer Class Reference	169
9.42.1	Detailed Description	170
9.42.2	Constructor & Destructor Documentation	170
9.42.2.1	~IConcatenationLayer()	170
9.42.3	Member Function Documentation	170
9.42.3.1	getAxis()	171
9.42.3.2	setAxis()	171
9.42.4	Member Data Documentation	171
9.42.4.1	mImpl	171
9.43	nvinfer1::IConditionLayer Class Reference	172
9.43.1	Detailed Description	172
9.43.2	Constructor & Destructor Documentation	172
9.43.2.1	~IConditionLayer()	172
9.43.3	Member Data Documentation	172
9.43.3.1	mImpl	173
9.44	nvinfer1::consistency::IConsistencyChecker Class Reference	173
9.44.1	Detailed Description	173
9.44.2	Constructor & Destructor Documentation	173
9.44.2.1	~IConsistencyChecker()	174
9.44.2.2	IConsistencyChecker() [1/3]	174
9.44.2.3	IConsistencyChecker() [2/3]	174
9.44.2.4	IConsistencyChecker() [3/3]	174
9.44.3	Member Function Documentation	174
9.44.3.1	operator=() [1/2]	174
9.44.3.2	operator=() [2/2]	174
9.44.3.3	validate()	175
9.44.4	Member Data Documentation	175
9.44.4.1	mImpl	175
9.45	nvinfer1::IConstantLayer Class Reference	175
9.45.1	Detailed Description	176
9.45.2	Constructor & Destructor Documentation	176
9.45.2.1	~IConstantLayer()	176
9.45.3	Member Function Documentation	176
9.45.3.1	getDimensions()	176
9.45.3.2	getWeights()	177
9.45.3.3	setDimensions()	177
9.45.3.4	setWeights()	177
9.45.4	Member Data Documentation	177
9.45.4.1	mImpl	178
9.46	nvinfer1::IConvolutionLayer Class Reference	178
9.46.1	Detailed Description	180
9.46.2	Constructor & Destructor Documentation	180
9.46.2.1	~IConvolutionLayer()	180
9.46.3	Member Function Documentation	180
9.46.3.1	getBiasWeights()	180
9.46.3.2	getDilation()	181
9.46.3.3	getDilationNd()	181
9.46.3.4	getKernelSize()	181
9.46.3.5	getKernelSizeNd()	182
9.46.3.6	getKernelWeights()	182
9.46.3.7	getNbGroups()	182

9.46.3.8	getNbOutputMaps()	182
9.46.3.9	getPadding()	183
9.46.3.10	getPaddingMode()	183
9.46.3.11	getPaddingNd()	183
9.46.3.12	getPostPadding()	184
9.46.3.13	getPrePadding()	184
9.46.3.14	getStride()	184
9.46.3.15	getStrideNd()	184
9.46.3.16	setBiasWeights()	185
9.46.3.17	setDilation()	185
9.46.3.18	setDilationNd()	185
9.46.3.19	setInput()	185
9.46.3.20	setKernelSize()	186
9.46.3.21	setKernelSizeNd()	186
9.46.3.22	setKernelWeights()	187
9.46.3.23	setNbGroups()	187
9.46.3.24	setNbOutputMaps()	187
9.46.3.25	setPadding()	188
9.46.3.26	setPaddingMode()	188
9.46.3.27	setPaddingNd()	188
9.46.3.28	setPostPadding()	189
9.46.3.29	setPrePadding()	189
9.46.3.30	setStride()	189
9.46.3.31	setStrideNd()	190
9.46.4	Member Data Documentation	190
9.46.4.1	mImpl	190
9.47	nvinfer1::ICudaEngine Class Reference	190
9.47.1	Detailed Description	192
9.47.2	Constructor & Destructor Documentation	192
9.47.2.1	~ICudaEngine()	193
9.47.3	Member Function Documentation	193
9.47.3.1	bindingIsInput()	193
9.47.3.2	createEngineInspector()	193
9.47.3.3	createExecutionContext()	194
9.47.3.4	createExecutionContextWithoutDeviceMemory()	194
9.47.3.5	destroy()	194
9.47.3.6	getBindingBytesPerComponent()	194
9.47.3.7	getBindingComponentsPerElement()	195
9.47.3.8	getBindingDataType()	195
9.47.3.9	getBindingDimensions()	196
9.47.3.10	getBindingFormat()	196
9.47.3.11	getBindingFormatDesc()	197
9.47.3.12	getBindingIndex()	197
9.47.3.13	getBindingName()	198
9.47.3.14	getBindingVectorizedDim()	198
9.47.3.15	getDeviceMemorySize()	198
9.47.3.16	getEngineCapability()	199
9.47.3.17	getErrorRecorder()	199
9.47.3.18	getLocation()	199
9.47.3.19	getMaxBatchSize()	200
9.47.3.20	getName()	200
9.47.3.21	getNbBindings()	201
9.47.3.22	getNbLayers()	201
9.47.3.23	getNbOptimizationProfiles()	201

9.47.3.24	getProfileDimensions()	201
9.47.3.25	getProfileShapeValues()	202
9.47.3.26	getProfilingVerbosity()	203
9.47.3.27	getTacticSources()	203
9.47.3.28	hasImplicitBatchDimension()	203
9.47.3.29	isExecutionBinding()	204
9.47.3.30	isRefittable()	204
9.47.3.31	isShapeBinding()	204
9.47.3.32	serialize()	205
9.47.3.33	setErrorRecorder()	205
9.47.4	Member Data Documentation	206
9.47.4.1	mImpl	206
9.48	nvinfer1::safe::ICudaEngine Class Reference	206
9.48.1	Detailed Description	207
9.48.2	Constructor & Destructor Documentation	207
9.48.2.1	ICudaEngine() [1/3]	207
9.48.2.2	~ICudaEngine()	207
9.48.2.3	ICudaEngine() [2/3]	208
9.48.2.4	ICudaEngine() [3/3]	208
9.48.3	Member Function Documentation	208
9.48.3.1	bindingIsInput()	208
9.48.3.2	createExecutionContext()	209
9.48.3.3	createExecutionContextWithoutDeviceMemory()	209
9.48.3.4	getBindingBytesPerComponent()	209
9.48.3.5	getBindingComponentsPerElement()	210
9.48.3.6	getBindingDataType()	210
9.48.3.7	getBindingDimensions()	211
9.48.3.8	getBindingFormat()	212
9.48.3.9	getBindingIndex()	212
9.48.3.10	getBindingName()	213
9.48.3.11	getBindingVectorizedDim()	213
9.48.3.12	getDeviceMemorySize()	214
9.48.3.13	getErrorRecorder()	214
9.48.3.14	getName()	215
9.48.3.15	getNbBindings()	215
9.48.3.16	operator=() [1/2]	216
9.48.3.17	operator=() [2/2]	216
9.48.3.18	setErrorRecorder()	216
9.49	nvinfer1::IDeconvolutionLayer Class Reference	217
9.49.1	Detailed Description	218
9.49.2	Constructor & Destructor Documentation	219
9.49.2.1	~IDeconvolutionLayer()	219
9.49.3	Member Function Documentation	219
9.49.3.1	getBiasWeights()	219
9.49.3.2	getDilationNd()	219
9.49.3.3	getKernelSize()	220
9.49.3.4	getKernelSizeNd()	220
9.49.3.5	getKernelWeights()	220
9.49.3.6	getNbGroups()	220
9.49.3.7	getNbOutputMaps()	221
9.49.3.8	getPadding()	221
9.49.3.9	getPaddingMode()	221
9.49.3.10	getPaddingNd()	222
9.49.3.11	getPostPadding()	222

9.49.3.12	getPrePadding()	222
9.49.3.13	getStride()	222
9.49.3.14	getStrideNd()	223
9.49.3.15	setBiasWeights()	223
9.49.3.16	setDilationNd()	223
9.49.3.17	setInput()	223
9.49.3.18	setKernelSize()	224
9.49.3.19	setKernelSizeNd()	225
9.49.3.20	setKernelWeights()	225
9.49.3.21	setNbGroups()	225
9.49.3.22	setNbOutputMaps()	226
9.49.3.23	setPadding()	226
9.49.3.24	setPaddingMode()	226
9.49.3.25	setPaddingNd()	227
9.49.3.26	setPostPadding()	227
9.49.3.27	setPrePadding()	227
9.49.3.28	setStride()	228
9.49.3.29	setStrideNd()	228
9.49.4	Member Data Documentation	228
9.49.4.1	mImpl	228
9.50	nvinfer1::IDequantizeLayer Class Reference	229
9.50.1	Detailed Description	229
9.50.2	Constructor & Destructor Documentation	230
9.50.2.1	~IDequantizeLayer()	230
9.50.3	Member Function Documentation	230
9.50.3.1	getAxis()	231
9.50.3.2	setAxis()	231
9.50.4	Member Data Documentation	231
9.50.4.1	mImpl	231
9.51	nvinfer1::IDimensionExpr Class Reference	231
9.51.1	Detailed Description	232
9.51.2	Constructor & Destructor Documentation	232
9.51.2.1	~IDimensionExpr()	232
9.51.3	Member Function Documentation	232
9.51.3.1	getConstantValue()	233
9.51.3.2	isConstant()	233
9.51.4	Member Data Documentation	233
9.51.4.1	mImpl	233
9.52	nvinfer1::IEinsumLayer Class Reference	233
9.52.1	Detailed Description	234
9.52.2	Constructor & Destructor Documentation	235
9.52.2.1	~IEinsumLayer()	235
9.52.3	Member Function Documentation	235
9.52.3.1	getEquation()	235
9.52.3.2	setEquation()	235
9.52.4	Member Data Documentation	235
9.52.4.1	mImpl	236
9.53	nvinfer1::IElementWiseLayer Class Reference	236
9.53.1	Detailed Description	236
9.53.2	Constructor & Destructor Documentation	237
9.53.2.1	~IElementWiseLayer()	237
9.53.3	Member Function Documentation	237
9.53.3.1	getOperation()	237
9.53.3.2	setOperation()	237

9.53.4	Member Data Documentation	238
9.53.4.1	mImpl	238
9.54	nvinfer1::IEngineInspector Class Reference	238
9.54.1	Detailed Description	239
9.54.2	Constructor & Destructor Documentation	239
9.54.2.1	~IEngineInspector()	239
9.54.3	Member Function Documentation	239
9.54.3.1	getEngineInformation()	239
9.54.3.2	getErrorRecorder()	240
9.54.3.3	getExecutionContext()	241
9.54.3.4	getLayerInformation()	241
9.54.3.5	setErrorRecorder()	242
9.54.3.6	setExecutionContext()	242
9.54.4	Member Data Documentation	242
9.54.4.1	mImpl	243
9.55	nvinfer1::IErrorRecorder Class Reference	243
9.55.1	Detailed Description	244
9.55.2	Member Typedef Documentation	244
9.55.2.1	ErrorDesc	244
9.55.2.2	RefCount	244
9.55.3	Constructor & Destructor Documentation	244
9.55.3.1	IErrorRecorder()	244
9.55.3.2	~IErrorRecorder()	245
9.55.4	Member Function Documentation	245
9.55.4.1	clear()	245
9.55.4.2	decRefCount()	245
9.55.4.3	getErrorCode()	246
9.55.4.4	getErrorDesc()	246
9.55.4.5	getNbErrors()	247
9.55.4.6	hasOverflowed()	248
9.55.4.7	incRefCount()	248
9.55.4.8	reportError()	248
9.55.5	Member Data Documentation	249
9.55.5.1	kMAX_DESC_LENGTH	249
9.56	nvinfer1::IExecutionContext Class Reference	249
9.56.1	Detailed Description	251
9.56.2	Constructor & Destructor Documentation	251
9.56.2.1	~IExecutionContext()	251
9.56.3	Member Function Documentation	251
9.56.3.1	allInputDimensionsSpecified()	252
9.56.3.2	allInputShapesSpecified()	252
9.56.3.3	destroy()	252
9.56.3.4	enqueue()	253
9.56.3.5	enqueueV2()	254
9.56.3.6	execute()	254
9.56.3.7	executeV2()	255
9.56.3.8	getBindingDimensions()	256
9.56.3.9	getDebugSync()	256
9.56.3.10	getEngine()	257
9.56.3.11	getEnqueueEmitsProfile()	257
9.56.3.12	getErrorRecorder()	257
9.56.3.13	getName()	258
9.56.3.14	getOptimizationProfile()	258
9.56.3.15	getProfiler()	258

9.56.3.16	getShapeBinding()	258
9.56.3.17	getStrides()	259
9.56.3.18	reportToProfiler()	259
9.56.3.19	setBindingDimensions()	260
9.56.3.20	setDebugSync()	261
9.56.3.21	setDeviceMemory()	261
9.56.3.22	setEnqueueEmitsProfile()	262
9.56.3.23	setErrorRecorder()	262
9.56.3.24	setInputShapeBinding()	262
9.56.3.25	setName()	263
9.56.3.26	setOptimizationProfile()	263
9.56.3.27	setOptimizationProfileAsync()	264
9.56.3.28	setProfiler()	265
9.56.4	Member Data Documentation	266
9.56.4.1	mImpl	266
9.57	nvinfer1::safe::IExecutionContext Class Reference	266
9.57.1	Detailed Description	267
9.57.2	Constructor & Destructor Documentation	267
9.57.2.1	IExecutionContext() [1/3]	267
9.57.2.2	~IExecutionContext()	267
9.57.2.3	IExecutionContext() [2/3]	267
9.57.2.4	IExecutionContext() [3/3]	267
9.57.3	Member Function Documentation	267
9.57.3.1	enqueueV2()	268
9.57.3.2	getEngine()	268
9.57.3.3	getErrorBuffer()	269
9.57.3.4	getErrorRecorder()	269
9.57.3.5	getName()	270
9.57.3.6	getStrides()	270
9.57.3.7	operator=() [1/2]	270
9.57.3.8	operator=() [2/2]	271
9.57.3.9	setDeviceMemory()	271
9.57.3.10	setErrorBuffer()	271
9.57.3.11	setErrorRecorder()	272
9.57.3.12	setName()	273
9.58	nvinfer1::IExprBuilder Class Reference	273
9.58.1	Detailed Description	274
9.58.2	Constructor & Destructor Documentation	274
9.58.2.1	~IExprBuilder()	274
9.58.3	Member Function Documentation	274
9.58.3.1	constant()	274
9.58.3.2	operation()	275
9.58.4	Member Data Documentation	275
9.58.4.1	mImpl	275
9.59	nvinfer1::IFillLayer Class Reference	275
9.59.1	Detailed Description	276
9.59.2	Constructor & Destructor Documentation	277
9.59.2.1	~IFillLayer()	277
9.59.3	Member Function Documentation	277
9.59.3.1	getAlpha()	277
9.59.3.2	getBeta()	278
9.59.3.3	getDimensions()	278
9.59.3.4	getOperation()	278
9.59.3.5	setAlpha()	278

9.59.3.6	setBeta()	279
9.59.3.7	setDimensions()	279
9.59.3.8	setInput()	280
9.59.3.9	setOperation()	281
9.59.4	Member Data Documentation	281
9.59.4.1	mImpl	281
9.60	nvinfer1::IFullyConnectedLayer Class Reference	281
9.60.1	Detailed Description	282
9.60.2	Constructor & Destructor Documentation	283
9.60.2.1	~IFullyConnectedLayer()	283
9.60.3	Member Function Documentation	283
9.60.3.1	getBiasWeights()	283
9.60.3.2	getKernelWeights()	283
9.60.3.3	getNbOutputChannels()	284
9.60.3.4	setBiasWeights()	284
9.60.3.5	setInput()	284
9.60.3.6	setKernelWeights()	285
9.60.3.7	setNbOutputChannels()	285
9.60.4	Member Data Documentation	285
9.60.4.1	mImpl	286
9.61	nvinfer1::IGatherLayer Class Reference	286
9.61.1	Detailed Description	287
9.61.2	Constructor & Destructor Documentation	288
9.61.2.1	~IGatherLayer()	288
9.61.3	Member Function Documentation	289
9.61.3.1	getGatherAxis()	289
9.61.3.2	getMode()	289
9.61.3.3	getNbElementWiseDims()	289
9.61.3.4	setGatherAxis()	289
9.61.3.5	setMode()	290
9.61.3.6	setNbElementWiseDims()	290
9.61.4	Member Data Documentation	291
9.61.4.1	mImpl	291
9.62	nvinfer1::IGpuAllocator Class Reference	291
9.62.1	Detailed Description	291
9.62.2	Constructor & Destructor Documentation	291
9.62.2.1	~IGpuAllocator()	291
9.62.2.2	IGpuAllocator()	292
9.62.3	Member Function Documentation	292
9.62.3.1	allocate()	292
9.62.3.2	deallocate()	292
9.62.3.3	free()	293
9.62.3.4	reallocate()	294
9.63	nvinfer1::IHostMemory Class Reference	295
9.63.1	Detailed Description	296
9.63.2	Constructor & Destructor Documentation	296
9.63.2.1	~IHostMemory()	296
9.63.3	Member Function Documentation	296
9.63.3.1	data()	296
9.63.3.2	destroy()	296
9.63.3.3	size()	297
9.63.3.4	type()	297
9.63.4	Member Data Documentation	297
9.63.4.1	mImpl	297

9.64	nvinfer1::IIdentityLayer Class Reference	297
9.64.1	Detailed Description	298
9.64.2	Constructor & Destructor Documentation	298
9.64.2.1	~IIdentityLayer()	298
9.64.3	Member Data Documentation	298
9.64.3.1	mImpl	299
9.65	nvinfer1::IIfConditional Class Reference	299
9.65.1	Detailed Description	300
9.65.2	Constructor & Destructor Documentation	300
9.65.2.1	~IIfConditional()	300
9.65.3	Member Function Documentation	300
9.65.3.1	addInput()	300
9.65.3.2	addOutput()	301
9.65.3.3	getName()	301
9.65.3.4	setCondition()	301
9.65.3.5	setName()	302
9.65.4	Member Data Documentation	302
9.65.4.1	mImpl	302
9.66	nvinfer1::IIfConditionalBoundaryLayer Class Reference	302
9.66.1	Detailed Description	303
9.66.2	Constructor & Destructor Documentation	303
9.66.2.1	~IIfConditionalBoundaryLayer()	303
9.66.3	Member Function Documentation	303
9.66.3.1	getConditional()	303
9.66.4	Member Data Documentation	303
9.66.4.1	mBoundary	304
9.67	nvinfer1::IIfConditionalInputLayer Class Reference	304
9.67.1	Detailed Description	304
9.67.2	Constructor & Destructor Documentation	304
9.67.2.1	~IIfConditionalInputLayer()	305
9.67.3	Member Data Documentation	305
9.67.3.1	mImpl	305
9.68	nvinfer1::IIfConditionalOutputLayer Class Reference	305
9.68.1	Detailed Description	306
9.68.2	Constructor & Destructor Documentation	306
9.68.2.1	~IIfConditionalOutputLayer()	306
9.68.3	Member Data Documentation	306
9.68.3.1	mImpl	306
9.69	nvinfer1::IInt8Calibrator Class Reference	306
9.69.1	Detailed Description	307
9.69.2	Constructor & Destructor Documentation	307
9.69.2.1	~IInt8Calibrator()	307
9.69.3	Member Function Documentation	307
9.69.3.1	getAlgorithm()	307
9.69.3.2	getBatch()	308
9.69.3.3	getBatchSize()	308
9.69.3.4	readCalibrationCache()	308
9.69.3.5	writeCalibrationCache()	309
9.70	nvinfer1::IInt8EntropyCalibrator Class Reference	309
9.70.1	Detailed Description	310
9.70.2	Constructor & Destructor Documentation	310
9.70.2.1	~IInt8EntropyCalibrator()	310
9.70.3	Member Function Documentation	310
9.70.3.1	getAlgorithm()	310

9.71	<code>nvinfer1::IInt8EntropyCalibrator2</code> Class Reference	310
9.71.1	Detailed Description	311
9.71.2	Constructor & Destructor Documentation	311
9.71.2.1	<code>~IInt8EntropyCalibrator2()</code>	311
9.71.3	Member Function Documentation	311
9.71.3.1	<code>getAlgorithm()</code>	311
9.72	<code>nvinfer1::IInt8LegacyCalibrator</code> Class Reference	311
9.72.1	Detailed Description	312
9.72.2	Constructor & Destructor Documentation	312
9.72.2.1	<code>~IInt8LegacyCalibrator()</code>	312
9.72.3	Member Function Documentation	312
9.72.3.1	<code>getAlgorithm()</code>	312
9.72.3.2	<code>getQuantile()</code>	313
9.72.3.3	<code>getRegressionCutoff()</code>	313
9.72.3.4	<code>readHistogramCache()</code>	313
9.72.3.5	<code>writeHistogramCache()</code>	313
9.73	<code>nvinfer1::IInt8MinMaxCalibrator</code> Class Reference	314
9.73.1	Detailed Description	314
9.73.2	Constructor & Destructor Documentation	314
9.73.2.1	<code>~IInt8MinMaxCalibrator()</code>	314
9.73.3	Member Function Documentation	315
9.73.3.1	<code>getAlgorithm()</code>	315
9.74	<code>nvinfer1::IIteratorLayer</code> Class Reference	315
9.74.1	Constructor & Destructor Documentation	316
9.74.1.1	<code>~IIteratorLayer()</code>	316
9.74.2	Member Function Documentation	316
9.74.2.1	<code>getAxis()</code>	316
9.74.2.2	<code>getReverse()</code>	316
9.74.2.3	<code>setAxis()</code>	316
9.74.2.4	<code>setReverse()</code>	317
9.74.3	Member Data Documentation	317
9.74.3.1	<code>mImpl</code>	317
9.75	<code>nvinfer1::ILayer</code> Class Reference	317
9.75.1	Detailed Description	319
9.75.2	Constructor & Destructor Documentation	319
9.75.2.1	<code>~ILayer()</code>	319
9.75.3	Member Function Documentation	319
9.75.3.1	<code>getInput()</code>	319
9.75.3.2	<code>getName()</code>	320
9.75.3.3	<code>getNbInputs()</code>	320
9.75.3.4	<code>getNbOutputs()</code>	320
9.75.3.5	<code>getOutput()</code>	320
9.75.3.6	<code>getOutputType()</code>	320
9.75.3.7	<code>getPrecision()</code>	321
9.75.3.8	<code>getType()</code>	321
9.75.3.9	<code>outputTypeIsSet()</code>	321
9.75.3.10	<code>precisionIsSet()</code>	322
9.75.3.11	<code>resetOutputType()</code>	322
9.75.3.12	<code>resetPrecision()</code>	323
9.75.3.13	<code>setInput()</code>	323
9.75.3.14	<code>setName()</code>	323
9.75.3.15	<code>setOutputType()</code>	324
9.75.3.16	<code>setPrecision()</code>	324
9.75.4	Member Data Documentation	325

9.75.4.1	mLayer	325
9.76	nvinfer1::ILogger Class Reference	325
9.76.1	Detailed Description	326
9.76.2	Member Enumeration Documentation	326
9.76.2.1	Severity	326
9.76.3	Constructor & Destructor Documentation	327
9.76.3.1	ILogger()	327
9.76.3.2	~ILogger()	327
9.76.4	Member Function Documentation	327
9.76.4.1	log()	327
9.77	nvinfer1::ILoop Class Reference	328
9.77.1	Detailed Description	328
9.77.2	Constructor & Destructor Documentation	328
9.77.2.1	~ILoop()	329
9.77.3	Member Function Documentation	329
9.77.3.1	addIterator()	329
9.77.3.2	addLoopOutput()	329
9.77.3.3	addRecurrence()	329
9.77.3.4	addTripLimit()	330
9.77.3.5	getName()	330
9.77.3.6	setName()	330
9.77.4	Member Data Documentation	330
9.77.4.1	mImpl	331
9.78	nvinfer1::ILoopBoundaryLayer Class Reference	331
9.78.1	Constructor & Destructor Documentation	331
9.78.1.1	~ILoopBoundaryLayer()	331
9.78.2	Member Function Documentation	332
9.78.2.1	getLoop()	332
9.78.3	Member Data Documentation	332
9.78.3.1	mBoundary	332
9.79	nvinfer1::ILoopOutputLayer Class Reference	332
9.79.1	Detailed Description	333
9.79.2	Constructor & Destructor Documentation	333
9.79.2.1	~ILoopOutputLayer()	333
9.79.3	Member Function Documentation	334
9.79.3.1	getAxis()	334
9.79.3.2	getLoopOutput()	334
9.79.3.3	setAxis()	334
9.79.3.4	setInput()	334
9.79.4	Member Data Documentation	335
9.79.4.1	mImpl	335
9.80	nvinfer1::ILRNLayer Class Reference	335
9.80.1	Detailed Description	336
9.80.2	Constructor & Destructor Documentation	336
9.80.2.1	~ILRNLayer()	336
9.80.3	Member Function Documentation	337
9.80.3.1	getAlpha()	337
9.80.3.2	getBeta()	337
9.80.3.3	getK()	337
9.80.3.4	getWindowSize()	338
9.80.3.5	setAlpha()	338
9.80.3.6	setBeta()	338
9.80.3.7	setK()	339
9.80.3.8	setWindowSize()	339

9.80.4	Member Data Documentation	339
9.80.4.1	mImpl	339
9.81	nvinfer1::IMatrixMultiplyLayer Class Reference	340
9.81.1	Detailed Description	340
9.81.2	Constructor & Destructor Documentation	341
9.81.2.1	~IMatrixMultiplyLayer()	341
9.81.3	Member Function Documentation	341
9.81.3.1	getOperation()	341
9.81.3.2	setOperation()	341
9.81.4	Member Data Documentation	342
9.81.4.1	mImpl	342
9.82	nvinfer1::INetworkDefinition Class Reference	342
9.82.1	Detailed Description	346
9.82.2	Constructor & Destructor Documentation	346
9.82.2.1	~INetworkDefinition()	346
9.82.3	Member Function Documentation	346
9.82.3.1	addActivation()	346
9.82.3.2	addAssertion()	347
9.82.3.3	addConcatenation()	347
9.82.3.4	addConstant()	348
9.82.3.5	addConvolution()	349
9.82.3.6	addConvolutionNd()	349
9.82.3.7	addDeconvolution()	350
9.82.3.8	addDeconvolutionNd()	351
9.82.3.9	addDequantize()	352
9.82.3.10	addEinsum()	352
9.82.3.11	addElementWise()	353
9.82.3.12	addFill()	353
9.82.3.13	addFullyConnected()	354
9.82.3.14	addGather()	355
9.82.3.15	addGatherV2()	355
9.82.3.16	addIdentity()	356
9.82.3.17	addIfConditional()	356
9.82.3.18	addInput()	357
9.82.3.19	addLoop()	358
9.82.3.20	addLRN()	358
9.82.3.21	addMatrixMultiply()	359
9.82.3.22	addPadding()	360
9.82.3.23	addPaddingNd()	360
9.82.3.24	addParametricReLU()	361
9.82.3.25	addPluginV2()	361
9.82.3.26	addPooling()	362
9.82.3.27	addPoolingNd()	363
9.82.3.28	addQuantize()	363
9.82.3.29	addRaggedSoftMax()	364
9.82.3.30	addReduce()	365
9.82.3.31	addResize()	365
9.82.3.32	addRNNv2()	366
9.82.3.33	addScale()	367
9.82.3.34	addScaleNd()	368
9.82.3.35	addScatter()	369
9.82.3.36	addSelect()	370
9.82.3.37	addShape()	371
9.82.3.38	addShuffle()	372

9.82.3.39	addSlice()	372
9.82.3.40	addSoftMax()	373
9.82.3.41	addTopK()	374
9.82.3.42	addUnary()	374
9.82.3.43	destroy()	375
9.82.3.44	getErrorRecorder()	376
9.82.3.45	getInput()	376
9.82.3.46	getLayer()	376
9.82.3.47	getName()	377
9.82.3.48	getNbInputs()	377
9.82.3.49	getNbLayers()	378
9.82.3.50	getNbOutputs()	378
9.82.3.51	getOutput()	378
9.82.3.52	hasExplicitPrecision()	379
9.82.3.53	hasImplicitBatchDimension()	379
9.82.3.54	markOutput()	380
9.82.3.55	markOutputForShapes()	380
9.82.3.56	removeTensor()	380
9.82.3.57	setErrorRecorder()	381
9.82.3.58	setName()	381
9.82.3.59	setWeightsName()	382
9.82.3.60	unmarkOutput()	382
9.82.3.61	unmarkOutputForShapes()	383
9.82.4	Member Data Documentation	383
9.82.4.1	mImpl	383
9.83	nvinfer1::INoCopy Class Reference	384
9.83.1	Detailed Description	384
9.83.2	Constructor & Destructor Documentation	385
9.83.2.1	INoCopy() [1/3]	385
9.83.2.2	~INoCopy()	385
9.83.2.3	INoCopy() [2/3]	385
9.83.2.4	INoCopy() [3/3]	385
9.83.3	Member Function Documentation	385
9.83.3.1	operator=() [1/2]	385
9.83.3.2	operator=() [2/2]	386
9.84	nvonnxparser::IONnxConfig Class Reference	386
9.84.1	Detailed Description	387
9.84.2	Member Typedef Documentation	387
9.84.2.1	Verbosity	387
9.84.3	Constructor & Destructor Documentation	387
9.84.3.1	~IONnxConfig()	387
9.84.4	Member Function Documentation	387
9.84.4.1	addVerbosity()	388
9.84.4.2	destroy()	388
9.84.4.3	getFullTextFileName()	388
9.84.4.4	getModelDtype()	389
9.84.4.5	getModelFileName()	389
9.84.4.6	getPrintLayerInfo()	389
9.84.4.7	getTextFileName()	390
9.84.4.8	getVerbosityLevel()	390
9.84.4.9	reduceVerbosity()	390
9.84.4.10	setFullTextFileName()	390
9.84.4.11	setModelDtype()	391
9.84.4.12	setModelFileName()	391

9.84.4.13	setPrintLayerInfo()	392
9.84.4.14	setTextFileName()	392
9.84.4.15	setVerbosityLevel()	392
9.85	nvinfer1::IOptimizationProfile Class Reference	393
9.85.1	Detailed Description	394
9.85.2	Constructor & Destructor Documentation	394
9.85.2.1	~IOptimizationProfile()	394
9.85.3	Member Function Documentation	394
9.85.3.1	getDimensions()	394
9.85.3.2	getExtraMemoryTarget()	395
9.85.3.3	getNbShapeValues()	395
9.85.3.4	getShapeValues()	395
9.85.3.5	isValid()	395
9.85.3.6	setDimensions()	396
9.85.3.7	setExtraMemoryTarget()	396
9.85.3.8	setShapeValues()	397
9.85.4	Member Data Documentation	398
9.85.4.1	mImpl	398
9.86	nvinfer1::IPaddingLayer Class Reference	398
9.86.1	Detailed Description	399
9.86.2	Constructor & Destructor Documentation	399
9.86.2.1	~IPaddingLayer()	399
9.86.3	Member Function Documentation	400
9.86.3.1	getPostPadding()	400
9.86.3.2	getPostPaddingNd()	400
9.86.3.3	getPrePadding()	400
9.86.3.4	getPrePaddingNd()	400
9.86.3.5	setPostPadding()	401
9.86.3.6	setPostPaddingNd()	401
9.86.3.7	setPrePadding()	402
9.86.3.8	setPrePaddingNd()	402
9.86.4	Member Data Documentation	402
9.86.4.1	mImpl	402
9.87	nvinfer1::IParametricReLULayer Class Reference	403
9.87.1	Detailed Description	403
9.87.2	Constructor & Destructor Documentation	403
9.87.2.1	~IParametricReLULayer()	403
9.87.3	Member Data Documentation	404
9.87.3.1	mImpl	404
9.88	nvonnxparser::IParser Class Reference	404
9.88.1	Detailed Description	405
9.88.2	Constructor & Destructor Documentation	405
9.88.2.1	~IParser()	405
9.88.3	Member Function Documentation	405
9.88.3.1	clearErrors()	405
9.88.3.2	destroy()	405
9.88.3.3	getError()	406
9.88.3.4	getNbErrors()	406
9.88.3.5	parse()	406
9.88.3.6	parseFromFile()	407
9.88.3.7	parseWithWeightDescriptors()	407
9.88.3.8	supportsModel()	408
9.88.3.9	supportsOperator()	408
9.89	nvonnxparser::IParserError Class Reference	408

9.89.1	Detailed Description	409
9.89.2	Constructor & Destructor Documentation	409
9.89.2.1	~IParserError()	409
9.89.3	Member Function Documentation	409
9.89.3.1	code()	409
9.89.3.2	desc()	410
9.89.3.3	file()	410
9.89.3.4	func()	410
9.89.3.5	line()	410
9.89.3.6	node()	410
9.90	nvinfer1::consistency::IPluginChecker Class Reference	411
9.90.1	Detailed Description	411
9.90.2	Constructor & Destructor Documentation	411
9.90.2.1	IPluginChecker() [1/3]	412
9.90.2.2	~IPluginChecker()	412
9.90.2.3	IPluginChecker() [2/3]	412
9.90.2.4	IPluginChecker() [3/3]	412
9.90.3	Member Function Documentation	412
9.90.3.1	operator=() [1/2]	412
9.90.3.2	operator=() [2/2]	412
9.90.3.3	validate()	413
9.91	nvinfer1::IPluginCreator Class Reference	413
9.91.1	Detailed Description	414
9.91.2	Constructor & Destructor Documentation	414
9.91.2.1	IPluginCreator()	414
9.91.2.2	~IPluginCreator()	414
9.91.3	Member Function Documentation	415
9.91.3.1	createPlugin()	415
9.91.3.2	deserializePlugin()	415
9.91.3.3	getFieldNames()	416
9.91.3.4	getPluginName()	416
9.91.3.5	getPluginNamespace()	416
9.91.3.6	getPluginVersion()	417
9.91.3.7	getTensorRTVersion()	417
9.91.3.8	setPluginNamespace()	418
9.92	nvcaffeparser1::IPluginFactoryV2 Class Reference	418
9.92.1	Detailed Description	418
9.92.2	Constructor & Destructor Documentation	419
9.92.2.1	~IPluginFactoryV2()	419
9.92.3	Member Function Documentation	419
9.92.3.1	createPlugin()	419
9.92.3.2	isPluginV2()	419
9.93	nvinfer1::IPluginRegistry Class Reference	420
9.93.1	Detailed Description	420
9.93.2	Constructor & Destructor Documentation	421
9.93.2.1	~IPluginRegistry()	421
9.93.3	Member Function Documentation	421
9.93.3.1	deregisterCreator()	421
9.93.3.2	getErrorRecorder()	422
9.93.3.3	getPluginCreator()	422
9.93.3.4	getPluginCreatorList()	423
9.93.3.5	registerCreator()	423
9.93.3.6	setErrorRecorder()	423
9.94	nvinfer1::IPluginV2 Class Reference	424

9.94.1	Detailed Description	425
9.94.2	Member Function Documentation	425
9.94.2.1	clone()	426
9.94.2.2	configureWithFormat()	426
9.94.2.3	destroy()	427
9.94.2.4	enqueue()	427
9.94.2.5	getNbOutputs()	428
9.94.2.6	getOutputDimensions()	428
9.94.2.7	getPluginNamespace()	429
9.94.2.8	getPluginType()	429
9.94.2.9	getPluginVersion()	430
9.94.2.10	getSerializationSize()	430
9.94.2.11	getTensorRTVersion()	431
9.94.2.12	getWorkspaceSize()	431
9.94.2.13	initialize()	432
9.94.2.14	serialize()	432
9.94.2.15	setPluginNamespace()	433
9.94.2.16	supportsFormat()	433
9.94.2.17	terminate()	434
9.95	nvinfer1::IPluginV2DynamicExt Class Reference	435
9.95.1	Detailed Description	436
9.95.2	Constructor & Destructor Documentation	436
9.95.2.1	~IPluginV2DynamicExt()	436
9.95.3	Member Function Documentation	436
9.95.3.1	clone()	436
9.95.3.2	configurePlugin()	437
9.95.3.3	enqueue()	438
9.95.3.4	getOutputDimensions()	438
9.95.3.5	getTensorRTVersion()	439
9.95.3.6	getWorkspaceSize()	439
9.95.3.7	supportsFormatCombination()	440
9.95.4	Member Data Documentation	440
9.95.4.1	kFORMAT_COMBINATION_LIMIT	440
9.96	nvinfer1::IPluginV2Ext Class Reference	441
9.96.1	Detailed Description	442
9.96.2	Constructor & Destructor Documentation	442
9.96.2.1	IPluginV2Ext()	442
9.96.2.2	~IPluginV2Ext()	442
9.96.3	Member Function Documentation	442
9.96.3.1	attachToContext()	442
9.96.3.2	canBroadcastInputAcrossBatch()	443
9.96.3.3	clone()	444
9.96.3.4	configurePlugin()	444
9.96.3.5	configureWithFormat()	445
9.96.3.6	detachFromContext()	445
9.96.3.7	getOutputDataType()	446
9.96.3.8	getTensorRTVersion()	446
9.96.3.9	isOutputBroadcastAcrossBatch()	446
9.97	nvinfer1::IPluginV2IOExt Class Reference	447
9.97.1	Detailed Description	448
9.97.2	Member Function Documentation	448
9.97.2.1	configurePlugin()	448
9.97.2.2	getTensorRTVersion()	449
9.97.2.3	supportsFormatCombination()	449

9.98	nvinfer1::IPluginV2Layer Class Reference	450
9.98.1	Detailed Description	451
9.98.2	Constructor & Destructor Documentation	451
9.98.2.1	~IPluginV2Layer()	451
9.98.3	Member Function Documentation	451
9.98.3.1	getPlugin()	451
9.98.4	Member Data Documentation	452
9.98.4.1	mImpl	452
9.99	nvinfer1::IPoolingLayer Class Reference	452
9.99.1	Detailed Description	454
9.99.2	Constructor & Destructor Documentation	454
9.99.2.1	~IPoolingLayer()	454
9.99.3	Member Function Documentation	454
9.99.3.1	getAverageCountExcludesPadding()	454
9.99.3.2	getBlendFactor()	454
9.99.3.3	getPadding()	455
9.99.3.4	getPaddingMode()	455
9.99.3.5	getPaddingNd()	455
9.99.3.6	getPoolingType()	456
9.99.3.7	getPostPadding()	456
9.99.3.8	getPrePadding()	456
9.99.3.9	getStride()	456
9.99.3.10	getStrideNd()	457
9.99.3.11	getWindowSize()	457
9.99.3.12	getWindowSizeNd()	457
9.99.3.13	setAverageCountExcludesPadding()	458
9.99.3.14	setBlendFactor()	458
9.99.3.15	setPadding()	458
9.99.3.16	setPaddingMode()	459
9.99.3.17	setPaddingNd()	459
9.99.3.18	setPoolingType()	459
9.99.3.19	setPostPadding()	460
9.99.3.20	setPrePadding()	460
9.99.3.21	setStride()	460
9.99.3.22	setStrideNd()	461
9.99.3.23	setWindowSize()	461
9.99.3.24	setWindowSizeNd()	461
9.99.4	Member Data Documentation	462
9.99.4.1	mImpl	462
9.100	nvinfer1::IProfiler Class Reference	462
9.100.1	Detailed Description	462
9.100.2	Constructor & Destructor Documentation	462
9.100.2.1	~IProfiler()	462
9.100.3	Member Function Documentation	463
9.100.3.1	reportLayerTime()	463
9.101	nvinfer1::IQuantizeLayer Class Reference	464
9.101.1	Detailed Description	465
9.101.2	Constructor & Destructor Documentation	465
9.101.2.1	~IQuantizeLayer()	466
9.101.3	Member Function Documentation	466
9.101.3.1	getAxis()	466
9.101.3.2	setAxis()	466
9.101.4	Member Data Documentation	466
9.101.4.1	mImpl	466

9.102	<code>nvinfer1::IRaggedSoftMaxLayer</code> Class Reference	467
9.102.1	Detailed Description	467
9.102.2	Constructor & Destructor Documentation	467
9.102.2.1	<code>~IRaggedSoftMaxLayer()</code>	468
9.102.3	Member Data Documentation	468
9.102.3.1	<code>mImpl</code>	468
9.103	<code>nvinfer1::IRecurrenceLayer</code> Class Reference	468
9.103.1	Constructor & Destructor Documentation	469
9.103.1.1	<code>~IRecurrenceLayer()</code>	469
9.103.2	Member Function Documentation	469
9.103.2.1	<code>setInput()</code>	469
9.103.3	Member Data Documentation	469
9.103.3.1	<code>mImpl</code>	470
9.104	<code>nvinfer1::IReduceLayer</code> Class Reference	470
9.104.1	Detailed Description	470
9.104.2	Constructor & Destructor Documentation	471
9.104.2.1	<code>~IReduceLayer()</code>	471
9.104.3	Member Function Documentation	471
9.104.3.1	<code>getKeepDimensions()</code>	471
9.104.3.2	<code>getOperation()</code>	471
9.104.3.3	<code>getReduceAxes()</code>	472
9.104.3.4	<code>setKeepDimensions()</code>	472
9.104.3.5	<code>setOperation()</code>	472
9.104.3.6	<code>setReduceAxes()</code>	472
9.104.4	Member Data Documentation	473
9.104.4.1	<code>mImpl</code>	473
9.105	<code>nvinfer1::IRefitter</code> Class Reference	473
9.105.1	Detailed Description	474
9.105.2	Constructor & Destructor Documentation	474
9.105.2.1	<code>~IRefitter()</code>	474
9.105.3	Member Function Documentation	474
9.105.3.1	<code>destroy()</code>	475
9.105.3.2	<code>getAll()</code>	475
9.105.3.3	<code>getAllWeights()</code>	475
9.105.3.4	<code>getDynamicRangeMax()</code>	476
9.105.3.5	<code>getDynamicRangeMin()</code>	476
9.105.3.6	<code>getErrorRecorder()</code>	476
9.105.3.7	<code>getLogger()</code>	477
9.105.3.8	<code>getMaxThreads()</code>	477
9.105.3.9	<code>getMissing()</code>	477
9.105.3.10	<code>getMissingWeights()</code>	478
9.105.3.11	<code>getTensorsWithDynamicRange()</code>	478
9.105.3.12	<code>refitCudaEngine()</code>	479
9.105.3.13	<code>setDynamicRange()</code>	479
9.105.3.14	<code>setErrorRecorder()</code>	479
9.105.3.15	<code>setMaxThreads()</code>	480
9.105.3.16	<code>setNamedWeights()</code>	480
9.105.3.17	<code>setWeights()</code>	481
9.105.4	Member Data Documentation	481
9.105.4.1	<code>mImpl</code>	481
9.106	<code>nvinfer1::IResizeLayer</code> Class Reference	481
9.106.1	Detailed Description	483
9.106.2	Constructor & Destructor Documentation	483
9.106.2.1	<code>~IResizeLayer()</code>	484

9.106.3 Member Function Documentation	484
9.106.3.1 getAlignCorners()	484
9.106.3.2 getCoordinateTransformation()	484
9.106.3.3 getNearestRounding()	484
9.106.3.4 getOutputDimensions()	485
9.106.3.5 getResizeMode()	485
9.106.3.6 getScales()	485
9.106.3.7 getSelectorForSinglePixel()	486
9.106.3.8 setAlignCorners()	486
9.106.3.9 setCoordinateTransformation()	486
9.106.3.10 setInput()	486
9.106.3.11 setNearestRounding()	487
9.106.3.12 setOutputDimensions()	487
9.106.3.13 setResizeMode()	488
9.106.3.14 setScales()	488
9.106.3.15 setSelectorForSinglePixel()	489
9.106.4 Member Data Documentation	489
9.106.4.1 mImpl	489
9.107 nvinfer1::IRNNv2Layer Class Reference	490
9.107.1 Detailed Description	491
9.107.2 Constructor & Destructor Documentation	491
9.107.2.1 ~IRNNv2Layer()	491
9.107.3 Member Function Documentation	491
9.107.3.1 getBiasForGate()	492
9.107.3.2 getCellState()	492
9.107.3.3 getDataLength()	492
9.107.3.4 getDirection()	492
9.107.3.5 getHiddenSize()	493
9.107.3.6 getHiddenState()	493
9.107.3.7 getInputMode()	493
9.107.3.8 getLayerCount()	493
9.107.3.9 getMaxSeqLength()	493
9.107.3.10 getOperation()	494
9.107.3.11 getSequenceLengths()	494
9.107.3.12 getWeightsForGate()	494
9.107.3.13 setBiasForGate()	494
9.107.3.14 setCellState()	495
9.107.3.15 setDirection()	495
9.107.3.16 setHiddenState()	496
9.107.3.17 setInputMode()	496
9.107.3.18 setOperation()	496
9.107.3.19 setSequenceLengths()	497
9.107.3.20 setWeightsForGate()	497
9.107.4 Member Data Documentation	498
9.107.4.1 mImpl	498
9.108 nvinfer1::IRuntime Class Reference	498
9.108.1 Detailed Description	499
9.108.2 Constructor & Destructor Documentation	499
9.108.2.1 ~IRuntime()	500
9.108.3 Member Function Documentation	500
9.108.3.1 deserializeCudaEngine() [1/2]	500
9.108.3.2 deserializeCudaEngine() [2/2]	500
9.108.3.3 destroy()	501
9.108.3.4 getDLACore()	501

9.108.3.5	getErrorRecorder()	501
9.108.3.6	getLogger()	502
9.108.3.7	getMaxThreads()	502
9.108.3.8	getNbDLACores()	502
9.108.3.9	setDLACore()	502
9.108.3.10	setErrorRecorder()	503
9.108.3.11	setGpuAllocator()	503
9.108.3.12	setMaxThreads()	504
9.108.4	Member Data Documentation	504
9.108.4.1	mImpl	504
9.109	nvinfer1::safe::IRuntime Class Reference	504
9.109.1	Detailed Description	505
9.109.2	Constructor & Destructor Documentation	505
9.109.2.1	IRuntime() [1/3]	505
9.109.2.2	~IRuntime()	506
9.109.2.3	IRuntime() [2/3]	506
9.109.2.4	IRuntime() [3/3]	506
9.109.3	Member Function Documentation	506
9.109.3.1	deserializeCudaEngine()	506
9.109.3.2	getErrorRecorder()	507
9.109.3.3	operator=() [1/2]	507
9.109.3.4	operator=() [2/2]	508
9.109.3.5	setErrorRecorder()	508
9.109.3.6	setGpuAllocator()	508
9.110	nvinfer1::IScaleLayer Class Reference	509
9.110.1	Detailed Description	510
9.110.2	Constructor & Destructor Documentation	510
9.110.2.1	~IScaleLayer()	511
9.110.3	Member Function Documentation	511
9.110.3.1	getChannelAxis()	511
9.110.3.2	getMode()	511
9.110.3.3	getPower()	511
9.110.3.4	getScale()	512
9.110.3.5	getShift()	512
9.110.3.6	setChannelAxis()	512
9.110.3.7	setMode()	513
9.110.3.8	setPower()	513
9.110.3.9	setScale()	513
9.110.3.10	setShift()	513
9.110.4	Member Data Documentation	514
9.110.4.1	mImpl	514
9.111	nvinfer1::IScatterLayer Class Reference	514
9.111.1	Detailed Description	515
9.111.2	Constructor & Destructor Documentation	516
9.111.2.1	~IScatterLayer()	516
9.111.3	Member Function Documentation	516
9.111.3.1	getAxis()	516
9.111.3.2	getMode()	516
9.111.3.3	setAxis()	516
9.111.3.4	setMode()	517
9.111.4	Member Data Documentation	517
9.111.4.1	mImpl	517
9.112	nvinfer1::ISelectLayer Class Reference	517
9.112.1	Detailed Description	517

9.112.2 Constructor & Destructor Documentation	518
9.112.2.1 ~ISelectLayer()	518
9.112.3 Member Data Documentation	518
9.112.3.1 mImpl	518
9.113 nvinfer1::IShapeLayer Class Reference	518
9.113.1 Detailed Description	519
9.113.2 Constructor & Destructor Documentation	519
9.113.2.1 ~IShapeLayer()	519
9.113.3 Member Data Documentation	519
9.113.3.1 mImpl	519
9.114 nvinfer1::IShuffleLayer Class Reference	520
9.114.1 Detailed Description	521
9.114.2 Constructor & Destructor Documentation	521
9.114.2.1 ~IShuffleLayer()	521
9.114.3 Member Function Documentation	521
9.114.3.1 getFirstTranspose()	521
9.114.3.2 getReshapeDimensions()	522
9.114.3.3 getSecondTranspose()	522
9.114.3.4 getZeroIsPlaceholder()	522
9.114.3.5 setFirstTranspose()	522
9.114.3.6 setInput()	523
9.114.3.7 setReshapeDimensions()	524
9.114.3.8 setSecondTranspose()	524
9.114.3.9 setZeroIsPlaceholder()	525
9.114.4 Member Data Documentation	525
9.114.4.1 mImpl	525
9.115 nvinfer1::ISliceLayer Class Reference	525
9.115.1 Detailed Description	526
9.115.2 Constructor & Destructor Documentation	527
9.115.2.1 ~ISliceLayer()	527
9.115.3 Member Function Documentation	527
9.115.3.1 getMode()	527
9.115.3.2 getSize()	528
9.115.3.3 getStart()	528
9.115.3.4 getStride()	528
9.115.3.5 setInput()	528
9.115.3.6 setMode()	529
9.115.3.7 setSize()	529
9.115.3.8 setStart()	530
9.115.3.9 setStride()	530
9.115.4 Member Data Documentation	531
9.115.4.1 mImpl	531
9.116 nvinfer1::ISoftMaxLayer Class Reference	531
9.116.1 Detailed Description	532
9.116.2 Constructor & Destructor Documentation	532
9.116.2.1 ~ISoftMaxLayer()	532
9.116.3 Member Function Documentation	532
9.116.3.1 getAxes()	532
9.116.3.2 setAxes()	533
9.116.4 Member Data Documentation	533
9.116.4.1 mImpl	533
9.117 nvinfer1::ITensor Class Reference	533
9.117.1 Detailed Description	535
9.117.2 Constructor & Destructor Documentation	535

9.117.2.1	~ITensor()	535
9.117.3	Member Function Documentation	535
9.117.3.1	dynamicRangeIsSet()	536
9.117.3.2	getAllowedFormats()	536
9.117.3.3	getBroadcastAcrossBatch()	536
9.117.3.4	getDimensions()	536
9.117.3.5	getDynamicRangeMax()	537
9.117.3.6	getDynamicRangeMin()	537
9.117.3.7	getLocation()	537
9.117.3.8	getName()	538
9.117.3.9	getType()	538
9.117.3.10	isExecutionTensor()	538
9.117.3.11	isNetworkInput()	539
9.117.3.12	isNetworkOutput()	539
9.117.3.13	isShapeTensor()	539
9.117.3.14	resetDynamicRange()	540
9.117.3.15	setAllowedFormats()	540
9.117.3.16	setBroadcastAcrossBatch()	540
9.117.3.17	setDimensions()	541
9.117.3.18	setDynamicRange()	541
9.117.3.19	setLocation()	542
9.117.3.20	setName()	542
9.117.3.21	setType()	543
9.117.4	Member Data Documentation	543
9.117.4.1	mImpl	543
9.118	nvinfer1::ITimingCache Class Reference	543
9.118.1	Detailed Description	544
9.118.2	Constructor & Destructor Documentation	544
9.118.2.1	~ITimingCache()	544
9.118.3	Member Function Documentation	544
9.118.3.1	combine()	544
9.118.3.2	reset()	545
9.118.3.3	serialize()	545
9.118.4	Member Data Documentation	546
9.118.4.1	mImpl	546
9.119	nvinfer1::ITopKLayer Class Reference	546
9.119.1	Detailed Description	547
9.119.2	Constructor & Destructor Documentation	547
9.119.2.1	~ITopKLayer()	547
9.119.3	Member Function Documentation	547
9.119.3.1	getK()	547
9.119.3.2	getOperation()	548
9.119.3.3	getReduceAxes()	548
9.119.3.4	setK()	548
9.119.3.5	setOperation()	548
9.119.3.6	setReduceAxes()	549
9.119.4	Member Data Documentation	549
9.119.4.1	mImpl	549
9.120	nvinfer1::ITripLimitLayer Class Reference	549
9.120.1	Constructor & Destructor Documentation	550
9.120.1.1	~ITripLimitLayer()	550
9.120.2	Member Function Documentation	550
9.120.2.1	getTripLimit()	550
9.120.3	Member Data Documentation	550

9.120.3.1 mImpl	550
9.121 nvuffparser::IuffParser Class Reference	550
9.121.1 Detailed Description	551
9.121.2 Constructor & Destructor Documentation	551
9.121.2.1 ~IuffParser()	551
9.121.3 Member Function Documentation	552
9.121.3.1 destroy()	552
9.121.3.2 getErrorRecorder()	552
9.121.3.3 getUffRequiredVersionMajor()	552
9.121.3.4 getUffRequiredVersionMinor()	552
9.121.3.5 getUffRequiredVersionPatch()	553
9.121.3.6 parse()	553
9.121.3.7 parseBuffer()	553
9.121.3.8 registerInput()	554
9.121.3.9 registerOutput()	554
9.121.3.10 setErrorRecorder()	554
9.121.3.11 setPluginNamespace()	555
9.122 nvinfer1::UnaryLayer Class Reference	555
9.122.1 Detailed Description	555
9.122.2 Constructor & Destructor Documentation	556
9.122.2.1 ~UnaryLayer()	556
9.122.3 Member Function Documentation	556
9.122.3.1 getOperation()	556
9.122.3.2 setOperation()	556
9.122.4 Member Data Documentation	556
9.122.4.1 mImpl	557
9.123 nvinfer1::plugin::NMSParameters Struct Reference	557
9.123.1 Detailed Description	557
9.123.2 Member Data Documentation	558
9.123.2.1 backgroundLabelId	558
9.123.2.2 iouThreshold	558
9.123.2.3 isNormalized	558
9.123.2.4 keepTopK	558
9.123.2.5 numClasses	558
9.123.2.6 scoreThreshold	558
9.123.2.7 shareLocation	559
9.123.2.8 topK	559
9.124 nvinfer1::Permutation Struct Reference	559
9.124.1 Member Data Documentation	559
9.124.1.1 order	559
9.125 nvinfer1::PluginField Class Reference	560
9.125.1 Detailed Description	560
9.125.2 Constructor & Destructor Documentation	560
9.125.2.1 PluginField()	560
9.125.3 Member Data Documentation	560
9.125.3.1 data	561
9.125.3.2 length	561
9.125.3.3 name	561
9.125.3.4 type	561
9.126 nvinfer1::PluginFieldCollection Struct Reference	561
9.126.1 Detailed Description	562
9.126.2 Member Data Documentation	562
9.126.2.1 fields	562
9.126.2.2 nbFields	562

9.127	<code>nvinfer1::PluginRegistrar< T ></code> Class Template Reference	562
9.127.1	Detailed Description	562
9.127.2	Constructor & Destructor Documentation	563
9.127.2.1	<code>PluginRegistrar()</code>	563
9.128	<code>nvinfer1::safe::PluginRegistrar< T ></code> Class Template Reference	563
9.128.1	Detailed Description	563
9.128.2	Constructor & Destructor Documentation	564
9.128.2.1	<code>PluginRegistrar()</code>	564
9.129	<code>nvinfer1::PluginTensorDesc</code> Struct Reference	564
9.129.1	Detailed Description	564
9.129.2	Member Data Documentation	565
9.129.2.1	<code>dims</code>	565
9.129.2.2	<code>format</code>	565
9.129.2.3	<code>scale</code>	565
9.129.2.4	<code>type</code>	565
9.130	<code>PluginVersion</code> Struct Reference	565
9.130.1	Detailed Description	566
9.131	<code>nvinfer1::plugin::PriorBoxParameters</code> Struct Reference	566
9.131.1	Detailed Description	566
9.131.2	Member Data Documentation	567
9.131.2.1	<code>aspectRatios</code>	567
9.131.2.2	<code>clip</code>	567
9.131.2.3	<code>flip</code>	567
9.131.2.4	<code>imgH</code>	567
9.131.2.5	<code>imgW</code>	568
9.131.2.6	<code>maxSize</code>	568
9.131.2.7	<code>minSize</code>	568
9.131.2.8	<code>numAspectRatios</code>	568
9.131.2.9	<code>numMaxSize</code>	568
9.131.2.10	<code>numMinSize</code>	568
9.131.2.11	<code>loffset</code>	568
9.131.2.12	<code>stepH</code>	569
9.131.2.13	<code>stepW</code>	569
9.131.2.14	<code>variance</code>	569
9.132	<code>nvinfer1::plugin::Quadruple</code> Struct Reference	569
9.132.1	Detailed Description	569
9.132.2	Member Data Documentation	569
9.132.2.1	<code>data</code>	570
9.133	<code>nvinfer1::plugin::RegionParameters</code> Struct Reference	570
9.133.1	Detailed Description	570
9.133.2	Member Data Documentation	570
9.133.2.1	<code>classes</code>	571
9.133.2.2	<code>coords</code>	571
9.133.2.3	<code>num</code>	571
9.133.2.4	<code>smTree</code>	571
9.134	<code>nvinfer1::plugin::RPROIParams</code> Struct Reference	571
9.134.1	Detailed Description	571
9.134.2	Member Data Documentation	572
9.134.2.1	<code>anchorsRatioCount</code>	572
9.134.2.2	<code>anchorsScaleCount</code>	572
9.134.2.3	<code>featureStride</code>	572
9.134.2.4	<code>iouThreshold</code>	572
9.134.2.5	<code>minBoxSize</code>	573
9.134.2.6	<code>nmsMaxOut</code>	573

9.134.2.7 poolingH	573
9.134.2.8 poolingW	573
9.134.2.9 preNmsTop	573
9.134.2.10 spatialScale	573
9.135 nvinfer1::plugin::softmaxTree Struct Reference	573
9.135.1 Detailed Description	574
9.135.2 Member Data Documentation	574
9.135.2.1 child	574
9.135.2.2 group	574
9.135.2.3 groupOffset	574
9.135.2.4 groups	574
9.135.2.5 groupSize	575
9.135.2.6 leaf	575
9.135.2.7 n	575
9.135.2.8 name	575
9.135.2.9 parent	575
9.136 nvinfer1::Weights Class Reference	575
9.136.1 Detailed Description	576
9.136.2 Member Data Documentation	576
9.136.2.1 count	576
9.136.2.2 type	576
9.136.2.3 values	576
10 File Documentation	577
10.1 NvCaffeParser.h File Reference	577
10.1.1 Detailed Description	578
10.2 NvCaffeParser.h	578
10.3 NvInfer.h File Reference	579
10.3.1 Detailed Description	585
10.4 NvInfer.h	585
10.5 NvInferConsistency.h File Reference	626
10.5.1 Function Documentation	626
10.5.1.1 createConsistencyChecker_INTERNAL()	627
10.6 NvInferConsistency.h	627
10.7 NvInferLegacyDims.h File Reference	628
10.7.1 Detailed Description	629
10.8 NvInferLegacyDims.h	629
10.9 NvInferPlugin.h File Reference	630
10.9.1 Detailed Description	631
10.9.2 Function Documentation	631
10.9.2.1 createAnchorGeneratorPlugin()	632
10.9.2.2 createBatchedNMSPlugin()	632
10.9.2.3 createInstanceNormalizationPlugin()	632
10.9.2.4 createNMSPlugin()	633
10.9.2.5 createNormalizePlugin()	633
10.9.2.6 createPriorBoxPlugin()	633
10.9.2.7 createRegionPlugin()	634
10.9.2.8 createReorgPlugin()	634
10.9.2.9 createRPNROIPlugin()	634
10.9.2.10 createSplitPlugin()	635
10.9.2.11 initLibNvInferPlugins()	635
10.10 NvInferPlugin.h	636
10.11 NvInferPluginUtils.h File Reference	636
10.11.1 Detailed Description	637

10.12NvInferPluginUtils.h	638
10.13NvInferRuntime.h File Reference	639
10.13.1 Detailed Description	642
10.13.2 Macro Definition Documentation	642
10.13.2.1 REGISTER_TENSORRT_PLUGIN	642
10.13.3 Function Documentation	642
10.13.3.1 getLogger()	642
10.13.3.2 getPluginRegistry()	642
10.14NvInferRuntime.h	643
10.15NvInferRuntimeCommon.h File Reference	655
10.15.1 Detailed Description	657
10.15.2 Macro Definition Documentation	657
10.15.2.1 NV_TENSORRT_VERSION	657
10.15.2.2 TENSORRTAPI	658
10.15.2.3 TRT_DEPRECATED	658
10.15.2.4 TRT_DEPRECATED_API	658
10.15.2.5 TRT_DEPRECATED_ENUM	658
10.15.2.6 TRTNOEXCEPT	658
10.15.3 Function Documentation	658
10.15.3.1 getInferLibVersion()	658
10.16NvInferRuntimeCommon.h	659
10.17NvInferSafeRuntime.h File Reference	666
10.17.1 Detailed Description	667
10.17.2 Macro Definition Documentation	667
10.17.2.1 REGISTER_SAFE_TENSORRT_PLUGIN	667
10.18NvInferSafeRuntime.h	668
10.19NvInferVersion.h File Reference	670
10.19.1 Detailed Description	670
10.19.2 Macro Definition Documentation	670
10.19.2.1 NV_TENSORRT_BUILD	670
10.19.2.2 NV_TENSORRT_LWS_MAJOR	671
10.19.2.3 NV_TENSORRT_LWS_MINOR	671
10.19.2.4 NV_TENSORRT_LWS_PATCH	671
10.19.2.5 NV_TENSORRT_MAJOR	671
10.19.2.6 NV_TENSORRT_MINOR	671
10.19.2.7 NV_TENSORRT_PATCH	671
10.19.2.8 NV_TENSORRT_SONAME_MAJOR	672
10.19.2.9 NV_TENSORRT_SONAME_MINOR	672
10.19.2.10 NV_TENSORRT_SONAME_PATCH	672
10.20NvInferVersion.h	672
10.21NvOnnxConfig.h File Reference	673
10.21.1 Detailed Description	673
10.22NvOnnxConfig.h	673
10.23NvUffParser.h File Reference	674
10.23.1 Detailed Description	675
10.23.2 Macro Definition Documentation	675
10.23.2.1 UFF_REQUIRED_VERSION_MAJOR	675
10.23.2.2 UFF_REQUIRED_VERSION_MINOR	675
10.23.2.3 UFF_REQUIRED_VERSION_PATCH	675
10.24NvUffParser.h	676
10.25NvUtils.h File Reference	677
10.25.1 Detailed Description	677
10.26NvUtils.h	678
10.27NvOnnxParser.h File Reference	678

10.27.1 Detailed Description	679
10.27.2 Macro Definition Documentation	679
10.27.2.1 NV_ONNX_PARSER_MAJOR	679
10.27.2.2 NV_ONNX_PARSER_MINOR	680
10.27.2.3 NV_ONNX_PARSER_PATCH	680
10.27.3 Typedef Documentation	680
10.27.3.1 SubGraph_t	680
10.27.3.2 SubGraphCollection_t	680
10.27.4 Function Documentation	680
10.27.4.1 createNvOnnxParser_INTERNAL()	680
10.27.4.2 getNvOnnxParserVersion()	681
10.28 NvOnnxParser.h	681
Index	683

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	655
NvInferSafeRuntime.h	666
NvInferVersion.h	670

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 `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`

Use `delete` instead. Deprecated in TensorRT 8.0.

Member `nvcaffeparser1::ICaffeParser::destroy () noexcept=0`

Use `delete` instead. Deprecated in TensorRT 8.0.

Member `nvinfer1::IAlgorithm::getAlgorithmIOInfo (int32_t index) const noexcept`

Use `IAlgorithm::getAlgorithmIOInfoByIndex` instead. Deprecated in TensorRT 8.0

Member `nvinfer1::IBuilder::buildEngineWithConfig (INetworkDefinition &network, IBuilderConfig &config) noexcept`

Use `IBuilder::buildSerializedNetwork`. Deprecated in TensorRT 8.0

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

Use `delete` instead. Deprecated in TensorRT 8.0

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`

Use `delete` instead. Deprecated in TensorRT 8.0

Member `nvinfer1::IBuilderConfig::getMaxWorkspaceSize () const noexcept`

Use `IBuilderConfig::getMemoryPoolLimit` with `MemoryPoolType::kWORKSPACE`. Deprecated in TensorRT 8.3

Member `nvinfer1::IBuilderConfig::getMinTimingIterations () const noexcept`

Use `getAvgTimingIterations()` instead. Deprecated in TensorRT 8.4.

Member `nvinfer1::IBuilderConfig::setMaxWorkspaceSize (std::size_t workspaceSize) noexcept`

Use `IBuilderConfig::setMemoryPoolLimit` with `MemoryPoolType::kWORKSPACE`. Deprecated in TensorRT 8.3

Member `nvinfer1::IBuilderConfig::setMinTimingIterations (int32_t minTiming) noexcept`

Use `setAvgTimingIterations()` instead. Deprecated in TensorRT 8.4.

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::destroy () noexcept`

Use `delete` instead. Deprecated in TRT 8.0.

Member `nvinfer1::ICudaEngine::getMaxBatchSize () const noexcept`

Deprecated in TensorRT 8.4.

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`

Use `delete` instead. Deprecated in TRT 8.0.

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::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::setOptimizationProfile (int32_t profileIndex) noexcept`**
Superseded by `setOptimizationProfileAsync`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0.
- Class `nvinfer1::IFullyConnectedLayer`**
Use `IMatrixMultiplyLayer` instead. Deprecated in TensorRT 8.4.
- Member `nvinfer1::IGpuAllocator::free (void *const memory) noexcept=0`**
Superseded by `deallocate`. Deprecated in TensorRT 8.0.
- Member `nvinfer1::IHostMemory::destroy () noexcept`**
Use `delete` instead. Deprecated in TRT 8.0.
- 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`**
Use `addMatrixMultiply` instead. Deprecated in TensorRT 8.4.
- 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`**
Superseded by `addSlice`. Deprecated in TensorRT 8.0
- 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`**
Superseded by `INetworkDefinition::addLoop`. Deprecated prior to TensorRT 8.0 and will be removed in 9.0
- Member `nvinfer1::INetworkDefinition::destroy () noexcept`**
Use `delete` instead. Deprecated in TensorRT 8.0
- 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
- 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::IRfitter::destroy () noexcept`

Use `delete` instead. Deprecated in TRT 8.0.

Member `nvinfer1::IResizeLayer::getAlignCorners () const noexcept`

Superseded by `IResizeLayer::getCoordinateTransformation()`. Deprecated in TensorRT 8.0

Member `nvinfer1::IResizeLayer::setAlignCorners (bool alignCorners) noexcept`

Superseded by `IResizeLayer::setCoordinateTransformation()`. Deprecated in TensorRT 8.0

Class `nvinfer1::IRNNv2Layer`

Use `INetworkDefinition::addLoop` instead. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Member `nvinfer1::IRuntime::deserializeCudaEngine (void const *blob, std::size_t size, IPluginFactory *pluginFactory) noexcept`

Deprecated in TensorRT 8.0.

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

Use `delete` instead. Deprecated in TRT 8.0.

Member `nvinfer1::kDEFAULT`

Use `kLAYER_NAMES_ONLY`. Deprecated in TensorRT 8.0.

Member `nvinfer1::kDEFAULT`

Use `kSTANDARD`. Deprecated in TensorRT 8.0.

Member `nvinfer1::kSAFE_DLA`

Use `kDLA_STANDALONE`. Deprecated in TensorRT 8.0.

Member `nvinfer1::kSAFE_GPU`

Use `kSAFETY`. Deprecated in TensorRT 8.0.

Member `nvinfer1::kVERBOSE`

Use `kDETAILED`. Deprecated in TensorRT 8.0.

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

Deprecated in TensorRT 8.0.

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

Deprecated in TensorRT 8.0.

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

Deprecated in TensorRT 8.0.

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

Use `delete` instead. Deprecated in TRT 8.0.

Member [nvuffparser::createUffParser \(\) noexcept](#)

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

Member [nvuffparser::IUffParser::destroy \(\) noexcept=0](#)

Use `delete` instead. Deprecated in TRT 8.0.

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

nvcaffeparser1	The TensorRT Caffe parser API namespace	25
nvinfer1	The TensorRT API version 1 namespace	26
nvinfer1::consistency	72
nvinfer1::impl	72
nvinfer1::plugin	73
nvinfer1::safe	The safety subset of TensorRT's API version 1 namespace	74
nvinfer1::utils	75
nvonnxparser	The TensorRT ONNX parser API namespace	78
nvuffparser	The TensorRT UFF parser API namespace	80

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

nvinfer1::plugin::DetectionOutputParameters	83
Dims	86
nvinfer1::Dims32	89
nvinfer1::Dims2	87
nvinfer1::DimsHW	93
nvinfer1::Dims3	88
nvinfer1::Dims4	90
nvinfer1::DimsExprs	92
nvinfer1::DynamicPluginTensorDesc	95
nvinfer1::impl::EnumMaxImpl< T >	96
nvinfer1::impl::EnumMaxImpl< ActivationType >	97
nvinfer1::impl::EnumMaxImpl< AllocatorFlag >	97
nvinfer1::impl::EnumMaxImpl< DataType >	98
nvinfer1::impl::EnumMaxImpl< ElementWiseOperation >	99
nvinfer1::impl::EnumMaxImpl< EngineCapability >	100
nvinfer1::impl::EnumMaxImpl< ErrorCode >	100
nvinfer1::impl::EnumMaxImpl< ILogger::Severity >	101
nvinfer1::impl::EnumMaxImpl< PaddingMode >	102
nvinfer1::impl::EnumMaxImpl< PoolingType >	103
nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation >	103
nvinfer1::impl::EnumMaxImpl< ResizeMode >	104
nvinfer1::impl::EnumMaxImpl< ResizeRoundMode >	105
nvinfer1::impl::EnumMaxImpl< ResizeSelector >	105
nvinfer1::impl::EnumMaxImpl< TensorFormat >	106
nvinfer1::impl::EnumMaxImpl< TensorLocation >	107
nvuffparser::FieldCollection	108
nvuffparser::FieldMap	108
nvinfer1::safe::FloatingPointErrorInformation	110
nvinfer1::plugin::GridAnchorParameters	111
nvinfer1::IAlgorithmSelector	124
nvcaffeparser1::IBinaryProtoBlob	130

nvcaffeparser1::IBlobNameToTensor	132
nvcaffeparser1::ICaffeParser	164
nvinfer1::consistency::IConsistencyChecker	173
nvinfer1::safe::ICudaEngine	206
nvinfer1::IErrorRecorder	243
nvinfer1::safe::IExecutionContext	266
nvinfer1::IGpuAllocator	291
nvinfer1::IInt8Calibrator	306
nvinfer1::IInt8EntropyCalibrator	309
nvinfer1::IInt8EntropyCalibrator2	310
nvinfer1::IInt8LegacyCalibrator	311
nvinfer1::IInt8MinMaxCalibrator	314
nvinfer1::ILogger	325
nvinfer1::INoCopy	384
nvinfer1::IAlgorithm	117
nvinfer1::IAlgorithmContext	120
nvinfer1::IAlgorithmIOInfo	122
nvinfer1::IAlgorithmVariant	126
nvinfer1::IBuilder	133
nvinfer1::IBuilderConfig	143
nvinfer1::ICudaEngine	190
nvinfer1::IDimensionExpr	231
nvinfer1::IEngineInspector	238
nvinfer1::IExecutionContext	249
nvinfer1::IExprBuilder	273
nvinfer1::IHostMemory	295
nvinfer1::IIfConditional	299
nvinfer1::ILayer	317
nvinfer1::IActivationLayer	113
nvinfer1::IAssertionLayer	128
nvinfer1::IConcatenationLayer	169
nvinfer1::IConstantLayer	175
nvinfer1::IConvolutionLayer	178
nvinfer1::IDeconvolutionLayer	217
nvinfer1::IDequantizeLayer	229
nvinfer1::IEinsumLayer	233
nvinfer1::IElementWiseLayer	236
nvinfer1::IFillLayer	275
nvinfer1::IFullyConnectedLayer	281
nvinfer1::IGatherLayer	286
nvinfer1::IIdentityLayer	297
nvinfer1::IIfConditionalBoundaryLayer	302
nvinfer1::IConditionLayer	172
nvinfer1::IIfConditionalInputLayer	304
nvinfer1::IIfConditionalOutputLayer	305
nvinfer1::ILRNLayer	335
nvinfer1::ILoopBoundaryLayer	331
nvinfer1::IIteratorLayer	315
nvinfer1::ILoopOutputLayer	332
nvinfer1::IRecurrenceLayer	468
nvinfer1::ITripLimitLayer	549
nvinfer1::IMatrixMultiplyLayer	340
nvinfer1::IPaddingLayer	398
nvinfer1::IParametricReLULayer	403

nvinfer1::IPluginV2Layer	450
nvinfer1::IPoolingLayer	452
nvinfer1::IQuantizeLayer	464
nvinfer1::IRNNv2Layer	490
nvinfer1::IRaggedSoftMaxLayer	467
nvinfer1::IReduceLayer	470
nvinfer1::IResizeLayer	481
nvinfer1::IScaleLayer	509
nvinfer1::IScatterLayer	514
nvinfer1::ISelectLayer	517
nvinfer1::IShapeLayer	518
nvinfer1::IShuffleLayer	520
nvinfer1::ISliceLayer	525
nvinfer1::ISoftMaxLayer	531
nvinfer1::ITopKLayer	546
nvinfer1::UnaryLayer	555
nvinfer1::ILoop	328
nvinfer1::INetworkDefinition	342
nvinfer1::IOptimizationProfile	393
nvinfer1::IRefitter	473
nvinfer1::IRuntime	498
nvinfer1::ITensor	533
nvinfer1::ITimingCache	543
nvonnxparser::IONnxConfig	386
nvonnxparser::IParser	404
nvonnxparser::IParserError	408
nvinfer1::IPluginCreator	413
nvinfer1::consistency::IPluginChecker	411
nvcaffeparser1::IPluginFactoryV2	418
nvinfer1::IPluginRegistry	420
nvinfer1::IPluginV2	424
nvinfer1::IPluginV2Ext	441
nvinfer1::IPluginV2DynamicExt	435
nvinfer1::IPluginV2IOExt	447
nvinfer1::IProfiler	462
nvinfer1::safe::IRuntime	504
nvuffparser::IUffParser	550
nvinfer1::plugin::NMSParameters	557
nvinfer1::Permutation	559
nvinfer1::PluginField	560
nvinfer1::PluginFieldCollection	561
nvinfer1::PluginRegistrar< T >	562
nvinfer1::safe::PluginRegistrar< T >	563
nvinfer1::PluginTensorDesc	564
PluginVersion	565
nvinfer1::plugin::PriorBoxParameters	566
nvinfer1::plugin::Quadruple	569
nvinfer1::plugin::RegionParameters	570
nvinfer1::plugin::RPROIParams	571
nvinfer1::plugin::softmaxTree	573
nvinfer1::Weights	575

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:	83
Dims	Structure to define the dimensions of a tensor	86
nvinfer1::Dims2	Descriptor for two-dimensional data	87
nvinfer1::Dims3	Descriptor for three-dimensional data	88
nvinfer1::Dims32	89
nvinfer1::Dims4	Descriptor for four-dimensional data	90
nvinfer1::DimsExprs	92
nvinfer1::DimsHW	Descriptor for two-dimensional spatial data	93
nvinfer1::DynamicPluginTensorDesc	95
nvinfer1::impl::EnumMaxImpl< T >	Declaration of EnumMaxImpl struct to store maximum number of elements in an enumeration type	96
nvinfer1::impl::EnumMaxImpl< ActivationType >	97
nvinfer1::impl::EnumMaxImpl< AllocatorFlag >	Maximum number of elements in AllocatorFlag enum	97
nvinfer1::impl::EnumMaxImpl< DataType >	Maximum number of elements in DataType enum	98
nvinfer1::impl::EnumMaxImpl< ElementWiseOperation >	99
nvinfer1::impl::EnumMaxImpl< EngineCapability >	Maximum number of elements in EngineCapability enum	100
nvinfer1::impl::EnumMaxImpl< ErrorCode >	Maximum number of elements in ErrorCode enum	100

nvinfer1::impl::EnumMaxImpl< ILogger::Severity >	
Maximum number of elements in ILogger::Severity enum	101
nvinfer1::impl::EnumMaxImpl< PaddingMode >	102
nvinfer1::impl::EnumMaxImpl< PoolingType >	103
nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation >	103
nvinfer1::impl::EnumMaxImpl< ResizeMode >	104
nvinfer1::impl::EnumMaxImpl< ResizeRoundMode >	105
nvinfer1::impl::EnumMaxImpl< ResizeSelector >	105
nvinfer1::impl::EnumMaxImpl< TensorFormat >	
Maximum number of elements in TensorFormat enum	106
nvinfer1::impl::EnumMaxImpl< TensorLocation >	
Maximum number of elements in TensorLocation enum	107
nvuffparser::FieldCollection	108
nvuffparser::FieldMap	
An array of field params used as a layer parameter for plugin layers	108
nvinfer1::safe::FloatingPointErrorInformation	
Space to record information about floating point runtime errors	110
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:	111
nvinfer1::IActivationLayer	
An Activation layer in a network definition	113
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() .”	117
nvinfer1::IAlgorithmContext	
Describes the context and requirements, that could be fulfilled by one or more instances of IAlgorithm	120
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()	122
nvinfer1::IAlgorithmSelector	
Interface implemented by application for selecting and reporting algorithms of a layer provided by the builder	124
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()	126
nvinfer1::IAssertionLayer	
An assertion layer in a network	128
nvcaffeparser1::IBinaryProtoBlob	
Object used to store and query data extracted from a binaryproto file using the ICaffeParser	130
nvcaffeparser1::IBlobNameToTensor	
Object used to store and query Tensors after they have been extracted from a Caffe model using the ICaffeParser	132
nvinfer1::IBuilder	
Builds an engine from a network definition	133
nvinfer1::IBuilderConfig	
Holds properties for configuring a builder to produce an engine	143
nvcaffeparser1::ICaffeParser	
Class used for parsing Caffe models	164

nvinfer1::IConcatenationLayer	
A concatenation layer in a network definition	169
nvinfer1::IConditionLayer	172
nvinfer1::consistency::IConsistencyChecker	
Validates a serialized engine blob	173
nvinfer1::IConstantLayer	
Layer that represents a constant value	175
nvinfer1::IConvolutionLayer	
A convolution layer in a network definition	178
nvinfer1::ICudaEngine	
An engine for executing inference on a built network, with functionally unsafe features	190
nvinfer1::safe::ICudaEngine	
A functionally safe engine for executing inference on a built network	206
nvinfer1::IDeconvolutionLayer	
A deconvolution layer in a network definition	217
nvinfer1::IDequantizeLayer	
A Dequantize layer in a network definition	229
nvinfer1::IDimensionExpr	231
nvinfer1::IEinsumLayer	
An Einsum layer in a network	233
nvinfer1::IElementWiseLayer	
A elementwise layer in a network definition	236
nvinfer1::IEngineInspector	
An engine inspector which prints out the layer information of an engine or an execution context	238
nvinfer1::IErrorRecorder	
Reference counted application-implemented error reporting interface for TensorRT objects	243
nvinfer1::IExecutionContext	
Context for executing inference using an engine, with functionally unsafe features	249
nvinfer1::safe::IExecutionContext	
Functionally safe context for executing inference using an engine	266
nvinfer1::IExprBuilder	273
nvinfer1::IFillLayer	
Generate an output tensor with specified mode	275
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:	281
nvinfer1::IGatherLayer	
A Gather layer in a network definition. Supports several kinds of gathering	286
nvinfer1::IGpuAllocator	
Application-implemented class for controlling allocation on the GPU	291
nvinfer1::IHostMemory	
Class to handle library allocated memory that is accessible to the user	295
nvinfer1::IIdentityLayer	
A layer that represents the identity function	297
nvinfer1::IIfConditional	299
nvinfer1::IIfConditionalBoundaryLayer	302
nvinfer1::IIfConditionalInputLayer	304
nvinfer1::IIfConditionalOutputLayer	305
nvinfer1::IInt8Calibrator	
Application-implemented interface for calibration	306
nvinfer1::IInt8EntropyCalibrator	309
nvinfer1::IInt8EntropyCalibrator2	310

nvinfer1::Int8LegacyCalibrator	311
nvinfer1::Int8MinMaxCalibrator	314
nvinfer1::IteratorLayer	315
nvinfer1::ILayer	
Base class for all layer classes in a network definition	317
nvinfer1::ILogger	
Application-implemented logging interface for the builder, refitter and runtime	325
nvinfer1::ILoop	328
nvinfer1::ILoopBoundaryLayer	331
nvinfer1::ILoopOutputLayer	332
nvinfer1::ILRNLayer	
A LRN layer in a network definition	335
nvinfer1::IMatrixMultiplyLayer	
Layer that represents a Matrix Multiplication	340
nvinfer1::INetworkDefinition	
A network definition for input to the builder	342
nvinfer1::INoCopy	
Forward declaration of IEngineInspector for use by other interfaces	384
nvonnxparser::IOnnxConfig	
Configuration Manager Class	386
nvinfer1::IOptimizationProfile	
Optimization profile for dynamic input dimensions and shape tensors	393
nvinfer1::IPaddingLayer	
Layer that represents a padding operation	398
nvinfer1::IParametricReLULayer	
Layer that represents a parametric ReLU operation	403
nvonnxparser::IParser	
Object for parsing ONNX models into a TensorRT network definition	404
nvonnxparser::IParserError	
Object containing information about an error	408
nvinfer1::consistency::IPluginChecker	
Consistency Checker plugin class for user implemented Plugins	411
nvinfer1::IPluginCreator	
Plugin creator class for user implemented layers	413
nvcaffeparser1::IPluginFactoryV2	
Plugin factory used to configure plugins	418
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	420
nvinfer1::IPluginV2	
Plugin class for user-implemented layers	424
nvinfer1::IPluginV2DynamicExt	435
nvinfer1::IPluginV2Ext	
Plugin class for user-implemented layers	441
nvinfer1::IPluginV2IOExt	
Plugin class for user-implemented layers	447
nvinfer1::IPluginV2Layer	
Layer type for pluginV2	450
nvinfer1::IPoolingLayer	
A Pooling layer in a network definition	452

nvinfer1::IProfiler	Application-implemented interface for profiling	462
nvinfer1::IQuantizeLayer	A Quantize layer in a network definition	464
nvinfer1::IRaggedSoftMaxLayer	A RaggedSoftmax layer in a network definition	467
nvinfer1::IRecurrenceLayer	468
nvinfer1::IReduceLayer	Layer that represents a reduction across a non-bool tensor	470
nvinfer1::IRefitter	Updates weights in an engine	473
nvinfer1::IResizeLayer	A resize layer in a network definition	481
nvinfer1::IRNNv2Layer	An RNN layer in a network definition, version 2	490
nvinfer1::IRuntime	Allows a serialized functionally unsafe engine to be deserialized	498
nvinfer1::safe::IRuntime	Allows a serialized functionally safe engine to be deserialized	504
nvinfer1::IScaleLayer	A Scale layer in a network definition	509
nvinfer1::IScatterLayer	A scatter layer in a network definition. Supports several kinds of scattering	514
nvinfer1::ISelectLayer	517
nvinfer1::IShapeLayer	Layer type for getting shape of a tensor	518
nvinfer1::IShuffleLayer	Layer type for shuffling data	520
nvinfer1::ISliceLayer	Slices an input tensor into an output tensor based on the offset and strides	525
nvinfer1::ISoftMaxLayer	A Softmax layer in a network definition	531
nvinfer1::ITensor	A tensor in a network definition	533
nvinfer1::ITimingCache	Class to handle tactic timing info collected from builder	543
nvinfer1::ITopKLayer	Layer that represents a TopK reduction	546
nvinfer1::ITripLimitLayer	549
nvuffparser::IuffParser	Class used for parsing models described using the UFF format	550
nvinfer1::IUnaryLayer	Layer that represents an unary operation	555
nvinfer1::plugin::NMSPParameters	The <code>NMSPParameters</code> are used by the <code>BatchedNMSPPlugin</code> for performing the <code>non_max_suppression</code> operation over boxes for object detection networks	557
nvinfer1::Permutation	559
nvinfer1::PluginField	Structure containing plugin attribute field names and associated data This information can be parsed to decode necessary plugin metadata	560
nvinfer1::PluginFieldCollection	Plugin field collection struct	561

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	562
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	563
nvinfer1::PluginTensorDesc	Fields that a plugin might see for an input or output	564
PluginVersion	Definition of plugin versions	565
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:	566
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	569
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	570
nvinfer1::plugin::RPROIParams	RPROIParams is used to create the RPROIPlugin instance. It contains:	571
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	573
nvinfer1::Weights	An array of weights used as a layer parameter	575

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

NvCaffeParser.h	577
NvInfer.h	579
NvInferConsistency.h	626
NvInferLegacyDims.h	628
NvInferPlugin.h	630
NvInferPluginUtils.h	636
NvInferRuntime.h	639
NvInferRuntimeCommon.h	655
NvInferSafeRuntime.h	666
NvInferVersion.h	670
NvOnnxConfig.h	673
NvUffParser.h	674
NvUtils.h	677
NvOnnxParser.h	678

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::select←Algorithms()`.
- 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 [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 [IHostMemory](#)
 - Class to handle library allocated memory that is accessible to the user.*
- class [IIdentityLayer](#)
 - A layer that represents the identity function.*
- class [IIfConditional](#)
- class [IIfConditionalBoundaryLayer](#)
- class [IIfConditionalInputLayer](#)
- class [IIfConditionalOutputLayer](#)
- class [IInt8Calibrator](#)
 - Application-implemented interface for calibration.*
- class [IInt8EntropyCalibrator](#)
- class [IInt8EntropyCalibrator2](#)
- class [IInt8LegacyCalibrator](#)
- class [IInt8MinMaxCalibrator](#)
- class [IIteratorLayer](#)
- class [ILayer](#)
 - Base class for all layer classes in a network definition.*
- class [ILogger](#)
 - Application-implemented logging interface for the builder, refitter and runtime.*
- class [ILoop](#)
- class [ILoopBoundaryLayer](#)
- class [ILoopOutputLayer](#)
- class [ILRNLayer](#)
 - A LRN layer in a network definition.*
- class [IMatrixMultiplyLayer](#)
 - Layer that represents a Matrix Multiplication.*

- class [INetworkDefinition](#)
A network definition for input to the builder.
- class [INoCopy](#)
Forward declaration of [IEngineInspector](#) for use by other interfaces.
- class [IOptimizationProfile](#)
Optimization profile for dynamic input dimensions and shape tensors.
- class [IPaddingLayer](#)
Layer that represents a padding operation.
- class [IParametricReLULayer](#)
Layer that represents a parametric ReLU operation.
- class [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 [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., $1U \ll \text{TensorFormat::kCHW4} \mid 1U \ll \text{TensorFormat::kCHW32}$.*
- using [QuantizationFlags](#) = uint32_t
 - Represents one or more QuantizationFlag values using binary OR operations.*
- using [BuilderFlags](#) = uint32_t
 - Represents one or more QuantizationFlag values using binary OR operations, e.g., $1U \ll \text{BuilderFlag::kFP16} \mid 1U \ll \text{BuilderFlag::kDEBUG}$.*
- using [NetworkDefinitionCreationFlags](#) = uint32_t
 - Represents one or more NetworkDefinitionCreationFlag flags using binary OR operations. e.g., $1U \ll \text{NetworkDefinitionCreationFlag::kEXPLICIT_BATCH}$.*
- 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`
AsciiChar is the type used by TensorRT to represent valid ASCII characters.
- using `Dims` = `Dims32`
- using `PluginFormat` = `TensorFormat`
PluginFormat is reserved for backward compatibility.
- using `AllocatorFlags` = `uint32_t`

Enumerations

- enum class `LayerType` : `int32_t` {
`kCONVOLUTION` = 0 , `kFULLY_CONNECTED` = 1 , `kACTIVATION` = 2 , `kPOOLING` = 3 ,
`kLRN` = 4 , `kSCALE` = 5 , `kSOFTMAX` = 6 , `kDECONVOLUTION` = 7 ,
`kCONCATENATION` = 8 , `KELEMENTWISE` = 9 , `kPLUGIN` = 10 , `kUNARY` = 11 ,
`kPADDING` = 12 , `kSHUFFLE` = 13 , `kREDUCE` = 14 , `kTOPK` = 15 ,
`kGATHER` = 16 , `kMATRIX_MULTIPLY` = 17 , `kRAGGED_SOFTMAX` = 18 , `kCONSTANT` = 19 ,
`krnn_v2` = 20 , `kIDENTITY` = 21 , `kPLUGIN_V2` = 22 , `kSLICE` = 23 ,
`kSHAPE` = 24 , `kPARAMETRIC_RELU` = 25 , `kRESIZE` = 26 , `kTRIP_LIMIT` = 27 ,
`kRECURRENCE` = 28 , `kITERATOR` = 29 , `kLOOP_OUTPUT` = 30 , `kSELECT` = 31 ,
`kFILL` = 32 , `kQUANTIZE` = 33 , `kDEQUANTIZE` = 34 , `kCONDITION` = 35 ,
`kCONDITIONAL_INPUT` = 36 , `kCONDITIONAL_OUTPUT` = 37 , `kSCATTER` = 38 , `keINSUM` = 39 ,
`kASSERTION` = 40 }
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 }
- 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 [SliceMode](#) : int32_t {
[kDEFAULT](#) = 0 , [kWRAP](#) = 1 , [kCLAMP](#) = 2 , [kFILL](#) = 3 ,
[kREFLECT](#) = 4 }
- Controls how ISliceLayer handles 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 [ResizeMode](#) : int32_t { [kNEAREST](#) = 0 , [kLINEAR](#) = 1 }
- Enumerates various modes of resize in the resize layer. Resize mode set using setResizeMode().*
- 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 }
- 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 [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 }

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 }

The type for memory pools used by TensorRT.

- 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 **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 }

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 }

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 }
- *Format of the input/output tensors.*
- enum class `PluginVersion` : `uint8_t` { `kV2` = 0 , `kV2_EXT` = 1 , `kV2_IOEXT` = 2 , `kV2_DYNAMICEXT` = 3 }
- enum class `PluginFieldType` : `int32_t` {
`kFLOAT16` = 0 , `kFLOAT32` = 1 , `kFLOAT64` = 2 , `kINT8` = 3 ,
`kINT16` = 4 , `kINT32` = 5 , `kCHAR` = 6 , `kDIMS` = 7 ,
`kUNKNOWN` = 8 }
- enum class `AllocatorFlag` : `int32_t` { `kRESIZABLE` = 0 }
- enum class `ErrorCode` : `int32_t` {
`kSUCCESS` = 0 , `kUNSPECIFIED_ERROR` = 1 , `kINTERNAL_ERROR` = 2 , `kINVALID_ARGUMENT` = 3 ,
`kINVALID_CONFIG` = 4 , `kFAILED_ALLOCATION` = 5 , `kFAILED_INITIALIZATION` = 6 , `kFAILED_EXECUTION`
= 7 ,
`kFAILED_COMPUTATION` = 8 , `kINVALID_STATE` = 9 , `kUNSUPPORTED_STATE` = 10 }
- *Error codes that can be returned by TensorRT during execution.*

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< SliceMode > () 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< CalibrationAlgoType > () noexcept`
- `template<> constexpr int32_t EnumMax< QuantizationFlag > () noexcept`
- `template<> constexpr int32_t EnumMax< BuilderFlag > () noexcept`
- `template<> constexpr int32_t EnumMax< MemoryPoolType > () noexcept`
- `template<> constexpr int32_t EnumMax< NetworkDefinitionCreationFlag > () noexcept`
- `nvinfer1::IPluginRegistry * getBuilderPluginRegistry (nvinfer1::EngineCapability capability) noexcept`
Return the plugin registry for the given capability or nullptr if no registry exists.
- `template<> constexpr int32_t EnumMax< DimensionOperation > () noexcept`
Maximum number of elements in DimensionOperation enum.
- `template<> constexpr int32_t EnumMax< WeightsRole > () noexcept`
Maximum number of elements in WeightsRole enum.
- `template<> constexpr int32_t EnumMax< DeviceType > () noexcept`
Maximum number of elements in DeviceType enum.

- `template<> constexpr int32_t EnumMax< 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.

8.2.2.3 BuilderFlags

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

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

See also

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

8.2.2.4 char_t

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

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

8.2.2.5 Dims

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

Alias for [Dims32](#).

Warning

: This alias might change in the future.

8.2.2.6 NetworkDefinitionCreationFlags

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

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

See also

[IBuilder::createNetworkV2](#)

8.2.2.7 PluginFormat

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

PluginFormat is reserved for backward compatibility.

See also

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

8.2.2.8 QuantizationFlags

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

Represents one or more QuantizationFlag values using binary OR operations.

See also

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

8.2.2.9 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\(\)](#)

8.2.2.10 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 BuilderFlag

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

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

See also

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

Enumerator

kFP16	Enable FP16 layer selection, with FP32 fallback.
kINT8	Enable Int8 layer selection, with FP32 fallback with FP16 fallback if kFP16 also specified.
kDEBUG	Enable debugging of layers via synchronizing after every layer.

Enumerator

kGPU_FALLBACK	Enable layers marked to execute on GPU if layer cannot execute on DLA.
kSTRICT_TYPES	Legacy flag with effect similar to setting all of these three flags: <ul style="list-style-type: none"> * kPREFER_PRECISION_CONSTRAINTS * kDIRECT_IO * kREJECT_EMPTY_ALGORITHMS <p>except that if the direct I/O requirement cannot be met and kDIRECT_IO was not set, instead of the build failing, the build falls back as if kDIRECT_IO was not set.</p> <p>\deprecated Deprecatd in TensorRT 8.2</p>
kREFIT	Enable building a refittable engine.
kDISABLE_TIMING_CACHE	Disable reuse of timing information across identical layers.
kTF32	Allow (but not require) computations on tensors of type DataType::kFLOAT to use TF32. TF32 computes inner products by rounding the inputs to 10-bit mantissas before multiplying, but accumulates the sum using 23-bit mantissas. Enabled by default.
kSPARSE_WEIGHTS	Allow the builder to examine weights and use optimized functions when weights have suitable sparsity.
kSAFETY_SCOPE	Change the allowed parameters in the EngineCapability::kSTANDARD flow to match the restrictions that EngineCapability::kSAFETY check against for DeviceType::kGPU and EngineCapability::kDLA_STANDALONE check against the DeviceType::kDLA case. This flag is forced to true if EngineCapability::kSAFETY at build time if it is unset. This flag is only supported in NVIDIA Drive(R) products.
kOBEY_PRECISION_CONSTRAINTS	Require that layers execute in specified precisions. Build fails otherwise.
kPREFER_PRECISION_CONSTRAINTS	Prefer that layers execute in specified precisions. Fall back (with warning) to another precision if build would otherwise fail.
kDIRECT_IO	Require that no reformats be inserted between a layer and a network I/O tensor for which ITensor::setAllowedFormats was called. Build fails if a reformat is required for functional correctness.
kREJECT_EMPTY_ALGORITHMS	Fail if IAlgorithmSelector::selectAlgorithms returns an empty set of algorithms.

8.2.3.4 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	

8.2.3.5 DataType

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

The type of weights and tensors.

Enumerator

kFLOAT	32-bit floating point format.
kHALF	IEEE 16-bit floating-point format.
kINT8	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.

8.2.3.6 DeviceType

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

The device that this layer/network will execute on.

Enumerator

kGPU	GPU Device.
kDLA	DLA Core.

8.2.3.7 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.

8.2.3.8 ElementWiseOperation

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

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

Operations kAND, kOR, and kXOR must have inputs of [DataType](#) kBOOL.

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

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

See also

[IElementWiseLayer](#)

Enumerator

kSUM	Sum of the two elements.
kPROD	Product of the two elements.
kMAX	Maximum of the two elements.
kMIN	Minimum of the two elements.
kSUB	Subtract the second element from the first.
kDIV	Divide the first element by the second.

Enumerator

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.

8.2.3.9 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 [sample← Nvmedia](#) for an example of integrating NvMediaDLA 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 Use kSTANDARD. Deprecated in TensorRT 8.0.
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 Use kSAFETY. Deprecated in TensorRT 8.0.
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 Use kDLA_STANDALONE. Deprecated in TensorRT 8.0.

8.2.3.10 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	<p>An error occurred due to the network not being supported on the device due to constraints of the hardware or system. An example is running a unsafe layer in a safety certified context, or a resource requirement for the current network is greater than the capabilities of the target device. The network is otherwise correct, but the network and hardware combination is problematic. This can be recoverable. Examples:</p> <ul style="list-style-type: none"> • Scratch space requests larger than available device memory and can be recovered by increasing allowed workspace size. • Tensor size exceeds the maximum element count and can be recovered by reducing the maximum batch size.
--------------------	--

8.2.3.11 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.

8.2.3.12 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.

8.2.3.13 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.

8.2.3.14 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.

Enumerator

kDECONVOLUTION	Deconvolution layer.
kCONCATENATION	Concatenation layer.
kELEMENTWISE	Elementwise layer.
kPLUGIN	Plugin layer.
kUNARY	UnaryOp operation Layer.
kPADDING	Padding layer.
kSHUFFLE	Shuffle layer.
kREDUCE	Reduce layer.
kTOPK	TopK layer.
kGATHER	Gather layer.
kMATRIX_MULTIPLY	Matrix multiply layer.
kRAGGED_SOFTMAX	Ragged softmax layer.
kCONSTANT	Constant layer.
kRNN_V2	RNNv2 layer.
kIDENTITY	Identity layer.
kPLUGIN_V2	PluginV2 layer.
kSLICE	Slice layer.
kSHAPE	Shape layer.
kPARAMETRIC_RELU	Parametric ReLU layer.
kRESIZE	Resize Layer.
kTRIP_LIMIT	Loop Trip limit layer.
kRECURRENCE	Loop Recurrence layer.
kITERATOR	Loop Iterator layer.
kLOOP_OUTPUT	Loop output layer.
kSELECT	Select layer.
kFILL	Fill layer.
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.

8.2.3.15 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.

8.2.3.16 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 .

8.2.3.17 MemoryPoolType

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

The type for memory pools used by TensorRT.

See also

[IBuilderConfig::setMemoryPoolLimit](#), [IBuilderConfig::getMemoryPoolLimit](#)

Enumerator

kWORKSPACE	kWORKSPACE is used by TensorRT to store intermediate buffers within an operation. This is equivalent to the deprecated IBuilderConfig::setMaxWorkspaceSize and overrides that value. This defaults to max device memory. Set to a smaller value to restrict tactics that use over the threshold en masse. For more targeted removal of tactics use the IAlgorithmSelector interface.
kDLA_MANAGED_SRAM	kDLA_MANAGED_SRAM is a fast software managed RAM used by DLA to communicate within a layer. The size of this pool must be at least 4 KiB and must be a power of 2. This defaults to 1 MiB. Orin has capacity of 1 MiB per core, and Xavier shares 4 MiB across all of its accelerator cores.
kDLA_LOCAL_DRAM	kDLA_LOCAL_DRAM is host RAM used by DLA to share intermediate tensor data across operations. The size of this pool must be at least 4 KiB and must be a power of 2. This defaults to 1 GiB.
kDLA_GLOBAL_DRAM	kDLA_GLOBAL_DRAM is host RAM used by DLA to store weights and metadata for execution. The size of this pool must be at least 4 KiB and must be a power of 2. This defaults to 512 MiB.

8.2.3.18 NetworkDefinitionCreationFlag

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

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

See also

[IBuilder::createNetworkV2](#)

Enumerator

kEXPLICIT_BATCH	Mark the network to be an explicit batch network. Dynamic shape support requires that the <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.
kEXPLICIT_PRECISION	Deprecated. This flag has no effect now, but is only kept for backward compatibility.

8.2.3.19 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.

8.2.3.20 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:**
 $\text{EXPLICIT_ROUND_DOWN} - ((\text{EXPLICIT_ROUND_DOWN} - 1) * S >= I + B)$
- **CAFFE_ROUND_UP:**
 $\text{EXPLICIT_ROUND_UP} - ((\text{EXPLICIT_ROUND_UP} - 1) * S >= I + B)$

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:
 $M = \{6, 6\} + \{3, 3\} + \{2, 2\} ==> \{11, 11\}$
 $O ==> \text{floor}((M - F) / S) + 1$
 $==> \text{floor}((\{11, 11\} - \{3, 3\}) / \{2, 2\}) + \{1, 1\}$
 $==> \text{floor}(\{8, 8\} / \{2, 2\}) + \{1, 1\}$
 $==> \{5, 5\}$

- **EXPLICIT_ROUND_UP:**

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

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

- **SAME_UPPER:**

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

- **SAME_LOWER:**

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

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

Pooling Example 2:

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

- **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 <code>prePadding</code> \leq <code>postPadding</code> .
kSAME_LOWER	Use SAME padding, with <code>prePadding</code> \geq <code>postPadding</code> .
kCAFFE_ROUND_DOWN	Use CAFFE padding, rounding output size down, uses <code>prePadding</code> value.
kCAFFE_ROUND_UP	Use CAFFE padding, rounding output size up, uses <code>prePadding</code> value.

8.2.3.21 PluginFieldType

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

Enumerator

kFLOAT16	FP16 field type.
kFLOAT32	FP32 field type.
kFLOAT64	FP64 field type.
kINT8	INT8 field type.
kINT16	INT16 field type.
kINT32	INT32 field type.
kCHAR	char field type.
kDIMS	nvinfer1::Dims field type.
kUNKNOWN	Unknown field type.

8.2.3.22 PluginVersion

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

Enumerator

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

8.2.3.23 PoolingType

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

The type of pooling to perform in a pooling layer.

Enumerator

kMAX	
kAVERAGE	
kMAX_AVERAGE_BLEND	

8.2.3.24 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 Use kLAYER_NAMES_ONLY. Deprecated in TensorRT 8.0.
kVERBOSE	Deprecated Use kDETAILED. Deprecated in TensorRT 8.0.

8.2.3.25 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.
--------------------------	---

8.2.3.26 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	

8.2.3.27 ResizeCoordinateTransformation

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

The resize coordinate transformation function.

See also

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

Enumerator

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

8.2.3.28 ResizeMode

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

Enumerates various modes of resize in the resize layer. Resize mode set using `setResizeMode()`.

Enumerator

kNEAREST	ND (0 < N <= 8) nearest neighbor resizing.
kLINEAR	Can handle linear (1D), bilinear (2D), and trilinear (3D) resizing.

8.2.3.29 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.

8.2.3.30 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.

8.2.3.31 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.

8.2.3.32 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).

8.2.3.33 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.

8.2.3.34 RNNOperation

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

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

Equation definitions

The equations below have the following naming convention:

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

Equations

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

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

See also

[IRNNv2Layer](#)

Enumerator

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

8.2.3.35 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.

8.2.3.36 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.

8.2.3.37 SliceMode

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

Controls how [ISliceLayer](#) handles out of bounds coordinates.

See also

[ISliceLayer](#)

Enumerator

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

8.2.3.38 `TacticSource`

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

List of tactic sources for TensorRT.

See also

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

Enumerator

kCUBLAS	cuBLAS tactics. Note Disabling kCUBLAS will cause the cublas handle passed to plugins in <code>attachToContext</code> to be null.
kCUBLAS_LT	cuBLAS LT tactics
kCUDNN	cuDNN tactics
kEDGE_MASK_CONVOLUTIONS	Enables convolution tactics implemented with edge mask tables. These tactics tradeoff memory for performance by consuming additional memory space proportional to the input size.

8.2.3.39 `TensorFormat`

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

Format of the input/output tensors.

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

See also

[IPluginV2::supportsFormat\(\)](#), [safe::ICudaEngine::getBindingFormat\(\)](#)

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

Enumerator

kLINEAR	Row major linear format. For a tensor with dimensions $\{N, C, H, W\}$ or $\{\text{numbers, channels, columns, rows}\}$, the dimensional index corresponds to $\{3, 2, 1, 0\}$ and thus the order is W minor. For DLA usage, the tensor sizes are limited to C,H,W in the range $[1,8192]$.
kCHW2	Two wide channel vectorized row major format. This format is bound to FP16. It is only available for dimensions ≥ 3 . For a tensor with dimensions $\{N, C, H, W\}$, the memory layout is equivalent to a C array with dimensions $[N][(C+1)/2][H][W][2]$, with the tensor coordinates (n, c, h, w) mapping to array subscript $[n][c/2][h][w][c\%2]$.
kHWC8	Eight channel format where C is padded to a multiple of 8. This format is bound to FP16. It is only available for dimensions ≥ 3 . For a tensor with dimensions $\{N, C, H, W\}$, the memory layout is equivalent to the array with dimensions $[N][H][W][(C+7)/8*8]$, with the tensor coordinates (n, c, h, w) mapping to array subscript $[n][h][w][c]$.
kCHW4	Four wide channel vectorized row major format. This format is bound to INT8 or FP16. It is only available for dimensions ≥ 3 . For INT8, the C dimension must be a build-time constant. For a tensor with dimensions $\{N, C, H, W\}$, the memory layout is equivalent to a C array with dimensions $[N][(C+3)/4][H][W][4]$, with the tensor coordinates (n, c, h, w) mapping to array subscript $[n][c/4][h][w][c\%4]$. Deprecated usage: If running on the DLA, this format can be used for acceleration with the caveat that C must be equal or lesser than 4. If used as DLA input and the build option kGPU_FALLBACK is not specified, it needs to meet line stride requirement of DLA format. Column stride in bytes should be a multiple of 32 on Xavier and 64 on Orin.
kCHW16	Sixteen wide channel vectorized row major format. This format is bound to FP16. It is only available for dimensions ≥ 3 . For a tensor with dimensions $\{N, C, H, W\}$, the memory layout is equivalent to a C array with dimensions $[N][(C+15)/16][H][W][16]$, with the tensor coordinates (n, c, h, w) mapping to array subscript $[n][c/16][h][w][c\%16]$. For DLA usage, this format maps to the native image 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 image format for INT8, and the tensor sizes are limited to C,H,W in the range $[1,8192]$.
kDHWC8	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 FP32 and is only available for dimensions ≥ 3 .

Enumerator

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 $[n][h][w][c]$.

8.2.3.40 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.

8.2.3.41 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.

8.2.3.42 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.

8.2.3.43 UnaryOperation

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

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

Operations kNOT must have inputs of [DataType](#) kBOOL.

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

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

See also

[IUnaryLayer](#)

Enumerator

kEXP	Exponentiation.
kLOG	Log (base e).
kSQRT	Square root.
kRECIP	Reciprocal.
kABS	Absolute value.
kNEG	Negation.
kSIN	Sine.
kCOS	Cosine.
kTAN	Tangent.
kSINH	Hyperbolic sine.
kCOSH	Hyperbolic cosine.
kASIN	Inverse sine.
kACOS	Inverse cosine.

Enumerator

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 float datatype.

8.2.3.44 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< BuilderFlag >()

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

Maximum number of builder flags in BuilderFlag enum.

See also

[BuilderFlag](#)

8.2.4.3 EnumMax< CalibrationAlgoType >()

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

Maximum number of elements in CalibrationAlgoType enum.

See also

[DataType](#)

8.2.4.4 EnumMax< DeviceType >()

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

Maximum number of elements in DeviceType enum.

See also

[DeviceType](#)

8.2.4.5 EnumMax< DimensionOperation >()

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

Maximum number of elements in DimensionOperation enum.

See also

[DimensionOperation](#)

8.2.4.6 EnumMax< FillOperation >()

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

Maximum number of elements in FillOperation enum.

See also

[FillOperation](#)

8.2.4.7 EnumMax< GatherMode >()

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

Maximum number of elements in GatherMode enum.

See also

[GatherMode](#)

8.2.4.8 EnumMax< LayerInformationFormat >()

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

Maximum number of layer information formats in LayerInformationFormat enum.

See also

[LayerInformationFormat](#)

8.2.4.9 EnumMax< LayerType >()

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

Maximum number of elements in LayerType enum.

See also

[LayerType](#)

8.2.4.10 EnumMax< LoopOutput >()

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

Maximum number of elements in LoopOutput enum.

See also

[DataType](#)

8.2.4.11 EnumMax< MatrixOperation >()

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

Maximum number of elements in MatrixOperation enum.

See also

[DataType](#)

8.2.4.12 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](#)

8.2.4.13 EnumMax< NetworkDefinitionCreationFlag >()

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

Maximum number of elements in NetworkDefinitionCreationFlag enum.

See also

[NetworkDefinitionCreationFlag](#)

8.2.4.14 EnumMax< OptProfileSelector >()

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

Number of different values of OptProfileSelector enum.

See also

[OptProfileSelector](#)

8.2.4.15 EnumMax< ProfilingVerbosity >()

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

Maximum number of profile verbosity levels in ProfilingVerbosity enum.

See also

[ProfilingVerbosity](#)

8.2.4.16 EnumMax< QuantizationFlag >()

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

Maximum number of quantization flags in QuantizationFlag enum.

See also

[QuantizationFlag](#)

8.2.4.17 EnumMax< ReduceOperation >()

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

Maximum number of elements in ReduceOperation enum.

See also

[ReduceOperation](#)

8.2.4.18 EnumMax< RNNDirection >()

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

Maximum number of elements in RNNDirection enum.

See also

[RNNDirection](#)

8.2.4.19 EnumMax< RNNGateType >()

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

Maximum number of elements in RNNGateType enum.

See also

[RNNGateType](#)

8.2.4.20 EnumMax< RNNInputMode >()

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

Maximum number of elements in RNNInputMode enum.

See also

[RNNInputMode](#)

8.2.4.21 EnumMax< RNNOperation >()

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

Maximum number of elements in RNNOperation enum.

See also

[RNNOperation](#)

8.2.4.22 EnumMax< ScaleMode >()

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

Maximum number of elements in ScaleMode enum.

See also

[ScaleMode](#)

8.2.4.23 EnumMax< ScatterMode >()

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

Maximum number of elements in ScatterMode enum.

See also

[ScatterMode](#)

8.2.4.24 EnumMax< SliceMode >()

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

Maximum number of elements in SliceMode enum.

See also

[SliceMode](#)

8.2.4.25 EnumMax< TacticSource >()

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

Maximum number of tactic sources in TacticSource enum.

See also

[TacticSource](#)

8.2.4.26 EnumMax< TopKOperation >()

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

Maximum number of elements in TopKOperation enum.

See also

[TopKOperation](#)

8.2.4.27 EnumMax< TripLimit >()

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

Maximum number of elements in TripLimit enum.

See also

[DataType](#)

8.2.4.28 EnumMax< UnaryOperation >()

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

Maximum number of elements in UnaryOperation enum.

See also

[UnaryOperation](#)

8.2.4.29 EnumMax< WeightsRole >()

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

Maximum number of elements in WeightsRole enum.

See also

[WeightsRole](#)

8.2.4.30 `getBuilderPluginRegistry()`

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

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

Engine capabilities `EngineCapability::kSTANDARD` and `EngineCapability::kSAFETY` have distinct plugin registries. Use `IPluginRegistry::registerCreator` from the registry to register plugins. Plugins registered in a registry associated with a specific engine capability are only available when building engines with that engine capability.

There is no plugin registry for `EngineCapability::kDLA_STANDALONE`.

8.3 `nvinfer1::consistency` Namespace Reference

Classes

- class `IConsistencyChecker`
Validates a serialized engine blob.
- class `IPluginChecker`
Consistency Checker plugin class for user implemented Plugins.

8.4 `nvinfer1::impl` Namespace Reference

Classes

- struct `EnumMaxImpl`
Declaration of `EnumMaxImpl` struct to store maximum number of elements in an enumeration type.
- struct `EnumMaxImpl< ActivationType >`
- struct `EnumMaxImpl< AllocatorFlag >`
Maximum number of elements in `AllocatorFlag` enum.
- struct `EnumMaxImpl< DataType >`
Maximum number of elements in `DataType` enum.
- struct `EnumMaxImpl< ElementWiseOperation >`
- struct `EnumMaxImpl< EngineCapability >`
Maximum number of elements in `EngineCapability` enum.
- struct `EnumMaxImpl< ErrorCode >`
Maximum number of elements in `ErrorCode` enum.
- struct `EnumMaxImpl< ILogger::Severity >`
Maximum number of elements in `ILogger::Severity` enum.
- struct `EnumMaxImpl< PaddingMode >`
- struct `EnumMaxImpl< PoolingType >`
- struct `EnumMaxImpl< ResizeCoordinateTransformation >`
- struct `EnumMaxImpl< ResizeMode >`
- struct `EnumMaxImpl< ResizeRoundMode >`
- struct `EnumMaxImpl< ResizeSelector >`
- struct `EnumMaxImpl< TensorFormat >`
Maximum number of elements in `TensorFormat` enum.
- struct `EnumMaxImpl< TensorLocation >`
Maximum number of elements in `TensorLocation` enum.

8.5 nvinfer1::plugin Namespace Reference

Classes

- struct [DetectionOutputParameters](#)
The DetectionOutput plugin layer generates the detection output based on location and confidence predictions by doing non maximum suppression. This plugin first decodes the bounding boxes based on the anchors generated. It then performs non_max_suppression on the decoded bounding boxes. [DetectionOutputParameters](#) defines a set of parameters for creating the DetectionOutput plugin layer. It contains:
- struct [GridAnchorParameters](#)
The Anchor Generator plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions (H x W). [GridAnchorParameters](#) defines a set of parameters for creating the plugin layer for all feature maps. It contains:
- struct [NMSPParameters](#)
The [NMSPParameters](#) are used by the BatchedNMSPPlugin for performing the non_max_suppression operation over boxes for object detection networks.
- struct [PriorBoxParameters](#)
The PriorBox plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions (H x W). [PriorBoxParameters](#) defines a set of parameters for creating the PriorBox plugin layer. It contains:
- struct [Quadruple](#)
The Permute plugin layer permutes the input tensor by changing the memory order of the data. [Quadruple](#) defines a structure that contains an array of 4 integers. They can represent the permute orders or the strides in each dimension.
- struct [RegionParameters](#)
The Region plugin layer performs region proposal calculation: generate 5 bounding boxes per cell (for yolo9000, generate 3 bounding boxes per cell). For each box, calculating its probabilities of objects detections from 80 pre-defined classifications (yolo9000 has 9418 pre-defined classifications, and these 9418 items are organized as work-tree structure). [RegionParameters](#) defines a set of parameters for creating the Region plugin layer.
- struct [RPROIParams](#)
[RPROIParams](#) is used to create the RPROIPlugin instance. It contains:
- struct [softmaxTree](#)
When performing yolo9000, [softmaxTree](#) is helping to do softmax on confidence scores, for element to get the precise classification through word-tree structured classification definition.

Enumerations

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

8.5.1 Enumeration Type Documentation

8.5.1.1 CodeTypeSSD

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

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

Enumerator

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

8.6 nvinfer1::safe Namespace Reference

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

Classes

- struct [FloatingPointErrorInformation](#)
Space to record information about floating point runtime errors.
- class [ICudaEngine](#)
A functionally safe engine for executing inference on a built network.
- class [IExecutionContext](#)
Functionally safe context for executing inference using an engine.
- class [IRuntime](#)
Allows a serialized functionally safe engine to be deserialized.
- class [PluginRegistrar](#)
Register the plugin creator to the registry The static registry object will be instantiated when the plugin library is loaded. This static object will register all creators available in the library to the registry.

Functions

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

8.6.1 Detailed Description

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

8.6.2 Function Documentation

8.6.2.1 createInferRuntime()

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

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

This class is the logging class for the runtime.

Usage considerations

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

8.6.2.2 getSafePluginRegistry()

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

Return the safe plugin registry.

Usage considerations

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

8.7 nvinfer1::utils Namespace Reference

Functions

- **TRT_DEPRECATED** bool [reshapeWeights](#) ([Weights](#) const &input, int32_t const *shape, int32_t const *shape←
Order, void *data, int32_t nbDims) noexcept
Reformat the input weights of the given shape based on the new order of dimensions.
- **TRT_DEPRECATED** bool [reorderSubBuffers](#) (void *input, int32_t const *order, int32_t num, int32_t size) 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 fcco order. TensorFlow however formats the sub-buffers in icfo order. This helper function solves this in a generic fashion.

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

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

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

Returns

True on success, false on failure.

See also

[reshapeWeights\(\)](#)

Deprecated Deprecated in TensorRT 8.0.

Warning

This file will be removed in TensorRT 10.0.

8.7.1.2 reshapeWeights()

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

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

Parameters

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

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

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

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

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

Returns

True on success, false on failure.

Deprecated Deprecated in TensorRT 8.0.

Warning

This file will be removed in TensorRT 10.0.

8.7.1.3 transposeSubBuffers()

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

Transpose num sub-buffers of height * width.

Parameters

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

Returns

True on success, false on failure.

Deprecated Deprecated in TensorRT 8.0.

Warning

This file will be removed in TensorRT 10.0.

8.8 nvonnxparser Namespace Reference

The TensorRT ONNX parser API namespace.

Classes

- class [IOnnxConfig](#)
Configuration Manager Class.
- class [IParser](#)
an object for parsing ONNX models into a TensorRT network definition
- class [IParserError](#)
an object containing information about an error

Enumerations

- enum class `ErrorCode` : int {
`kSUCCESS` = 0, `kINTERNAL_ERROR` = 1, `kMEM_ALLOC_FAILED` = 2, `kMODEL_DESERIALIZE_FAILED` = 3,
`kINVALID_VALUE` = 4, `kINVALID_GRAPH` = 5, `kINVALID_NODE` = 6, `kUNSUPPORTED_GRAPH` = 7,
`kUNSUPPORTED_NODE` = 8 }

the type of parser error

Functions

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

8.8.1 Detailed Description

The TensorRT ONNX parser API namespace.

8.8.2 Enumeration Type Documentation

8.8.2.1 ErrorCode

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

the type of parser error

Enumerator

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

8.8.3 Function Documentation

8.8.3.1 createONNXConfig()

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

8.8.3.2 EnumMax()

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

8.8.3.3 EnumMax< ErrorCode >()

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

8.9 nvuffparser Namespace Reference

The TensorRT UFF parser API namespace.

Classes

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

Enumerations

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

Functions

- `IuffParser * createUffParser ()` noexcept
Creates a `IuffParser` object.
- `void shutdownProtobufLibrary (void)` noexcept
Shuts down protocol buffers library.

8.9.1 Detailed Description

The TensorRT UFF parser API namespace.

8.9.2 Enumeration Type Documentation

8.9.2.1 FieldType

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

The possible field types for custom layer.

Enumerator

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

8.9.2.2 UffInputOrder

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

The different possible supported input order.

Enumerator

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

8.9.3 Function Documentation

8.9.3.1 createUffParser()

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

Creates a [IUffParser](#) object.

Returns

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

See also

[nvuffparser::IUffParser](#)

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

8.9.3.2 shutdownProtobufLibrary()

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

Shuts down protocol buffers library.

Note

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

Chapter 9

Class Documentation

9.1 nvinfer1::plugin::DetectionOutputParameters Struct Reference

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

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

Public Attributes

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

9.1.1 Detailed Description

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

Parameters

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

9.1.2 Member Data Documentation

9.1.2.1 backgroundLabelId

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

9.1.2.2 codeType

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

9.1.2.3 confidenceThreshold

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

9.1.2.4 confSigmoid

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

9.1.2.5 inputOrder

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

9.1.2.6 isBatchAgnostic

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

9.1.2.7 isNormalized

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

9.1.2.8 keepTopK

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

9.1.2.9 nmsThreshold

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

9.1.2.10 numClasses

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

9.1.2.11 shareLocation

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

9.1.2.12 topK

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

9.1.2.13 varianceEncodedInTarget

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

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

- [NvInferPluginUtils.h](#)

9.2 Dims Class Reference

Structure to define the dimensions of a tensor.

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

9.2.1 Detailed Description

Structure to define the dimensions of a tensor.

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

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

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

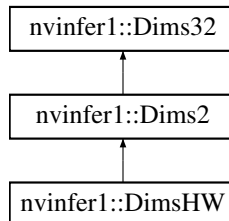
- [NvInferRuntimeCommon.h](#)

9.3 nvinfer1::Dims2 Class Reference

Descriptor for two-dimensional data.

```
#include <NvInferLegacyDims.h>
```

Inheritance diagram for nvinfer1::Dims2:



Public Member Functions

- [Dims2](#) ()
Construct an empty [Dims2](#) object.
- [Dims2](#) (int32_t d0, int32_t d1)
Construct a [Dims2](#) from 2 elements.

Additional Inherited Members

9.3.1 Detailed Description

Descriptor for two-dimensional data.

9.3.2 Constructor & Destructor Documentation

9.3.2.1 Dims2() [1/2]

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

Construct an empty [Dims2](#) object.

9.3.2.2 Dims2() [2/2]

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

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

Parameters

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

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

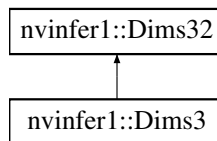
- [NvInferLegacyDims.h](#)

9.4 nvinfer1::Dims3 Class Reference

Descriptor for three-dimensional data.

```
#include <NvInferLegacyDims.h>
```

Inheritance diagram for nvinfer1::Dims3:



Public Member Functions

- [Dims3 \(\)](#)
Construct an empty [Dims3](#) object.
- [Dims3 \(int32_t d0, int32_t d1, int32_t d2\)](#)
Construct a [Dims3](#) from 3 elements.

Additional Inherited Members

9.4.1 Detailed Description

Descriptor for three-dimensional data.

9.4.2 Constructor & Destructor Documentation

9.4.2.1 Dims3() [1/2]

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

Construct an empty [Dims3](#) object.

9.4.2.2 Dims3() [2/2]

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

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

Parameters

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

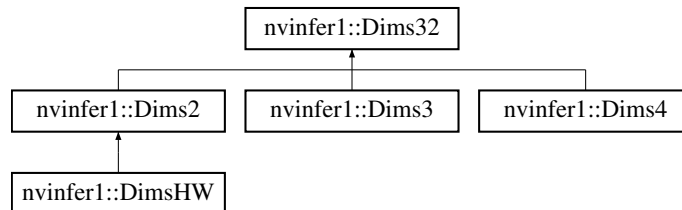
The documentation for this class was generated from the following file:

- [NvInferLegacyDims.h](#)

9.5 nvinfer1::Dims32 Class Reference

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

Inheritance diagram for nvinfer1::Dims32:



Public Attributes

- `int32_t nbDims`
The rank (number of dimensions).
- `int32_t d [MAX_DIMS]`
The extent of each dimension.

Static Public Attributes

- static constexpr int32_t [MAX_DIMS](#) {8}
The maximum rank (number of dimensions) supported for a tensor.

9.5.1 Member Data Documentation

9.5.1.1 d

```
int32_t nvinfer1::Dims32::d[MAX\_DIMS]
```

The extent of each dimension.

9.5.1.2 MAX_DIMS

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

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

9.5.1.3 nbDims

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

The rank (number of dimensions).

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

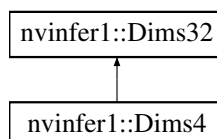
- [NvInferRuntimeCommon.h](#)

9.6 nvinfer1::Dims4 Class Reference

Descriptor for four-dimensional data.

```
#include <NvInferLegacyDims.h>
```

Inheritance diagram for nvinfer1::Dims4:



Public Member Functions

- [Dims4](#) ()
Construct an empty [Dims4](#) object.
- [Dims4](#) (int32_t d0, int32_t d1, int32_t d2, int32_t d3)
Construct a [Dims4](#) from 4 elements.

Additional Inherited Members

9.6.1 Detailed Description

Descriptor for four-dimensional data.

9.6.2 Constructor & Destructor Documentation

9.6.2.1 Dims4() [1/2]

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

Construct an empty [Dims4](#) object.

9.6.2.2 Dims4() [2/2]

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

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

Parameters

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

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

- [NvInferLegacyDims.h](#)

9.7 nvinfer1::DimsExprs Class Reference

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

Public Attributes

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

9.7.1 Detailed Description

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

9.7.2 Member Data Documentation

9.7.2.1 d

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

The extent of each dimension.

9.7.2.2 nbDims

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

The number of dimensions.

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

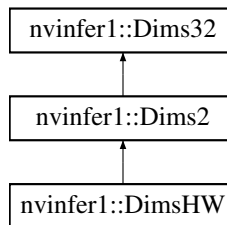
- [NvInferRuntime.h](#)

9.8 nvinfer1::DimsHW Class Reference

Descriptor for two-dimensional spatial data.

```
#include <NvInferLegacyDims.h>
```

Inheritance diagram for nvinfer1::DimsHW:



Public Member Functions

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

Additional Inherited Members

9.8.1 Detailed Description

Descriptor for two-dimensional spatial data.

9.8.2 Constructor & Destructor Documentation

9.8.2.1 DimsHW() [1/2]

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

Construct an empty [DimsHW](#) object.

9.8.2.2 DimsHW() [2/2]

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

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

Parameters

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

9.8.3 Member Function Documentation

9.8.3.1 h() [1/2]

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

Get the height.

Returns

The height.

9.8.3.2 h() [2/2]

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

Get the height.

Returns

The height.

9.8.3.3 w() [1/2]

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

Get the width.

Returns

The width.

9.8.3.4 w() [2/2]

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

Get the width.

Returns

The width.

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

- [NvInferLegacyDims.h](#)

9.9 nvinfer1::DynamicPluginTensorDesc Class Reference

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

Public Attributes

- [PluginTensorDesc desc](#)
Information required to interpret a pointer to tensor data, except that desc.dims has -1 in place of any runtime dimension.
- [Dims min](#)
Lower bounds on tensor's dimensions.
- [Dims max](#)
Upper bounds on tensor's dimensions.

9.9.1 Detailed Description

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

9.9.2 Member Data Documentation

9.9.2.1 desc

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

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

9.9.2.2 max

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

Upper bounds on tensor's dimensions.

9.9.2.3 min

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

Lower bounds on tensor's dimensions.

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

- [NvInferRuntime.h](#)

9.10 `nvinfer1::impl::EnumMaxImpl< T >` Struct Template Reference

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

9.10.1 Detailed Description

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

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

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

- [NvInferRuntimeCommon.h](#)

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

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

Static Public Attributes

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

9.11.1 Detailed Description

Maximum number of elements in ActivationType enum.

See also

[ActivationType](#)

9.11.2 Member Data Documentation

9.11.2.1 kVALUE

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

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

- [NvInfer.h](#)

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

Maximum number of elements in AllocatorFlag enum.

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

Static Public Attributes

- static constexpr int32_t [kVALUE](#) = 1
maximum number of elements in AllocatorFlag enum

9.12.1 Detailed Description

Maximum number of elements in AllocatorFlag enum.

See also

[AllocatorFlag](#)

9.12.2 Member Data Documentation

9.12.2.1 kVALUE

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

maximum number of elements in AllocatorFlag enum

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

- [NvInferRuntimeCommon.h](#)

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

Maximum number of elements in DataType enum.

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

Static Public Attributes

- static constexpr int32_t [kVALUE](#) = 5

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 = 5 [static], [constexpr]
```

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

- [NvInferRuntimeCommon.h](#)

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

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

Static Public Attributes

- static constexpr int32_t [kVALUE](#) = 14

9.14.1 Detailed Description

Maximum number of elements in ElementWiseOperation enum.

See also

[ElementWiseOperation](#)

9.14.2 Member Data Documentation

9.14.2.1 kVALUE

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

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

- [NvInfer.h](#)

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

Maximum number of elements in EngineCapability enum.

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

Static Public Attributes

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

9.15.1 Detailed Description

Maximum number of elements in EngineCapability enum.

See also

[EngineCapability](#)

9.15.2 Member Data Documentation

9.15.2.1 kVALUE

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

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

- [NvInferRuntime.h](#)

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

Maximum number of elements in ErrorCode enum.

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

Static Public Attributes

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

9.16.1 Detailed Description

Maximum number of elements in `ErrorCode` enum.

See also

[ErrorCode](#)

9.16.2 Member Data Documentation

9.16.2.1 `kVALUE`

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

Declaration of `kVALUE`.

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

- [NvInferRuntimeCommon.h](#)

9.17 `nvinfer1::impl::EnumMaxImpl< ILogger::Severity >` Struct Reference

Maximum number of elements in `ILogger::Severity` enum.

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

Static Public Attributes

- static constexpr int32_t `kVALUE` = 5
Declaration of `kVALUE` that represents maximum number of elements in `ILogger::Severity` enum.

9.17.1 Detailed Description

Maximum number of elements in `ILogger::Severity` enum.

See also

[ILogger::Severity](#)

9.17.2 Member Data Documentation

9.17.2.1 kVALUE

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

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

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

- [NvInferRuntimeCommon.h](#)

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

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

Static Public Attributes

- static constexpr int32_t `kVALUE` = 6

9.18.1 Detailed Description

Maximum number of elements in `PaddingMode` enum.

See also

[PaddingMode](#)

9.18.2 Member Data Documentation

9.18.2.1 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](#)

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

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

Static Public Attributes

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

9.19.1 Detailed Description

Maximum number of elements in PoolingType enum.

See also

[PoolingType](#)

9.19.2 Member Data Documentation

9.19.2.1 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](#)

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

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

Static Public Attributes

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

9.20.1 Detailed Description

Maximum number of elements in `ResizeCoordinateTransformation` enum.

See also

[ResizeCoordinateTransformation](#)

9.20.2 Member Data Documentation

9.20.2.1 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](#)

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

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

Static Public Attributes

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

9.21.1 Detailed Description

Maximum number of elements in `ResizeMode` enum.

See also

[ResizeMode](#)

9.21.2 Member Data Documentation

9.21.2.1 kVALUE

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

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

- [NvInfer.h](#)

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

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

Static Public Attributes

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

9.22.1 Detailed Description

Maximum number of elements in ResizeRoundMode enum.

See also

[ResizeRoundMode](#)

9.22.2 Member Data Documentation

9.22.2.1 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](#)

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

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


Static Public Attributes

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

9.23.1 Detailed Description

Maximum number of elements in ResizeSelector enum.

See also

[ResizeSelector](#)

9.23.2 Member Data Documentation

9.23.2.1 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](#)

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

Maximum number of elements in TensorFormat enum.

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

Static Public Attributes

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

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

9.24.1 Detailed Description

Maximum number of elements in TensorFormat enum.

See also

[TensorFormat](#)

9.24.2 Member Data Documentation

9.24.2.1 `kVALUE`

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

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

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

- [NvInferRuntimeCommon.h](#)

9.25 `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

9.25.1 Detailed Description

Maximum number of elements in `TensorLocation` enum.

See also

[TensorLocation](#)

9.25.2 Member Data Documentation

9.25.2.1 `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](#)

9.26 nvuffparser::FieldCollection Struct Reference

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

Public Attributes

- `int32_t nbFields`
- `FieldMap const * fields`

9.26.1 Member Data Documentation

9.26.1.1 fields

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

9.26.1.2 nbFields

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

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

- [NvUffParser.h](#)

9.27 nvuffparser::FieldMap Class Reference

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

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

Public Member Functions

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

Public Attributes

- `char const * name`
- `void const * data`
- `FieldType type = FieldType::kUNKNOWN`
- `int32_t length = 1`

9.27.1 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

9.27.2 Constructor & Destructor Documentation

9.27.2.1 FieldMap()

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

9.27.3 Member Data Documentation

9.27.3.1 data

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

9.27.3.2 length

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

9.27.3.3 name

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

9.27.3.4 type

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

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

- [NvUffParser.h](#)

9.28 nvinfer1::safe::FloatingPointErrorInformation Struct Reference

Space to record information about floating point runtime errors.

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

Public Attributes

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

9.28.1 Detailed Description

Space to record information about floating point runtime errors.

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

9.28.2 Member Data Documentation

9.28.2.1 nbInfErrors

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

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

9.28.2.2 nbNanErrors

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

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

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

- [NvInferSafeRuntime.h](#)

9.29 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]

9.29.1 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.

9.29.2 Member Data Documentation

9.29.2.1 aspectRatios

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

9.29.2.2 H

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

9.29.2.3 maxSize

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

9.29.2.4 minSize

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

9.29.2.5 numAspectRatios

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

9.29.2.6 variance

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

9.29.2.7 W

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

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

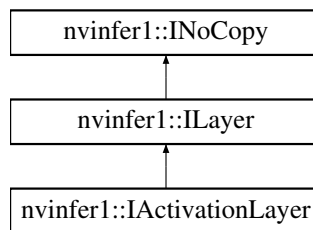
- [NvInferPluginUtils.h](#)

9.30 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`

9.30.1 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.

9.30.2 Constructor & Destructor Documentation

9.30.2.1 ~IActivationLayer()

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

9.30.3 Member Function Documentation

9.30.3.1 getActivationType()

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

Get the type of activation to be performed.

See also

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

9.30.3.2 `getAlpha()`

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

Get the alpha parameter.

See also

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

9.30.3.3 `getBeta()`

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

Get the beta parameter.

See also

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

9.30.3.4 `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](#)

9.30.3.5 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\(\)](#)

9.30.3.6 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\(\)](#)

9.30.4 Member Data Documentation

9.30.4.1 mImpl

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

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

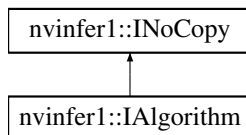
- [NvInfer.h](#)

9.31 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`

9.31.1 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.

9.31.2 Constructor & Destructor Documentation**9.31.2.1 ~IAlgorithm()**

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

9.31.3 Member Function Documentation**9.31.3.1 getAlgorithmIOInfo()**

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

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

Parameters

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

Returns

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

Deprecated Use [IAlgorithm::getAlgorithmIOInfoByIndex](#) instead. Deprecated in TensorRT 8.0

9.31.3.2 getAlgorithmIOInfoByIndex()

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

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

Parameters

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

Returns

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

9.31.3.3 getAlgorithmVariant()

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

Returns the algorithm variant.

9.31.3.4 getTimingMSec()

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

The time in milliseconds to execute the algorithm.

9.31.3.5 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.

9.31.4 Member Data Documentation

9.31.4.1 mImpl

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

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

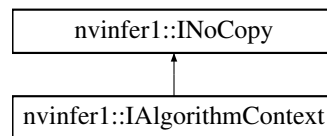
- [NvInfer.h](#)

9.32 nvinfer1::IAlgorithmContext Class Reference

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

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

Inheritance diagram for nvinfer1::IAlgorithmContext:



Public Member Functions

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

Protected Member Functions

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

Protected Attributes

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

9.32.1 Detailed Description

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

See also

[IAlgorithm](#)

Warning

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

9.32.2 Constructor & Destructor Documentation

9.32.2.1 ~IAlgorithmContext()

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

9.32.3 Member Function Documentation

9.32.3.1 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.

9.32.3.2 getName()

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

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

9.32.3.3 getNbInputs()

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

Return number of inputs of the algorithm.

9.32.3.4 getNbOutputs()

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

Return number of outputs of the algorithm.

9.32.4 Member Data Documentation

9.32.4.1 mImpl

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

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

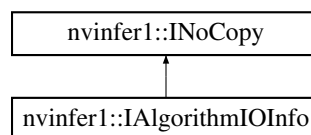
- [NvInfer.h](#)

9.33 nvinfer1::IAlgorithmIOInfo Class Reference

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

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

Inheritance diagram for nvinfer1::IAlgorithmIOInfo:



Public Member Functions

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

Protected Member Functions

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

Protected Attributes

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

9.33.1 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.

9.33.2 Constructor & Destructor Documentation

9.33.2.1 ~IAlgorithmIOInfo()

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

9.33.3 Member Function Documentation

9.33.3.1 getDataTypes()

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

Return DataType of the input/output of algorithm.

9.33.3.2 getStrides()

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

Return strides of the input/output tensor of algorithm.

9.33.3.3 getTensorFormat()

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

Return TensorFormat of the input/output of algorithm.

9.33.4 Member Data Documentation

9.33.4.1 mImpl

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

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

- [NvInfer.h](#)

9.34 nvinfer1::IAlgorithmSelector Class Reference

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

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

Public Member Functions

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

9.34.1 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 [INetworkDefiniton](#). For example, an algorithm might be implementing a conglomeration of multiple [ILayers](#) in [INetworkDefinition](#).

9.34.2 Constructor & Destructor Documentation

9.34.2.1 ~IAlgorithmSelector()

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

9.34.3 Member Function Documentation

9.34.3.1 reportAlgorithms()

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

Called by TensorRT to report choices it made.

Note

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

Parameters

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

9.34.3.2 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:

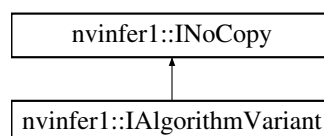
- [NvInfer.h](#)

9.35 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 [~IAAlgorithmVariant](#) () noexcept=default

Protected Attributes

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

9.35.1 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 [IAAlgorithmSelector::selectAlgorithms\(\)](#)

See also

[IAAlgorithmIOInfo](#), [IAAlgorithm](#), [IAAlgorithmSelector::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.

9.35.2 Constructor & Destructor Documentation

9.35.2.1 [~IAAlgorithmVariant\(\)](#)

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

9.35.3 Member Function Documentation

9.35.3.1 getImplementation()

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

Return implementation of the algorithm.

9.35.3.2 getTactic()

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

Return tactic of the algorithm.

9.35.4 Member Data Documentation

9.35.4.1 mImpl

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

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

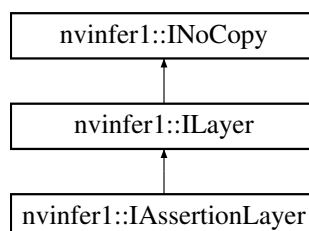
- [NvInfer.h](#)

9.36 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](#)

9.36.1 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.

9.36.2 Constructor & Destructor Documentation

9.36.2.1 ~IAssertionLayer()

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

9.36.3 Member Function Documentation

9.36.3.1 getMessage()

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

Return the assertion message.

See also

[setMessage\(\)](#)

9.36.3.2 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\(\)](#)

9.36.4 Member Data Documentation

9.36.4.1 mImpl

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

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

- [NvInfer.h](#)

9.37 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

9.37.1 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.

9.37.2 Constructor & Destructor Documentation

9.37.2.1 ~IBinaryProtoBlob()

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

9.37.3 Member Function Documentation

9.37.3.1 destroy()

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

Deprecated Use `delete` instead. Deprecated in TensorRT 8.0.

Warning

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

9.37.3.2 `getData()`

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

9.37.3.3 `getDataType()`

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

9.37.3.4 `getDimensions()`

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

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

- [NvCaffeParser.h](#)

9.38 `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](#) ()

9.38.1 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.

9.38.2 Constructor & Destructor Documentation**9.38.2.1 ~IBlobNameToTensor()**

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

9.38.3 Member Function Documentation**9.38.3.1 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:

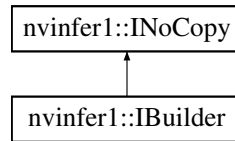
- [NvCaffeParser.h](#)

9.39 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.

Protected Attributes

- `apiv::VBuilder * mImpl`

Additional Inherited Members

9.39.1 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.

9.39.2 Constructor & Destructor Documentation

9.39.2.1 ~IBuilder()

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

9.39.3 Member Function Documentation

9.39.3.1 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 Use [IBuilder::buildSerializedNetwork](#). Deprecated in TensorRT 8.0

9.39.3.2 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](#)

9.39.3.3 createBuilderConfig()

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

Create a builder configuration object.

See also

[IBuilderConfig](#)

9.39.3.4 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., $1U \ll \text{NetworkDefinitionCreationFlag::kEXPLICIT_BATCH}$
--------------	--

See also

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

9.39.3.5 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](#)

9.39.3.6 `destroy()`

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

Destroy this object.

Deprecated Use `delete` instead. Deprecated in TensorRT 8.0

Warning

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

9.39.3.7 `getErrorRecorder()`

`ILoggerRecorder * 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 `ILoggerRecorder` object that has been registered.

See also

[setErrorRecorder\(\)](#)

9.39.3.8 `getLogger()`

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

get the logger with which the builder was created

Returns

the logger

9.39.3.9 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\(\)](#)

9.39.3.10 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.

9.39.3.11 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\(\)](#)

9.39.3.12 getNbDLACores()

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

Return the number of DLA engines available to this builder.

9.39.3.13 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.

9.39.3.14 platformHasFastFp16()

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

Determine whether the platform has fast native fp16.

9.39.3.15 platformHasFastInt8()

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

Determine whether the platform has fast native int8.

9.39.3.16 platformHasTf32()

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

Determine whether the platform has TF32 support.

9.39.3.17 reset()

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

Resets the builder state to default values.

9.39.3.18 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\(\)](#)

9.39.3.19 setGpuAllocator()

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

Set the GPU allocator.

Parameters

<i>allocator</i>	Set the GPU allocator to be used by the builder. All GPU memory acquired will use this allocator. If NULL is passed, the default allocator will be used.
------------------	--

Default: uses cudaMalloc/cudaFree.

Note

This allocator will be passed to any engines created via the builder; thus the lifetime of the allocator must span the lifetime of those engines as well as that of the builder. If nullptr is passed, the default allocator will be used.

9.39.3.20 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\(\)](#)

9.39.3.21 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.

9.39.4 Member Data Documentation

9.39.4.1 mImpl

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

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

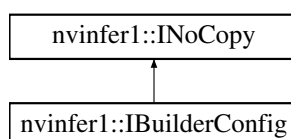
- [NvInfer.h](#)

9.40 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` (`IAgorithmSelector` *selector) noexcept
Set Algorithm Selector.
- `IAgorithmSelector` * `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.

Protected Attributes

- `apiv::VBuilderConfig` * `mImpl`

Additional Inherited Members

9.40.1 Detailed Description

Holds properties for configuring a builder to produce an engine.

See also

[BuilderFlags](#)

9.40.2 Constructor & Destructor Documentation

9.40.2.1 ~IBuilderConfig()

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

9.40.3 Member Function Documentation

9.40.3.1 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.

9.40.3.2 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.

9.40.3.3 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\(\)](#)

9.40.3.4 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\(\)](#)

9.40.3.5 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

9.40.3.6 destroy()

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

Delete this [IBuilderConfig](#).

De-allocates any internally allocated memory.

Deprecated Use `delete` instead. Deprecated in TensorRT 8.0

Warning

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

9.40.3.7 getAlgorithmSelector()

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

Get Algorithm Selector.

9.40.3.8 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\(\)](#)

9.40.3.9 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.

9.40.3.10 getDefaultDeviceType()

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

Get the default DeviceType which was set by setDefaultDeviceType.

By default it returns [DeviceType::kGPU](#).

9.40.3.11 `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.

9.40.3.12 `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.

9.40.3.13 `getEngineCapability()`

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

Query EngineCapability flow configured for the builder.

By default it returns `EngineCapability::kSTANDARD`.

See also

[setEngineCapability\(\)](#)

9.40.3.14 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.

9.40.3.15 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\(\)](#)

9.40.3.16 getInt8Calibrator()

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

Get Int8 Calibration interface.

9.40.3.17 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 Use [IBuilderConfig::getMemoryPoolLimit](#) with [MemoryPoolType::kWORKSPACE](#). Deprecated in TensorRT 8.3

9.40.3.18 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](#)

9.40.3.19 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 Use [getAvgTimingIterations\(\)](#) instead. Deprecated in TensorRT 8.4.

9.40.3.20 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.

9.40.3.21 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\(\)](#)

9.40.3.22 `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](#)

9.40.3.23 `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.

9.40.3.24 `getQuantizationFlags()`

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

Get the quantization flags.

Returns

The quantization flags as a bitmask.

See also

[setQuantizationFlag\(\)](#)

9.40.3.25 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

9.40.3.26 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](#)

9.40.3.27 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\(\)](#)

9.40.3.28 reset()

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

Resets the builder configuration to defaults.

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

9.40.3.29 resetDeviceType()

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

reset the DeviceType for this layer

See also

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

9.40.3.30 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.
-----------------	---

9.40.3.31 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\(\)](#)

9.40.3.32 setCalibrationProfile()

```
bool nvinfer1::IBuilderConfig::setCalibrationProfile (
    OptimizationProfile 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 <code>nullptr</code> . MIN and MAX values will be overwritten by <code>kOPT</code> .
----------------	---

Returns

True if the calibration profile was set correctly.

9.40.3.33 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\(\)](#)

9.40.3.34 setDeviceType()

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

Set the device that this layer must execute on.

Parameters

<i>deviceType</i>	that this layer must execute on. If DeviceType is not set or is reset, TensorRT will use the default DeviceType set in the builder.
-------------------	---

Note

The device type for a layer must be compatible with the safety flow (if specified). For example a layer cannot be marked for DLA execution while the builder is configured for kSAFE_GPU.

See also

[getDeviceType\(\)](#)

9.40.3.35 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\(\)](#)

9.40.3.36 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.

9.40.3.37 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\(\)](#)

9.40.3.38 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\(\)](#)

9.40.3.39 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.

9.40.3.40 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 Use [IBuilderConfig::setMemoryPoolLimit](#) with [MemoryPoolType::kWORKSPACE](#). Deprecated in TensorRT 8.3

9.40.3.41 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](#)

9.40.3.42 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 Use [setAvgTimingIterations\(\)](#) instead. Deprecated in TensorRT 8.4.

9.40.3.43 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\(\)](#)

9.40.3.44 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](#)

9.40.3.45 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\(\)](#)

9.40.3.46 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\(\)](#)

9.40.3.47 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.

By default, kCUBLAS, kCUDNN, and kEDGE_MASK_CONVOLUTIONS are always enabled. kCUBLAS_LT is always enabled for x86 platforms and only enabled for non-x86 platforms when CUDA >= 11.0.

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.

9.40.3.48 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.

9.40.4 Member Data Documentation

9.40.4.1 mImpl

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

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

- [NvInfer.h](#)

9.41 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.

9.41.1 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.

9.41.2 Constructor & Destructor Documentation

9.41.2.1 ~ICaffeParser()

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

9.41.3 Member Function Documentation

9.41.3.1 destroy()

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

Destroy this [ICaffeParser](#) object.

Deprecated Use `delete` instead. Deprecated in TensorRT 8.0.

Warning

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

9.41.3.2 `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\(\)](#)

9.41.3.3 `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](#)

9.41.3.4 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](#)

9.41.3.5 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](#)

9.41.3.6 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\(\)](#)

9.41.3.7 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 .
----------------	---

9.41.3.8 setPluginNamespace()

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

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

9.41.3.9 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:

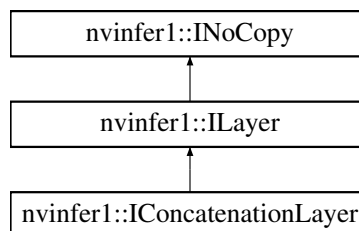
- [NvCaffeParser.h](#)

9.42 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](#)

9.42.1 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.

9.42.2 Constructor & Destructor Documentation

9.42.2.1 [~IConcatenationLayer](#)()

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

9.42.3 Member Function Documentation

9.42.3.1 `getAxis()`

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

Get the axis along which concatenation occurs.

See also

[setAxis\(\)](#)

9.42.3.2 `setAxis()`

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

Set the axis along which concatenation occurs.

0 is the major axis (excluding the batch dimension). The default is the number of non-batch axes in the tensor minus three (e.g. for an NCHW input it would be 0), or 0 if there are fewer than 3 non-batch axes.

When running this layer on the DLA, only concat across the Channel axis is valid.

Parameters

<i>axis</i>	The axis along which concatenation occurs.
-------------	--

9.42.4 Member Data Documentation

9.42.4.1 `mImpl`

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

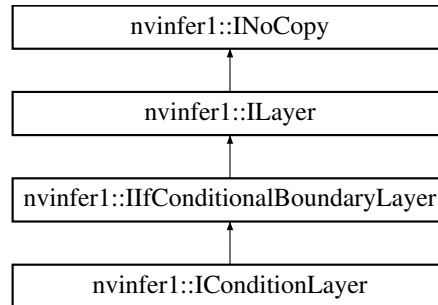
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.43 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

9.43.1 Detailed Description

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

9.43.2 Constructor & Destructor Documentation

9.43.2.1 ~IConditionLayer()

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

9.43.3 Member Data Documentation

9.43.3.1 mImpl

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

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

- [NvInfer.h](#)

9.44 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](#)

9.44.1 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.

9.44.2 Constructor & Destructor Documentation

9.44.2.1 ~IConsistencyChecker()

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

De-allocates any internally allocated memory.

9.44.2.2 IConsistencyChecker() [1/3]

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

9.44.2.3 IConsistencyChecker() [2/3]

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

9.44.2.4 IConsistencyChecker() [3/3]

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

9.44.3 Member Function Documentation**9.44.3.1 operator=() [1/2]**

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

9.44.3.2 operator=() [2/2]

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

9.44.3.3 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.

9.44.4 Member Data Documentation

9.44.4.1 mImpl

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

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

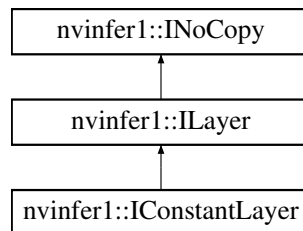
- [NvInferConsistency.h](#)

9.45 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](#)

9.45.1 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.

9.45.2 Constructor & Destructor Documentation

9.45.2.1 ~IConstantLayer()

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

9.45.3 Member Function Documentation

9.45.3.1 getDimensions()

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

Get the dimensions for the layer.

Returns

the dimensions for the layer

See also

[getDimensions](#)

9.45.3.2 getWeights()

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

Get the weights for the layer.

See also

[setWeights](#)

9.45.3.3 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](#)

9.45.3.4 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\(\)](#)

9.45.4 Member Data Documentation

9.45.4.1 mImpl

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

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

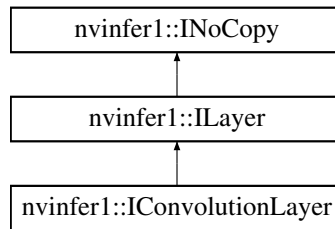
- [NvInfer.h](#)

9.46 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 [~IC convolutionLayer](#) () noexcept=default

Protected Attributes

- `apiv::VConvolutionLayer * mImpl`

9.46.1 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.

9.46.2 Constructor & Destructor Documentation

9.46.2.1 `~IConvolutionLayer()`

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

9.46.3 Member Function Documentation

9.46.3.1 `getBiasWeights()`

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

Get the bias weights for the convolution.

See also

[setBiasWeights\(\)](#)

9.46.3.2 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

9.46.3.3 getDilationNd()

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

Get the multi-dimension dilation of the convolution.

See also

[setDilation\(\)](#)

9.46.3.4 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

9.46.3.5 getKernelSizeNd()

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

Get the multi-dimension kernel size of the convolution.

See also

[setKernelSizeNd\(\)](#)

9.46.3.6 getKernelWeights()

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

Get the kernel weights of the convolution.

See also

[setKernelWeights\(\)](#)

9.46.3.7 getNbGroups()

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

Get the number of groups of the convolution.

See also

[setNbGroups\(\)](#)

9.46.3.8 getNbOutputMaps()

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

Get the number of output maps for the convolution.

See also

[setNbOutputMaps\(\)](#)

9.46.3.9 `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

9.46.3.10 `getPaddingMode()`

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

Get the padding mode.

Default: `kEXPLICIT_ROUND_DOWN`

See also

[setPaddingMode\(\)](#)

9.46.3.11 `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\(\)](#)

9.46.3.12 `getPostPadding()`

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

Get the post-padding.

See also

[setPostPadding\(\)](#)

9.46.3.13 `getPrePadding()`

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

Get the pre-padding.

See also

[setPrePadding\(\)](#)

9.46.3.14 `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

9.46.3.15 `getStrideNd()`

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

Get the multi-dimension stride of the convolution.

See also

[setStrideNd\(\)](#)

9.46.3.16 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\(\)](#)

9.46.3.17 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

9.46.3.18 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\(\)](#)

9.46.3.19 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

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.

9.46.3.20 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

9.46.3.21 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\(\)](#)

9.46.3.22 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\(\)](#)

9.46.3.23 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\(\)](#)

9.46.3.24 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\(\)](#)

9.46.3.25 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

9.46.3.26 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\(\)](#)

9.46.3.27 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\(\)](#)

9.46.3.28 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\(\)](#)

9.46.3.29 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\(\)](#)

9.46.3.30 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

9.46.3.31 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\(\)](#)

9.46.4 Member Data Documentation

9.46.4.1 mImpl

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

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

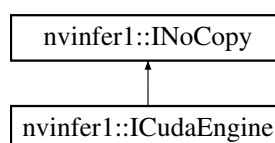
- [NvInfer.h](#)

9.47 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`
- `int32_t getNbBindings () const noexcept`
Get the number of binding indices.
- `int32_t getBindingIndex (char const *name) const noexcept`
Retrieve the binding index for a named tensor.
- `char const * getBindingName (int32_t bindingIndex) const noexcept`
Retrieve the name corresponding to a binding index.
- `bool bindingIsInput (int32_t bindingIndex) const noexcept`
Determine whether a binding is an input binding.
- `Dims getBindingDimensions (int32_t bindingIndex) const noexcept`
Get the dimensions of a binding.
- `DataType getBindingDataType (int32_t bindingIndex) const noexcept`
Determine the required data type for a buffer from its binding index.
- `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;.
- `TensorLocation getLocation (int32_t bindingIndex) const noexcept`
Get location of binding.
- `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.
- `int32_t getBindingBytesPerComponent (int32_t bindingIndex) const noexcept`
Return the number of bytes per component of an element.
- `int32_t getBindingComponentsPerElement (int32_t bindingIndex) const noexcept`
Return the number of components included in one element.
- `TensorFormat getBindingFormat (int32_t bindingIndex) const noexcept`
Return the binding format.
- `char const * getBindingFormatDesc (int32_t bindingIndex) const noexcept`
Return the human readable description of the tensor format.
- `int32_t getBindingVectorizedDim (int32_t bindingIndex) const noexcept`
Return the dimension index that the buffer is vectorized.
- `char const * getName () const noexcept`
Returns the name of the network associated with the engine.
- `int32_t getNbOptimizationProfiles () const noexcept`

- Get the number of optimization profiles defined for this engine.*

 - **Dims** `getProfileDimensions` (int32_t bindingIndex, int32_t profileIndex, **OptProfileSelector** select) const noexcept
- Get the minimum / optimum / maximum dimensions for a particular binding under an optimization profile.*

 - 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.*

 - bool `isShapeBinding` (int32_t bindingIndex) const noexcept

True if tensor is required as input for shape calculations or output from them.
- 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.

Protected Attributes

- apiv::VCudaEngine * `mImpl`

Additional Inherited Members

9.47.1 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.

9.47.2 Constructor & Destructor Documentation

9.47.2.1 ~ICudaEngine()

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

9.47.3 Member Function Documentation

9.47.3.1 bindingIsInput()

```
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.

See also

[getBindingIndex\(\)](#)

9.47.3.2 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](#).

9.47.3.3 createExecutionContext()

```
IExecutionContext * nvinfer1::ICudaEngine::createExecutionContext ( ) [inline], [noexcept]
```

Create an execution context.

If the engine supports dynamic shapes, each execution context in concurrent use must use a separate optimization profile. The first execution context created will call `setOptimizationProfile(0)` implicitly. For other execution contexts, `setOptimizationProfile()` must be called with unique profile index before calling `execute` or `enqueue`. If an error recorder has been set for the engine, it will also be passed to the execution context.

See also

[IExecutionContext](#).

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

9.47.3.4 createExecutionContextWithoutDeviceMemory()

```
IExecutionContext * 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.

9.47.3.5 destroy()

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

Destroy this object;

Deprecated Use `delete` instead. Deprecated in TRT 8.0.

Warning

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

9.47.3.6 getBindingBytesPerComponent()

```
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.
---------------------	--------------------

See also

[ICudaEngine::getBindingVectorizedDim\(\)](#)

9.47.3.7 getBindingComponentsPerElement()

```
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.
---------------------	--------------------

See also

[ICudaEngine::getBindingVectorizedDim\(\)](#)

9.47.3.8 getBindingDataType()

```
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.

See also

[getBindingIndex\(\)](#)

9.47.3.9 getBindingDimensions()

```
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].

See also

[getBindingIndex\(\)](#)

9.47.3.10 getBindingFormat()

```
TensorFormat nvinfer1::ICudaEngine::getBindingFormat (
    int32_t bindingIndex ) const [inline], [noexcept]
```

Return the binding format.

Parameters

<i>bindingIndex</i>	The binding Index.
---------------------	--------------------

9.47.3.11 getBindingFormatDesc()

```
char const * nvinfer1::ICudaEngine::getBindingFormatDesc (
    int32_t bindingIndex ) const [inline], [noexcept]
```

Return the human readable description of the tensor format.

The description includes the order, vectorization, data type, strides, and etc. 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.
---------------------	--------------------

9.47.3.12 getBindingIndex()

```
int32_t nvinfer1::ICudaEngine::getBindingIndex (
    char const * name ) const [inline], [noexcept]
```

Retrieve the binding index for a named tensor.

[IExecutionContext::enqueueV2\(\)](#) and [IExecutionContext::executeV2\(\)](#) require an array of buffers.

Engine bindings map from tensor names to indices in this array. Binding indices are assigned at engine build time, and take values in the range [0 ... n-1] where n is the total number of inputs and outputs.

To get the binding index of the name in an optimization profile with index $k > 0$, mangle the name by appending "[profile k]", as described for method [getBindingName\(\)](#).

Parameters

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

Returns

The binding index for the named tensor, or -1 if the name is not found.

See also

[getNbBindings\(\)](#) [getBindingName\(\)](#)

9.47.3.13 `getBindingName()`

```
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.

See also

[getBindingIndex\(\)](#)

9.47.3.14 `getBindingVectorizedDim()`

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

Return the dimension index that the buffer is vectorized.

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

Parameters

<i>bindingIndex</i>	The binding Index.
---------------------	--------------------

9.47.3.15 `getDeviceMemorySize()`

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

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

See also

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

9.47.3.16 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.

9.47.3.17 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\(\)](#)

9.47.3.18 getLocation()

```
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.

See also

[ITensor::setLocation\(\)](#) [ITensor::getLocation\(\)](#)

Parameters

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

Returns

The location of the bound tensor with given index.

9.47.3.19 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.

9.47.3.20 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.

9.47.3.21 getNbBindings()

```
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.

See also

[getBindingIndex\(\)](#);

9.47.3.22 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.

9.47.3.23 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\(\)](#)

9.47.3.24 getProfileDimensions()

```
Dims nvinfer1::ICudaEngine::getProfileDimensions (
    int32_t bindingIndex,
    int32_t profileIndex,
    OptProfileSelector select ) const [inline], [noexcept]
```

Get the minimum / optimum / maximum dimensions for a particular binding under an optimization profile.

Parameters

<i>bindingIndex</i>	The 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 <code>getNbOptimizationProfiles()-1</code> .
<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.

9.47.3.25 getProfileShapeValues()

```
int32_t const * nvinfer1::ICudaEngine::getProfileShapeValues (
    int32_t profileIndex,
    int32_t inputIndex,
    OptProfileSelector select ) const [inline], [noexcept]
```

Get minimum / optimum / maximum values for an input shape binding under an optimization profile.

Parameters

<i>profileIndex</i>	The profile index (must be between 0 and <code>getNbOptimizationProfiles()-1</code>)
<i>inputIndex</i>	The input index (must be between 0 and <code>getNbBindings() - 1</code>)
<i>select</i>	Whether to query the minimum, optimum, or maximum shape values for this binding.

Returns

If the binding is an input shape binding, return a pointer to an array that has the same number of elements as the corresponding tensor, i.e. 1 if `dims.nbDims == 0`, or `dims.d[0]` if `dims.nbDims == 1`, where `dims = getBindingDimensions(inputIndex)`. The array contains the elementwise minimum / optimum / maximum values for this shape binding under the profile. If either of the indices is out of range, or if the binding is not an input shape binding, return `nullptr`.

For backwards compatibility with earlier versions of TensorRT, a `bindingIndex` that does not belong to the profile is corrected as described for `getProfileDimensions`.

See also

[ICudaEngine::getProfileDimensions](#)

9.47.3.26 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\(\)](#)

9.47.3.27 getTacticSources()

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

return the tactic sources required by this engine.

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

See also

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

9.47.3.28 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](#)

9.47.3.29 isExecutionBinding()

```
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](#).

See also

[isShapeBinding\(\)](#)

9.47.3.30 isRefittable()

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

Return true if an engine can be refit.

See also

[nvinfer1::createInferRefitter\(\)](#)

9.47.3.31 isShapeBinding()

```
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.

See also

[isExecutionBinding\(\)](#)

9.47.3.32 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\(\)](#)

9.47.3.33 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\(\)](#)

9.47.4 Member Data Documentation

9.47.4.1 mImpl

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

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

- [NvInferRuntime.h](#)

9.48 nvinfer1::safe::ICudaEngine Class Reference

A functionally safe engine for executing inference on a built network.

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

Public Member Functions

- virtual `std::int32_t` [getNbBindings](#) () const noexcept=0
Get the number of binding indices.
- virtual `std::int32_t` [getBindingIndex](#) (`AsciiChar` const *const name) const noexcept=0
Retrieve the binding index for a named tensor.
- virtual `AsciiChar` const * [getBindingName](#) (`std::int32_t` const bindingIndex) const noexcept=0
Retrieve the name corresponding to a binding index.
- virtual `bool` [bindingIsInput](#) (`std::int32_t` const bindingIndex) const noexcept=0
Determine whether a binding is an input binding.
- virtual `Dims` [getBindingDimensions](#) (`std::int32_t` const bindingIndex) const noexcept=0
Get the dimensions of a binding.
- virtual `DataType` [getBindingDataType](#) (`std::int32_t` const bindingIndex) const noexcept=0
Determine the required data type for a buffer from its binding index.
- virtual `IEExecutionContext` * [createExecutionContext](#) () noexcept=0
Create an execution context.
- virtual `IEExecutionContext` * [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 `std::int32_t` [getBindingBytesPerComponent](#) (`std::int32_t` const bindingIndex) const noexcept=0
Return the number of bytes per component of an element.

- virtual `std::int32_t` `getBindingComponentsPerElement` (`std::int32_t` const bindingIndex) const noexcept=0
Return the number of components included in one element.
- virtual `TensorFormat` `getBindingFormat` (`std::int32_t` const bindingIndex) const noexcept=0
Return the binding format.
- virtual `std::int32_t` `getBindingVectorizedDim` (`std::int32_t` const bindingIndex) const noexcept=0
Return the dimension index that the buffer is vectorized.
- virtual `AsciiChar` const * `getName` () const noexcept=0
Returns the name of the network associated with the engine.
- virtual void `setErrorRecorder` (`IErrRecorder` *const recorder) noexcept=0
Set the ErrorRecorder for this interface.
- virtual `IErrRecorder` * `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

9.48.1 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.

9.48.2 Constructor & Destructor Documentation

9.48.2.1 `ICudaEngine()` [1/3]

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

9.48.2.2 `~ICudaEngine()`

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

9.48.2.3 ICudaEngine() [2/3]

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

9.48.2.4 ICudaEngine() [3/3]

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

9.48.3 Member Function Documentation**9.48.3.1 bindingIsInput()**

```
virtual bool nvinfer1::safe::ICudaEngine::bindingIsInput (
    std::int32_t const bindingIndex ) const [pure virtual], [noexcept]
```

Determine whether a binding is an input binding.

Parameters

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

Returns

True if the index corresponds to an input binding and the index is in range.

See also

[getBindingIndex\(\)](#)

Usage considerations

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

9.48.3.2 createExecutionContext()

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

Create an execution context.

See also

[safe::IExecutionContext](#).

Usage considerations

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

9.48.3.3 createExecutionContextWithoutDeviceMemory()

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

Create an execution context without any device memory allocated.

The memory for execution of this device context must be supplied by the application.

See also

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

Usage considerations

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

9.48.3.4 getBindingBytesPerComponent()

```
virtual std::int32_t nvinfer1::safe::ICudaEngine::getBindingBytesPerComponent (   
    std::int32_t const bindingIndex ) const [pure virtual], [noexcept]
```

Return the number of bytes per component of an element.

The vector component size is returned if [getBindingVectorizedDim\(\)](#) != -1.

Parameters

<i>bindingIndex</i>	The binding Index.
---------------------	--------------------

See also

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

Usage considerations

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

9.48.3.5 getBindingComponentsPerElement()

```
virtual std::int32_t nvinfer1::safe::ICudaEngine::getBindingComponentsPerElement (
    std::int32_t const bindingIndex ) const [pure virtual], [noexcept]
```

Return the number of components included in one element.

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

Parameters

<i>bindingIndex</i>	The binding Index.
---------------------	--------------------

See also

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

Usage considerations

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

9.48.3.6 getBindingDataType()

```
virtual DataType nvinfer1::safe::ICudaEngine::getBindingDataType (
    std::int32_t const bindingIndex ) const [pure virtual], [noexcept]
```

Determine the required data type for a buffer from its binding index.

Parameters

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

Returns

The type of the data in the buffer.

See also

[getBindingIndex\(\)](#)

Usage considerations

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

9.48.3.7 getBindingDimensions()

```
virtual Dims nvinfer1::safe::ICudaEngine::getBindingDimensions (  
    std::int32_t const bindingIndex ) const [pure virtual], [noexcept]
```

Get the dimensions of a binding.

Parameters

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

Returns

The dimensions of the binding if the index is in range, otherwise [Dims\(\)](#)

See also

[getBindingIndex\(\)](#)

Usage considerations

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

9.48.3.8 `getBindingFormat()`

```
virtual 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.
---------------------	--------------------

Usage considerations

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

9.48.3.9 `getBindingIndex()`

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

Retrieve the binding index for a named tensor.

[safe::IExecutionContext::enqueueV2\(\)](#) requires an array of buffers. Engine bindings map from tensor names to indices in this array. Binding indices are assigned at engine build time, and take values in the range [0 ... n-1] where n is the total number of inputs and outputs.

Warning

Strings passed to the runtime must be 1024 characters or less including NULL terminator and must be NULL terminated.

Parameters

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

Returns

The binding index for the named tensor, or -1 if the name is not found.

See also

[getNbBindings\(\)](#)

Usage considerations

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

9.48.3.10 getBindingName()

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

Retrieve the name corresponding to a binding index.

This is the reverse mapping to that provided by [getBindingIndex\(\)](#).

Parameters

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

Returns

The name corresponding to the index, or nullptr if the index is out of range.

See also

[getBindingIndex\(\)](#)

Usage considerations

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

9.48.3.11 getBindingVectorizedDim()

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

Return the dimension index that the buffer is vectorized.

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

Parameters

<i>bindingIndex</i>	The binding Index.
---------------------	--------------------

Usage considerations

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

9.48.3.12 `getDeviceMemorySize()`

```
virtual size_t nvinfer1::safe::ICudaEngine::getDeviceMemorySize ( ) const [pure virtual], [noexcept]
```

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

See also

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

Usage considerations

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

9.48.3.13 `getErrorRecorder()`

```
virtual IErrorRecorder * nvinfer1::safe::ICudaEngine::getErrorRecorder ( ) const [pure virtual], [noexcept]
```

Get the ErrorRecorder assigned to this interface.

Retrieves the assigned error recorder object for the given class. A nullptr will be returned if an error reporter has not been inherited from the [IRuntime](#), and `setErrorReporter()` has not been called.

Returns

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

See also

[setErrorRecorder\(\)](#)

Usage considerations

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

9.48.3.14 getName()

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

Returns the name of the network associated with the engine.

The name is set during network creation and is retrieved after building or deserialization.

See also

[INetworkDefinition::setName\(\)](#), [INetworkDefinition::getName\(\)](#)

Returns

A null-terminated C-style string representing the name of the network.

Usage considerations

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

9.48.3.15 getNbBindings()

```
virtual std::int32_t nvinfer1::safe::ICudaEngine::getNbBindings ( ) const [pure virtual], [noexcept]
```

Get the number of binding indices.

See also

[getBindingIndex\(\)](#)

Usage considerations

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

9.48.3.16 operator=() [1/2]

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

9.48.3.17 operator=() [2/2]

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

9.48.3.18 setErrorRecorder()

```
virtual void nvinfer1::safe::ICudaEngine::setErrorRecorder (
    IErrorRecorder *const recorder ) [pure virtual], [noexcept]
```

Set the ErrorRecorder for this interface.

Assigns the ErrorRecorder to this interface. The ErrorRecorder will track all errors during execution. This function will call `incRefCount` of the registered ErrorRecorder at least once. Setting recorder to `nullptr` unregisters the recorder with the interface, resulting in a call to `decRefCount` if a recorder has been registered.

Parameters

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

See also

[getErrorRecorder\(\)](#)

Usage considerations

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

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

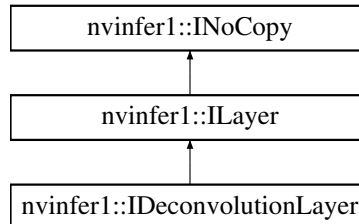
- [NvInferSafeRuntime.h](#)

9.49 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`

9.49.1 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.

9.49.2 Constructor & Destructor Documentation

9.49.2.1 ~IDeconvolutionLayer()

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

9.49.3 Member Function Documentation

9.49.3.1 getBiasWeights()

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

Get the bias weights for the deconvolution.

See also

[getBiasWeights\(\)](#)

9.49.3.2 getDilationNd()

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

Get the multi-dimension dilation of the deconvolution.

See also

[setDilationNd\(\)](#)

9.49.3.3 `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

9.49.3.4 `getKernelSizeNd()`

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

Get the multi-dimension kernel size of the deconvolution.

See also

[setKernelSizeNd\(\)](#)

9.49.3.5 `getKernelWeights()`

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

Get the kernel weights for the deconvolution.

See also

[setNbGroups\(\)](#)

9.49.3.6 `getNbGroups()`

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

Get the number of groups for a deconvolution.

See also

[setNbGroups\(\)](#)

9.49.3.7 getNbOutputMaps()

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

Get the number of output feature maps for the deconvolution.

See also

[setNbOutputMaps\(\)](#)

9.49.3.8 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

9.49.3.9 getPaddingMode()

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

Get the padding mode.

Default: `kEXPLICIT_ROUND_DOWN`

See also

[setPaddingMode\(\)](#)

9.49.3.10 `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\(\)](#)

9.49.3.11 `getPostPadding()`

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

Get the padding.

See also

[setPostPadding\(\)](#)

9.49.3.12 `getPrePadding()`

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

Get the pre-padding.

See also

[setPrePadding\(\)](#)

9.49.3.13 `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

9.49.3.14 getStrideNd()

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

Get the multi-dimension stride of the deconvolution.

See also

[setStrideNd\(\)](#)

9.49.3.15 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\(\)](#)

9.49.3.16 setDilationNd()

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

Set the multi-dimension dilation of the deconvolution.

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

See also

[getDilationNd\(\)](#)

9.49.3.17 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

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.

9.49.3.18 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

9.49.3.19 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\(\)](#)

9.49.3.20 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\(\)](#)

9.49.3.21 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\(\)](#)

9.49.3.22 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\(\)](#)

9.49.3.23 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

9.49.3.24 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\(\)](#)

9.49.3.25 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\(\)](#)

9.49.3.26 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\(\)](#)

9.49.3.27 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\(\)](#)

9.49.3.28 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

9.49.3.29 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\(\)](#)

9.49.4 Member Data Documentation

9.49.4.1 mImpl

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

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

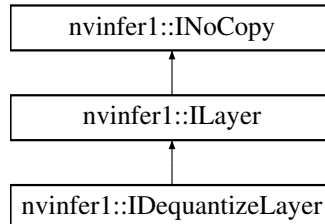
- [NvInfer.h](#)

9.50 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](#)

9.50.1 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: $output = (input - zeroPt) * scale$

The first input (index 0) is the tensor to be quantized. The second (index 1) and third (index 2) are the `scale` and `zeroPt` respectively. Each of `scale` and `zeroPt` must be either a scalar, or a 1D tensor.

The `zeroPt` tensor is optional, and if not set, will be assumed to be zero. Its data type must be `DataType::kINT8`. `zeroPt` must only contain zero-valued coefficients, because only symmetric quantization is supported. The `scale` value must be either a scalar for per-tensor quantization, or a 1D tensor for per-channel quantization. All `scale` coefficients must have positive values. The size of the 1-D `scale` tensor must match the size of the quantization axis. The size of the `scale` must match the size of the `zeroPt`.

The subgraph which terminates with the `scale` tensor must be a build-time constant. The same restrictions apply to the `zeroPt`. The output type, if constrained, must be constrained to `DataType::kINT8`. The input type, if constrained, must be constrained to `DataType::kFLOAT` (FP16 input is not supported). 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` 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.

9.50.2 Constructor & Destructor Documentation

9.50.2.1 `~IDequantizeLayer()`

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

9.50.3 Member Function Documentation

9.50.3.1 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.

9.50.3.2 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).

9.50.4 Member Data Documentation

9.50.4.1 mImpl

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

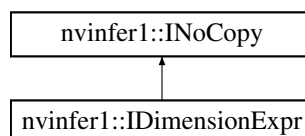
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.51 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`

9.51.1 Detailed Description

An `IDimensionExpr` represents an integer expression constructed from constants, input dimensions, and binary operations. These expressions are can be used in overrides of `IPluginV2DynamicExt::getOutputDimensions` 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](#)

9.51.2 Constructor & Destructor Documentation

9.51.2.1 `~IDimensionExpr()`

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

9.51.3 Member Function Documentation

9.51.3.1 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()`.

9.51.3.2 isConstant()

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

Return true if expression is a build-time constant.

9.51.4 Member Data Documentation

9.51.4.1 mImpl

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

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

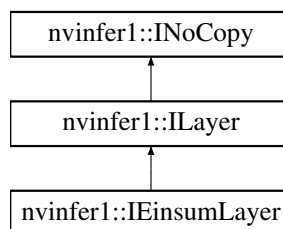
- [NvInferRuntime.h](#)

9.52 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`

9.52.1 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.

9.52.2 Constructor & Destructor Documentation

9.52.2.1 ~IEinsumLayer()

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

9.52.3 Member Function Documentation

9.52.3.1 getEquation()

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

Return the equation.

See also

[setEquation\(\)](#)

9.52.3.2 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\(\)](#)

9.52.4 Member Data Documentation

9.52.4.1 mImpl

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

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

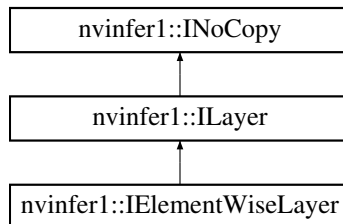
- [NvInfer.h](#)

9.53 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](#)

9.53.1 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.

9.53.2 Constructor & Destructor Documentation

9.53.2.1 ~IElementWiseLayer()

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

9.53.3 Member Function Documentation

9.53.3.1 getOperation()

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

Get the binary operation for the layer.

See also

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

9.53.3.2 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\(\)](#)

9.53.4 Member Data Documentation

9.53.4.1 mImpl

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

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

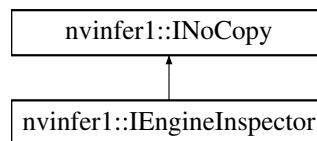
- [NvInfer.h](#)

9.54 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.
- `AsciiChar 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.
- `AsciiChar 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

9.54.1 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](#)

9.54.2 Constructor & Destructor Documentation

9.54.2.1 ~IEngineInspector()

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

9.54.3 Member Function Documentation

9.54.3.1 getEngineInformation()

```
AsciiChar const * nvinfer1::IEngineInspector::getEngineInformation (
    LayerInformationFormat format ) const [inline], [noexcept]
```

Get a string describing the information about all the layers in the current engine or the execution context.

Parameters

<i>layerIndex</i>	the index of the layer. It must lie in range [0, engine.getNbLayers()).
<i>format</i>	the format the layer information should be printed in.

Returns

A null-terminated C-style string describing the information about all the layers in the current engine or the execution context.

Warning

The content of the returned string may change when another execution context has been set, or when another [getLayerInformation\(\)](#) or [getEngineInformation\(\)](#) has been called.

In a multi-threaded environment, this function must be protected from other threads changing the inspection source. If the inspection source changes, the data that is being pointed to can change. Copy the string to another buffer before releasing the lock in order to guarantee consistency.

See also

[LayerInformationFormat](#)

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

See also

[setErrorRecorder\(\)](#)

9.54.3.3 `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\(\)](#)

9.54.3.4 `getLayerInformation()`

```
AsciiChar 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](#)

9.54.3.5 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\(\)](#)

9.54.3.6 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.

9.54.4 Member Data Documentation

9.54.4.1 mImpl

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

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

- [NvInferRuntime.h](#)

9.55 nvinfer1::IErrorRecorder Class Reference

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

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

Public Types

- using [ErrorDesc](#) = char const *
- using [RefCount](#) = int32_t

Public Member Functions

- [IErrorRecorder](#) ()=default
- virtual [~IErrorRecorder](#) () noexcept=default
- virtual int32_t [getNbErrors](#) () const noexcept=0
Return the number of errors.
- virtual [ErrorCode](#) [getErrorCode](#) (int32_t errorIdx) const noexcept=0
Returns the ErrorCode enumeration.
- virtual [ErrorDesc](#) [getErrorDesc](#) (int32_t errorIdx) const noexcept=0
Returns a null-terminated C-style string description of the error.
- virtual bool [hasOverflowed](#) () const noexcept=0
Determine if the error stack has overflowed.
- virtual void [clear](#) () noexcept=0
Clear the error stack on the error recorder.
- virtual bool [reportError](#) ([ErrorCode](#) val, [ErrorDesc](#) desc) noexcept=0
Report an error to the error recorder with the corresponding enum and description.
- virtual [RefCount](#) [incRefCount](#) () noexcept=0
Increments the refcount for the current ErrorRecorder.
- virtual [RefCount](#) [decRefCount](#) () noexcept=0
Decrements the refcount for the current ErrorRecorder.

Static Public Attributes

- static constexpr size_t [kMAX_DESC_LENGTH](#) {127U}

9.55.1 Detailed Description

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

The error reporting mechanism is a user defined object that interacts with the internal state of the object that it is assigned to in order to determine information about abnormalities in execution. The error recorder gets both an error enum that is more descriptive than pass/fail and also a string description that gives more detail on the exact failure modes. In the safety context, the error strings are all limited to 1024 characters in length.

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

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

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

9.55.2 Member Typedef Documentation

9.55.2.1 ErrorDesc

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

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

9.55.2.2 RefCount

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

A typedef of a 32bit integer for reference counting.

9.55.3 Constructor & Destructor Documentation

9.55.3.1 IErrorRecorder()

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

9.55.3.2 ~IErrorRecorder()

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

9.55.4 Member Function Documentation

9.55.4.1 clear()

```
virtual void nvinfer1::IErrorRecorder::clear ( ) [pure virtual], [noexcept]
```

Clear the error stack on the error recorder.

Removes all the tracked errors by the error recorder. This function must guarantee that after this function is called, and as long as no error occurs, the next call to `getNbErrors` will return zero.

See also

[getNbErrors](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.55.4.2 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.

9.55.4.3 `getErrorCode()`

```
virtual ErrorCode nvinfer1::IErrorRecorder::getErrorCode (
    int32_t errorIdx ) const [pure virtual], [noexcept]
```

Returns the ErrorCode enumeration.

Parameters

<i>errorIdx</i>	A 32-bit integer that indexes into the error array.
-----------------	---

The errorIdx specifies what error code from 0 to `getNbErrors()-1` that the application wants to analyze and return the error code enum.

Returns

Returns the enum corresponding to errorIdx.

See also

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

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.55.4.4 `getErrorDesc()`

```
virtual ErrorDesc nvinfer1::IErrorRecorder::getErrorDesc (
    int32_t errorIdx ) const [pure virtual], [noexcept]
```

Returns a null-terminated C-style string description of the error.

Parameters

<i>errorIdx</i>	A 32-bit integer that indexes into the error array.
-----------------	---

For the error specified by the idx value, return the string description of the error. The error string is a null-terminated C-style string. In the safety context there is a constant length requirement to remove any dynamic memory allocations and the error message may be truncated. The format of the string is "`<EnumAsStr> - <Description>`".

Returns

Returns a string representation of the error along with a description of the error.

See also

[getErrorCode](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.55.4.5 getNbErrors()

```
virtual int32_t nvinfer1::IErrorRecorder::getNbErrors ( ) const [pure virtual], [noexcept]
```

Return the number of errors.

Determines the number of errors that occurred between the current point in execution and the last time that the [clear\(\)](#) was executed. Due to the possibility of asynchronous errors occurring, a TensorRT API can return correct results, but still register errors with the Error Recorder. The value of `getNbErrors` must monotonically increase until [clear\(\)](#) is called.

Returns

Returns the number of errors detected, or 0 if there are no errors.

See also

[clear](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.55.4.6 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.

9.55.4.7 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.

9.55.4.8 reportError()

```
virtual bool nvinfer1::IErrorRecorder::reportError (
    ErrorCode val,
    ErrorDesc desc ) [pure virtual], [noexcept]
```

Report an error to the error recorder with the corresponding enum and description.

Parameters

<i>val</i>	The error code enum that is being reported.
<i>desc</i>	The string description of the error.

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

Returns

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

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.55.5 Member Data Documentation

9.55.5.1 kMAX_DESC_LENGTH

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

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

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

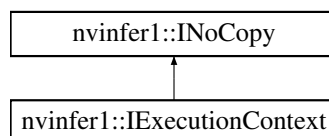
- [NvInferRuntimeCommon.h](#)

9.56 nvinfer1::IExecutionContext Class Reference

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

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

Inheritance diagram for nvinfer1::IExecutionContext:



Public Member Functions

- virtual `~IExecutionContext ()` noexcept=default
- `TRT_DEPRECATED` `bool execute (int32_t batchSize, void *const *bindings)` noexcept
Synchronously execute inference on a batch.
- `TRT_DEPRECATED` `bool enqueue (int32_t batchSize, void *const *bindings, cudaStream_t stream, cudaEvent_t *inputConsumed)` noexcept
Asynchronously execute inference on a batch.
- void `setDebugSync (bool sync)` noexcept
Set the debug sync flag.
- bool `getDebugSync ()` const noexcept
Get the debug sync flag.
- void `setProfiler (IProfiler *profiler)` noexcept
Set the profiler.
- `IProfiler *` `getProfiler ()` const noexcept
Get the profiler.
- `ICudaEngine` const & `getEngine ()` const noexcept
Get the associated engine.
- `TRT_DEPRECATED` void `destroy ()` noexcept
Destroy this object.
- void `setName (char const *name)` noexcept
Set the name of the execution context.
- char const * `getName ()` const noexcept
Return the name of the execution context.
- void `setDeviceMemory (void *memory)` noexcept
Set the device memory for use by this execution context.
- `Dims` `getStrides (int32_t bindingIndex)` const noexcept
Return the strides of the buffer for the given binding.
- `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.
- bool `setBindingDimensions (int32_t bindingIndex, Dims dimensions)` noexcept
Set the dynamic dimensions of a binding.
- `Dims` `getBindingDimensions (int32_t bindingIndex)` const noexcept
Get the dynamic dimensions of a binding.
- bool `setInputShapeBinding (int32_t bindingIndex, int32_t const *data)` noexcept
Set values of input tensor required by shape calculations.
- 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.
 - bool [enqueueV2](#) (void *const *bindings, cudaStream_t stream, cudaEvent_t *inputConsumed) noexcept
Asynchronously execute inference.
 - bool [setOptimizationProfileAsync](#) (int32_t profileIndex, cudaStream_t stream) noexcept
Select an optimization profile for the current context with async semantics.
 - void [setEnqueueEmitsProfile](#) (bool enqueueEmitsProfile) noexcept
Set whether enqueue emits layer timing to the profiler.
 - bool [getEnqueueEmitsProfile](#) () const noexcept
Get the enqueueEmitsProfile state.
 - bool [reportToProfiler](#) () const noexcept
Calculate layer timing info for the current optimization profile in [IExecutionContext](#) and update the profiler after one iteration of inference launch.

Protected Attributes

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

Additional Inherited Members

9.56.1 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.

9.56.2 Constructor & Destructor Documentation

9.56.2.1 ~IExecutionContext()

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

9.56.3 Member Function Documentation

9.56.3.1 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.

See also

[setBindingDimensions\(bindingIndex,dimensions\)](#)

9.56.3.2 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.

See also

[isShapeBinding\(bindingIndex\)](#)

9.56.3.3 destroy()

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

Destroy this object.

Deprecated Use `delete` instead. Deprecated in TRT 8.0.

Warning

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

9.56.3.4 `enqueue()`

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

Asynchronously execute inference on a batch.

This method requires an array of input and output buffers. The mapping from tensor names to indices can be queried using `ICudaEngine::getBindingIndex()`

Parameters

<i>batchSize</i>	The batch size. This is at most the max batch size value supplied to the builder when the engine was built. If the network is created with <code>NetworkDefinitionCreationFlag::kEXPLICIT_BATCH</code> flag, please use <code>enqueueV2()</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.
<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.

9.56.3.5 enqueueV2()

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

Asynchronously execute inference.

This method requires an array of input and output buffers. The mapping from tensor names to indices can be queried using [ICudaEngine::getBindingIndex\(\)](#). This method only works for execution contexts built with full dimension networks.

Parameters

<i>bindings</i>	An array of pointers to input and output buffers for the network.
<i>stream</i>	A cuda stream on which the inference kernels will be enqueued
<i>inputConsumed</i>	An optional event which will be signaled when the input buffers can be refilled with new data

Returns

True if the kernels were enqueued successfully.

See also

[ICudaEngine::getBindingIndex\(\)](#) [ICudaEngine::getMaxBatchSize\(\)](#)

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.

9.56.3.6 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 NetworkDefinitionCreationFlag::kEXPLICIT_BATCH flag, please use executeV2() instead, and this batchSize 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\(\)](#)

9.56.3.7 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\(\)](#)

9.56.3.8 `getBindingDimensions()`

```
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](#).

See also

[ICudaEngine::getProfileDimensions](#)

9.56.3.9 `getDebugSync()`

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

Get the debug sync flag.

See also

[setDebugSync\(\)](#)

9.56.3.10 getEngine()

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

Get the associated engine.

See also

[ICudaEngine](#)

9.56.3.11 getEnqueueEmitsProfile()

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

Get the enqueueEmitsProfile state.

Returns

The enqueueEmitsProfile state.

See also

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

9.56.3.12 getErrorRecorder()

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

Get the ErrorRecorder assigned to this interface.

Retrieves the assigned error recorder object for the given class. A nullptr will be returned if an error handler has not been set.

Returns

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

See also

[setErrorRecorder\(\)](#)

9.56.3.13 getName()

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

Return the name of the execution context.

See also

[setName\(\)](#)

9.56.3.14 getOptimizationProfile()

```
int32_t nvinfer1::IExecutionContext::getOptimizationProfile ( ) const [inline], [noexcept]
```

Get the index of the currently selected optimization profile.

If the profile index has not been set yet (implicitly to 0 for the first execution context to be created, or explicitly for all subsequent contexts), an invalid value of -1 will be returned and all calls to [enqueueV2\(\)](#) or [executeV2\(\)](#) will fail until a valid profile index has been set.

9.56.3.15 getProfiler()

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

Get the profiler.

See also

[IProfiler](#) [setProfiler\(\)](#)

9.56.3.16 getShapeBinding()

```
bool nvinfer1::IExecutionContext::getShapeBinding (
    int32_t bindingIndex,
    int32_t * data ) const [inline], [noexcept]
```

Get values of an input tensor required for shape calculations or an output tensor produced by shape calculations.

Parameters

<i>bindingIndex</i>	index of an input or output tensor for which <code>ICudaEngine::isShapeBinding(bindingIndex)</code> is true.
<i>data</i>	pointer to where values will be written. The number of values written is the product of the dimensions returned by <code>getBindingDimensions(bindingIndex)</code> .

If `ICudaEngine::bindingIsInput(bindingIndex)` is false, then both `allInputDimensionsSpecified()` and `allInputShapesSpecified()` must be true before calling this method. The method will also fail if no valid optimization profile has been set for the current execution context, i.e. if `getOptimizationProfile()` returns -1.

See also

`isShapeBinding(bindingIndex)`

9.56.3.17 getStrides()

```
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.
---------------------	--------------------

See also

`getBindingComponentsPerElement`

9.56.3.18 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\(\)](#)

9.56.3.19 setBindingDimensions()

```
bool nvinfer1::IExecutionContext::setBindingDimensions (
    int32_t bindingIndex,
    Dims dimensions ) [inline], [noexcept]
```

Set the dynamic dimensions of a 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`.

See also

[ICudaEngine::getBindingIndex](#)

9.56.3.20 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()`.

See also

[getDebugSync\(\)](#)

9.56.3.21 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()` to run the network, the memory is in use from the invocation of `enqueueV2()` 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\(\)](#)

9.56.3.22 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\(\)](#)

9.56.3.23 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\(\)](#)

9.56.3.24 setInputShapeBinding()

```
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`.

9.56.3.25 setName()

```
void nvinfer1::IExecutionContext::setName (
    char const * name ) [inline], [noexcept]
```

Set the name of the execution context.

This method copies the name string.

See also

[getName\(\)](#)

9.56.3.26 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\(\)](#) or [enqueueV2\(\)](#).

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\(\)](#) operations. To avoid these calls during [enqueueV2\(\)](#), use [setOptimizationProfileAsync\(\)](#) instead.

If the associated CUDA engine has dynamic inputs, this method must be called at least once with a unique `profileIndex` before calling `execute` or `enqueue` (i.e. the profile index may not be in use by another execution context that has not been destroyed yet). For the first execution context that is created for an engine, `setOptimizationProfile(0)` is called implicitly.

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 either [executeV2\(\)](#) or [enqueueV2\(\)](#).

Warning

This function will trigger layer resource updates on the next call of [enqueueV2\(\)/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\(\)](#)

9.56.3.27 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 cudaMemcpyAsyncs 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\(\)](#) or [enqueueV2\(\)](#). If the associated CUDA engine has inputs with dynamic shapes, the optimization profile must be set with a unique `profileIndex` before calling `execute` or `enqueue`. For the first execution context that is created for an engine, `setOptimizationProfile(0)` is called implicitly.

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 either [executeV2\(\)](#) or [enqueueV2\(\)](#).

Warning

This function will trigger layer resource updates on the next call of [enqueueV2\(\)/executeV2\(\)](#), 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\(\)](#)

9.56.3.28 setProfiler()

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

Set the profiler.

See also

[IProfiler getProfiler\(\)](#)

9.56.4 Member Data Documentation

9.56.4.1 mImpl

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

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

- [NvInferRuntime.h](#)

9.57 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 [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 bool [enqueueV2](#) (void *const *const bindings, cudaStream_t const stream, cudaEvent_t *const input←Consumed) noexcept=0
Asynchronously execute inference on a batch.
- [IExecutionContext](#) ()=default
- virtual [~IExecutionContext](#) () noexcept=default
- [IExecutionContext](#) ([IExecutionContext](#) const &)=delete
- [IExecutionContext](#) ([IExecutionContext](#) &&)=delete
- [IExecutionContext](#) & operator= ([IExecutionContext](#) const &) &=delete
- [IExecutionContext](#) & operator= ([IExecutionContext](#) &&) &=delete
- virtual void [setErrorBuffer](#) ([FloatingPointErrorInformation](#) *const buffer) noexcept=0
Set error buffer output for floating point errors.
- virtual [FloatingPointErrorInformation](#) * [getErrorBuffer](#) () const noexcept=0
Get error buffer output for floating point errors.

9.57.1 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 [IExecutionContext](#) from multiple threads at any given time. Each concurrent execution must have its own instance of an [IExecutionContext](#).

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

9.57.2 Constructor & Destructor Documentation

9.57.2.1 IExecutionContext() [1/3]

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

9.57.2.2 ~IExecutionContext()

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

9.57.2.3 IExecutionContext() [2/3]

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

9.57.2.4 IExecutionContext() [3/3]

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

9.57.3 Member Function Documentation

9.57.3.1 enqueueV2()

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

Asynchronously execute inference on a batch.

This method requires an array of input and output buffers. The mapping from tensor names to indices can be queried using [safe::ICudaEngine::getBindingIndex\(\)](#). This method only works for an execution context built from a network without an implicit batch dimension.

Parameters

<i>bindings</i>	An array of pointers to input and output buffers for the network.
<i>stream</i>	A cuda stream on which the inference kernels will be enqueued.
<i>inputConsumed</i>	An optional event which will be signaled when the input buffers can be refilled with new data.

Returns

True if the kernels were enqueued successfully.

See also

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

Usage considerations

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

9.57.3.2 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

9.57.3.3 `getErrorBuffer()`

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

Get error buffer output for floating point errors.

Returns

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

See also

[setErrorBuffer\(\)](#)

Usage considerations

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

9.57.3.4 `getErrorRecorder()`

```
virtual IErrorRecorder * nvinfer1::safe::IExecutionContext::getErrorRecorder ( ) const [pure  
virtual], [noexcept]
```

get the ErrorRecorder assigned to this interface.

Retrieves the assigned error recorder object for the given class. A default error recorder does not exist, so a nullptr will be returned if `setErrorRecorder` has not been called.

Returns

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

See also

[setErrorRecorder\(\)](#)

Usage considerations

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

9.57.3.5 getName()

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

Return the name of the execution context.

See also

[setName\(\)](#)

Usage considerations

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

9.57.3.6 getStrides()

```
virtual 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.
---------------------	--------------------

Usage considerations

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

9.57.3.7 operator=() [1/2]

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

9.57.3.8 operator=() [2/2]

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

9.57.3.9 setDeviceMemory()

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

Set the device memory for use by this execution context.

If using [enqueueV2\(\)](#) to run the network, The memory is in use from the invocation of [enqueueV2\(\)](#) until network execution is complete. Releasing or otherwise using the memory for other purposes during this time will result in undefined behavior.

Warning

Do not release or use for other purposes the memory set here during network execution.

See also

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

Usage considerations

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

9.57.3.10 setErrorBuffer()

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

Set error buffer output for floating point errors.

The error buffer output must be allocated in device memory and will be used for subsequent calls to [enqueueV2](#). Checking the contents of the error buffer after inference is the responsibility of the application. The pointer passed here must have alignment adequate for the [FloatingPointErrorInformation](#) struct.

Warning

Do not release or use the contents of the error buffer for any other purpose before synchronizing on the CUDA stream passed to `enqueueV2`.

Parameters

<i>buffer</i>	The device memory to use as floating point error buffer
---------------	---

See also

[getErrorBuffer\(\)](#)

Usage considerations

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

9.57.3.11 setErrorRecorder()

```
virtual void nvinfer1::safe::IExecutionContext::setErrorRecorder (
    IErrorRecorder *const recorder ) [pure virtual], [noexcept]
```

Set the ErrorRecorder for this interface.

Assigns the ErrorRecorder to this interface. The ErrorRecorder will track all errors during execution. This function will call `incRefCount` of the registered ErrorRecorder at least once. Setting recorder to `nullptr` unregisters the recorder with the interface, resulting in a call to `decRefCount` if a recorder has been registered.

Parameters

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

See also

[getErrorRecorder\(\)](#)

Usage considerations

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

9.57.3.12 setName()

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

Set the name of the execution context.

This method copies the name string.

Warning

Strings passed to the runtime must be 1024 characters or less including NULL terminator and must be NULL terminated.

See also

[getName\(\)](#)

Usage considerations

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

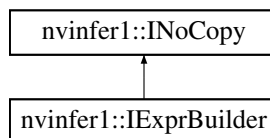
The documentation for this class was generated from the following file:

- [NvInferSafeRuntime.h](#)

9.58 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`

9.58.1 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](#)

9.58.2 Constructor & Destructor Documentation

9.58.2.1 ~IExprBuilder()

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

9.58.3 Member Function Documentation

9.58.3.1 constant()

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

Return pointer to `IDimensionExp` for given value.

9.58.3.2 operation()

```
IDimensionExpr const * nvinfer1::IExprBuilder::operation (
    DimensionOperation op,
    IDimensionExpr const & first,
    IDimensionExpr const & second ) [inline], [noexcept]
```

Return pointer to IDimensionExp that represents the given operation applied to first and second. Returns nullptr if op is not a valid DimensionOperation.

9.58.4 Member Data Documentation

9.58.4.1 mImpl

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

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

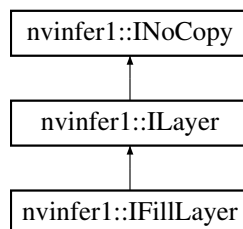
- [NvInferRuntime.h](#)

9.59 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](#)

9.59.1 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 a 1D Int32 or Float tensor with length not exceeding `2*Dims::MAX_DIMS`.
- 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.

9.59.2 Constructor & Destructor Documentation

9.59.2.1 ~IFillLayer()

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

9.59.3 Member Function Documentation

9.59.3.1 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](#)

9.59.3.2 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](#)

9.59.3.3 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](#)

9.59.3.4 getOperation()

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

Get the fill operation for the layer.

See also

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

9.59.3.5 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;

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

See also

[getAlpha](#)

9.59.3.6 setBeta()

```
void nvinfer1::IFillLayer::setBeta (
    double beta ) [inline], [noexcept]
```

Set the beta parameter.

Parameters

<i>beta</i>	has different meanings for each operator:
-------------	---

Operation | Usage `kLinspace` | the delta value, defaults to 1.0; `kRandomUniform` | the maximal value, defaults to 1.0;

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

See also

[getBeta](#)

9.59.3.7 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](#)

9.59.3.8 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.

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.

9.59.3.9 setOperation()

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

Set the fill operation for the layer.

See also

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

9.59.4 Member Data Documentation

9.59.4.1 mImpl

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

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

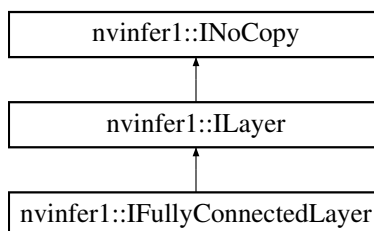
- [NvInfer.h](#)

9.60 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`

9.60.1 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 Use [IMatrixMultiplyLayer](#) instead. Deprecated in TensorRT 8.4.

9.60.2 Constructor & Destructor Documentation

9.60.2.1 ~IFullyConnectedLayer()

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

9.60.3 Member Function Documentation

9.60.3.1 getBiasWeights()

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

Get the bias weights.

See also

[setBiasWeightsWeights\(\)](#)

9.60.3.2 getKernelWeights()

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

Get the kernel weights.

See also

[setKernelWeights\(\)](#)

9.60.3.3 `getNbOutputChannels()`

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

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

See also

[setNbOutputChannels\(\)](#)

9.60.3.4 `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\(\)](#)

9.60.3.5 `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.

9.60.3.6 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\(\)](#)

9.60.3.7 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\(\)](#)

9.60.4 Member Data Documentation

9.60.4.1 mImpl

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

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

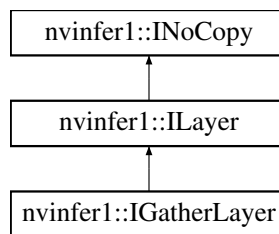
- [NvInfer.h](#)

9.61 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`

9.61.1 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.

9.61.2 Constructor & Destructor Documentation

9.61.2.1 ~IGatherLayer()

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

9.61.3 Member Function Documentation

9.61.3.1 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\(\)](#)

9.61.3.2 getMode()

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

Get the gather mode.

See also

[setMode\(\)](#)

9.61.3.3 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\(\)](#)

9.61.3.4 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\(\)](#)

9.61.3.5 setMode()

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

Set the gather mode.

See also

[getMode\(\)](#)

9.61.3.6 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\(\)](#)

9.61.4 Member Data Documentation

9.61.4.1 mImpl

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

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

- [NvInfer.h](#)

9.62 nvinfer1::IGpuAllocator Class Reference

Application-implemented class for controlling allocation on the GPU.

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

Public Member Functions

- virtual void * [allocate](#) (uint64_t const size, uint64_t const alignment, [AllocatorFlags](#) const flags) noexcept=0
- virtual [TRT_DEPRECATED](#) void [free](#) (void *const memory) noexcept=0
- virtual [~IGpuAllocator](#) ()=default
- [IGpuAllocator](#) ()=default
- virtual void * [reallocate](#) (void *, uint64_t, uint64_t) noexcept
- virtual bool [deallocate](#) (void *const memory) noexcept

9.62.1 Detailed Description

Application-implemented class for controlling allocation on the GPU.

9.62.2 Constructor & Destructor Documentation

9.62.2.1 [~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.

9.62.2.2 IGpuAllocator()

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

9.62.3 Member Function Documentation

9.62.3.1 allocate()

```
virtual void * nvinfer1::IGpuAllocator::allocate (
    uint64_t const size,
    uint64_t const alignment,
    AllocatorFlags const flags ) [pure virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle acquisition of GPU memory.

Parameters

<i>size</i>	The size of the memory required.
<i>alignment</i>	The required alignment of memory. Alignment will be zero or a power of 2 not exceeding the alignment guaranteed by <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.

If an allocation request of size 0 is made, `nullptr` should be returned.

If an allocation request cannot be satisfied, `nullptr` should be returned.

Note

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

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads.

9.62.3.2 deallocate()

```
virtual bool nvinfer1::IGpuAllocator::deallocate (
    void *const memory ) [inline], [virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle release of GPU memory.

TensorRT may pass a `nullptr` to this function if it was previously returned by [allocate\(\)](#).

Parameters

<i>memory</i>	The acquired memory.
---------------	----------------------

Returns

True if the acquired memory is released successfully.

Note

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

If user-implemented [free\(\)](#) might hit an error condition, the user should override [deallocate\(\)](#) as the primary implementation and override [free\(\)](#) to call [deallocate\(\)](#) for backwards compatibility.

See also

[free\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads.

9.62.3.3 free()

```
virtual TRT\_DEPRECATED void nvinfer1::IGpuAllocator::free (  
    void *const memory ) [pure virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle release of GPU memory.

TensorRT may pass a nullptr to this function if it was previously returned by [allocate\(\)](#).

Parameters

<i>memory</i>	The acquired memory.
---------------	----------------------

Note

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

See also

[deallocate\(\)](#)

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

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.

9.62.3.4 `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 $\min(\text{oldSize}, \text{newSize})$ bytes unchanged and return the original address
- move $\min(\text{oldSize}, \text{newSize})$ bytes to a new location of sufficient size and return its address
- return `nullptr`, to indicate that the request could not be fulfilled.

If `nullptr` is returned, TensorRT will assume that `resize()` is not implemented, and that the allocation at `baseAddr` is still valid.

This method is made available for use cases where delegating the resize strategy to the application provides an opportunity to improve memory management. One possible implementation is to allocate a large virtual device buffer and progressively commit physical memory with `cuMemMap`. `CU_MEM_ALLOC_GRANULARITY_RECOMMENDED` is suggested in this case.

TensorRT may call `realloc` to increase the buffer by relatively small amounts.

Parameters

<i>baseAddr</i>	the address of the original allocation.
<i>alignment</i>	The alignment used by the original allocation.
<i>newSize</i>	The new memory size required.

Returns

the address of the reallocated memory

Note

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

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads.

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

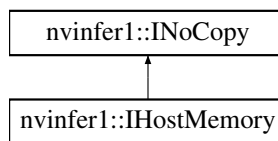
- [NvInferRuntimeCommon.h](#)

9.63 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

9.63.1 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.

9.63.2 Constructor & Destructor Documentation

9.63.2.1 ~IHostMemory()

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

9.63.3 Member Function Documentation

9.63.3.1 data()

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

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

9.63.3.2 destroy()

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

Destroy the allocated memory.

Deprecated Use `delete` instead. Deprecated in TRT 8.0.

Warning

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

9.63.3.3 size()

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

The size in bytes of the data that was allocated.

9.63.3.4 type()

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

The type of the memory that was allocated.

9.63.4 Member Data Documentation**9.63.4.1 mImpl**

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

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

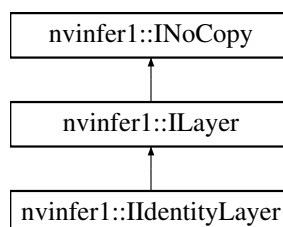
- [NvInferRuntime.h](#)

9.64 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

9.64.1 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)
```

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.

9.64.2 Constructor & Destructor Documentation

9.64.2.1 ~IIdentityLayer()

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

9.64.3 Member Data Documentation

9.64.3.1 mImpl

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

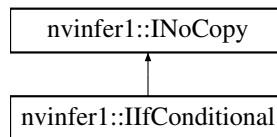
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.65 nvinfer1::IIfConditional Class Reference

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

Inheritance diagram for nvinfer1::IIfConditional:



Public Member Functions

- [IConditionLayer](#) * [setCondition](#) ([ITensor](#) &condition) noexcept
Set the condition tensor for this If-Conditional construct.
- [IIfConditionalOutputLayer](#) * [addOutput](#) ([ITensor](#) &>trueSubgraphOutput, [ITensor](#) &>falseSubgraphOutput) noexcept
Add an If-conditional output.
- [IIfConditionalInputLayer](#) * [addInput](#) ([ITensor](#) &input) noexcept
Add an If-conditional input.
- void [setName](#) (char const *name) noexcept
Set the name of the conditional.
- char const * [getName](#) () const noexcept
Return the name of the conditional.

Protected Member Functions

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

Protected Attributes

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

9.65.1 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.

9.65.2 Constructor & Destructor Documentation

9.65.2.1 `~IIfConditional()`

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

9.65.3 Member Function Documentation

9.65.3.1 `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](#)

9.65.3.2 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](#)

9.65.3.3 getName()

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

Return the name of the conditional.

See also

[setName\(\)](#)

9.65.3.4 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](#)

9.65.3.5 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.

See also

[getName\(\)](#)

9.65.4 Member Data Documentation

9.65.4.1 mImpl

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

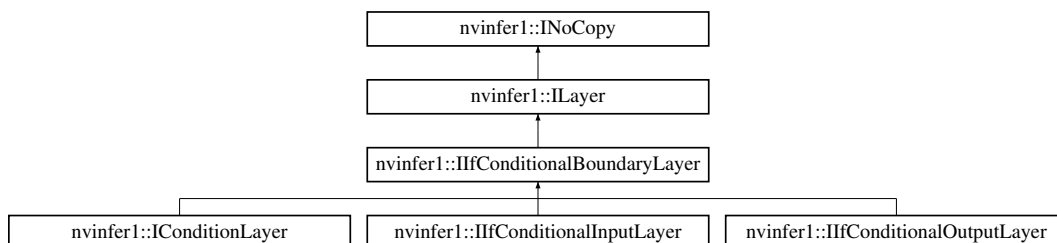
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.66 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`

9.66.1 Detailed Description

This is a base class for Conditional boundary layers.

Boundary layers are used to demarcate the boundaries of Conditionals.

9.66.2 Constructor & Destructor Documentation

9.66.2.1 `~IIfConditionalBoundaryLayer()`

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

9.66.3 Member Function Documentation

9.66.3.1 `getConditional()`

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

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

9.66.4 Member Data Documentation

9.66.4.1 mBoundary

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

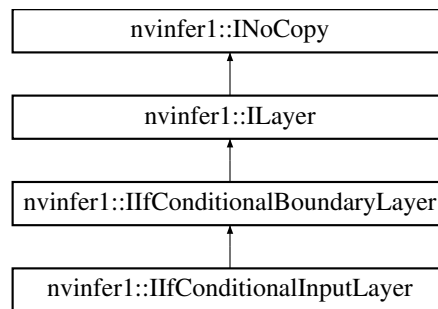
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.67 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

9.67.1 Detailed Description

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

9.67.2 Constructor & Destructor Documentation

9.67.2.1 ~IIfConditionalInputLayer()

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

9.67.3 Member Data Documentation

9.67.3.1 mImpl

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

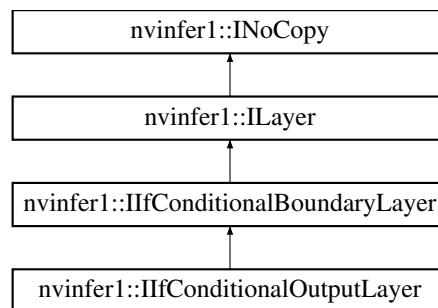
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.68 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

9.68.1 Detailed Description

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

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

9.68.2 Constructor & Destructor Documentation

9.68.2.1 ~IIfConditionalOutputLayer()

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

9.68.3 Member Data Documentation

9.68.3.1 mImpl

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

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

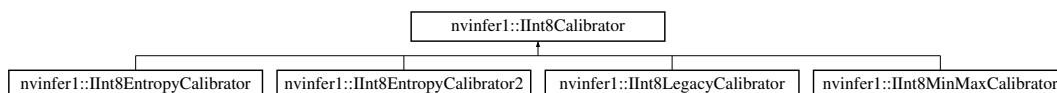
- [NvInfer.h](#)

9.69 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

9.69.1 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.

9.69.2 Constructor & Destructor Documentation

9.69.2.1 ~IInt8Calibrator()

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

9.69.3 Member Function Documentation

9.69.3.1 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](#).

9.69.3.2 `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\(\)](#)

9.69.3.3 `getBatchSize()`

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

Get the batch size used for calibration batches.

Returns

The batch size.

9.69.3.4 `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 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.

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.

9.69.3.5 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\(\)](#)

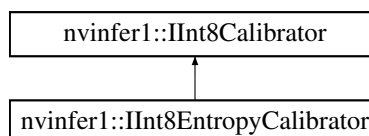
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.70 nvinfer1::IInt8EntropyCalibrator Class Reference

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

Inheritance diagram for nvinfer1::IInt8EntropyCalibrator:



Public Member Functions

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

9.70.1 Detailed Description

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

9.70.2 Constructor & Destructor Documentation

9.70.2.1 ~IInt8EntropyCalibrator()

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

9.70.3 Member Function Documentation

9.70.3.1 getAlgorithm()

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

Signal that this is the entropy calibrator.

Implements [nvinfer1::IInt8Calibrator](#).

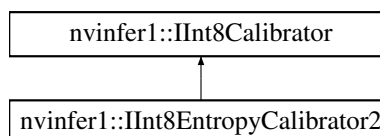
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.71 nvinfer1::IInt8EntropyCalibrator2 Class Reference

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

Inheritance diagram for nvinfer1::IInt8EntropyCalibrator2:



Public Member Functions

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

9.71.1 Detailed Description

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

9.71.2 Constructor & Destructor Documentation

9.71.2.1 ~IInt8EntropyCalibrator2()

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

9.71.3 Member Function Documentation

9.71.3.1 getAlgorithm()

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

Signal that this is the entropy calibrator 2.

Implements [nvinfer1::IInt8Calibrator](#).

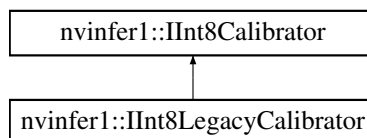
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.72 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

9.72.1 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.

9.72.2 Constructor & Destructor Documentation

9.72.2.1 `~IInt8LegacyCalibrator()`

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

9.72.3 Member Function Documentation

9.72.3.1 `getAlgorithm()`

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

Signal that this is the legacy calibrator.

Implements [nvinfer1::IInt8Calibrator](#).

9.72.3.2 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.

9.72.3.3 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

9.72.3.4 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.

9.72.3.5 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\(\)](#)

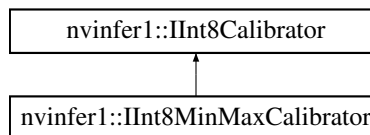
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.73 nvinfer1::IInt8MinMaxCalibrator Class Reference

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

Inheritance diagram for nvinfer1::IInt8MinMaxCalibrator:



Public Member Functions

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

9.73.1 Detailed Description

MinMax Calibrator. It supports per activation tensor scaling.

9.73.2 Constructor & Destructor Documentation

9.73.2.1 ~IInt8MinMaxCalibrator()

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

9.73.3 Member Function Documentation

9.73.3.1 getAlgorithm()

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

Signal that this is the MinMax Calibrator.

Implements [nvinfer1::IInt8Calibrator](#).

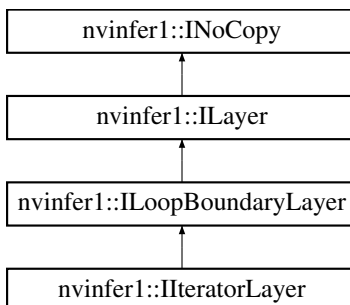
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.74 nvinfer1::IteratorLayer Class Reference

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

Inheritance diagram for nvinfer1::IteratorLayer:



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 `~IIteratorLayer () noexcept=default`

Protected Attributes

- `apiv::VIteratorLayer * mImpl`

9.74.1 Constructor & Destructor Documentation

9.74.1.1 `~IIteratorLayer()`

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

9.74.2 Member Function Documentation

9.74.2.1 `getAxis()`

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

Get axis being iterated over.

9.74.2.2 `getReverse()`

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

True if and only if reversing input.

9.74.2.3 `setAxis()`

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

Set axis to iterate over.

9.74.2.4 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.

9.74.3 Member Data Documentation

9.74.3.1 mImpl

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

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

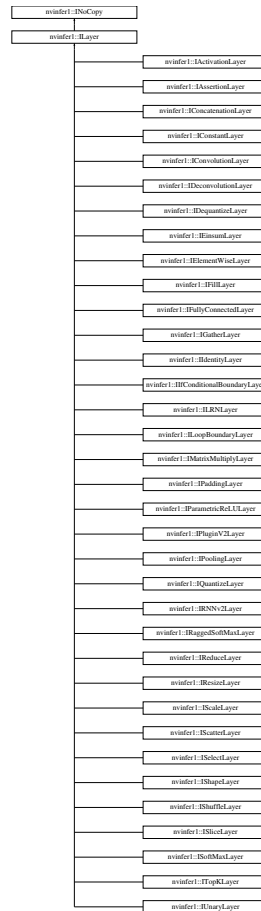
- [NvInfer.h](#)

9.75 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

Protected Member Functions

- `virtual ~ILayer ()` noexcept=default

Protected Attributes

- `apiv::VLayer * mLayer`

9.75.1 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.

9.75.2 Constructor & Destructor Documentation

9.75.2.1 ~ILayer()

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

9.75.3 Member Function Documentation

9.75.3.1 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](#)).

9.75.3.2 getName()

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

Return the name of a layer.

See also

[setName\(\)](#)

9.75.3.3 getNbInputs()

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

Get the number of inputs of a layer.

9.75.3.4 getNbOutputs()

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

Get the number of outputs of a layer.

9.75.3.5 getOutput()

```
ITensor * nvinfer1::ILayer::getOutput (
    int32_t index ) const [inline], [noexcept]
```

Get the layer output corresponding to the given index.

Returns

The indexed output tensor, or nullptr if the index is out of range or the tensor is optional ([IRNNv2Layer](#)).

9.75.3.6 getOutputType()

```
DataType nvinfer1::ILayer::getOutputType (
    int32_t index ) const [inline], [noexcept]
```

get the output type of this layer

Parameters

<i>index</i>	the index of the output
--------------	-------------------------

Returns

the output precision. If no precision has been set, `DataType::kFLOAT` will be returned, unless the output type is inherently `DataType::kINT32`.

See also

[getOutputType\(\)](#) [outputTypeIsSet\(\)](#) [resetOutputType\(\)](#)

9.75.3.7 getPrecision()

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

get the computational precision of this layer

Returns

the computational precision

See also

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

9.75.3.8 getType()

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

Return the type of a layer.

See also

[LayerType](#)

9.75.3.9 outputTypeIsSet()

```
bool nvinfer1::ILayer::outputTypeIsSet (
    int32_t index ) const [inline], [noexcept]
```

whether the output type has been set for this layer

Parameters

<i>index</i>	the index of the output
--------------	-------------------------

Returns

whether the output type has been explicitly set

See also

[setOutputType\(\)](#) [getOutputType\(\)](#) [resetOutputType\(\)](#)

9.75.3.10 precisionIsSet()

```
bool nvinfer1::ILayer::precisionIsSet ( ) const [inline], [noexcept]
```

whether the computational precision has been set for this layer

Returns

whether the computational precision has been explicitly set

See also

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

9.75.3.11 resetOutputType()

```
void nvinfer1::ILayer::resetOutputType (
    int32_t index ) [inline], [noexcept]
```

reset the output type for this layer

Parameters

<i>index</i>	the index of the output
--------------	-------------------------

See also

[setOutputType\(\)](#) [getOutputType\(\)](#) [outputTypeIsSet\(\)](#)

9.75.3.12 resetPrecision()

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

reset the computational precision for this layer

See also

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

9.75.3.13 setInput()

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

Replace an input of this layer with a specific tensor.

Parameters

<i>index</i>	the index of the input to modify.
<i>tensor</i>	the new input tensor

Except for [IFillLayer](#), [ILoopOutputLayer](#), [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.

9.75.3.14 setName()

```
void nvinfer1::ILayer::setName (
    char const * name ) [inline], [noexcept]
```

Set the name of a layer.

This method copies the name string.

See also

[getName\(\)](#)

9.75.3.15 setOutputType()

```
void nvinfer1::ILayer::setOutputType (
    int32_t index,
    DataType dataType ) [inline], [noexcept]
```

Set the output type of this layer.

Setting the output type constrains TensorRT to choose implementations which generate output data with the given type. If it is not set, TensorRT will select output type based on layer computational precision. TensorRT could still choose non-conforming output type based on fastest implementation. To force choosing the requested output type, set exactly one of the following flags, which differ in what happens if no such implementation exists:

- [BuilderFlag::kOBEY_PRECISION_CONSTRAINTS](#) - build fails with an error message.
- [BuilderFlag::kPREFER_PRECISION_CONSTRAINTS](#) - TensorRT falls back to an implementation with a non-conforming output type.

In case layer precision is not specified, or falling back, the output type depends on the chosen implementation, based on performance considerations and the flags specified to the builder.

This method cannot be used to set the data type of the second output tensor of the TopK layer. The data type of the second output tensor of the topK layer is always Int32. Also the output type of all layers that are shape operations must be [DataType::kINT32](#), and all attempts to set the output type to some other data type will be ignored except for issuing an error message.

Note that the layer output type is generally not identical to the data type of the output tensor, as TensorRT may insert implicit reformatting operations to convert the former to the latter. Calling `layer->setOutputType(i, type)` has no effect on the data type of the *i*-th output tensor of layer, and users need to call `layer->getOutput(i)->setType(type)` to change the tensor data type. This is particularly relevant if the tensor is marked as a network output, since only `setType()` [but not `setOutputType()`] will affect the data representation in the corresponding output binding.

Parameters

<i>index</i>	the index of the output to set
<i>dataType</i>	the type of the output

See also

[getOutputType\(\)](#) [outputTypeIsSet\(\)](#) [resetOutputType\(\)](#)

9.75.3.16 setPrecision()

```
void nvinfer1::ILayer::setPrecision (
    DataType dataType ) [inline], [noexcept]
```

Set the computational precision of this layer.

Setting the precision allows TensorRT to choose an implementation which run at this computational precision. TensorRT could still choose a non-conforming fastest implementation that ignores the requested precision. To force choosing an implementation with the requested precision, set exactly one of the following flags, which differ in what happens if no such implementation exists:

- [BuilderFlag::kOBEY_PRECISION_CONSTRAINTS](#) - build fails with an error message.
- [BuilderFlag::kPREFER_PRECISION_CONSTRAINTS](#) - TensorRT falls back to an implementation without the requested precision.

If precision is not set, or falling back, TensorRT will select the layer computational precision and layer input type based on global performance considerations and the flags specified to the builder.

For a [IdentityLayer](#): If it casts to/from float/half/int8, 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\(\)](#)

9.75.4 Member Data Documentation

9.75.4.1 mLayer

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

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

- [NvInfer.h](#)

9.76 nvinfer1::ILogger Class Reference

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

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

Public Types

- enum class `Severity` : `int32_t` {
`kINTERNAL_ERROR` = 0 , `kERROR` = 1 , `kWARNING` = 2 , `kINFO` = 3 ,
`kVERBOSE` = 4 }

Public Member Functions

- virtual void `log` (`Severity` severity, `AsciiChar` const *msg) noexcept=0
- `ILogger` ()=default
- virtual `~ILogger` ()=default

9.76.1 Detailed Description

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

The logger used to create an instance of `IBuilder`, `IRuntime` or `IRefitter` is used for all objects created through that interface. The logger should be valid until all objects created are released.

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

9.76.2 Member Enumeration Documentation

9.76.2.1 Severity

```
enum class nvinfer1::ILogger::Severity : int32_t [strong]
```

The severity corresponding to a log message.

Enumerator

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

9.76.3 Constructor & Destructor Documentation

9.76.3.1 ILogger()

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

9.76.3.2 ~ILogger()

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

9.76.4 Member Function Documentation

9.76.4.1 log()

```
virtual void nvinfer1::ILogger::log (
    Severity severity,
    AsciiChar const * msg ) [pure virtual], [noexcept]
```

A callback implemented by the application to handle logging messages;

Parameters

<i>severity</i>	The severity of the message.
<i>msg</i>	A null-terminated log message.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime, or if the same logger is used for multiple runtimes, builders, or refitters.

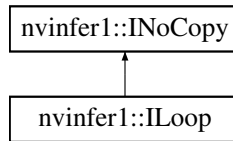
The documentation for this class was generated from the following file:

- [NvInferRuntimeCommon.h](#)

9.77 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](#)

9.77.1 Detailed Description

Helper for creating a recurrent subgraph.

An [ILoop](#) cannot be added to an [INetworkDefinition](#) where [hasImplicitBatchDimensions\(\)](#) returns true.

9.77.2 Constructor & Destructor Documentation

9.77.2.1 ~ILoop()

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

9.77.3 Member Function Documentation

9.77.3.1 addIterator()

```
IIteratorLayer * nvinfer1::ILoop::addIterator (
    ITensor & tensor,
    int32_t axis = 0,
    bool reverse = false ) [inline], [noexcept]
```

Return layer that subscripts tensor by loop iteration.

For reverse=false, this is equivalent to addGather(tensor, I, 0) where I is a scalar tensor containing the loop iteration number. For reverse=true, this is equivalent to addGather(tensor, M-1-I, 0) where M is the trip count computed from TripLimits of kind kCOUNT.

9.77.3.2 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](#).

9.77.3.3 addRecurrence()

```
IRecurrenceLayer * nvinfer1::ILoop::addRecurrence (
    ITensor & initialValue ) [inline], [noexcept]
```

Create a recurrence layer for this loop with initialValue as its first input.

[IRecurrenceLayer](#) requires exactly two inputs. The 2nd input must be added, via method [IRecurrenceLayer::set←Input\(1,...\)](#) before an Engine can be built.

9.77.3.4 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.

9.77.3.5 getName()

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

Return the name of the loop.

See also

[setName\(\)](#)

9.77.3.6 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.

See also

[getName\(\)](#)

9.77.4 Member Data Documentation

9.77.4.1 mImpl

```
apiv::VLoop* nvinfer1::ILoop::mImpl [protected]
```

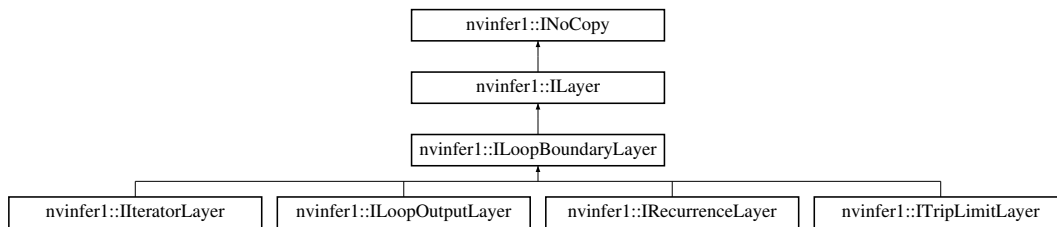
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.78 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`

9.78.1 Constructor & Destructor Documentation

9.78.1.1 ~ILoopBoundaryLayer()

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

9.78.2 Member Function Documentation

9.78.2.1 getLoop()

```
ILoop * nvinfer1::ILoopBoundaryLayer::getLoop ( ) const [inline], [noexcept]
```

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

9.78.3 Member Data Documentation

9.78.3.1 mBoundary

```
apiv::VLoopBoundaryLayer* nvinfer1::ILoopBoundaryLayer::mBoundary [protected]
```

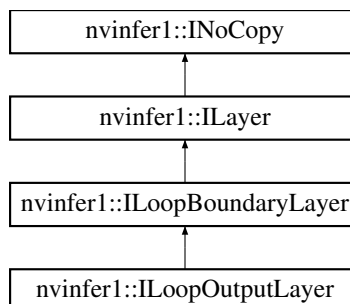
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.79 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 contention 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`

9.79.1 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`.

9.79.2 Constructor & Destructor Documentation

9.79.2.1 `~ILoopOutputLayer()`

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

9.79.3 Member Function Documentation

9.79.3.1 `getAxis()`

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

Get axis being concatenated over.

9.79.3.2 `getLoopOutput()`

```
LoopOutput nvinfer1::ILoopOutputLayer::getLoopOutput ( ) const [inline], [noexcept]
```

9.79.3.3 `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.

9.79.3.4 `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.

9.79.4 Member Data Documentation

9.79.4.1 mImpl

```
apiv::VLoopOutputLayer* nvinfer1::ILoopOutputLayer::mImpl [protected]
```

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

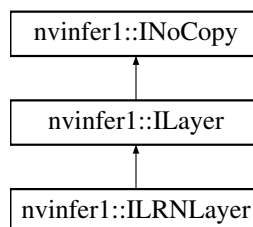
- [NvInfer.h](#)

9.80 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](#)

9.80.1 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.

9.80.2 Constructor & Destructor Documentation

9.80.2.1 [~ILRNLayer](#)()

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

9.80.3 Member Function Documentation

9.80.3.1 getAlpha()

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

Get the LRN alpha value.

See also

[setAlpha\(\)](#)

9.80.3.2 getBeta()

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

Get the LRN beta value.

See also

[setBeta\(\)](#)

9.80.3.3 getK()

```
float nvinfer1::ILRNLayer::getK ( ) const [inline], [noexcept]
```

Get the LRN K value.

See also

[setK\(\)](#)

9.80.3.4 getWindowSize()

```
int32_t nvinfer1::ILRNLayer::getWindowSize ( ) const [inline], [noexcept]
```

Get the LRN window size.

See also

[getWindowStride\(\)](#)

9.80.3.5 setAlpha()

```
void nvinfer1::ILRNLayer::setAlpha (
    float alpha ) [inline], [noexcept]
```

Set the LRN alpha value.

The valid range is [-1e20, 1e20].

See also

[getAlpha\(\)](#)

9.80.3.6 setBeta()

```
void nvinfer1::ILRNLayer::setBeta (
    float beta ) [inline], [noexcept]
```

Set the LRN beta value.

The valid range is [0.01, 1e5f].

See also

[getBeta\(\)](#)

9.80.3.7 setK()

```
void nvinfer1::ILRNLayer::setK (
    float k ) [inline], [noexcept]
```

Set the LRN K value.

The valid range is [1e-5, 1e10].

See also

[getK\(\)](#)

9.80.3.8 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\(\)](#)

9.80.4 Member Data Documentation

9.80.4.1 mImpl

```
apiv::VLRNLayer* nvinfer1::ILRNLayer::mImpl [protected]
```

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

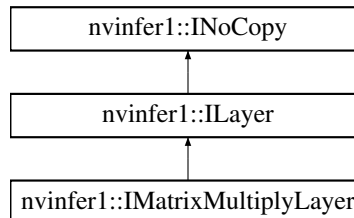
- [NvInfer.h](#)

9.81 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](#)

9.81.1 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.

9.81.2 Constructor & Destructor Documentation

9.81.2.1 ~IMatrixMultiplyLayer()

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

9.81.3 Member Function Documentation

9.81.3.1 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\(\)](#)

9.81.3.2 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\(\)](#)

9.81.4 Member Data Documentation

9.81.4.1 mImpl

```
apiv::VMatrixMultiplyLayer* nvinfer1::IMatrixMultiplyLayer::mImpl [protected]
```

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

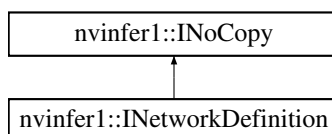
- [NvInfer.h](#)

9.82 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.
- `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 keep←Dimensions)` 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)`
*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.*
- `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.
- `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*.*

- [IParametricReLU](#) * [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.

Protected Attributes

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

Additional Inherited Members

9.82.1 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.

9.82.2 Constructor & Destructor Documentation

9.82.2.1 ~INetworkDefinition()

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

9.82.3 Member Function Documentation

9.82.3.1 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.

9.82.3.2 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.

9.82.3.3 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.

9.82.3.4 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.

9.82.3.5 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. <code>Weights{}</code> 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

9.82.3.6 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. <code>Weights{}</code> 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.

9.82.3.7 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. <code>Weights{}</code> 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

9.82.3.8 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. <code>Weights{}</code> 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.

9.82.3.9 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.

9.82.3.10 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.

9.82.3.11 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.

9.82.3.12 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.

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.

9.82.3.13 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 Use addMatrixMultiply instead. Deprecated in TensorRT 8.4.

9.82.3.14 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.

9.82.3.15 addGatherV2()

```
IGatherLayer * nvinfer1::INetworkDefinition::addGatherV2 (
    ITensor & data,
    ITensor & indices,
    GatherMode mode ) [inline]
```

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.

9.82.3.16 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.

9.82.3.17 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.

9.82.3.18 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.

See also

[ITensor](#)

Returns

The new tensor or nullptr if there is an error.

9.82.3.19 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.

9.82.3.20 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.

9.82.3.21 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.

9.82.3.22 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

9.82.3.23 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 Superseded by addSlice. Deprecated in TensorRT 8.0

9.82.3.24 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.

9.82.3.25 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.

9.82.3.26 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

9.82.3.27 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.

9.82.3.28 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.

9.82.3.29 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.

9.82.3.30 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.

9.82.3.31 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.

9.82.3.32 addRNNv2()

```
TRT_DEPRECATED IRNNv2Layer * nvinfer1::INetworkDefinition::addRNNv2 (
    ITensor & input,
    int32_t layerCount,
    int32_t hiddenSize,
    int32_t maxSeqLen,
    RNNOperation op ) [inline], [noexcept]
```

Add an `layerCount` deep RNN layer to the network with `hiddenSize` internal states that can take a batch with fixed or variable sequence lengths.

Parameters

<i>input</i>	The input tensor to the layer (see below).
<i>layerCount</i>	The number of layers in the RNN.
<i>hiddenSize</i>	Size of the internal hidden state for each layer.
<i>maxSeqLen</i>	Maximum sequence length for the input.
<i>op</i>	The type of RNN to execute.

By default, the layer is configured with `RNNDirection::kUNIDIRECTION` and `RNNInputMode::kLINEAR`. To change these settings, use `IRNNv2Layer::setDirection()` and `IRNNv2Layer::setInputMode()`.

Weights and biases for the added layer should be set using `IRNNv2Layer::setWeightsForGate()` and `IRNNv2Layer::setBiasForGate()` prior to building an engine using this network.

The input tensors must be of the type `DataType::kFLOAT` or `DataType::kHALF`. The layout of the weights is row major and must be the same datatype as the input tensor. `weights` contain 8 matrices and `bias` contains 8 vectors.

See `IRNNv2Layer::setWeightsForGate()` and `IRNNv2Layer::setBiasForGate()` for details on the required input format for `weights` and `bias`.

The input `ITensor` should contain zero or more index dimensions $\{N_1, \dots, N_p\}$, followed by two dimensions, defined as follows:

- `S_max` is the maximum allowed sequence length (number of RNN iterations)
- `E` specifies the embedding length (unless `::kSKIP` is set, in which case it should match `getHiddenSize()`).

By default, all sequences in the input are assumed to be size `maxSeqLen`. To provide explicit sequence lengths for each input sequence in the batch, use [IRNNv2Layer::setSequenceLengths\(\)](#).

The RNN layer outputs up to three tensors.

The first output tensor is the output of the final RNN layer across all timesteps, with dimensions $\{N_1, \dots, N_p, S_{\max}, H\}$:

- $N_1 \dots N_p$ are the index dimensions specified by the input tensor
- `S_max` is the maximum allowed sequence length (number of RNN iterations)
- `H` is an output hidden state (equal to `getHiddenSize()` or $2 \times \text{getHiddenSize}()$)

The second tensor is the final hidden state of the RNN across all layers, and if the RNN is an LSTM (i.e. `getOperation()` is `::kLSTM`), then the third tensor is the final cell state of the RNN across all layers. Both the second and third output tensors have dimensions $\{N_1, \dots, N_p, L, H\}$:

- $N_1 \dots N_p$ are the index dimensions specified by the input tensor
- `L` is the number of layers in the RNN, equal to `getLayerCount()` if `getDirection` is `::kUNIDIRECTION`, and $2 \times \text{getLayerCount}()$ if `getDirection` is `::kBIDIRECTION`. In the bi-directional case, layer `l`'s final forward hidden state is stored in $L = 2 * l$, and final backward hidden state is stored in $L = 2 * l + 1$.
- `H` is the hidden state for each layer, equal to `getHiddenSize()`.

See also

[IRNNv2Layer](#)

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

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.

9.82.3.33 addScale()

```
IScaleLayer * nvinfer1::INetworkDefinition::addScale (
    ITensor & input,
    ScaleMode mode,
    Weights shift,
    Weights scale,
    Weights power ) [inline], [noexcept]
```

Add a Scale layer to the network.

Parameters

<i>input</i>	The input tensor to the layer. This tensor is required to have a minimum of 3 dimensions in implicit batch mode and a minimum of 4 dimensions in explicit batch mode.
<i>mode</i>	The scaling mode.
<i>shift</i>	The shift value.
<i>scale</i>	The scale value.
<i>power</i>	The power value.

If the weights are available, then the size of weights are dependent on the ScaleMode. For `::kUNIFORM`, the number of weights equals 1. For `::kCHANNEL`, the number of weights equals the channel dimension. For `::kELEMENTWISE`, the number of weights equals the product of the last three dimensions of the input.

See also

[addScaleNd](#)

[IScaleLayer](#)

Warning

Int32 tensors are not valid input tensors.

Returns

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

9.82.3.34 addScaleNd()

```
IScaleLayer * nvinfer1::INetworkDefinition::addScaleNd (
    ITensor & input,
    ScaleMode mode,
    Weights shift,
    Weights scale,
    Weights power,
    int32_t channelAxis ) [inline], [noexcept]
```

Add a multi-dimension scale layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
<i>mode</i>	The scaling mode.
<i>shift</i>	The shift value.
<i>scale</i>	The scale value.
<i>power</i>	The power value.
<i>channelAxis</i>	The channel axis.

If the weights are available, then the size of weights are dependent on the ScaleMode. For `::kUNIFORM`, the number of weights equals 1. For `::kCHANNEL`, the number of weights equals the channel dimension. For `::kELEMENTWISE`, the number of weights equals the product of all input dimensions at channelAxis and beyond.

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

channelAxis can also be set explicitly using `setChannelAxis()`.

See also

[IScaleLayer](#)

`setChannelAxis()`

Warning

Int32 tensors are not valid input tensors.

Only 2D or 3D scale is supported.

Returns

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

9.82.3.35 addScatter()

```
IScatterLayer * nvinfer1::INetworkDefinition::addScatter (
    ITensor & data,
    ITensor & indices,
    ITensor & updates,
    ScatterMode mode ) [inline], [noexcept]
```

Add a Scatter layer to the network with specified mode and axis=0.

Parameters

<i>input</i>	The input tensor to be updated with additional values.
<i>indices</i>	indices of the elements to be updated.
<i>updates</i>	values to be used for updates.

See also

[IScatterLayer](#)

`input` tensor data type must be `DataType::kFLOAT`. `indices` tensor data type must be `DataType::kINT32`. `updates` tensor data type must be `DataType::kFLOAT`.

Returns

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

9.82.3.36 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.

9.82.3.37 addShape()

```
IShapeLayer * nvinfer1::INetworkDefinition::addShape (
    ITensor & input ) [inline], [noexcept]
```

Add a shape layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
--------------	--------------------------------

See also

[IShapeLayer](#)

Warning

addShape is only supported when hasImplicitBatchDimensions is false.
input to addShape cannot contain wildcard dimension values.

Returns

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

9.82.3.38 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.

9.82.3.39 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.

9.82.3.40 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.

9.82.3.41 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>	Number of elements to keep.
<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> 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.

9.82.3.42 addUnary()

```
IUnaryLayer * nvinfer1::INetworkDefinition::addUnary (
    ITensor & input,
    UnaryOperation operation ) [inline], [noexcept]
```

Add a unary layer to the network.

Parameters

<i>input</i>	The input tensor to the layer.
<i>operation</i>	The operation to apply.

See also

[UnaryLayer](#)

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

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

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

Returns

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

9.82.3.43 destroy()

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

Destroy this [INetworkDefinition](#) object.

Deprecated Use `delete` instead. Deprecated in TensorRT 8.0

Warning

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

9.82.3.44 `getErrorRecorder()`

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

get the ErrorRecorder assigned to this interface.

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

Returns

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

See also

[setErrorRecorder\(\)](#)

9.82.3.45 `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\(\)](#)

9.82.3.46 `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\(\)](#)

9.82.3.47 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.

9.82.3.48 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\(\)](#)

9.82.3.49 `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\(\)](#)

9.82.3.50 `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\(\)](#)

9.82.3.51 `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\(\)](#)

9.82.3.52 hasExplicitPrecision()

```
TRT_DEPRECATED bool nvinfer1::INetworkDefinition::hasExplicitPrecision ( ) const [inline], [noexcept]
```

True if network is an explicit precision network.

Deprecated Deprecated in TensorRT 8.0

[hasExplicitPrecision\(\)](#) is true if and only if this [INetworkDefinition](#) was created with [createNetworkV2\(\)](#) with [NetworkDefinitionCreationFlag::kEXPLICIT_PRECISION](#) set.

See also

[createNetworkV2](#)

Returns

True if network has explicit precision, false otherwise.

9.82.3.53 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](#)

9.82.3.54 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.

9.82.3.55 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\(\)](#)

9.82.3.56 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()`.

9.82.3.57 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\(\)](#)

9.82.3.58 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.

See also

[INetworkDefinition::getName\(\)](#), [ISafeCudaEngine::getName\(\)](#)

Returns

none

9.82.3.59 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.

9.82.3.60 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\(\)](#)

9.82.3.61 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.

9.82.4 Member Data Documentation

9.82.4.1 mImpl

```
apiv::VNetworkDefinition* nvinfer1::INetworkDefinition::mImpl [protected]
```

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

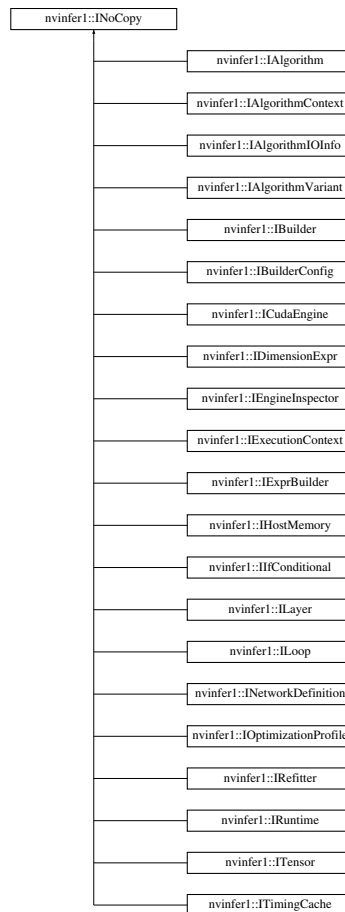
- [NvInfer.h](#)

9.83 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

9.83.1 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.

9.83.2 Constructor & Destructor Documentation

9.83.2.1 INoCopy() [1/3]

```
nvinfer1::INoCopy::INoCopy ( ) [protected], [default]
```

9.83.2.2 ~INoCopy()

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

9.83.2.3 INoCopy() [2/3]

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

9.83.2.4 INoCopy() [3/3]

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

9.83.3 Member Function Documentation

9.83.3.1 operator=() [1/2]

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


9.83.3.2 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](#)

9.84 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.

9.84.1 Detailed Description

Configuration Manager Class.

9.84.2 Member Typedef Documentation

9.84.2.1 Verbosity

`nvonnxparser::IOnnxConfig::Verbosity`

Defines Verbosity level.

9.84.3 Constructor & Destructor Documentation

9.84.3.1 ~IOnnxConfig()

```
virtual nvonnxparser::IOnnxConfig::~IOnnxConfig ( ) [virtual], [default], [noexcept]
```

9.84.4 Member Function Documentation

9.84.4.1 addVerbosity()

```
virtual void nvonnxparser::IOnnxConfig::addVerbosity ( ) [pure virtual], [noexcept]
```

Increase the Verbosity Level.

Returns

The Verbosity Level.

See also

[reduceVerbosity\(\)](#), [setVerbosity\(Verbosity\)](#)

9.84.4.2 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.

9.84.4.3 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\(\)](#)

9.84.4.4 getModelDtype()

```
virtual nvinfer1::DataType nvonnxparser::IOnnxConfig::getModelDtype ( ) const [pure virtual],  
[noexcept]
```

Get the Model Data Type.

Returns

[DataType](#) [nvinfer1::DataType](#)

See also

[setModelDtype\(\)](#) and [#DataType](#)

9.84.4.5 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\(\)](#)

9.84.4.6 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\(\)](#)

9.84.4.7 `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()`

9.84.4.8 `getVerbosityLevel()`

```
virtual Verbosity nvonnxparser::IOnnxConfig::getVerbosityLevel ( ) const [pure virtual], [noexcept]
```

Get the Verbosity Level.

Returns

The Verbosity Level.

See also

[addVerbosity\(\)](#), [reduceVerbosity\(\)](#)

9.84.4.9 `reduceVerbosity()`

```
virtual void nvonnxparser::IOnnxConfig::reduceVerbosity ( ) [pure virtual], [noexcept]
```

Reduce the Verbosity Level.

See also

[addVerbosity\(\)](#), [setVerbosity\(Verbosity\)](#)

9.84.4.10 `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\(\)](#)

9.84.4.11 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\(\)](#)

9.84.4.12 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\(\)](#)

9.84.4.13 setPrintLayerInfo()

```
virtual void nvonnxparser::IOnnxConfig::setPrintLayerInfo (
    bool ) [pure virtual], [noexcept]
```

Set whether the layer information will be printed.

See also

[getPrintLayerInfo\(\)](#)

9.84.4.14 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\(\)](#)

9.84.4.15 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:

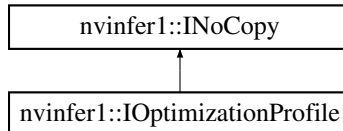
- [NvOnnxConfig.h](#)

9.85 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](#)

9.85.1 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\(\)](#)

9.85.2 Constructor & Destructor Documentation

9.85.2.1 ~IOptimizationProfile()

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

9.85.3 Member Function Documentation

9.85.3.1 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.

9.85.3.2 getExtraMemoryTarget()

```
float nvinfer1::IOptimizationProfile::getExtraMemoryTarget ( ) const [inline], [noexcept]
```

Get the extra memory target that has been defined for this profile.

9.85.3.3 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.

9.85.3.4 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.

9.85.3.5 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

9.85.3.6 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) $0 \leq \text{minDims.d}[i] \leq \text{optDims.d}[i] \leq \text{maxDims.d}[i]$ for $i = 0, \dots, \text{networkDims.nbDims}-1$ (3) if $\text{networkDims.d}[i] \neq -1$, then $\text{minDims.d}[i] == \text{optDims.d}[i] == \text{maxDims.d}[i] == \text{networkDims.d}[i]$

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 to be the same.

9.85.3.7 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

9.85.3.8 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. 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.

9.85.4 Member Data Documentation

9.85.4.1 mImpl

```
apiv::VOptimizationProfile* nvinfer1::IOptimizationProfile::mImpl [protected]
```

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

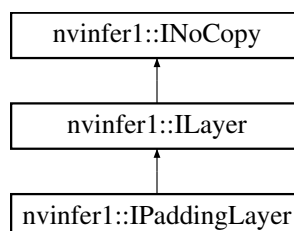
- [NvInferRuntime.h](#)

9.86 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`

9.86.1 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.

9.86.2 Constructor & Destructor Documentation

9.86.2.1 `~IPaddingLayer()`

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

9.86.3 Member Function Documentation

9.86.3.1 `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

9.86.3.2 `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](#)

9.86.3.3 `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

9.86.3.4 `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](#)

9.86.3.5 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

9.86.3.6 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](#)

9.86.3.7 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

9.86.3.8 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](#)

9.86.4 Member Data Documentation

9.86.4.1 mImpl

```
apiv::VPaddingLayer* nvinfer1::IPaddingLayer::mImpl [protected]
```

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

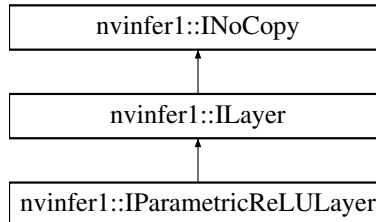
- [NvInfer.h](#)

9.87 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 `~IParametricReLU Layer () noexcept=default`

Protected Attributes

- `apiv::VParametricReLU Layer * mImpl`

Additional Inherited Members

9.87.1 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.

9.87.2 Constructor & Destructor Documentation

9.87.2.1 ~IParametricReLU Layer()

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

9.87.3 Member Data Documentation

9.87.3.1 mImpl

```
apiv::VParametricReLULayer* nvinfer1::IParametricReLULayer::mImpl [protected]
```

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

- [NvInfer.h](#)

9.88 nvonnxparser::IParser Class Reference

an object for parsing ONNX models into a TensorRT network definition

```
#include <NvOnnxParser.h>
```

Public Member Functions

- virtual bool [parse](#) (void const *serialized_onnx_model, size_t serialized_onnx_model_size, const char *model_path=nullptr)=0
Parse a serialized ONNX model into the TensorRT network. This method has very limited diagnostics. If parsing the serialized model fails for any reason (e.g. unsupported IR version, unsupported opset, etc.) it is the user responsibility to intercept and report the error. To obtain a better diagnostic, use the parseFromFile method below.
- virtual bool [parseFromFile](#) (const char *onnxModelFile, int verbosity)=0
Parse an onnx model file, which can be a binary protobuf or a text onnx model calls parse method inside.
- virtual bool [supportsModel](#) (void const *serialized_onnx_model, size_t serialized_onnx_model_size, [SubGraphCollection.t](#) &sub_graph_collection, const char *model_path=nullptr)=0
Check whether TensorRT supports a particular ONNX model. If the function returns True, one can proceed to engine building without having to call parse or parseFromFile.
- virtual bool [parseWithWeightDescriptors](#) (void const *serialized_onnx_model, size_t serialized_onnx_model_size)=0
Parse a serialized ONNX model into the TensorRT network with consideration of user provided weights.
- virtual bool [supportsOperator](#) (const char *op_name) const =0
Returns whether the specified operator may be supported by the parser.
- virtual [TRT_DEPRECATED](#) void [destroy](#) ()=0
destroy this object
- virtual int [getNbErrors](#) () const =0
Get the number of errors that occurred during prior calls to parse.
- virtual [IParserError](#) const * [getError](#) (int index) const =0
Get an error that occurred during prior calls to parse.
- virtual void [clearErrors](#) ()=0
Clear errors from prior calls to parse.
- virtual [~IParser](#) () noexcept=default

9.88.1 Detailed Description

an object for parsing ONNX models into a TensorRT network definition

9.88.2 Constructor & Destructor Documentation

9.88.2.1 ~IParser()

```
virtual nvonnxparser::IParser::~~IParser ( ) [virtual], [default], [noexcept]
```

9.88.3 Member Function Documentation

9.88.3.1 clearErrors()

```
virtual void nvonnxparser::IParser::clearErrors ( ) [pure virtual]
```

Clear errors from prior calls to `parse`.

See also

[getNbErrors\(\)](#) [getError\(\)](#) [IParserError](#)

9.88.3.2 destroy()

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

destroy this object

Warning

deprecated and planned on being removed in TensorRT 10.0

9.88.3.3 `getError()`

```
virtual IParserError const * nvonnxparser::IParser::getError (
    int index ) const [pure virtual]
```

Get an error that occurred during prior calls to `parse`.

See also

[getNbErrors\(\)](#) [clearErrors\(\)](#) [IParserError](#)

9.88.3.4 `getNbErrors()`

```
virtual int nvonnxparser::IParser::getNbErrors ( ) const [pure virtual]
```

Get the number of errors that occurred during prior calls to `parse`.

See also

[getError\(\)](#) [clearErrors\(\)](#) [IParserError](#)

9.88.3.5 `parse()`

```
virtual bool nvonnxparser::IParser::parse (
    void const * serialized_onnx_model,
    size_t serialized_onnx_model_size,
    const char * model_path = nullptr ) [pure virtual]
```

Parse a serialized ONNX model into the TensorRT network. This method has very limited diagnostics. If parsing the serialized model fails for any reason (e.g. unsupported IR version, unsupported opset, etc.) it is the user responsibility to intercept and report the error. To obtain a better diagnostic, use the `parseFromFile` method below.

Parameters

<i>serialized_onnx_model</i>	Pointer to the serialized ONNX model
<i>serialized_onnx_model_size</i>	Size of the serialized ONNX model in bytes
<i>model_path</i>	Absolute path to the model file for loading external weights if required

Returns

true if the model was parsed successfully

See also

[getNbErrors\(\)](#) [getError\(\)](#)

9.88.3.6 parseFromFile()

```
virtual bool nvonnxparser::IParser::parseFromFile (
    const char * onnxModelFile,
    int verbosity ) [pure virtual]
```

Parse an onnx model file, which can be a binary protobuf or a text onnx model calls parse method inside.

Parameters

<i>File</i>	name
<i>Verbosity</i>	Level

Returns

true if the model was parsed successfully

9.88.3.7 parseWithWeightDescriptors()

```
virtual bool nvonnxparser::IParser::parseWithWeightDescriptors (
    void const * serialized_onnx_model,
    size_t serialized_onnx_model_size ) [pure virtual]
```

Parse a serialized ONNX model into the TensorRT network with consideration of user provided weights.

Parameters

<i>serialized_onnx_model</i>	Pointer to the serialized ONNX model
<i>serialized_onnx_model_size</i>	Size of the serialized ONNX model in bytes

Returns

true if the model was parsed successfully

See also

[getNbErrors\(\)](#) [getError\(\)](#)

9.88.3.8 supportsModel()

```
virtual bool nvonnxparser::IParser::supportsModel (
    void const * serialized_onnx_model,
    size_t serialized_onnx_model_size,
    SubGraphCollection_t & sub_graph_collection,
    const char * model_path = nullptr ) [pure virtual]
```

Check whether TensorRT supports a particular ONNX model. If the function returns True, one can proceed to engine building without having to call `parse` or `parseFromFile`.

Parameters

<i>serialized_onnx_model</i>	Pointer to the serialized ONNX model
<i>serialized_onnx_model_size</i>	Size of the serialized ONNX model in bytes
<i>sub_graph_collection</i>	Container to hold supported subgraphs
<i>model_path</i>	Absolute path to the model file for loading external weights if required

Returns

true if the model is supported

9.88.3.9 supportsOperator()

```
virtual bool nvonnxparser::IParser::supportsOperator (
    const char * op_name ) const [pure virtual]
```

Returns whether the specified operator may be supported by the parser.

Note that a result of true does not guarantee that the operator will be supported in all cases (i.e., this function may return false-positives).

Parameters

<i>op_name</i>	The name of the ONNX operator to check for support
----------------	--

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

- [NvOnnxParser.h](#)

9.89 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](#) ()

9.89.1 Detailed Description

an object containing information about an error

9.89.2 Constructor & Destructor Documentation

9.89.2.1 [~IParserError\(\)](#)

```
virtual nvonnxparser::IParserError::~~IParserError ( ) [inline], [protected], [virtual]
```

9.89.3 Member Function Documentation

9.89.3.1 `code()`

```
virtual ErrorCode nvonnxparser::IParserError::code ( ) const [pure virtual]
```

the error code

9.89.3.2 desc()

```
virtual const char * nvonnxparser::IParserError::desc ( ) const [pure virtual]
```

description of the error

9.89.3.3 file()

```
virtual const char * nvonnxparser::IParserError::file ( ) const [pure virtual]
```

source file in which the error occurred

9.89.3.4 func()

```
virtual const char * nvonnxparser::IParserError::func ( ) const [pure virtual]
```

source function in which the error occurred

9.89.3.5 line()

```
virtual int nvonnxparser::IParserError::line ( ) const [pure virtual]
```

source line at which the error occurred

9.89.3.6 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:

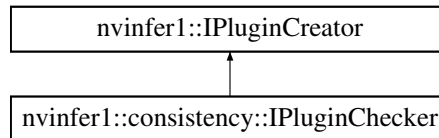
- [NvOnnxParser.h](#)

9.90 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

9.90.1 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.

9.90.2 Constructor & Destructor Documentation

9.90.2.1 IPluginChecker() [1/3]

```
nvinfer1::consistency::IPluginChecker::IPluginChecker ( ) [default]
```

9.90.2.2 ~IPluginChecker()

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

9.90.2.3 IPluginChecker() [2/3]

```
nvinfer1::consistency::IPluginChecker::IPluginChecker (
    IPluginChecker const & ) [protected], [default]
```

9.90.2.4 IPluginChecker() [3/3]

```
nvinfer1::consistency::IPluginChecker::IPluginChecker (
    IPluginChecker && ) [protected], [default]
```

9.90.3 Member Function Documentation**9.90.3.1 operator=() [1/2]**

```
IPluginChecker & nvinfer1::consistency::IPluginChecker::operator= (
    IPluginChecker && ) & [protected], [default]
```

9.90.3.2 operator=() [2/2]

```
IPluginChecker & nvinfer1::consistency::IPluginChecker::operator= (
    IPluginChecker const & ) & [protected], [default]
```

9.90.3.3 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:

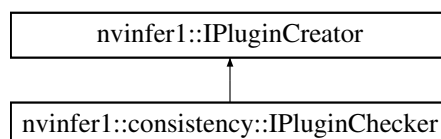
- [NvInferConsistency.h](#)

9.91 nvinfer1::IPluginCreator Class Reference

Plugin creator class for user implemented layers.

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

Inheritance diagram for nvinfer1::IPluginCreator:



Public Member Functions

- virtual `int32_t getTensorRTVersion ()` const noexcept
Return the version of the API the plugin creator was compiled with.
- virtual `AsciiChar const * getPluginName ()` const noexcept=0
Return the plugin name.
- virtual `AsciiChar const * getPluginVersion ()` const noexcept=0
Return the plugin version.
- virtual `PluginFieldCollection const * getFieldNames ()` noexcept=0
Return a list of fields that needs to be passed to createPlugin.
- virtual `IPluginV2 * createPlugin (AsciiChar const *name, PluginFieldCollection const *fc)` noexcept=0
Return a plugin object. Return nullptr in case of error.
- virtual `IPluginV2 * deserializePlugin (AsciiChar const *name, void const *serialData, size_t serialLength)` noexcept=0
Called during deserialization of plugin layer. Return a plugin object.
- virtual `void setPluginNamespace (AsciiChar const *pluginNamespace)` noexcept=0
Set the namespace of the plugin creator based on the plugin library it belongs to. This can be set while registering the plugin creator.
- virtual `AsciiChar const * getPluginNamespace ()` const noexcept=0
Return the namespace of the plugin creator object.
- `IPluginCreator ()`=default
- virtual `~IPluginCreator ()`=default

9.91.1 Detailed Description

Plugin creator class for user implemented layers.

See also

[IPlugin](#) and [IPluginFactory](#)

9.91.2 Constructor & Destructor Documentation

9.91.2.1 IPluginCreator()

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

9.91.2.2 ~IPluginCreator()

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

9.91.3 Member Function Documentation

9.91.3.1 createPlugin()

```
virtual IPluginV2 * nvinfer1::IPluginCreator::createPlugin (  
    AsciiChar const * name,  
    PluginFieldCollection const * fc ) [pure virtual], [noexcept]
```

Return a plugin object. Return nullptr in case of error.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when deserializing multiple engines concurrently sharing plugins.

9.91.3.2 deserializePlugin()

```
virtual IPluginV2 * nvinfer1::IPluginCreator::deserializePlugin (  
    AsciiChar const * name,  
    void const * serialData,  
    size_t serialLength ) [pure virtual], [noexcept]
```

Called during deserialization of plugin layer. Return a plugin object.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when deserializing multiple engines concurrently sharing plugins.

9.91.3.3 getFieldNames()

```
virtual PluginFieldCollection const * nvinfer1::IPluginCreator::getFieldNames ( ) [pure virtual],  
[noexcept]
```

Return a list of fields that needs to be passed to createPlugin.

See also

[PluginFieldCollection](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when deserializing multiple engines concurrently sharing plugins.

9.91.3.4 getPluginName()

```
virtual AsciiChar const * nvinfer1::IPluginCreator::getPluginName ( ) const [pure virtual], [noexcept]
```

Return the plugin name.

Warning

The string returned must be 1024 bytes or less including the NULL terminator and must be NULL terminated.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when deserializing multiple engines concurrently sharing plugins.

9.91.3.5 getPluginNamespace()

```
virtual AsciiChar const * nvinfer1::IPluginCreator::getPluginNamespace ( ) const [pure virtual],  
[noexcept]
```

Return the namespace of the plugin creator object.

Warning

The string returned must be 1024 bytes or less including the NULL terminator and must be NULL terminated.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when deserializing multiple engines concurrently sharing plugins.

9.91.3.6 getPluginVersion()

```
virtual AsciiChar const * nvinfer1::IPluginCreator::getPluginVersion ( ) const [pure virtual],  
[noexcept]
```

Return the plugin version.

Warning

The string returned must be 1024 bytes or less including the NULL terminator and must be NULL terminated.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when deserializing multiple engines concurrently sharing plugins.

9.91.3.7 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.

9.91.3.8 setPluginNamespace()

```
virtual void nvinfer1::IPluginCreator::setPluginNamespace (
    AsciiChar const * pluginNamespace ) [pure virtual], [noexcept]
```

Set the namespace of the plugin creator based on the plugin library it belongs to. This can be set while registering the plugin creator.

See also

[IPluginRegistry::registerCreator\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when deserializing multiple engines concurrently sharing plugins.

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

- [NvInferRuntimeCommon.h](#)

9.92 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

9.92.1 Detailed Description

Plugin factory used to configure plugins.

9.92.2 Constructor & Destructor Documentation

9.92.2.1 ~IPluginFactoryV2()

```
virtual nvcaffeparser1::IPluginFactoryV2::~IPluginFactoryV2 ( ) [virtual], [default], [noexcept]
```

9.92.3 Member Function Documentation

9.92.3.1 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

9.92.3.2 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](#)

9.93 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 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
Set 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

9.93.1 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.

9.93.2 Constructor & Destructor Documentation

9.93.2.1 ~IPluginRegistry()

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

9.93.3 Member Function Documentation

9.93.3.1 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

9.93.3.2 `getErrorRecorder()`

```
virtual IErrorRecorder * nvinfer1::IPluginRegistry::getErrorRecorder ( ) const [pure virtual],
[noexcept]
```

Set 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

9.93.3.3 `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

9.93.3.4 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

9.93.3.5 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.

9.93.3.6 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

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

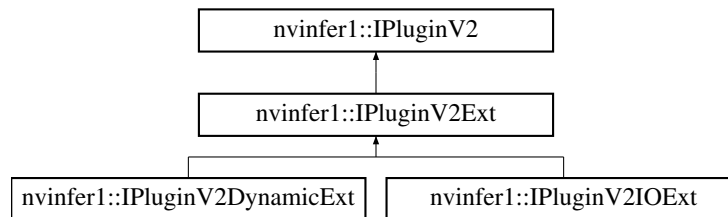
- [NvInferRuntimeCommon.h](#)

9.94 nvinfer1::IPluginV2 Class Reference

Plugin class for user-implemented layers.

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

Inheritance diagram for nvinfer1::IPluginV2:



Public Member Functions

- virtual `int32_t` [getTensorRTVersion](#) () const noexcept
Return the API version with which this plugin was built.
- virtual `AsciiChar` const * [getPluginType](#) () const noexcept=0
Return the plugin type. Should match the plugin name returned by the corresponding plugin creator.
- virtual `AsciiChar` const * [getPluginVersion](#) () const noexcept=0
Return the plugin version. Should match the plugin version returned by the corresponding plugin creator.
- virtual `int32_t` [getNbOutputs](#) () const noexcept=0
Get the number of outputs from the layer.

- virtual [Dims](#) `getOutputDimensions` (int32_t index, [Dims](#) const *inputs, int32_t nbInputDims) noexcept=0
Get the dimension of an output tensor.
- virtual bool `supportsFormat` ([DataType](#) type, [PluginFormat](#) format) const noexcept=0
Check format support.
- virtual void `configureWithFormat` ([Dims](#) const *inputDims, int32_t nbInputs, [Dims](#) const *outputDims, int32_t nbOutputs, [DataType](#) type, [PluginFormat](#) format, int32_t maxBatchSize) noexcept=0
Configure the layer.
- virtual int32_t `initialize` () noexcept=0
Initialize the layer for execution. This is called when the engine is created.
- virtual void `terminate` () noexcept=0
Release resources acquired during plugin layer initialization. This is called when the engine is destroyed.
- virtual size_t `getWorkspaceSize` (int32_t maxBatchSize) const noexcept=0
Find the workspace size required by the layer.
- virtual int32_t `enqueue` (int32_t batchSize, void const *const *inputs, void *const *outputs, void *workspace, cudaStream_t stream) noexcept=0
Execute the layer.
- virtual size_t `getSerializationSize` () const noexcept=0
Find the size of the serialization buffer required.
- virtual void `serialize` (void *buffer) const noexcept=0
Serialize the layer.
- virtual void `destroy` () noexcept=0
Destroy the plugin object. This will be called when the network, builder or engine is destroyed.
- virtual [IPluginV2](#) * `clone` () const noexcept=0
Clone the plugin object. This copies over internal plugin parameters and returns a new plugin object with these parameters.
- virtual void `setPluginNamespace` ([AsciiChar](#) const *pluginNamespace) noexcept=0
Set the namespace that this plugin object belongs to. Ideally, all plugin objects from the same plugin library should have the same namespace.
- virtual [AsciiChar](#) const * `getPluginNamespace` () const noexcept=0
Return the namespace of the plugin object.

9.94.1 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](#)

9.94.2 Member Function Documentation

9.94.2.1 clone()

```
virtual IPluginV2 * nvinfer1::IPluginV2::clone ( ) const [pure virtual], [noexcept]
```

Clone the plugin object. This copies over internal plugin parameters and returns a new plugin object with these parameters.

The TensorRT runtime calls `clone()` to clone the plugin when an execution context is created for an engine, after the engine has been created. The runtime does not call `initialize()` on the cloned plugin, so the cloned plugin should be created in an initialized state.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when creating multiple execution contexts.

Implemented in `nvinfer1::IPluginV2DynamicExt`, and `nvinfer1::IPluginV2Ext`.

9.94.2.2 configureWithFormat()

```
virtual void nvinfer1::IPluginV2::configureWithFormat (
    Dims const * inputDims,
    int32_t nbInputs,
    Dims const * outputDims,
    int32_t nbOutputs,
    DataType type,
    PluginFormat format,
    int32_t batchSize ) [pure virtual], [noexcept]
```

Configure the layer.

This function is called by the builder prior to `initialize()`. It provides an opportunity for the layer to make algorithm choices on the basis of its weights, dimensions, and maximum batch size.

Parameters

<i>inputDims</i>	The input tensor dimensions.
<i>nbInputs</i>	The number of inputs.
<i>outputDims</i>	The output tensor dimensions.
<i>nbOutputs</i>	The number of outputs.
<i>type</i>	The data type selected for the engine.
<i>format</i>	The format selected for the engine.
<i>maxBatchSize</i>	The maximum batch size.

The dimensions passed here do not include the outermost batch size (i.e. for 2-D image networks, they will be 3-dimensional CHW dimensions).

Warning

for the format field, the values `PluginFormat::kCHW4`, `PluginFormat::kCHW16`, and `PluginFormat::kCHW32` will not be passed in, this is to keep backward compatibility with TensorRT 5.x series. Use `PluginV2IOExt` or `PluginV2DynamicExt` for other `PluginFormats`.

`DataType::kBOOL` 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](#).

9.94.2.3 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.

9.94.2.4 enqueue()

```
virtual int32_t nvinfer1::IPluginV2::enqueue (
    int32_t batchSize,
    void const *const * inputs,
    void *const * outputs,
    void * workspace,
    cudaStream_t stream ) [pure virtual], [noexcept]
```

Execute the layer.

Parameters

<i>batchSize</i>	The number of inputs in the batch.
<i>inputs</i>	The memory for the input tensors.
<i>outputs</i>	The memory for the output tensors.
<i>workspace</i>	Workspace for execution.
<i>stream</i>	The stream in which to execute the kernels.

Returns

0 for success, else non-zero (which will cause engine termination).

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when multiple execution contexts are used during runtime.

9.94.2.5 getNbOutputs()

```
virtual int32_t nvinfer1::IPluginV2::getNbOutputs ( ) const [pure virtual], [noexcept]
```

Get the number of outputs from the layer.

Returns

The number of outputs.

This function is called by the implementations of [INetworkDefinition](#) and [IBuilder](#). In particular, it is called prior to any call to [initialize\(\)](#).

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.94.2.6 getOutputDimensions()

```
virtual Dims nvinfer1::IPluginV2::getOutputDimensions (
    int32_t index,
    Dims const * inputs,
    int32_t nbInputDims ) [pure virtual], [noexcept]
```

Get the dimension of an output tensor.

Parameters

<i>index</i>	The index of the output tensor.
<i>inputs</i>	The input tensors.
<i>nbInputDims</i>	The number of input tensors.

This function is called by the implementations of [INetworkDefinition](#) and [IBuilder](#). In particular, it is called prior to any call to [initialize\(\)](#).

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

Note

In any non-IPluginV2DynamicExt plugin, batch size should not be included in the returned dimensions, even if the plugin is expected to be run in a network with explicit batch mode enabled. Please see the [TensorRT Developer Guide](#) for more details on how plugin inputs and outputs behave.

9.94.2.7 getPluginNamespace()

```
virtual AsciiChar const * nvinfer1::IPluginV2::getPluginNamespace ( ) const [pure virtual], [noexcept]
```

Return the namespace of the plugin object.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.94.2.8 getPluginType()

```
virtual AsciiChar const * nvinfer1::IPluginV2::getPluginType ( ) const [pure virtual], [noexcept]
```

Return the plugin type. Should match the plugin name returned by the corresponding plugin creator.

See also

[IPluginCreator::getPluginName\(\)](#)

Warning

The string returned must be 1024 bytes or less including the NULL terminator and must be NULL terminated.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.94.2.9 getPluginVersion()

```
virtual AsciiChar const * nvinfer1::IPluginV2::getPluginVersion ( ) const [pure virtual], [noexcept]
```

Return the plugin version. Should match the plugin version returned by the corresponding plugin creator.

See also

[IPluginCreator::getPluginVersion\(\)](#)

Warning

The string returned must be 1024 bytes or less including the NULL terminator and must be NULL terminated.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.94.2.10 getSerializationSize()

```
virtual size_t nvinfer1::IPluginV2::getSerializationSize ( ) const [pure virtual], [noexcept]
```

Find the size of the serialization buffer required.

Returns

The size of the serialization buffer.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.94.2.11 getTensorRTVersion()

```
virtual int32_t nvinfer1::IPluginV2::getTensorRTVersion ( ) const [inline], [virtual], [noexcept]
```

Return the API version with which this plugin was built.

Do not override this method as it is used by the TensorRT library to maintain backwards-compatibility with plugins.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, the implementation provided here is safe to call from any thread.

Reimplemented in [nvinfer1::IPluginV2DynamicExt](#), [nvinfer1::IPluginV2Ext](#), and [nvinfer1::IPluginV2IOExt](#).

9.94.2.12 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 should be sufficient for any batch size up to the maximum.

Returns

The workspace size.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin. However, TensorRT will not call this method from two threads simultaneously on a given clone of a plugin.

9.94.2.13 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.

9.94.2.14 serialize()

```
virtual void nvinfer1::IPluginV2::serialize (
    void * buffer ) const [pure virtual], [noexcept]
```

Serialize the layer.

Parameters

<i>buffer</i>	A pointer to a buffer to serialize data. Size of buffer must be equal to value returned by <code>getSerializationSize</code> .
---------------	--

See also

[getSerializationSize\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.94.2.15 setPluginNamespace()

```
virtual void nvinfer1::IPluginV2::setPluginNamespace (
    AsciiChar const * pluginNamespace ) [pure virtual], [noexcept]
```

Set the namespace that this plugin object belongs to. Ideally, all plugin objects from the same plugin library should have the same namespace.

Parameters

<i>pluginNamespace</i>	The namespace for the plugin object.
------------------------	--------------------------------------

Warning

The string `pluginNamespace` must be 1024 bytes or less including the NULL terminator and must be NULL terminated.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.94.2.16 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 `PluginV2IOExt` or `PluginV2DynamicExt` for other `PluginFormats`.

`DataType:kBOOL` 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.

9.94.2.17 terminate()

```
virtual void nvinfer1::IPluginV2::terminate ( ) [pure virtual], [noexcept]
```

Release resources acquired during plugin layer initialization. This is called when the engine is destroyed.

See also

[initialize\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when using multiple execution contexts using this plugin. However, TensorRT will not call this method from two threads simultaneously on a given clone of a plugin.

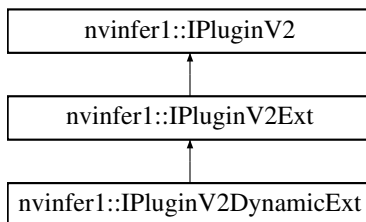
The documentation for this class was generated from the following file:

- [NvInferRuntimeCommon.h](#)

9.95 nvinfer1::IPluginV2DynamicExt Class Reference

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

Inheritance diagram for nvinfer1::IPluginV2DynamicExt:



Public Member Functions

- [IPluginV2DynamicExt * clone](#) () const noexcept override=0
Clone the plugin object. This copies over internal plugin parameters as well and returns a new plugin object with these parameters. If the source plugin is pre-configured with [configurePlugin\(\)](#), the returned object should also be pre-configured. The returned object should allow [attachToContext\(\)](#) with a new execution context. Cloned plugin objects can share the same per-engine immutable resource (e.g. weights) with the source object (e.g. via ref-counting) to avoid duplication.
- virtual [DimsExprs getOutputDimensions](#) (int32_t outputIndex, [DimsExprs](#) const *inputs, int32_t nbInputs, [IExprBuilder](#) &exprBuilder) noexcept=0
Get expressions for computing dimensions of an output tensor from dimensions of the input tensors.
- virtual bool [supportsFormatCombination](#) (int32_t pos, [PluginTensorDesc](#) const *inOut, int32_t nbInputs, int32_t nbOutputs) noexcept=0
Return true if plugin supports the format and datatype for the input/output indexed by pos.
- virtual void [configurePlugin](#) ([DynamicPluginTensorDesc](#) const *in, int32_t nbInputs, [DynamicPluginTensorDesc](#) const *out, int32_t nbOutputs) noexcept=0
Configure the plugin.
- virtual size_t [getWorkspaceSize](#) ([PluginTensorDesc](#) const *inputs, int32_t nbInputs, [PluginTensorDesc](#) const *outputs, int32_t nbOutputs) const noexcept=0
Find the workspace size required by the layer.
- virtual int32_t [enqueue](#) ([PluginTensorDesc](#) const *inputDesc, [PluginTensorDesc](#) const *outputDesc, void const *const *inputs, void *const *outputs, void *workspace, cudaStream_t stream) noexcept=0
Execute the layer.

Static Public Attributes

- static constexpr int32_t [kFORMAT_COMBINATION_LIMIT](#) = 100

Protected Member Functions

- int32_t [getTensorRTVersion](#) () const noexcept override
Return the API version with which this plugin was built. The upper byte reserved by TensorRT and is used to differentiate this from [IPluginV2](#).
- virtual [~IPluginV2DynamicExt](#) () noexcept

9.95.1 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`.

9.95.2 Constructor & Destructor Documentation

9.95.2.1 ~IPluginV2DynamicExt()

```
virtual nvinfer1::IPluginV2DynamicExt::~~IPluginV2DynamicExt ( ) [inline], [protected], [virtual],
[noexcept]
```

9.95.3 Member Function Documentation

9.95.3.1 clone()

```
IPluginV2DynamicExt * nvinfer1::IPluginV2DynamicExt::clone ( ) const [override], [pure virtual],
[noexcept]
```

Clone the plugin object. This copies over internal plugin parameters as well and returns a new plugin object with these parameters. If the source plugin is pre-configured with `configurePlugin()`, the returned object should also be pre-configured. The returned object should allow `attachToContext()` with a new execution context. Cloned plugin objects can share the same per-engine immutable resource (e.g. weights) with the source object (e.g. via ref-counting) to avoid duplication.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

Implements [nvinfer1::IPluginV2Ext](#).

9.95.3.2 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.

9.95.3.3 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).

9.95.3.4 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;
```

9.95.3.5 `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](#).

9.95.3.6 `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.

9.95.3.7 supportsFormatCombination()

```
virtual bool nvinfer1::IPluginV2DynamicExt::supportsFormatCombination (
    int32_t pos,
    PluginTensorDesc const * inOut,
    int32_t nbInputs,
    int32_t nbOutputs ) [pure virtual], [noexcept]
```

Return true if plugin supports the format and datatype for the input/output indexed by pos.

For this method inputs are numbered 0..(nbInputs-1) and outputs are numbered nbInputs..(nbInputs+nbOutputs-1). Using this numbering, pos is an index into InOut, where $0 \leq \text{pos} < \text{nbInputs} + \text{nbOutputs} - 1$.

TensorRT invokes this method to ask if the input/output indexed by pos supports the format/datatype specified by `inOut[pos].format` and `inOut[pos].type`. The override should return true if that format/datatype at `inOut[pos]` are supported by the plugin. If support is conditional on other input/output formats/datatypes, the plugin can make its result conditional on the formats/datatypes in `inOut[0..pos-1]`, which will be set to values that the plugin supports. The override should not inspect `inOut[pos+1..nbInputs+nbOutputs-1]`, which will have invalid values. In other words, the decision for pos must be based on `inOut[0..pos]` only.

Some examples:

- A definition for a plugin that supports only FP16 NCHW:

```
return inOut.format[pos] == TensorFormat::kLINEAR && inOut.type[pos] == DataType::kHALF;
```

- A definition for a plugin that supports only FP16 NCHW for its two inputs, and FP32 NCHW for its single output:

```
return inOut.format[pos] == TensorFormat::kLINEAR && (inOut.type[pos] == pos < 2 ? DataType::kHALF :
    DataType::kFLOAT);
```

- A definition for a "polymorphic" plugin with two inputs and one output that supports any format or type, but the inputs and output must have the same format and type:

```
return pos == 0 || (inOut.format[pos] == inOut.format[0] && inOut.type[pos] == inOut.type[0]);
```

Warning: TensorRT will stop asking for formats once it finds `kFORMAT_COMBINATION_LIMIT` on combinations.

9.95.4 Member Data Documentation

9.95.4.1 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:

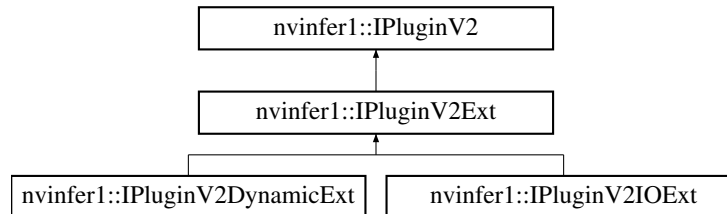
- [NvInferRuntime.h](#)

9.96 nvinfer1::IPluginV2Ext Class Reference

Plugin class for user-implemented layers.

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

Inheritance diagram for nvinfer1::IPluginV2Ext:



Public Member Functions

- virtual `nvinfer1::DataType` `getOutputDataType` (`int32_t` index, `nvinfer1::DataType` const *inputTypes, `int32_t` nbInputs) const noexcept=0
Return the DataType of the plugin output at the requested index.
- virtual `bool` `isOutputBroadcastAcrossBatch` (`int32_t` outputIndex, `bool` const *inputIsBroadcasted, `int32_t` nbInputs) const noexcept=0
Return true if output tensor is broadcast across a batch.
- virtual `bool` `canBroadcastInputAcrossBatch` (`int32_t` inputIndex) const noexcept=0
Return true if plugin can use input that is broadcast across batch without replication.
- virtual `void` `configurePlugin` (`Dims` const *inputDims, `int32_t` nbInputs, `Dims` const *outputDims, `int32_t` nbOutputs, `DataType` const *inputTypes, `DataType` const *outputTypes, `bool` const *inputIsBroadcast, `bool` const *outputIsBroadcast, `PluginFormat` floatFormat, `int32_t` maxBatchSize) noexcept=0
Configure the layer with input and output data types.
- `IPluginV2Ext` ()=default
- `~IPluginV2Ext` () override=default
- virtual `void` `attachToContext` (`cudaContext` *, `cudaContext` *, `IGpuAllocator` *) noexcept
Attach the plugin object to an execution context and grant the plugin the access to some context resource.
- virtual `void` `detachFromContext` () noexcept
Detach the plugin object from its execution context.
- `IPluginV2Ext` * `clone` () const noexcept override=0
Clone the plugin object. This copies over internal plugin parameters as well and returns a new plugin object with these parameters. If the source plugin is pre-configured with `configurePlugin()`, the returned object should also be pre-configured. The returned object should allow `attachToContext()` with a new execution context. Cloned plugin objects can share the same per-engine immutable resource (e.g. weights) with the source object (e.g. via ref-counting) to avoid duplication.

Protected Member Functions

- `int32_t` `getTensorRTVersion` () const noexcept override
Return the API version with which this plugin was built. The upper byte reserved by TensorRT and is used to differentiate this from `IPluginV2`.
- `void` `configureWithFormat` (`Dims` const *, `int32_t`, `Dims` const *, `int32_t`, `DataType`, `PluginFormat`, `int32_t`) noexcept override
Derived classes should not implement this. In a C++11 API it would be override final.

9.96.1 Detailed Description

Plugin class for user-implemented layers.

Plugins are a mechanism for applications to implement custom layers. This interface provides additional capabilities to the [IPluginV2](#) interface by supporting different output data types and broadcast across batch.

See also

[IPluginV2](#)

9.96.2 Constructor & Destructor Documentation

9.96.2.1 IPluginV2Ext()

```
nvinfer1::IPluginV2Ext::IPluginV2Ext ( ) [default]
```

9.96.2.2 ~IPluginV2Ext()

```
nvinfer1::IPluginV2Ext::~~IPluginV2Ext ( ) [override], [default]
```

9.96.3 Member Function Documentation

9.96.3.1 attachToContext()

```
virtual void nvinfer1::IPluginV2Ext::attachToContext (
    cudnnContext * ,
    cublasContext * ,
    IAllocator * ) [inline], [virtual], [noexcept]
```

Attach the plugin object to an execution context and grant the plugin the access to some context resource.

Parameters

<i>cudnn</i>	The CUDNN context handle of the execution context
<i>cublas</i>	The cublas context handle of the execution context
<i>allocator</i>	The allocator used by the execution context

This function is called automatically for each plugin when a new execution context is created. If the context was created without resources, this method is not called until the resources are assigned. It is also called if new resources are assigned to the context.

If the plugin needs per-context resource, it can be allocated here. The plugin can also get context-owned CUDNN and CUBLAS context here.

Note

In the automotive safety context, the CUDNN and CUBLAS parameters will be nullptr because CUDNN and CUBLAS is not used by the safe runtime.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.96.3.2 canBroadcastInputAcrossBatch()

```
virtual bool nvinfer1::IPluginV2Ext::canBroadcastInputAcrossBatch (
    int32_t inputIndex ) const [pure virtual], [noexcept]
```

Return true if plugin can use input that is broadcast across batch without replication.

Parameters

<i>inputIndex</i>	Index of input that could be broadcast.
-------------------	---

For each input whose tensor is semantically broadcast across a batch, TensorRT calls this method before calling `configurePlugin`. If `canBroadcastInputAcrossBatch` returns true, TensorRT will not replicate the input tensor; i.e., there will be a single copy that the plugin should share across the batch. If it returns false, TensorRT will replicate the input tensor so that it appears like a non-broadcasted tensor.

This method is called only for inputs that can be broadcast.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.96.3.3 clone()

```
IPluginV2Ext * nvinfer1::IPluginV2Ext::clone ( ) const [override], [pure virtual], [noexcept]
```

Clone the plugin object. This copies over internal plugin parameters as well and returns a new plugin object with these parameters. If the source plugin is pre-configured with [configurePlugin\(\)](#), the returned object should also be pre-configured. The returned object should allow [attachToContext\(\)](#) with a new execution context. Cloned plugin objects can share the same per-engine immutable resource (e.g. weights) with the source object (e.g. via ref-counting) to avoid duplication.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

Implements [nvinfer1::IPluginV2](#).

Implemented in [nvinfer1::IPluginV2DynamicExt](#).

9.96.3.4 configurePlugin()

```
virtual void nvinfer1::IPluginV2Ext::configurePlugin (
    Dims const * inputDims,
    int32_t nbInputs,
    Dims const * outputDims,
    int32_t nbOutputs,
    DataType const * inputTypes,
    DataType const * outputTypes,
    bool const * inputIsBroadcast,
    bool const * outputIsBroadcast,
    PluginFormat floatFormat,
    int32_t maxBatchSize ) [pure virtual], [noexcept]
```

Configure the layer with input and output data types.

This function is called by the builder prior to [initialize\(\)](#). It provides an opportunity for the layer to make algorithm choices on the basis of its weights, dimensions, data types and maximum batch size.

Parameters

<i>inputDims</i>	The input tensor dimensions.
<i>nbInputs</i>	The number of inputs.
<i>outputDims</i>	The output tensor dimensions.
<i>nbOutputs</i>	The number of outputs.
<i>inputTypes</i>	The data types selected for the plugin inputs.
<i>outputTypes</i>	The data types selected for the plugin outputs.
<i>inputIsBroadcast</i>	True for each input that the plugin must broadcast across the batch.
<i>outputIsBroadcast</i>	True for each output that TensorRT will broadcast across the batch.
<i>floatFormat</i>	The format selected for the engine for the floating point inputs/outputs.

The dimensions passed here do not include the outermost batch size (i.e. for 2-D image networks, they will be 3-dimensional CHW dimensions). When `inputIsBroadcast` or `outputIsBroadcast` is true, the outermost batch size for that input or output should be treated as if it is one. `inputIsBroadcast[i]` is true only if the input is semantically broadcast across the batch and `canBroadcastInputAcrossBatch(i)` returned true. `outputIsBroadcast[i]` is true only if `isOutputBroadcastAcrossBatch(i)` returns true.

Warning

for the `floatFormat` field, the values `PluginFormat::kCHW4`, `PluginFormat::kCHW16`, and `PluginFormat::kCHW32` will not be passed in, this is to keep backward compatibility with TensorRT 5.x series. Use `PluginV2IOExt` or `PluginV2DynamicExt` for other `PluginFormats`.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin. However, TensorRT will not call this method from two threads simultaneously on a given clone of a plugin.

9.96.3.5 configureWithFormat()

```
void nvinfer1::IPluginV2Ext::configureWithFormat (
    Dims const * ,
    int32_t ,
    Dims const * ,
    int32_t ,
    DataType ,
    PluginFormat ,
    int32_t ) [inline], [override], [protected], [virtual], [noexcept]
```

Derived classes should not implement this. In a C++11 API it would be `override final`.

Implements [nvinfer1::IPluginV2](#).

9.96.3.6 detachFromContext()

```
virtual void nvinfer1::IPluginV2Ext::detachFromContext ( ) [inline], [virtual], [noexcept]
```

Detach the plugin object from its execution context.

This function is called automatically for each plugin when a execution context is destroyed or the context resources are unassigned from the context.

If the plugin owns per-context resource, it can be released here.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.96.3.7 `getOutputDataType()`

```
virtual nvinfer1::DataType nvinfer1::IPluginV2Ext::getOutputDataType (
    int32_t index,
    nvinfer1::DataType const * inputTypes,
    int32_t nbInputs ) const [pure virtual], [noexcept]
```

Return the `DataType` of the plugin output at the requested index.

The default behavior should be to return the type of the first input, or `DataType::kFLOAT` if the layer has no inputs. The returned data type must have a format that is supported by the plugin.

See also

[supportsFormat\(\)](#)

Warning

`DataType::kBOOL` 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.

9.96.3.8 `getTensorRTVersion()`

```
int32_t nvinfer1::IPluginV2Ext::getTensorRTVersion ( ) const [inline], [override], [protected],
[virtual], [noexcept]
```

Return the API version with which this plugin was built. The upper byte reserved by TensorRT and is used to differentiate this from `IPluginV2`.

Do not override this method as it is used by the TensorRT library to maintain backwards-compatibility with plugins.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, the implementation provided here is safe to call from any thread.

Reimplemented from `nvinfer1::IPluginV2`.

Reimplemented in `nvinfer1::IPluginV2IOExt`.

9.96.3.9 `isOutputBroadcastAcrossBatch()`

```
virtual bool nvinfer1::IPluginV2Ext::isOutputBroadcastAcrossBatch (
    int32_t outputIndex,
    bool const * inputIsBroadcasted,
    int32_t nbInputs ) const [pure virtual], [noexcept]
```

Return true if output tensor is broadcast across a batch.

Parameters

<i>outputIndex</i>	The index of the output
<i>inputsBroadcasted</i>	The ith element is true if the tensor for the ith input is broadcast across a batch.
<i>nbInputs</i>	The number of inputs

The values in `inputsBroadcasted` refer to broadcasting at the semantic level, i.e. are unaffected by whether method `canBroadcastInputAcrossBatch` requests physical replication of the values.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

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

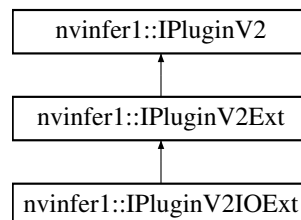
- [NvInferRuntimeCommon.h](#)

9.97 nvinfer1::IPluginV2IOExt Class Reference

Plugin class for user-implemented layers.

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

Inheritance diagram for `nvinfer1::IPluginV2IOExt`:



Public Member Functions

- virtual void `configurePlugin` (`PluginTensorDesc` const *in, `int32_t` nbInput, `PluginTensorDesc` const *out, `int32_t` nbOutput) noexcept=0
Configure the layer.
- virtual bool `supportsFormatCombination` (`int32_t` pos, `PluginTensorDesc` const *inOut, `int32_t` nbInputs, `int32_t` nbOutputs) const noexcept=0
Return true if plugin supports the format and datatype for the input/output indexed by pos.

Protected Member Functions

- `int32_t getTensorRTVersion ()` const noexcept override

Return the API version with which this plugin was built. The upper byte is reserved by TensorRT and is used to differentiate this from [IPluginV2](#) and [IPluginV2Ext](#).

9.97.1 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](#)

9.97.2 Member Function Documentation

9.97.2.1 `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 I/O [PluginTensorDesc](#) and the maximum batch size.

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.

9.97.2.2 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](#).

9.97.2.3 supportsFormatCombination()

```
virtual bool nvinfer1::IPluginV2IOExt::supportsFormatCombination (
    int32_t pos,
    PluginTensorDesc const * inOut,
    int32_t nbInputs,
    int32_t nbOutputs ) const [pure virtual], [noexcept]
```

Return true if plugin supports the format and datatype for the input/output indexed by pos.

For this method inputs are numbered 0..(nbInputs-1) and outputs are numbered nbInputs..(nbInputs+nbOutputs-1). Using this numbering, pos is an index into InOut, where $0 \leq \text{pos} < \text{nbInputs} + \text{nbOutputs} - 1$.

TensorRT invokes this method to ask if the input/output indexed by pos supports the format/datatype specified by `inOut[pos].format` and `inOut[pos].type`. The override should return true if that format/datatype at `inOut[pos]` are supported by the plugin. If support is conditional on other input/output formats/datatypes, the plugin can make its result conditional on the formats/datatypes in `inOut[0..pos-1]`, which will be set to values that the plugin supports. The override should not inspect `inOut[pos+1..nbInputs+nbOutputs-1]`, which will have invalid values. In other words, the decision for pos must be based on `inOut[0..pos]` only.

Some examples:

- A definition for a plugin that supports only FP16 NCHW:

```
return inOut.format[pos] == TensorFormat::kLINEAR && inOut.type[pos] == DataType::kHALF;
```

- A definition for a plugin that supports only FP16 NCHW for its two inputs, and FP32 NCHW for its single output:

```
return inOut.format[pos] == TensorFormat::kLINEAR &&
       (inOut.type[pos] == (pos < 2 ? DataType::kHALF : DataType::kFLOAT));
```

- A definition for a "polymorphic" plugin with two inputs and one output that supports any format or type, but the inputs and output must have the same format and type:

```
return pos == 0 || (inOut.format[pos] == inOut.format[0] && inOut.type[pos] == inOut.type[0]);
```

Warning: TensorRT will stop asking for formats once it finds kFORMAT_COMBINATION_LIMIT on combinations.

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

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

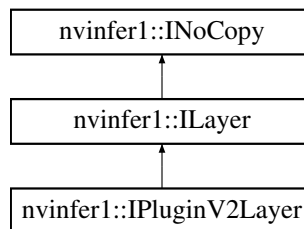
- [NvInferRuntimeCommon.h](#)

9.98 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](#)

9.98.1 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.

9.98.2 Constructor & Destructor Documentation

9.98.2.1 ~IPluginV2Layer()

```
virtual nvinfer1::IPluginV2Layer::~~IPluginV2Layer ( ) [protected], [virtual], [default], [noexcept]
```

9.98.3 Member Function Documentation

9.98.3.1 getPlugin()

```
IPluginV2 & nvinfer1::IPluginV2Layer::getPlugin ( ) [inline], [noexcept]
```

Get the plugin for the layer.

See also

[IPluginV2](#)

9.98.4 Member Data Documentation

9.98.4.1 mImpl

```
apiv::VPluginV2Layer* nvinfer1::IPluginV2Layer::mImpl [protected]
```

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

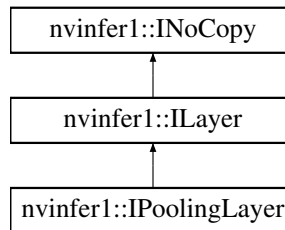
- [NvInfer.h](#)

9.99 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: $max_average_blendPool = (1-blendFactor)*maxPool + blendFactor*avgPool$ blendFactor is a user value in $[0,1]$ with the default value of 0.0 This value only applies for the kMAX↔AVERAGE_BLEND mode.*
- float [getBlendFactor \(\)](#) const noexcept
*Get the blending factor for the max_average_blend mode: $max_average_blendPool = (1-blendFactor)*maxPool + blendFactor*avgPool$ blendFactor is a user value in $[0,1]$ with the default value of 0.0 In modes other than kMAX↔AVERAGE_BLEND, blendFactor is ignored.*
- void [setAverageCountExcludesPadding](#) (bool exclusive) noexcept
Set whether average pooling uses as a denominator the overlap area between the window and the unpadded input. If this is not set, the denominator is the overlap between the pooling window and the padded input.
- bool [getAverageCountExcludesPadding \(\)](#) const noexcept
Get whether average pooling uses as a denominator the overlap area between the window and the unpadded input.
- void [setPrePadding](#) (Dims 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](#)

9.99.1 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.

9.99.2 Constructor & Destructor Documentation

9.99.2.1 `~IPoolingLayer()`

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

9.99.3 Member Function Documentation

9.99.3.1 `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\(\)](#)

9.99.3.2 `getBlendFactor()`

```
float nvinfer1::IPoolingLayer::getBlendFactor ( ) const [inline], [noexcept]
```

Get the blending factor for the `max_average_blend` mode: `max_average_blendPool = (1-blendFactor)*maxPool + blendFactor*avgPool` `blendFactor` is a user value in `[0,1]` with the default value of `0.0` In modes other than `kMAX_BLEND`, `blendFactor` is ignored.

See also

[setBlendFactor\(\)](#)

9.99.3.3 `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

9.99.3.4 `getPaddingMode()`

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

Get the padding mode.

Default: `kEXPLICIT_ROUND_DOWN`

See also

[setPaddingMode\(\)](#)

9.99.3.5 `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\(\)](#)

9.99.3.6 `getPoolingType()`

```
PoolingType nvinfer1::IPoolingLayer::getPoolingType ( ) const [inline], [noexcept]
```

Get the type of activation to be performed.

See also

[setPoolingType\(\)](#), [PoolingType](#)

9.99.3.7 `getPostPadding()`

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

Get the padding.

See also

[setPostPadding\(\)](#)

9.99.3.8 `getPrePadding()`

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

Get the pre-padding.

See also

[setPrePadding\(\)](#)

9.99.3.9 `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

9.99.3.10 getStrideNd()

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

Get the multi-dimension stride for pooling.

See also

[setStrideNd\(\)](#)

9.99.3.11 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

9.99.3.12 getWindowSizeNd()

```
Dims nvinfer1::IPoolingLayer::getWindowSizeNd ( ) const [inline], [noexcept]
```

Get the multi-dimension window size for pooling.

See also

[setWindowSizeNd\(\)](#)

9.99.3.13 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\(\)](#)

9.99.3.14 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\(\)](#)

9.99.3.15 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

9.99.3.16 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\(\)](#)

9.99.3.17 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\(\)](#)

9.99.3.18 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](#)

9.99.3.19 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\(\)](#)

9.99.3.20 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\(\)](#)

9.99.3.21 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

9.99.3.22 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\(\)](#)

9.99.3.23 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

9.99.3.24 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\(\)](#)

9.99.4 Member Data Documentation

9.99.4.1 mImpl

```
apiv::VPoolingLayer* nvinfer1::IPoolingLayer::mImpl [protected]
```

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

- [NvInfer.h](#)

9.100 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

9.100.1 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()`.

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.

9.100.2 Constructor & Destructor Documentation

9.100.2.1 ~IProfiler()

```
virtual nvinfer1::IProfiler::~~IProfiler ( ) [inline], [virtual], [noexcept]
```

9.100.3 Member Function Documentation

9.100.3.1 reportLayerTime()

```
virtual void nvinfer1::IProfiler::reportLayerTime (  
    char const * layerName,  
    float ms ) [pure virtual], [noexcept]
```

Layer time reporting callback.

Parameters

<i>layerName</i>	The name of the layer, set when constructing the network definition.
<i>ms</i>	The time in milliseconds to execute the layer.

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

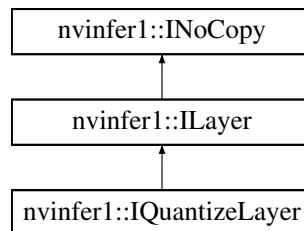
- [NvInferRuntime.h](#)

9.101 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`

9.101.1 Detailed Description

A Quantize layer in a network definition.

This layer accepts a floating-point data input tensor, and uses the `scale` and `zeroPt` inputs to quantize the data to an 8-bit signed integer according to: $\text{output} = \text{clamp}(\text{round}(\text{input} / \text{scale}) + \text{zeroPt})$

Rounding type is rounding-to-nearest ties-to-even (https://en.wikipedia.org/wiki/Rounding#Round_half_to_even). Clamping is in the range [-128, 127].

The first input (index 0) is the tensor to be quantized. The second (index 1) and third (index 2) are the `scale` and `zeroPt` respectively. Each of `scale` and `zeroPt` must be either a scalar, or a 1D tensor.

The `zeroPt` tensor is optional, and if not set, will be assumed to be zero. Its data type must be `DataType::kINT8`. `zeroPt` must only contain zero-valued coefficients, because only symmetric quantization is supported. The `scale` value must be either a scalar for per-tensor quantization, or a 1D tensor for per-channel quantization. All `scale` coefficients must have positive values. The size of the 1-D `scale` tensor must match the size of the quantization axis. The size of the `scale` must match the size of the `zeroPt`.

The subgraph which terminates with the `scale` tensor must be a build-time constant. The same restrictions apply to the `zeroPt`. The output type, if constrained, must be constrained to `DataType::kINT8`. The input type, if constrained, must be constrained to `DataType::kFLOAT` (FP16 input is not supported). 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.

9.101.2 Constructor & Destructor Documentation

9.101.2.1 ~IQuantizeLayer()

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

9.101.3 Member Function Documentation

9.101.3.1 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.

9.101.3.2 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).

9.101.4 Member Data Documentation

9.101.4.1 mImpl

```
apiv::VQuantizeLayer* nvinfer1::IQuantizeLayer::mImpl [protected]
```

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

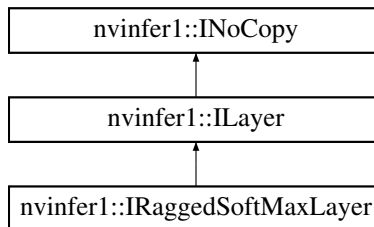
- [NvInfer.h](#)

9.102 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

9.102.1 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.

9.102.2 Constructor & Destructor Documentation

9.102.2.1 ~IRaggedSoftMaxLayer()

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

9.102.3 Member Data Documentation

9.102.3.1 mImpl

```
apiv::VRaggedSoftMaxLayer* nvinfer1::IRaggedSoftMaxLayer::mImpl [protected]
```

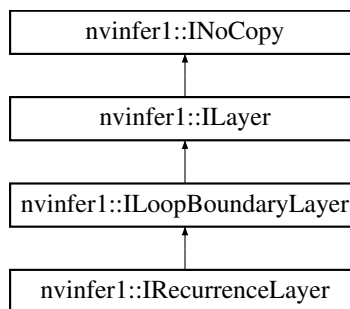
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.103 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`

9.103.1 Constructor & Destructor Documentation

9.103.1.1 ~IRecurrenceLayer()

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

9.103.2 Member Function Documentation

9.103.2.1 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.

9.103.3 Member Data Documentation

9.103.3.1 mImpl

```
apiv::VRecurrenceLayer* nvinfer1::IRecurrenceLayer::mImpl [protected]
```

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

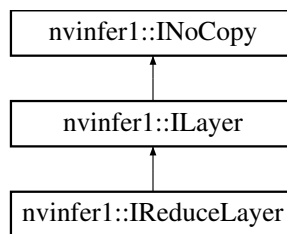
- [NvInfer.h](#)

9.104 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](#)

9.104.1 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.

9.104.2 Constructor & Destructor Documentation

9.104.2.1 ~IReduceLayer()

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

9.104.3 Member Function Documentation

9.104.3.1 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](#)

9.104.3.2 getOperation()

```
ReduceOperation nvinfer1::IReduceLayer::getOperation ( ) const [inline], [noexcept]
```

Get the reduce operation for the layer.

See also

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

9.104.3.3 `getReduceAxes()`

```
uint32_t nvinfer1::IReduceLayer::getReduceAxes ( ) const [inline], [noexcept]
```

Get the axes over which to reduce for the layer.

See also

[setReduceAxes](#)

9.104.3.4 `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](#)

9.104.3.5 `setOperation()`

```
void nvinfer1::IReduceLayer::setOperation (
    ReduceOperation op ) [inline], [noexcept]
```

Set the reduce operation for the layer.

See also

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

9.104.3.6 `setReduceAxes()`

```
void nvinfer1::IReduceLayer::setReduceAxes (
    uint32_t reduceAxes ) [inline], [noexcept]
```

Set the axes over which to reduce.

See also

[getReduceAxes](#)

9.104.4 Member Data Documentation

9.104.4.1 mImpl

apiv::VReduceLayer* nvinfer1::IReduceLayer::mImpl [protected]

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

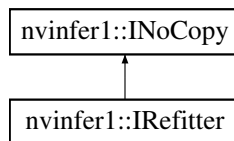
- [NvInfer.h](#)

9.105 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

9.105.1 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.

9.105.2 Constructor & Destructor Documentation

9.105.2.1 ~IRefitter()

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

9.105.3 Member Function Documentation

9.105.3.1 destroy()

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

Deprecated Use delete instead. Deprecated in TRT 8.0.

Warning

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

9.105.3.2 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.

9.105.3.3 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.

9.105.3.4 `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.

9.105.3.5 `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.

9.105.3.6 `getErrorRecorder()`

```
IErrorRecorder * nvinfer1::IRefitter::getErrorRecorder ( ) const [inline], [noexcept]
```

Get the `ErrorRecorder` assigned to this interface.

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

Returns

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

See also

[setErrorRecorder\(\)](#)

9.105.3.7 getLogger()

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

get the logger with which the refitter was created

Returns

the logger

9.105.3.8 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\(\)](#)

9.105.3.9 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.

9.105.3.10 `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.

9.105.3.11 `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 <code>tensorNames</code> .
<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.

9.105.3.12 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 `IExecutionContext`s associated with the engine should not be used afterwards. Instead, create new `IExecutionContext`s after refitting.

9.105.3.13 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.

9.105.3.14 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\(\)](#)

9.105.3.15 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.

9.105.3.16 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.

9.105.3.17 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.

9.105.4 Member Data Documentation

9.105.4.1 mImpl

```
apiv::VRefitter* nvinfer1::IRefitter::mImpl [protected]
```

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

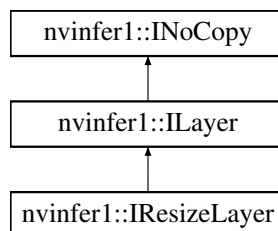
- [NvInferRuntime.h](#)

9.106 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 [setScale](#) (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 [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](#)

9.106.1 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.

9.106.2 Constructor & Destructor Documentation

9.106.2.1 ~IResizeLayer()

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

9.106.3 Member Function Documentation

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

9.106.3.2 getCoordinateTransformation()

```
ResizeCoordinateTransformation nvinfer1::IResizeLayer::getCoordinateTransformation ( ) const [inline], [noexcept]
```

Get coordinate transformation function.

Returns

The coordinate transformation function.

9.106.3.3 getNearestRounding()

```
ResizeRoundMode nvinfer1::IResizeLayer::getNearestRounding ( ) const [inline], [noexcept]
```

Get rounding mode for nearest neighbor resize.

Returns

The rounding mode.

9.106.3.4 `getOutputDimensions()`

```
Dims nvinfer1::IResizeLayer::getOutputDimensions ( ) const [inline], [noexcept]
```

Get the output dimensions.

Returns

The output dimensions.

9.106.3.5 `getResizeMode()`

```
ResizeMode nvinfer1::IResizeLayer::getResizeMode ( ) const [inline], [noexcept]
```

Get resize mode for an input tensor.

Returns

The resize mode.

9.106.3.6 `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.

9.106.3.7 getSelectorForSinglePixel()

```
ResizeSelector nvinfer1::IResizeLayer::getSelectorForSinglePixel ( ) const [inline], [noexcept]
```

Get the coordinate selector function when resized to single pixel.

Returns

The selector function.

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

9.106.3.9 setCoordinateTransformation()

```
void nvinfer1::IResizeLayer::setCoordinateTransformation (
    ResizeCoordinateTransformation coordTransform ) [inline], [noexcept]
```

Set coordinate transformation function.

The function maps a coordinate in the output tensor to a coordinate in the input tensor.

Default function is [ResizeCoordinateTransformation::kASYMMETRIC](#).

See also

[ResizeCoordinateTransformation](#)

9.106.3.10 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.

9.106.3.11 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](#)

9.106.3.12 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](#)

9.106.3.13 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](#)

9.106.3.14 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](#)

9.106.3.15 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](#)

9.106.4 Member Data Documentation

9.106.4.1 mImpl

```
apiv::VResizeLayer* nvinfer1::IResizeLayer::mImpl [protected]
```

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

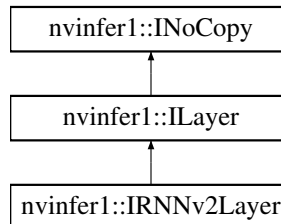
- [NvInfer.h](#)

9.107 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 maximum data 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`

9.107.1 Detailed Description

An RNN layer in a network definition, version 2.

This layer supersedes `IRNNLayer`.

Deprecated Use `INetworkDefinition::addLoop` instead. Deprecated prior to TensorRT 8.0 and will be removed in 9.0

Warning

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

9.107.2 Constructor & Destructor Documentation

9.107.2.1 ~IRNNv2Layer()

```
virtual nvinfer1::IRNNv2Layer::~~IRNNv2Layer ( ) [protected], [virtual], [default], [noexcept]
```

9.107.3 Member Function Documentation

9.107.3.1 `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\(\)](#)

9.107.3.2 `getCellState()`

```
ITensor * nvinfer1::IRNNv2Layer::getCellState ( ) const [inline], [noexcept]
```

Get the initial cell state of the RNN.

See also

[setCellState\(\)](#)

9.107.3.3 `getDataLength()`

```
int32_t nvinfer1::IRNNv2Layer::getDataLength ( ) const [inline], [noexcept]
```

Get the maximum data length of the RNN.

9.107.3.4 `getDirection()`

```
RNNDirection nvinfer1::IRNNv2Layer::getDirection ( ) const [inline], [noexcept]
```

Get the direction of the RNN layer.

See also

[setDirection\(\)](#), [RNNDirection](#)

9.107.3.5 getHiddenSize()

```
int32_t nvinfer1::IRNNv2Layer::getHiddenSize ( ) const [inline], [noexcept]
```

Get the hidden size of the RNN.

9.107.3.6 getHiddenState()

```
ITensor * nvinfer1::IRNNv2Layer::getHiddenState ( ) const [inline], [noexcept]
```

Get the initial hidden state of the RNN.

See also

[setHiddenState\(\)](#)

9.107.3.7 getInputMode()

```
RNNInputMode nvinfer1::IRNNv2Layer::getInputMode ( ) const [inline], [noexcept]
```

Get the input mode of the RNN layer.

See also

[setInputMode\(\)](#), [RNNInputMode](#)

9.107.3.8 getLayerCount()

```
int32_t nvinfer1::IRNNv2Layer::getLayerCount ( ) const [inline], [noexcept]
```

Get the layer count of the RNN.

9.107.3.9 getMaxSeqLength()

```
int32_t nvinfer1::IRNNv2Layer::getMaxSeqLength ( ) const [inline], [noexcept]
```

Get the maximum sequence length of the RNN.

9.107.3.10 `getOperation()`

```
RNNOperation nvinfer1::IRNNv2Layer::getOperation ( ) const [inline], [noexcept]
```

Get the operation of the RNN layer.

See also

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

9.107.3.11 `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\(\)](#)

9.107.3.12 `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\(\)](#)

9.107.3.13 `setBiasForGate()`

```
void nvinfer1::IRNNv2Layer::setBiasForGate (
    int32_t layerIndex,
    RNNGateType gate,
    bool isW,
    Weights bias ) [inline], [noexcept]
```

Set the bias parameters for an individual gate in the RNN.

The [DataType](#) for this structure must be `::kFLOAT` or `::kHALF`, and must be the same datatype as the input tensor.

Each bias vector has a fixed size, [getHiddenSize\(\)](#).

Parameters

<i>layerIndex</i>	The index of the layer that contains this gate. See the section Order of weight matrices in <code>IRNNLayer::setWeights()</code> for a description of the layer index.
<i>gate</i>	The name of the gate within the RNN layer. The gate name must correspond to one of the gates used by this layer's <code>RNNOperation</code> .
<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 <code>RNNOperation</code> 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 <code>getHiddenSize()</code> .

9.107.3.14 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 `::kUNIDIRECTION`, and $2 \times$ `getLayerCount()` if `getDirection` is `::kBIDIRECTION`. In the bi-directional case, layer 1's final forward hidden state is stored in $L = 2 * 1$, and final backward hidden state is stored in $L = 2 * 1 + 1$.
- 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`.

9.107.3.15 setDirection()

```
void nvinfer1::IRNNv2Layer::setDirection (
    RNNDirection op ) [inline], [noexcept]
```

Set the direction of the RNN layer.

The direction determines if the RNN is run as a unidirectional(left to right) or bidirectional(left to right and right to left). In the `::kBIDIRECTION` case the output is concatenated together, resulting in output size of $2 \times$ `getHiddenSize()`.

See also

[getDirection\(\)](#), [RNNDirection](#)

9.107.3.16 setHiddenState()

```
void nvinfer1::IRNNv2Layer::setHiddenState (
    ITensor & hidden ) [inline], [noexcept]
```

Set the initial hidden state of the RNN with the provided hidden [ITensor](#).

The hidden [ITensor](#) should have the dimensions $\{N_1, \dots, N_p, L, H\}$, where:

- $N_1 \dots N_p$ are the index dimensions specified by the input tensor
- L is the number of layers in the RNN, equal to [getLayerCount\(\)](#) if `getDirection` is `::kUNIDIRECTION`, and $2 \times$ [getLayerCount\(\)](#) if `getDirection` is `::kBIDIRECTION`. In the bi-directional case, layer l 's final forward hidden state is stored in $L = 2 * l$, and final backward hidden state is stored in $L = 2 * l + 1$.
- H is the hidden state for each layer, equal to [getHiddenSize\(\)](#).

9.107.3.17 setInputMode()

```
void nvinfer1::IRNNv2Layer::setInputMode (
    RNNInputMode op ) [inline], [noexcept]
```

Set the input mode of the RNN layer.

See also

[getInputMode\(\)](#), [RNNInputMode](#)

9.107.3.18 setOperation()

```
void nvinfer1::IRNNv2Layer::setOperation (
    RNNOperation op ) [inline], [noexcept]
```

Set the operation of the RNN layer.

See also

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

9.107.3.19 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 $\{N_1, \dots, N_p\}$ tensor, where $N_1..N_p$ 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](#).

9.107.3.20 setWeightsForGate()

```
void nvinfer1::IRNNv2Layer::setWeightsForGate (
    int32_t layerIndex,
    RNNGateType gate,
    bool isW,
    Weights weights ) [inline], [noexcept]
```

Set the weight parameters for an individual gate in the RNN.

The [DataType](#) for this structure must be `::kFLOAT` or `::kHALF`, and must be the same datatype as the input tensor.

Each parameter matrix is row-major in memory, and has the following dimensions:

```
Let K := { ::kUNIDIRECTION => 1
          { ::kBIDIRECTION => 2
            l := layer index (as described above)
            H := getHiddenSize()
            E := getDataLength() (the embedding length)
            isW := true if the matrix is an input (W) matrix, and false if
                  the matrix is a recurrent input (R) matrix.
if isW:
  if l < K and ::kSKIP:
    (numRows, numCols) := (0, 0) # input matrix is skipped
  elif l < K and ::kLINEAR:
    (numRows, numCols) := (H, E) # input matrix acts on input data size E
  elif l >= K:
    (numRows, numCols) := (H, K * H) # input matrix acts on previous hidden state
else: # not isW
  (numRows, numCols) := (H, H)
```

In other words, the input weights of the first layer of the RNN (if not skipped) transform a [getDataLength\(\)](#)-size column vector into a [getHiddenSize\(\)](#)-size column vector. The input weights of subsequent layers transform a $K * \text{getHiddenSize}()$ -size column vector into a [getHiddenSize\(\)](#)-size column vector. $K=2$ in the bidirectional case to account for the full hidden state being the concatenation of the forward and backward RNN hidden states.

The recurrent weight matrices for all layers all have shape (H, H) , both in the unidirectional and bidirectional cases. (In the bidirectional case, each recurrent weight matrix for the (forward or backward) RNN cell operates on the previous (forward or backward) RNN cell's hidden state, which is size H).

Parameters

<i>layerIndex</i>	The index of the layer that contains this gate. See the section
<i>gate</i>	The name of the gate within the RNN layer. The gate name must correspond to one of the gates used by this layer's RNNOperation .
<i>isW</i>	True if the weight parameters are for the input matrix $W[g]$ and false if they are for the recurrent input matrix $R[g]$. See RNNOperation for equations showing how these matrices are used in the RNN gate.
<i>weights</i>	The weight structure holding the weight parameters, which are stored as a row-major 2D matrix. See the layout of elements within a weight matrix in <code>IRNNLayer::setWeights()</code> for documentation on the expected dimensions of this matrix.

9.107.4 Member Data Documentation

9.107.4.1 mImpl

```
apiv::VRNNv2Layer* nvinfer1::IRNNv2Layer::mImpl [protected]
```

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

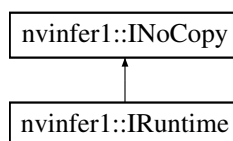
- [NvInfer.h](#)

9.108 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.

Protected Attributes

- `apiv::VRuntime * mImpl`

Additional Inherited Members

9.108.1 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.

9.108.2 Constructor & Destructor Documentation

9.108.2.1 ~IRuntime()

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

9.108.3 Member Function Documentation

9.108.3.1 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.

9.108.3.2 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.

9.108.3.3 destroy()

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

Destroy this object.

Deprecated Use delete instead. Deprecated in TRT 8.0.

Warning

Calling destroy on a managed pointer will result in a double-free error.

9.108.3.4 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.

9.108.3.5 getErrorRecorder()

```
IErrRecorder * nvinfer1::IRuntime::getErrorRecorder ( ) const [inline], [noexcept]
```

get the ErrorRecorder assigned to this interface.

Retrieves the assigned error recorder object for the given class. A nullptr will be returned if an error handler has not been set.

Returns

A pointer to the [IErrRecorder](#) object that has been registered.

See also

[setErrorRecorder\(\)](#)

9.108.3.6 `getLogger()`

```
ILogger * nvinfer1::IRuntime::getLogger ( ) const [inline], [noexcept]
```

get the logger with which the runtime was created

Returns

the logger

9.108.3.7 `getMaxThreads()`

```
int32_t nvinfer1::IRuntime::getMaxThreads ( ) const [inline], [noexcept]
```

Get the maximum number of threads that can be used by the runtime.

Retrieves the maximum number of threads that can be used by the runtime.

Returns

The maximum number of threads that can be used by the runtime.

See also

[setMaxThreads\(\)](#)

9.108.3.8 `getNbDLACores()`

```
int32_t nvinfer1::IRuntime::getNbDLACores ( ) const [inline], [noexcept]
```

Returns number of DLA hardware cores accessible or 0 if DLA is unavailable.

9.108.3.9 `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\(\)](#)

9.108.3.10 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\(\)](#)

9.108.3.11 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.

9.108.3.12 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.

9.108.4 Member Data Documentation**9.108.4.1 mImpl**

```
apiv::VRuntime* nvinfer1::IRuntime::mImpl [protected]
```

The documentation for this class was generated from the following file:

- [NvInferRuntime.h](#)

9.109 nvinfer1::safe::IRuntime Class Reference

Allows a serialized functionally safe engine to be deserialized.

```
#include <NvInferSafeRuntime.h>
```

Public Member Functions

- virtual [ICudaEngine](#) * [deserializeCudaEngine](#) (void const *const blob, std::size_t const size) noexcept=0
Deserialize an engine from a stream.
- virtual void [setGpuAllocator](#) ([IGpuAllocator](#) *const allocator) noexcept=0
Set the GPU allocator.
- virtual void [setErrorRecorder](#) ([IErrorRecorder](#) *const recorder) noexcept=0
Set the ErrorRecorder for this interface.
- virtual [IErrorRecorder](#) * [getErrorRecorder](#) () const noexcept=0
Get the ErrorRecorder assigned to this interface.
- [IRuntime](#) ()=default
- virtual [~IRuntime](#) () noexcept=default
- [IRuntime](#) ([IRuntime](#) const &)=delete
- [IRuntime](#) ([IRuntime](#) &&)=delete
- [IRuntime](#) & [operator=](#) ([IRuntime](#) const &) &=delete
- [IRuntime](#) & [operator=](#) ([IRuntime](#) &&) &=delete

9.109.1 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.

9.109.2 Constructor & Destructor Documentation

9.109.2.1 [IRuntime\(\)](#) [1/3]

```
nvinfer1::safe::IRuntime::IRuntime ( ) [default]
```


9.109.2.2 ~IRuntime()

```
virtual nvinfer1::safe::IRuntime::~IRuntime ( ) [virtual], [default], [noexcept]
```

9.109.2.3 IRuntime() [2/3]

```
nvinfer1::safe::IRuntime::IRuntime (
    IRuntime const & ) [delete]
```

9.109.2.4 IRuntime() [3/3]

```
nvinfer1::safe::IRuntime::IRuntime (
    IRuntime && ) [delete]
```

9.109.3 Member Function Documentation

9.109.3.1 deserializeCudaEngine()

```
virtual ICudaEngine * nvinfer1::safe::IRuntime::deserializeCudaEngine (
    void const *const blob,
    std::size_t const size ) [pure virtual], [noexcept]
```

Deserialize an engine from a stream.

If the serialized engine requires plugins the plugin creator must be registered by calling [IPluginRegistry::registerCreator\(\)](#) before calling [deserializeCudaEngine\(\)](#). Every plugin creator registered must have a unique combination of namespace, plugin name, and version.

Parameters

<i>blob</i>	The memory that holds the serialized engine.
<i>size</i>	The size of the memory in bytes.

Returns

The engine, or nullptr if it could not be deserialized.

See also

[IPluginRegistry::registerCreator\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes, if called from different instances of [safe::IRuntime](#). Calling `deserializeCudaEngine` of the same safety runtime from multiple threads is not guaranteed to be thread safe.

9.109.3.2 `getErrorRecorder()`

```
virtual IErrorRecorder * nvinfer1::safe::IRuntime::getErrorRecorder ( ) const [pure virtual],  
[noexcept]
```

Get the `ErrorRecorder` assigned to this interface.

Retrieves the assigned error recorder object for the given class. A default error recorder does not exist, so a `nullptr` will be returned if `setErrorRecorder` has not been called.

Returns

A pointer to the [IErrorRecorder](#) object that has been registered.

See also

[setErrorRecorder\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: Yes

9.109.3.3 `operator=()` [1/2]

```
IRuntime & nvinfer1::safe::IRuntime::operator= (  
    IRuntime && ) & [delete]
```

9.109.3.4 operator=() [2/2]

```
IRuntime & nvinfer1::safe::IRuntime::operator= (
    IRuntime const & ) & [delete]
```

9.109.3.5 setErrorRecorder()

```
virtual void nvinfer1::safe::IRuntime::setErrorRecorder (
    IErrorRecorder *const recorder ) [pure virtual], [noexcept]
```

Set the ErrorRecorder for this interface.

Assigns the ErrorRecorder to this interface. The ErrorRecorder will track all errors during execution. This function will call `incRefCount` of the registered ErrorRecorder at least once. Setting recorder to `nullptr` unregisters the recorder with the interface, resulting in a call to `decRefCount` if a recorder has been registered.

Parameters

<i>recorder</i>	The error recorder to register with this interface.
-----------------	---

See also

[getErrorRecorder\(\)](#)

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

9.109.3.6 setGpuAllocator()

```
virtual void nvinfer1::safe::IRuntime::setGpuAllocator (
    I_gpu_allocator *const allocator ) [pure virtual], [noexcept]
```

Set the GPU allocator.

Parameters

<i>allocator</i>	Set the GPU allocator to be used by the runtime. All GPU memory acquired will use this allocator. If NULL is passed, the default allocator will be used.
------------------	--

Default: uses cudaMalloc/cudaFree.

If nullptr is passed, the default allocator will be used.

Usage considerations

- Allowed context for the API call
 - Thread-safe: No

The documentation for this class was generated from the following file:

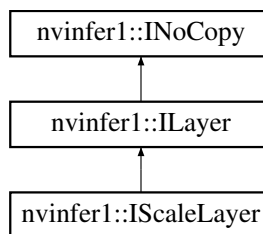
- [NvInferSafeRuntime.h](#)

9.110 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](#)

9.110.1 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.

9.110.2 Constructor & Destructor Documentation

9.110.2.1 ~IScaleLayer()

```
virtual nvinfer1::IScaleLayer::~~IScaleLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.110.3 Member Function Documentation

9.110.3.1 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\(\)](#)

9.110.3.2 getMode()

```
ScaleMode nvinfer1::IScaleLayer::getMode ( ) const [inline], [noexcept]
```

Get the scale mode.

See also

[setMode\(\)](#)

9.110.3.3 getPower()

```
Weights nvinfer1::IScaleLayer::getPower ( ) const [inline], [noexcept]
```

Get the power value.

See also

[setPower\(\)](#)

9.110.3.4 `getScale()`

```
Weights nvinfer1::IScaleLayer::getScale ( ) const [inline], [noexcept]
```

Get the scale value.

See also

[setScale\(\)](#)

9.110.3.5 `getShift()`

```
Weights nvinfer1::IScaleLayer::getShift ( ) const [inline], [noexcept]
```

Get the shift value.

See also

[setShift\(\)](#)

9.110.3.6 `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\(\)](#)

9.110.3.7 setMode()

```
void nvinfer1::IScaleLayer::setMode (  
    ScaleMode mode ) [inline], [noexcept]
```

Set the scale mode.

See also

[getMode\(\)](#)

9.110.3.8 setPower()

```
void nvinfer1::IScaleLayer::setPower (  
    Weights power ) [inline], [noexcept]
```

Set the power value.

See also

[getPower\(\)](#)

9.110.3.9 setScale()

```
void nvinfer1::IScaleLayer::setScale (  
    Weights scale ) [inline], [noexcept]
```

Set the scale value.

See also

[getScale\(\)](#)

9.110.3.10 setShift()

```
void nvinfer1::IScaleLayer::setShift (  
    Weights shift ) [inline], [noexcept]
```

Set the shift value.

See also

[getShift\(\)](#)

9.110.4 Member Data Documentation

9.110.4.1 mImpl

```
apiv::VScaleLayer* nvinfer1::IScaleLayer::mImpl [protected]
```

The documentation for this class was generated from the following file:

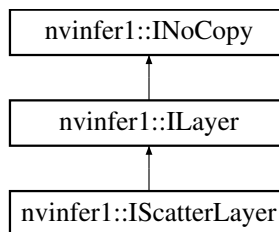
- [NvInfer.h](#)

9.111 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`

9.111.1 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.

9.111.2 Constructor & Destructor Documentation

9.111.2.1 ~IScatterLayer()

```
virtual nvinfer1::IScatterLayer::~~IScatterLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.111.3 Member Function Documentation

9.111.3.1 getAxis()

```
int32_t nvinfer1::IScatterLayer::getAxis ( ) const [inline], [noexcept]
```

Get the axis.

9.111.3.2 getMode()

```
ScatterMode nvinfer1::IScatterLayer::getMode ( ) const [inline], [noexcept]
```

Get the scatter mode.

See also

[setMode\(\)](#)

9.111.3.3 setAxis()

```
void nvinfer1::IScatterLayer::setAxis (
    int32_t axis ) [inline], [noexcept]
```

Set the axis used by ScatterMode::kELEMENTS.

The axis defaults to 0.

9.111.3.4 setMode()

```
void nvinfer1::IScatterLayer::setMode (
    ScatterMode mode ) [inline], [noexcept]
```

Set the scatter mode.

See also

[getMode\(\)](#)

9.111.4 Member Data Documentation

9.111.4.1 mImpl

```
apiv::VScatterLayer* nvinfer1::IScatterLayer::mImpl [protected]
```

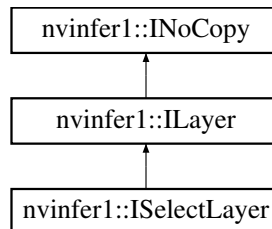
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.112 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

9.112.1 Detailed Description

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.112.2 Constructor & Destructor Documentation**9.112.2.1 ~ISelectLayer()**

```
virtual nvinfer1::ISelectLayer::~ISelectLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.112.3 Member Data Documentation**9.112.3.1 mImpl**

```
apiv::VSelectLayer* nvinfer1::ISelectLayer::mImpl [protected]
```

The documentation for this class was generated from the following file:

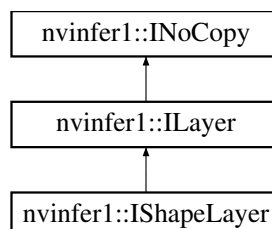
- [NvInfer.h](#)

9.113 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

9.113.1 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.

9.113.2 Constructor & Destructor Documentation

9.113.2.1 ~IShapeLayer()

```
virtual nvinfer1::IShapeLayer::~~IShapeLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.113.3 Member Data Documentation

9.113.3.1 mImpl

```
apiv::VShapeLayer* nvinfer1::IShapeLayer::mImpl [protected]
```

The documentation for this class was generated from the following file:

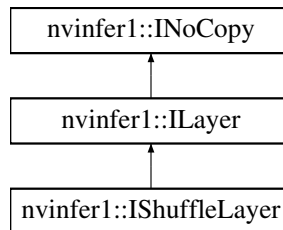
- [NvInfer.h](#)

9.114 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](#)

9.114.1 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.

9.114.2 Constructor & Destructor Documentation

9.114.2.1 ~IShuffleLayer()

```
virtual nvinfer1::IShuffleLayer::~IShuffleLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.114.3 Member Function Documentation

9.114.3.1 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](#)

9.114.3.2 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.

9.114.3.3 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](#)

9.114.3.4 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](#)

9.114.3.5 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](#)

9.114.3.6 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](#).

9.114.3.7 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.

9.114.3.8 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](#)

9.114.3.9 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\(\)](#);

9.114.4 Member Data Documentation

9.114.4.1 mImpl

```
apiv::VShuffleLayer* nvinfer1::IShuffleLayer::mImpl [protected]
```

The documentation for this class was generated from the following file:

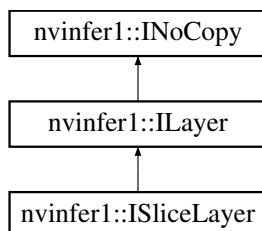
- [NvInfer.h](#)

9.115 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`

9.115.1 Detailed Description

Slices an input tensor into an output tensor based on the offset and strides.

The slice layer has two variants, static and dynamic. Static slice specifies the start, size, and stride dimensions at layer creation time via `Dims` and can use the get/set accessor functions of the `ISliceLayer`. Dynamic slice specifies one or more of start, size or stride as `ITensors`, by using `ILayer::setInput` to add a second, third, or fourth input respectively. The corresponding `Dims` are used if an input is missing or null.

An application can determine if the `ISliceLayer` has a dynamic output shape based on whether the size input (third input) is present and non-null.

The slice layer selects for each dimension a start location from within the input tensor, and copies elements to the output tensor using the specified stride across the input tensor. Start, size, and stride tensors must be 1D Int32 shape tensors if not specified via `Dims`.

An example of using slice on a tensor: `input = {{0, 2, 4}, {1, 3, 5}}` `start = {1, 0}` `size = {1, 2}` `stride = {1, 2}` `output = {{1, 5}}`

When the sliceMode is `kCLAMP` or `kREFLECT`, for each input dimension, if its size is 0 then the corresponding output dimension must be 0 too.

A slice layer can produce a shape tensor if the following conditions are met:

- start, size, and stride are build time constants, either as static [Dims](#) or as constant input tensors.
- The number of elements in the output tensor does not exceed $2 * \text{Dims}::\text{MAX_DIMS}$.

The input tensor is a shape tensor if the output is a shape tensor.

The following constraints must be satisfied to execute this layer on DLA:

- start, size, and stride are build time constants, either as static [Dims](#) or as constant input tensors.
- sliceMode is kDEFAULT.
- Strides are 1 for all dimensions.
- Slicing is not performed on the first dimension
- The input tensor has four dimensions

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.115.2 Constructor & Destructor Documentation

9.115.2.1 ~ISliceLayer()

```
virtual nvinfer1::ISliceLayer::~~ISliceLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.115.3 Member Function Documentation

9.115.3.1 getMode()

```
SliceMode nvinfer1::ISliceLayer::getMode ( ) const [inline], [noexcept]
```

Get the slice mode.

See also

[setMode\(\)](#)

9.115.3.2 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](#)

9.115.3.3 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](#)

9.115.3.4 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](#)

9.115.3.5 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 have the same data phylum as input data type. And 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.

9.115.3.6 setMode()

```
void nvinfer1::ISliceLayer::setMode (
    SliceMode mode ) [inline], [noexcept]
```

Set the slice mode.

See also

[getMode\(\)](#)

9.115.3.7 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](#)

9.115.3.8 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](#)

9.115.3.9 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](#)

9.115.4 Member Data Documentation

9.115.4.1 mImpl

apiv::VSliceLayer* nvinfer1::ISliceLayer::mImpl [protected]

The documentation for this class was generated from the following file:

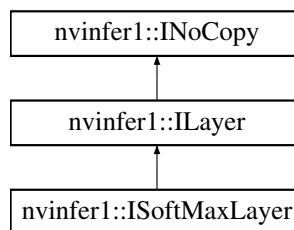
- [NvInfer.h](#)

9.116 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](#)

9.116.1 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.

9.116.2 Constructor & Destructor Documentation

9.116.2.1 ~ISoftMaxLayer()

```
virtual nvinfer1::ISoftMaxLayer::~~ISoftMaxLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.116.3 Member Function Documentation

9.116.3.1 getAxes()

```
uint32_t nvinfer1::ISoftMaxLayer::getAxes ( ) const [inline], [noexcept]
```

Get the axis along which softmax occurs.

See also

[setAxes\(\)](#)

9.116.3.2 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.
-------------	---

9.116.4 Member Data Documentation

9.116.4.1 mImpl

```
apiv::VSoftMaxLayer* nvinfer1::ISoftMaxLayer::mImpl [protected]
```

The documentation for this class was generated from the following file:

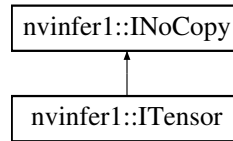
- [NvInfer.h](#)

9.117 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.

Protected Member Functions

- virtual `~ITensor ()` noexcept=default

Protected Attributes

- `apiv::VTensor * mImpl`

9.117.1 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.
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 ~ITensor()

```
virtual nvinfer1::ITensor::~~ITensor ( ) [protected], [virtual], [default], [noexcept]
```

9.117.3 Member Function Documentation

9.117.3.1 `dynamicRangeIsSet()`

```
bool nvinfer1::ITensor::dynamicRangeIsSet ( ) const [inline], [noexcept]
```

Query whether dynamic range is set.

Returns

True if dynamic range is set, false otherwise.

9.117.3.2 `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\(\)](#)

9.117.3.3 `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\(\)](#)

9.117.3.4 `getDimensions()`

```
Dims nvinfer1::ITensor::getDimensions ( ) const [inline], [noexcept]
```

Get the dimensions of a tensor.

Returns

The dimensions of the tensor.

Warning

`getDimensions()` returns a -1 for dimensions that are derived from a wildcard dimension.

See also

[setDimensions\(\)](#)

9.117.3.5 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.

9.117.3.6 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.

9.117.3.7 getLocation()

```
TensorLocation nvinfer1::ITensor::getLocation ( ) const [inline], [noexcept]
```

Get the storage location of a tensor.

Returns

The location of tensor data.

See also

[setLocation\(\)](#)

9.117.3.8 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\(\)](#)

9.117.3.9 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\(\)](#)

9.117.3.10 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.

9.117.3.11 isNetworkInput()

```
bool nvinfer1::ITensor::isNetworkInput ( ) const [inline], [noexcept]
```

Whether the tensor is a network input.

9.117.3.12 isNetworkOutput()

```
bool nvinfer1::ITensor::isNetworkOutput ( ) const [inline], [noexcept]
```

Whether the tensor is a network output.

9.117.3.13 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 be 0D or 1D, 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\(\)](#)

9.117.3.14 resetDynamicRange()

```
void nvinfer1::ITensor::resetDynamicRange ( ) [inline], [noexcept]
```

Undo effect of `setDynamicRange`.

9.117.3.15 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 <code>TensorFormat</code> values that are supported for this tensor.
----------------	---

See also

[ITensor::getAllowedFormats\(\)](#)

[TensorFormats](#)

9.117.3.16 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\(\)](#)

9.117.3.17 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\(\)](#)

9.117.3.18 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 \leq max.

9.117.3.19 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\(\)](#)

9.117.3.20 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.
-------------	-----------

See also

[getName\(\)](#)

9.117.3.21 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\(\)](#)

9.117.4 Member Data Documentation

9.117.4.1 mImpl

```
apiv::VTensor* nvinfer1::ITensor::mImpl [protected]
```

The documentation for this class was generated from the following file:

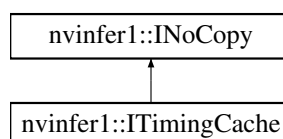
- [NvInfer.h](#)

9.118 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

9.118.1 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](#)

9.118.2 Constructor & Destructor Documentation

9.118.2.1 ~ITimingCache()

```
virtual nvinfer1::ITimingCache::~ITimingCache ( ) [virtual], [default], [noexcept]
```

9.118.3 Member Function Documentation

9.118.3.1 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!

9.118.3.2 reset()

```
bool nvinfer1::ITimingCache::reset ( ) [inline], [noexcept]
```

Empty the timing cache.

Returns

True if reset successfully, false otherwise.

9.118.3.3 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](#)

9.118.4 Member Data Documentation

9.118.4.1 mImpl

```
apiv::VTimingCache* nvinfer1::ITimingCache::mImpl [protected]
```

The documentation for this class was generated from the following file:

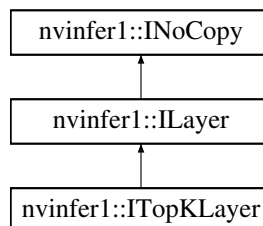
- [NvInfer.h](#)

9.119 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 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.

Protected Member Functions

- virtual [~ITopKLayer](#) () noexcept=default

Protected Attributes

- apiv::VTopKLayer * [mImpl](#)

9.119.1 Detailed Description

Layer that represents a TopK reduction.

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 ~ITopKLayer()

```
virtual nvinfer1::ITopKLayer::~~ITopKLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.119.3 Member Function Documentation

9.119.3.1 getK()

```
int32_t nvinfer1::ITopKLayer::getK ( ) const [inline], [noexcept]
```

Get the k value for the layer.

See also

[setK\(\)](#)

9.119.3.2 `getOperation()`

```
TopKOperation nvinfer1::ITopKLayer::getOperation ( ) const [inline], [noexcept]
```

Get the operation for the layer.

See also

[setOperation\(\)](#), [TopKOperation](#)

9.119.3.3 `getReduceAxes()`

```
uint32_t nvinfer1::ITopKLayer::getReduceAxes ( ) const [inline], [noexcept]
```

Get the axes to reduce for the layer.

See also

[setReduceAxes\(\)](#)

9.119.3.4 `setK()`

```
void nvinfer1::ITopKLayer::setK (
    int32_t k ) [inline], [noexcept]
```

Set the k value for the layer.

Currently only values up to 3840 are supported.

See also

[getK\(\)](#)

9.119.3.5 `setOperation()`

```
void nvinfer1::ITopKLayer::setOperation (
    TopKOperation op ) [inline], [noexcept]
```

Set the operation for the layer.

See also

[getOperation\(\)](#), [TopKOperation](#)

9.119.3.6 setReduceAxes()

```
void nvinfer1::ITopKLayer::setReduceAxes (
    uint32_t reduceAxes ) [inline], [noexcept]
```

Set which axes to reduce for the layer.

See also

[getReduceAxes\(\)](#)

9.119.4 Member Data Documentation

9.119.4.1 mImpl

```
apiv::VTopKLayer* nvinfer1::ITopKLayer::mImpl [protected]
```

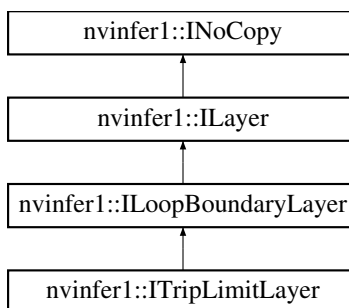
The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.120 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](#)

9.120.1 Constructor & Destructor Documentation

9.120.1.1 ~ITripLimitLayer()

```
virtual nvinfer1::ITripLimitLayer::~~ITripLimitLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.120.2 Member Function Documentation

9.120.2.1 getTripLimit()

```
TripLimit nvinfer1::ITripLimitLayer::getTripLimit ( ) const [inline], [noexcept]
```

9.120.3 Member Data Documentation

9.120.3.1 mImpl

```
apiv::VTripLimitLayer* nvinfer1::ITripLimitLayer::mImpl [protected]
```

The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.121 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](#) weights←Type=[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.

9.121.1 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.

9.121.2 Constructor & Destructor Documentation

9.121.2.1 [~IUffParser\(\)](#)

```
virtual nvuffparser::IUffParser::~~IUffParser ( ) [virtual], [default], [noexcept]
```

9.121.3 Member Function Documentation

9.121.3.1 destroy()

```
virtual TRT\_DEPRECATED void nvuffparser::IUffParser::destroy ( ) [pure virtual], [noexcept]
```

Deprecated Use delete instead. Deprecated in TRT 8.0.

9.121.3.2 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\(\)](#)

9.121.3.3 getUffRequiredVersionMajor()

```
virtual int32_t nvuffparser::IUffParser::getUffRequiredVersionMajor ( ) [pure virtual], [noexcept]
```

Return Version Major of the UFF.

9.121.3.4 getUffRequiredVersionMinor()

```
virtual int32_t nvuffparser::IUffParser::getUffRequiredVersionMinor ( ) [pure virtual], [noexcept]
```

Return Version Minor of the UFF.

9.121.3.5 getUffRequiredVersionPatch()

```
virtual int32_t nvuffparser::IUffParser::getUffRequiredVersionPatch ( ) [pure virtual], [noexcept]
```

Return Patch Version of the UFF.

9.121.3.6 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.

9.121.3.7 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.
<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.

9.121.3.8 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.

9.121.3.9 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.
-------------------	--------------

9.121.3.10 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\(\)](#)

9.121.3.11 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:

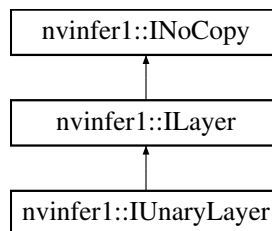
- [NvUffParser.h](#)

9.122 nvinfer1::UnaryLayer Class Reference

Layer that represents an unary operation.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::UnaryLayer:



Public Member Functions

- void [setOperation](#) ([UnaryOperation](#) op) noexcept
Set the unary operation for the layer.
- [UnaryOperation](#) [getOperation](#) () const noexcept
Get the unary operation for the layer.

Protected Member Functions

- virtual [~UnaryLayer](#) () noexcept=default

Protected Attributes

- apiv::VUnaryLayer * [mImpl](#)

9.122.1 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.

9.122.2 Constructor & Destructor Documentation**9.122.2.1 ~UnaryLayer()**

```
virtual nvinfer1::UnaryLayer::~UnaryLayer ( ) [protected], [virtual], [default], [noexcept]
```

9.122.3 Member Function Documentation**9.122.3.1 getOperation()**

```
UnaryOperation nvinfer1::UnaryLayer::getOperation ( ) const [inline], [noexcept]
```

Get the unary operation for the layer.

See also

[setOperation\(\)](#), [UnaryOperation](#)

9.122.3.2 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](#)

9.122.4 Member Data Documentation

9.122.4.1 mImpl

```
apiv::VUnaryLayer* nvinfer1::UnaryLayer::mImpl [protected]
```

The documentation for this class was generated from the following file:

- [NvInfer.h](#)

9.123 nvinfer1::plugin::NMSParameters Struct Reference

The [NMSParameters](#) are used by the BatchedNMSPugin 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](#)

9.123.1 Detailed Description

The [NMSParameters](#) are used by the BatchedNMSPugin 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.

9.123.2 Member Data Documentation

9.123.2.1 backgroundLabelId

```
int32_t nvinfer1::plugin::NMSParameters::backgroundLabelId
```

9.123.2.2 iouThreshold

```
float nvinfer1::plugin::NMSParameters::iouThreshold
```

9.123.2.3 isNormalized

```
bool nvinfer1::plugin::NMSParameters::isNormalized
```

9.123.2.4 keepTopK

```
int32_t nvinfer1::plugin::NMSParameters::keepTopK
```

9.123.2.5 numClasses

```
int32_t nvinfer1::plugin::NMSParameters::numClasses
```

9.123.2.6 scoreThreshold

```
float nvinfer1::plugin::NMSParameters::scoreThreshold
```

9.123.2.7 shareLocation

```
bool nvinfer1::plugin::NMSParameters::shareLocation
```

9.123.2.8 topK

```
int32_t nvinfer1::plugin::NMSParameters::topK
```

The documentation for this struct was generated from the following file:

- [NvInferPluginUtils.h](#)

9.124 nvinfer1::Permutation Struct Reference

```
#include <NvInfer.h>
```

Public Attributes

- int32_t [order](#) [[Dims::MAX_DIMS](#)]

9.124.1 Member Data Documentation

9.124.1.1 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](#)

9.125 nvinfer1::PluginField Class Reference

Structure containing plugin attribute field names and associated data This information can be parsed to decode necessary plugin metadata.

```
#include <NvInferRuntimeCommon.h>
```

Public Member Functions

- [PluginField](#) ([AsciiChar](#) const *const name_=nullptr, void const *const data_=nullptr, [PluginFieldType](#) const type_=[PluginFieldType::kUNKNOWN](#), int32_t const length_=0) noexcept

Public Attributes

- [AsciiChar](#) const * [name](#)
Plugin field attribute name.
- void const * [data](#)
Plugin field attribute data.
- [PluginFieldType](#) [type](#)
Plugin field attribute type.
- int32_t [length](#)
Number of data entries in the Plugin attribute.

9.125.1 Detailed Description

Structure containing plugin attribute field names and associated data This information can be parsed to decode necessary plugin metadata.

9.125.2 Constructor & Destructor Documentation

9.125.2.1 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]
```

9.125.3 Member Data Documentation

9.125.3.1 data

```
void const* nvinfer1::PluginField::data
```

Plugin field attribute data.

9.125.3.2 length

```
int32_t nvinfer1::PluginField::length
```

Number of data entries in the Plugin attribute.

9.125.3.3 name

```
AsciiChar const* nvinfer1::PluginField::name
```

Plugin field attribute name.

9.125.3.4 type

```
PluginFieldType nvinfer1::PluginField::type
```

Plugin field attribute type.

See also

[PluginFieldType](#)

The documentation for this class was generated from the following file:

- [NvInferRuntimeCommon.h](#)

9.126 nvinfer1::PluginFieldCollection Struct Reference

Plugin field collection struct.

```
#include <NvInferRuntimeCommon.h>
```


Public Attributes

- `int32_t nbFields`
Number of [PluginField](#) entries.
- `PluginField const * fields`
Pointer to [PluginField](#) entries.

9.126.1 Detailed Description

Plugin field collection struct.

9.126.2 Member Data Documentation

9.126.2.1 fields

```
PluginField const* nvinfer1::PluginFieldCollection::fields
```

Pointer to [PluginField](#) entries.

9.126.2.2 nbFields

```
int32_t nvinfer1::PluginFieldCollection::nbFields
```

Number of [PluginField](#) entries.

The documentation for this struct was generated from the following file:

- [NvInferRuntimeCommon.h](#)

9.127 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 \(\)](#)

9.127.1 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.

9.127.2 Constructor & Destructor Documentation**9.127.2.1 PluginRegistrar()**

```
template<typename T >
nvinfer1::PluginRegistrar< T >::PluginRegistrar ( ) [inline]
```

The documentation for this class was generated from the following file:

- [NvInferRuntime.h](#)

9.128 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 \(\)](#)

9.128.1 Detailed Description

```
template<typename T>
class nvinfer1::safe::PluginRegistrar< T >
```

Register the plugin creator to the registry The static registry object will be instantiated when the plugin library is loaded. This static object will register all creators available in the library to the registry.

Warning

Statically registering plugins should be avoided in the automotive safety context as the application developer should first register an error recorder with the plugin registry via `IPluginRegistry::setErrorRecorder()` before using `IPluginRegistry::registerCreator()` or other methods.

9.128.2 Constructor & Destructor Documentation

9.128.2.1 PluginRegistrar()

```
template<typename T >
nvinfer1::safe::PluginRegistrar< T >::PluginRegistrar ( ) [inline]
```

The documentation for this class was generated from the following file:

- [NvInferSafeRuntime.h](#)

9.129 nvinfer1::PluginTensorDesc Struct Reference

Fields that a plugin might see for an input or output.

```
#include <NvInferRuntimeCommon.h>
```

Public Attributes

- [Dims](#) `dims`
Dimensions.
- [DataType](#) `type`
- [TensorFormat](#) `format`
Tensor format.
- float `scale`
Scale for INT8 data type.

9.129.1 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](#)

9.129.2 Member Data Documentation

9.129.2.1 dims

`Dims` `nvinfer1::PluginTensorDesc::dims`

Dimensions.

9.129.2.2 format

`TensorFormat` `nvinfer1::PluginTensorDesc::format`

Tensor format.

9.129.2.3 scale

`float` `nvinfer1::PluginTensorDesc::scale`

Scale for INT8 data type.

9.129.2.4 type

`DataType` `nvinfer1::PluginTensorDesc::type`

Warning

`DataType:kBOOL` not supported.

The documentation for this struct was generated from the following file:

- [NvInferRuntimeCommon.h](#)

9.130 PluginVersion Struct Reference

Definition of plugin versions.

```
#include <NvInferRuntimeCommon.h>
```

9.130.1 Detailed Description

Definition of plugin versions.

Tag for plug-in versions. Used in upper byte of `getTensorRTVersion()`.

The documentation for this struct was generated from the following file:

- [NvInferRuntimeCommon.h](#)

9.131 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](#)

9.131.1 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.

Parameters

<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.

9.131.2 Member Data Documentation**9.131.2.1 aspectRatios**

```
float * nvinfer1::plugin::PriorBoxParameters::aspectRatios
```

9.131.2.2 clip

```
bool nvinfer1::plugin::PriorBoxParameters::clip
```

9.131.2.3 flip

```
bool nvinfer1::plugin::PriorBoxParameters::flip
```

9.131.2.4 imgH

```
int32_t nvinfer1::plugin::PriorBoxParameters::imgH
```

9.131.2.5 imgW

```
int32_t nvinfer1::plugin::PriorBoxParameters::imgW
```

9.131.2.6 maxSize

```
float * nvinfer1::plugin::PriorBoxParameters::maxSize
```

9.131.2.7 minSize

```
float* nvinfer1::plugin::PriorBoxParameters::minSize
```

9.131.2.8 numAspectRatios

```
int32_t nvinfer1::plugin::PriorBoxParameters::numAspectRatios
```

9.131.2.9 numMaxSize

```
int32_t nvinfer1::plugin::PriorBoxParameters::numMaxSize
```

9.131.2.10 numMinSize

```
int32_t nvinfer1::plugin::PriorBoxParameters::numMinSize
```

9.131.2.11 offset

```
float nvinfer1::plugin::PriorBoxParameters::offset
```

9.131.2.12 stepH

```
float nvinfer1::plugin::PriorBoxParameters::stepH
```

9.131.2.13 stepW

```
float nvinfer1::plugin::PriorBoxParameters::stepW
```

9.131.2.14 variance

```
float nvinfer1::plugin::PriorBoxParameters::variance[4]
```

The documentation for this struct was generated from the following file:

- [NvInferPluginUtils.h](#)

9.132 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]

9.132.1 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.

9.132.2 Member Data Documentation

9.132.2.1 data

```
int32_t nvinfer1::plugin::Quadruple::data[4]
```

The documentation for this struct was generated from the following file:

- [NvInferPluginUtils.h](#)

9.133 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`

9.133.1 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.

9.133.2 Member Data Documentation

9.133.2.1 classes

```
int32_t nvinfer1::plugin::RegionParameters::classes
```

9.133.2.2 coords

```
int32_t nvinfer1::plugin::RegionParameters::coords
```

9.133.2.3 num

```
int32_t nvinfer1::plugin::RegionParameters::num
```

9.133.2.4 smTree

```
softmaxTree* nvinfer1::plugin::RegionParameters::smTree
```

The documentation for this struct was generated from the following file:

- [NvInferPluginUtils.h](#)

9.134 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](#)

9.134.1 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.

9.134.2 Member Data Documentation

9.134.2.1 anchorsRatioCount

```
int32_t nvinfer1::plugin::RPROIParams::anchorsRatioCount
```

9.134.2.2 anchorsScaleCount

```
int32_t nvinfer1::plugin::RPROIParams::anchorsScaleCount
```

9.134.2.3 featureStride

```
int32_t nvinfer1::plugin::RPROIParams::featureStride
```

9.134.2.4 iouThreshold

```
float nvinfer1::plugin::RPROIParams::iouThreshold
```

9.134.2.5 minBoxSize

```
float nvinfer1::plugin::RPROIParams::minBoxSize
```

9.134.2.6 nmsMaxOut

```
int32_t nvinfer1::plugin::RPROIParams::nmsMaxOut
```

9.134.2.7 poolingH

```
int32_t nvinfer1::plugin::RPROIParams::poolingH
```

9.134.2.8 poolingW

```
int32_t nvinfer1::plugin::RPROIParams::poolingW
```

9.134.2.9 preNmsTop

```
int32_t nvinfer1::plugin::RPROIParams::preNmsTop
```

9.134.2.10 spatialScale

```
float nvinfer1::plugin::RPROIParams::spatialScale
```

The documentation for this struct was generated from the following file:

- [NvInferPluginUtils.h](#)

9.135 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`

9.135.1 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.

9.135.2 Member Data Documentation

9.135.2.1 child

```
int32_t* nvinfer1::plugin::softmaxTree::child
```

9.135.2.2 group

```
int32_t* nvinfer1::plugin::softmaxTree::group
```

9.135.2.3 groupOffset

```
int32_t* nvinfer1::plugin::softmaxTree::groupOffset
```

9.135.2.4 groups

```
int32_t nvinfer1::plugin::softmaxTree::groups
```

9.135.2.5 groupSize

```
int32_t* nvinfer1::plugin::softmaxTree::groupSize
```

9.135.2.6 leaf

```
int32_t* nvinfer1::plugin::softmaxTree::leaf
```

9.135.2.7 n

```
int32_t nvinfer1::plugin::softmaxTree::n
```

9.135.2.8 name

```
char** nvinfer1::plugin::softmaxTree::name
```

9.135.2.9 parent

```
int32_t* nvinfer1::plugin::softmaxTree::parent
```

The documentation for this struct was generated from the following file:

- [NvInferPluginUtils.h](#)

9.136 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.

9.136.1 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`).

9.136.2 Member Data Documentation

9.136.2.1 `count`

```
int64_t nvinfer1::Weights::count
```

The number of weights in the array.

9.136.2.2 `type`

```
DataType nvinfer1::Weights::type
```

The type of the weights.

9.136.2.3 `values`

```
void const* nvinfer1::Weights::values
```

The weight values, in a contiguous array.

The documentation for this class was generated from the following file:

- [NvInferRuntime.h](#)

Chapter 10

File Documentation

10.1 NvCaffeParser.h File Reference

```
#include "NvInfer.h"
```

Classes

- class [nvcaffeparser1::IBlobNameToTensor](#)
Object used to store and query Tensors after they have been extracted from a Caffe model using the [ICaffeParser](#).
- class [nvcaffeparser1::IBinaryProtoBlob](#)
Object used to store and query data extracted from a binaryproto file using the [ICaffeParser](#).
- class [nvcaffeparser1::IPluginFactoryV2](#)
Plugin factory used to configure plugins.
- class [nvcaffeparser1::ICaffeParser](#)
Class used for parsing Caffe models.

Namespaces

- namespace [nvcaffeparser1](#)
The TensorRT Caffe parser API namespace.

Functions

- [ICaffeParser * nvcaffeparser1::createCaffeParser \(\) noexcept](#)
Creates a [ICaffeParser](#) object.
- [void nvcaffeparser1::shutdownProtobufLibrary \(\) noexcept](#)
Shuts down protocol buffers library.

10.1.1 Detailed Description

This is the API for the Caffe Parser

10.2 NvCaffeParser.h

[Go to the documentation of this file.](#)

```

1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_CAFFE_PARSER_H
14 #define NV_CAFFE_PARSER_H
15
16 #include "NvInfer.h"
17
18
19 namespace nvcaffeparser1
20 {
21
22     class IBlobNameToTensor
23     {
24     public:
25         virtual nvinfer1::ITensor* find(char const* name) const noexcept = 0;
26
27     protected:
28         virtual ~IBlobNameToTensor() {}
29     };
30
31     class IBinaryProtoBlob
32     {
33     public:
34         virtual void const* getData() noexcept = 0;
35         virtual nvinfer1::Dims4 getDimensions() noexcept = 0;
36         virtual nvinfer1::DataType getDataType() noexcept = 0;
37         TRT_DEPRECATED virtual void destroy() noexcept = 0;
38         virtual ~IBinaryProtoBlob() noexcept = default;
39     };
40
41     class IPluginFactoryV2
42     {
43     public:
44         virtual bool isPluginV2(char const* layerName) noexcept = 0;
45
46         virtual nvinfer1::IPluginV2* createPlugin(char const* layerName, nvinfer1::Weights const* weights,
47             int32_t nbWeights, char const* libNamespace = "") noexcept = 0;
48
49         virtual ~IPluginFactoryV2() noexcept = default;
50     };
51
52     class ICaffeParser
53     {
54     public:
55         virtual IBlobNameToTensor const* parse(char const* deploy, char const* model,
56             nvinfer1::INetworkDefinition& network,
57             nvinfer1::DataType weightType) noexcept = 0;
58
59         virtual IBlobNameToTensor const* parseBuffers(uint8_t const* deployBuffer, std::size_t deployLength,
60             uint8_t const* modelBuffer, std::size_t modelLength, nvinfer1::INetworkDefinition& network,
61             nvinfer1::DataType weightType) noexcept = 0;
62
63         virtual IBinaryProtoBlob* parseBinaryProto(char const* fileName) noexcept = 0;
64
65         virtual void setProtobufBufferSize(size_t size) noexcept = 0;
66
67         TRT_DEPRECATED virtual void destroy() noexcept = 0;
68     };
69
70 }
71
72 #endif

```

```

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::IRaggedSoftMaxLayer](#)
A RaggedSoftmax layer in a network definition.
- class [nvinfer1::IIdentityLayer](#)
A layer that represents the identity function.
- class [nvinfer1::IConstantLayer](#)
Layer that represents a constant value.
- class [nvinfer1::IParametricReLULayer](#)
Layer that represents a parametric ReLU operation.
- struct [nvinfer1::impl::EnumMaxImpl< ResizeMode >](#)
- 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::IIfConditionalBoundaryLayer](#)
- class [nvinfer1::IConditionLayer](#)
- class [nvinfer1::IIfConditionalOutputLayer](#)
- class [nvinfer1::IIfConditionalInputLayer](#)
- class [nvinfer1::IIfConditional](#)

- class [nvinfer1::IRecurrenceLayer](#)
- class [nvinfer1::ILoopOutputLayer](#)
- class [nvinfer1::ITripLimitLayer](#)
- class [nvinfer1::IIteratorLayer](#)
- 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::INetworkDefinition](#)
A network definition for input to the builder.
- class [nvinfer1::IInt8Calibrator](#)
Application-implemented interface for calibration.
- class [nvinfer1::IInt8EntropyCalibrator](#)
- class [nvinfer1::IInt8EntropyCalibrator2](#)
- class [nvinfer1::IInt8MinMaxCalibrator](#)
- class [nvinfer1::IInt8LegacyCalibrator](#)
- class [nvinfer1::IAlgorithmIOInfo](#)
Carries information about input or output of the algorithm. [IAlgorithmIOInfo](#) for all the input and output along with [IAlgorithmVariant](#) denotes the variation of algorithm and can be used to select or reproduce an algorithm using [IAlgorithmSelector::selectAlgorithms\(\)](#).
- class [nvinfer1::IAlgorithmVariant](#)
provides a unique 128-bit identifier, which along with the input and output information denotes the variation of algorithm and can be used to select or reproduce an algorithm, using [IAlgorithmSelector::selectAlgorithms\(\)](#)
- class [nvinfer1::IAlgorithmContext](#)
Describes the context and requirements, that could be fulfilled by one or more instances of [IAlgorithm](#).
- class [nvinfer1::IAlgorithm](#)
Describes a variation of execution of a layer. An algorithm is represented by [IAlgorithmVariant](#) and the [IAlgorithmIOInfo](#) for each of its inputs and outputs. An algorithm can be selected or reproduced using [AlgorithmSelector::selectAlgorithms\(\)](#).”.
- class [nvinfer1::IAlgorithmSelector](#)
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.
- class [nvinfer1::IBuilderConfig](#)
Holds properties for configuring a builder to produce an engine.
- class [nvinfer1::IBuilder](#)
Builds an engine from a network definition.

Namespaces

- namespace `nvinfer1`
The TensorRT API version 1 namespace.
- namespace `nvinfer1::impl`

Typedefs

- using `nvinfer1::TensorFormats = uint32_t`
It is capable of representing one or more `TensorFormat` by binary OR operations, e.g., `IU << TensorFormat::kCHW4 | IU << TensorFormat::kCHW32`.
- using `nvinfer1::QuantizationFlags = uint32_t`
Represents one or more `QuantizationFlag` values using binary OR operations.
- using `nvinfer1::BuilderFlags = uint32_t`
Represents one or more `QuantizationFlag` values using binary OR operations, e.g., `IU << BuilderFlag::kFP16 | IU << BuilderFlag::kDEBUG`.
- using `nvinfer1::NetworkDefinitionCreationFlags = uint32_t`
Represents one or more `NetworkDefinitionCreationFlag` flags using binary OR operations. e.g., `IU << 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` }
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 }
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::SliceMode` : `int32_t` {
`nvinfer1::kDEFAULT` = 0, `nvinfer1::kWRAP` = 1, `nvinfer1::kCLAMP` = 2, `nvinfer1::kFILL` = 3,
`nvinfer1::kREFLECT` = 4 }
Controls how `ISliceLayer` handles 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::ResizeMode` : `int32_t` { `nvinfer1::kNEAREST` = 0 , `nvinfer1::kLINEAR` = 1 }

Enumerates various modes of resize in the resize layer. Resize mode set using `setResizeMode()`.

- 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 }

Enumerates the tensor fill operations that may be performed by a fill layer.

- enum class `nvinfer1::ScatterMode` : `int32_t` { `nvinfer1::kELEMENT` = 0 , `nvinfer1::kND` = 1 }

Control form of `IScatterLayer`.

- enum class `nvinfer1::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 }

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 }

The type for memory pools used by TensorRT.

- 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< SliceMode > () 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< CalibrationAlgoType > () noexcept`
- `template<> constexpr int32_t nvinfer1::EnumMax< QuantizationFlag > () noexcept`
- `template<> constexpr int32_t nvinfer1::EnumMax< BuilderFlag > () noexcept`
- `template<> constexpr int32_t nvinfer1::EnumMax< MemoryPoolType > () noexcept`
- `template<> constexpr int32_t nvinfer1::EnumMax< NetworkDefinitionCreationFlag > () noexcept`
- `nvinfer1::IPluginRegistry * nvinfer1::getBuilderPluginRegistry (nvinfer1::EngineCapability capability) noexcept`

Return the plugin registry for the given capability or nullptr if no registry exists.

10.3.1 Detailed Description

TensorRT Versioning follows Semantic Versioning Guidelines specified here: <https://semver.org/>

This is the top-level API file for TensorRT.

10.4 NvInfer.h

[Go to the documentation of this file.](#)

```

1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_INFERR_H
14 #define NV_INFERR_H
15
16 #include "NvInferLegacyDims.h"
17 #include "NvInferRuntime.h"
18

```



```

27 //
30
36
42 namespace nvinfer1
43 {
44
52 enum class LayerType : int32_t
53 {
54     kCONVOLUTION = 0,
55     kFULLY.CONNECTED = 1,
56     kACTIVATION = 2,
57     kPOOLING = 3,
58     kLRN = 4,
59     kSCALE = 5,
60     kSOFTMAX = 6,
61     kDECONVOLUTION = 7,
62     kCONCATENATION = 8,
63     kELEMENTWISE = 9,
64     kPLUGIN = 10,
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 };
96
102 template <>
103 constexpr inline int32_t EnumMax<LayerType>() noexcept
104 {
105     return 41;
106 }
107
114 using TensorFormats = uint32_t;
115
121 enum class ActivationType : int32_t
122 {
123     kRELU = 0,
124     kSIGMOID = 1,
125     kTANH = 2,
126     kLEAKY.RELU = 3,
127     kELU = 4,
128     kSELU = 5,
129     kSOFTSIGN = 6,
130     kSOFTPLUS = 7,
131     kCLIP = 8,
132     kHARD.SIGMOID = 9,
133     kSCALED.TANH = 10,
134     kTHRESHOLDED.RELU = 11
135 };
136
137 namespace impl
138 {
144 template <>
145 struct EnumMaxImpl<ActivationType>
146 {
147     static constexpr int32_t KVALUE = 12;

```

```

148 };
149 } // namespace impl
150
151 class ITensor : public INoCopy
152 {
153 public:
154     void setName(char const* name) noexcept
155     {
156         mImpl->setName(name);
157     }
158
159     char const* getName() const noexcept
160     {
161         return mImpl->getName();
162     }
163
164     void setDimensions(Dims dimensions) noexcept
165     {
166         mImpl->setDimensions(dimensions);
167     }
168
169     Dims getDimensions() const noexcept
170     {
171         return mImpl->getDimensions();
172     }
173
174     void setType(DataType type) noexcept
175     {
176         mImpl->setType(type);
177     }
178
179     DataType getType() const noexcept
180     {
181         return mImpl->getType();
182     }
183
184     bool setDynamicRange(float min, float max) noexcept
185     {
186         return mImpl->setDynamicRange(min, max);
187     }
188
189     bool isNetworkInput() const noexcept
190     {
191         return mImpl->isNetworkInput();
192     }
193
194     bool isNetworkOutput() const noexcept
195     {
196         return mImpl->isNetworkOutput();
197     }
198
199     void setBroadcastAcrossBatch(bool broadcastAcrossBatch) noexcept
200     {
201         mImpl->setBroadcastAcrossBatch(broadcastAcrossBatch);
202     }
203
204     bool getBroadcastAcrossBatch() const noexcept
205     {
206         return mImpl->getBroadcastAcrossBatch();
207     }
208
209     TensorLocation getLocation() const noexcept
210     {
211         return mImpl->getLocation();
212     }
213
214     void setLocation(TensorLocation location) noexcept
215     {
216         mImpl->setLocation(location);
217     }
218
219     bool dynamicRangeIsSet() const noexcept
220     {
221         return mImpl->dynamicRangeIsSet();
222     }
223
224     void resetDynamicRange() noexcept
225     {
226         mImpl->resetDynamicRange();
227     }
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368

```

```

374 float getDynamicRangeMin() const noexcept
375 {
376     return mImpl->getDynamicRangeMin();
377 }
378
384 float getDynamicRangeMax() const noexcept
385 {
386     return mImpl->getDynamicRangeMax();
387 }
388
403 void setAllowedFormats(TensorFormats formats) noexcept
404 {
405     mImpl->setAllowedFormats(formats);
406 }
407
416 TensorFormats getAllowedFormats() const noexcept
417 {
418     return mImpl->getAllowedFormats();
419 }
420
451 bool isShapeTensor() const noexcept
452 {
453     return mImpl->isShapeTensor();
454 }
455
474 bool isExecutionTensor() const noexcept
475 {
476     return mImpl->isExecutionTensor();
477 }
478
479 protected:
480     apiv::VTensor* mImpl;
481     virtual ITensor() noexcept = default;
482 };
483
491 class ILayer : public INoCopy
492 {
493 public:
494     LayerType getType() const noexcept
495     {
496         return mLayer->getType();
497     }
498
499     void setName(char const* name) noexcept
500     {
501         mLayer->setName(name);
502     }
503
504     char const* getName() const noexcept
505     {
506         return mLayer->getName();
507     }
508
509     int32_t getNbInputs() const noexcept
510     {
511         return mLayer->getNbInputs();
512     }
513
514     ITensor* getInput(int32_t index) const noexcept
515     {
516         return mLayer->getInput(index);
517     }
518
519     int32_t getNbOutputs() const noexcept
520     {
521         return mLayer->getNbOutputs();
522     }
523
524     ITensor* getOutput(int32_t index) const noexcept
525     {
526         return mLayer->getOutput(index);
527     }
528
529     void setInput(int32_t index, ITensor& tensor) noexcept
530     {
531         return mLayer->setInput(index, tensor);
532     }
533
534     void setPrecision(DataType dataType) noexcept
535     {

```

```

609     mLayer->setPrecision(dataType);
610 }
611
619 DataType getPrecision() const noexcept
620 {
621     return mLayer->getPrecision();
622 }
623
631 bool precisionIsSet() const noexcept
632 {
633     return mLayer->precisionIsSet();
634 }
635
641 void resetPrecision() noexcept
642 {
643     mLayer->resetPrecision();
644 }
645
679 void setOutputType(int32_t index, DataType dataType) noexcept
680 {
681     mLayer->setOutputType(index, dataType);
682 }
683
693 DataType getOutputType(int32_t index) const noexcept
694 {
695     return mLayer->getOutputType(index);
696 }
697
706 bool outputTypeIsSet(int32_t index) const noexcept
707 {
708     return mLayer->outputTypeIsSet(index);
709 }
710
718 void resetOutputType(int32_t index) noexcept
719 {
720     return mLayer->resetOutputType(index);
721 }
722
723 protected:
724     virtual ~ILayer() noexcept = default;
725     apiv::VLayer* mLayer;
726 };
727
950 enum class PaddingMode : int32_t
951 {
952     kEXPLICIT_ROUND_DOWN = 0,
953     kEXPLICIT_ROUND_UP = 1,
954     kSAME_UPPER = 2,
955     kSAME_LOWER = 3,
956     kCAFFE_ROUND_DOWN = 4,
957     kCAFFE_ROUND_UP = 5
958 };
959
960 namespace impl
961 {
962     template <>
963     struct EnumMaxImpl<PaddingMode>
964     {
965         static constexpr int32_t kVALUE = 6;
966     };
967 } // namespace impl
968
986 class IConvolutionLayer : public ILayer
987 {
988 public:
998     TRT_DEPRECATED void setKernelSize(DimsHW kernelSize) noexcept
999     {
1000         mImpl->setKernelSize(kernelSize);
1001     }
1002
1010     TRT_DEPRECATED DimsHW getKernelSize() const noexcept
1011     {
1012         return mImpl->getKernelSize();
1013     }
1014
1022     void setNbOutputMaps(int32_t nbOutputMaps) noexcept
1023     {
1024         mImpl->setNbOutputMaps(nbOutputMaps);
1025     }
1026
1032     int32_t getNbOutputMaps() const noexcept

```

```

1033     {
1034         return mImpl->getNbOutputMaps();
1035     }
1036
1048     TRT_DEPRECATED void setStride(DimsHW stride) noexcept
1049     {
1050         mImpl->setStride(stride);
1051     }
1052
1058     TRT_DEPRECATED DimsHW getStride() const noexcept
1059     {
1060         return mImpl->getStride();
1061     }
1062
1078     TRT_DEPRECATED void setPadding(DimsHW padding) noexcept
1079     {
1080         return mImpl->setPadding(padding);
1081     }
1082
1090     TRT_DEPRECATED DimsHW getPadding() const noexcept
1091     {
1092         return mImpl->getPadding();
1093     }
1094
1110     void setNbGroups(int32_t nbGroups) noexcept
1111     {
1112         mImpl->setNbGroups(nbGroups);
1113     }
1114
1120     int32_t getNbGroups() const noexcept
1121     {
1122         return mImpl->getNbGroups();
1123     }
1124
1134     void setKernelWeights(Weights weights) noexcept
1135     {
1136         mImpl->setKernelWeights(weights);
1137     }
1138
1144     Weights getKernelWeights() const noexcept
1145     {
1146         return mImpl->getKernelWeights();
1147     }
1148
1159     void setBiasWeights(Weights weights) noexcept
1160     {
1161         mImpl->setBiasWeights(weights);
1162     }
1163
1169     Weights getBiasWeights() const noexcept
1170     {
1171         return mImpl->getBiasWeights();
1172     }
1173
1185     TRT_DEPRECATED void setDilation(DimsHW dilation) noexcept
1186     {
1187         return mImpl->setDilation(dilation);
1188     }
1189
1197     TRT_DEPRECATED DimsHW getDilation() const noexcept
1198     {
1199         return mImpl->getDilation();
1200     }
1201
1214     void setPrePadding(Dims padding) noexcept
1215     {
1216         mImpl->setPrePadding(padding);
1217     }
1218
1224     Dims getPrePadding() const noexcept
1225     {
1226         return mImpl->getPrePadding();
1227     }
1228
1241     void setPostPadding(Dims padding) noexcept
1242     {
1243         mImpl->setPostPadding(padding);
1244     }
1245
1251     Dims getPostPadding() const noexcept
1252     {

```

```

1253     return mImpl->getPostPadding();
1254 }
1255
1265 void setPaddingMode(PaddingMode paddingMode) noexcept
1266 {
1267     mImpl->setPaddingMode(paddingMode);
1268 }
1269
1277 PaddingMode getPaddingMode() const noexcept
1278 {
1279     return mImpl->getPaddingMode();
1280 }
1281
1290 void setKernelSizeNd(Dims kernelSize) noexcept
1291 {
1292     mImpl->setKernelSizeNd(kernelSize);
1293 }
1294
1300 Dims getKernelSizeNd() const noexcept
1301 {
1302     return mImpl->getKernelSizeNd();
1303 }
1304
1315 void setStrideNd(Dims stride) noexcept
1316 {
1317     mImpl->setStrideNd(stride);
1318 }
1319
1325 Dims getStrideNd() const noexcept
1326 {
1327     return mImpl->getStrideNd();
1328 }
1329
1343 void setPaddingNd(Dims padding) noexcept
1344 {
1345     mImpl->setPaddingNd(padding);
1346 }
1347
1355 Dims getPaddingNd() const noexcept
1356 {
1357     return mImpl->getPaddingNd();
1358 }
1359
1369 void setDilationNd(Dims dilation) noexcept
1370 {
1371     mImpl->setDilationNd(dilation);
1372 }
1373
1379 Dims getDilationNd() const noexcept
1380 {
1381     return mImpl->getDilationNd();
1382 }
1383
1405 using ILayer::setInput;
1406
1407 protected:
1408     virtual ~IConvolutionLayer() noexcept = default;
1409     apiv::VConvolutionLayer* mImpl;
1410 };
1411
1443 class TRT_DEPRECATED IFullyConnectedLayer : public ILayer
1444 {
1445 public:
1453     void setNbOutputChannels(int32_t nbOutputs) noexcept
1454     {
1455         mImpl->setNbOutputChannels(nbOutputs);
1456     }
1457
1463     int32_t getNbOutputChannels() const noexcept
1464     {
1465         return mImpl->getNbOutputChannels();
1466     }
1467
1473     void setKernelWeights(Weights weights) noexcept
1474     {
1475         mImpl->setKernelWeights(weights);
1476     }
1477
1483     Weights getKernelWeights() const noexcept
1484     {
1485         return mImpl->getKernelWeights();

```

```

1486     }
1487
1495     void setBiasWeights(Weights weights) noexcept
1496     {
1497         mImpl->setBiasWeights(weights);
1498     }
1499
1505     Weights getBiasWeights() const noexcept
1506     {
1507         return mImpl->getBiasWeights();
1508     }
1509
1531     using ILayer::setInput;
1532
1533 protected:
1534     virtual ~IFullyConnectedLayer() noexcept = default;
1535     apiv::VFullyConnectedLayer* mImpl;
1536 };
1537
1551 class IActivationLayer : public ILayer
1552 {
1553 public:
1561     void setActivationType(ActivationType type) noexcept
1562     {
1563         mImpl->setActivationType(type);
1564     }
1565
1571     ActivationType getActivationType() const noexcept
1572     {
1573         return mImpl->getActivationType();
1574     }
1575
1586     void setAlpha(float alpha) noexcept
1587     {
1588         mImpl->setAlpha(alpha);
1589     }
1590
1600     void setBeta(float beta) noexcept
1601     {
1602         mImpl->setBeta(beta);
1603     }
1604
1609     float getAlpha() const noexcept
1610     {
1611         return mImpl->getAlpha();
1612     }
1613
1618     float getBeta() const noexcept
1619     {
1620         return mImpl->getBeta();
1621     }
1622
1623 protected:
1624     virtual ~IActivationLayer() noexcept = default;
1625     apiv::VActivationLayer* mImpl;
1626 };
1627
1633 enum class PoolingType : int32_t
1634 {
1635     kMAX = 0,           // Maximum over elements
1636     kAVERAGE = 1,     // Average over elements. If the tensor is padded, the count includes the
1637         padding
1637     kMAX_AVERAGE_BLEND = 2 // Blending between max and average pooling: (1-blendFactor)*maxPool +
1637         blendFactor*avgPool
1638 };
1639
1640 namespace impl
1641 {
1642     template <>
1643     struct EnumMaxImpl<PoolingType>
1644     {
1645         static constexpr int32_t kVALUE = 3;
1646     };
1647 } // namespace impl
1648
1665 class IPoolingLayer : public ILayer
1666 {
1667 public:
1675     void setPoolingType(PoolingType type) noexcept
1676     {
1677         mImpl->setPoolingType(type);

```

```
1678     }
1679
1685     PoolingType getPoolingType() const noexcept
1686     {
1687         return mImpl->getPoolingType();
1688     }
1689
1699     TRT_DEPRECATED void setWindowSize(DimsHW windowSize) noexcept
1700     {
1701         mImpl->setWindowSize(windowSize);
1702     }
1703
1711     TRT_DEPRECATED DimsHW getWindowSize() const noexcept
1712     {
1713         return mImpl->getWindowSize();
1714     }
1715
1727     TRT_DEPRECATED void setStride(DimsHW stride) noexcept
1728     {
1729         mImpl->setStride(stride);
1730     }
1731
1739     TRT_DEPRECATED DimsHW getStride() const noexcept
1740     {
1741         return mImpl->getStride();
1742     }
1743
1755     TRT_DEPRECATED void setPadding(DimsHW padding) noexcept
1756     {
1757         mImpl->setPadding(padding);
1758     }
1759
1769     TRT_DEPRECATED DimsHW getPadding() const noexcept
1770     {
1771         return mImpl->getPadding();
1772     }
1773
1784     void setBlendFactor(float blendFactor) noexcept
1785     {
1786         mImpl->setBlendFactor(blendFactor);
1787     }
1788
1797     float getBlendFactor() const noexcept
1798     {
1799         return mImpl->getBlendFactor();
1800     }
1801
1814     void setAverageCountExcludesPadding(bool exclusive) noexcept
1815     {
1816         mImpl->setAverageCountExcludesPadding(exclusive);
1817     }
1818
1825     bool getAverageCountExcludesPadding() const noexcept
1826     {
1827         return mImpl->getAverageCountExcludesPadding();
1828     }
1829
1843     void setPrePadding(Dims padding) noexcept
1844     {
1845         mImpl->setPrePadding(padding);
1846     }
1847
1853     Dims getPrePadding() const noexcept
1854     {
1855         return mImpl->getPrePadding();
1856     }
1857
1871     void setPostPadding(Dims padding) noexcept
1872     {
1873         mImpl->setPostPadding(padding);
1874     }
1875
1881     Dims getPostPadding() const noexcept
1882     {
1883         return mImpl->getPostPadding();
1884     }
1885
1894     void setPaddingMode(PaddingMode paddingMode) noexcept
1895     {
1896         mImpl->setPaddingMode(paddingMode);
1897     }
```



```

1898
1905     PaddingMode getPaddingMode() const noexcept
1906     {
1907         return mImpl->getPaddingMode();
1908     }
1909
1918     void setWindowSizeNd(Dims windowSize) noexcept
1919     {
1920         mImpl->setWindowSizeNd(windowSize);
1921     }
1922
1928     Dims getWindowSizeNd() const noexcept
1929     {
1930         return mImpl->getWindowSizeNd();
1931     }
1932
1943     void setStrideNd(Dims stride) noexcept
1944     {
1945         mImpl->setStrideNd(stride);
1946     }
1947
1953     Dims getStrideNd() const noexcept
1954     {
1955         return mImpl->getStrideNd();
1956     }
1957
1972     void setPaddingNd(Dims padding) noexcept
1973     {
1974         mImpl->setPaddingNd(padding);
1975     }
1976
1984     Dims getPaddingNd() const noexcept
1985     {
1986         return mImpl->getPaddingNd();
1987     }
1988
1989 protected:
1990     virtual ~IPoolingLayer() noexcept = default;
1991     apiv::VPoolingLayer* mImpl;
1992 };
1993
2003 class ILRNLayer : public ILayer
2004 {
2005 public:
2015     void setWindowSize(int32_t windowSize) noexcept
2016     {
2017         mImpl->setWindowSize(windowSize);
2018     }
2019
2025     int32_t getWindowSize() const noexcept
2026     {
2027         return mImpl->getWindowSize();
2028     }
2029
2036     void setAlpha(float alpha) noexcept
2037     {
2038         mImpl->setAlpha(alpha);
2039     }
2040
2046     float getAlpha() const noexcept
2047     {
2048         return mImpl->getAlpha();
2049     }
2050
2057     void setBeta(float beta) noexcept
2058     {
2059         mImpl->setBeta(beta);
2060     }
2061
2067     float getBeta() const noexcept
2068     {
2069         return mImpl->getBeta();
2070     }
2071
2078     void setK(float k) noexcept
2079     {
2080         mImpl->setK(k);
2081     }
2082
2088     float getK() const noexcept
2089     {

```

```

2090         return mImpl->getK();
2091     }
2092
2093 protected:
2094     virtual ~ILRNLayer() noexcept = default;
2095     apiv::VLRNLayer* mImpl;
2096 };
2097
2103 enum class ScaleMode : int32_t
2104 {
2105     kUNIFORM = 0,
2106     kCHANNEL = 1,
2107     kELEMENTWISE = 2
2108 };
2109
2115 template <>
2116 constexpr inline int32_t EnumMax<ScaleMode>() noexcept
2117 {
2118     return 3;
2119 }
2120
2147 class IScaleLayer : public ILayer
2148 {
2149 public:
2155     void setMode(ScaleMode mode) noexcept
2156     {
2157         mImpl->setMode(mode);
2158     }
2159
2165     ScaleMode getMode() const noexcept
2166     {
2167         return mImpl->getMode();
2168     }
2169
2175     void setShift(Weights shift) noexcept
2176     {
2177         mImpl->setShift(shift);
2178     }
2179
2185     Weights getShift() const noexcept
2186     {
2187         return mImpl->getShift();
2188     }
2189
2195     void setScale(Weights scale) noexcept
2196     {
2197         mImpl->setScale(scale);
2198     }
2199
2205     Weights getScale() const noexcept
2206     {
2207         return mImpl->getScale();
2208     }
2209
2215     void setPower(Weights power) noexcept
2216     {
2217         mImpl->setPower(power);
2218     }
2219
2225     Weights getPower() const noexcept
2226     {
2227         return mImpl->getPower();
2228     }
2229
2240     int32_t getChannelAxis() const noexcept
2241     {
2242         return mImpl->getChannelAxis();
2243     }
2244
2261     void setChannelAxis(int32_t channelAxis) noexcept
2262     {
2263         mImpl->setChannelAxis(channelAxis);
2264     }
2265
2266 protected:
2267     virtual ~IScaleLayer() noexcept = default;
2268     apiv::VScaleLayer* mImpl;
2269 };
2270
2292 class ISoftMaxLayer : public ILayer
2293 {

```

```

2294 public:
2325     void setAxes(uint32_t axes) noexcept
2326     {
2327         mImpl->setAxes(axes);
2328     }
2329
2335     uint32_t getAxes() const noexcept
2336     {
2337         return mImpl->getAxes();
2338     }
2339
2340 protected:
2341     virtual ~ISoftMaxLayer() noexcept = default;
2342     apiv::VSoftMaxLayer* mImpl;
2343 };
2344
2357 class IConcatenationLayer : public ILayer
2358 {
2359 public:
2370     void setAxis(int32_t axis) noexcept
2371     {
2372         mImpl->setAxis(axis);
2373     }
2374
2380     int32_t getAxis() const noexcept
2381     {
2382         return mImpl->getAxis();
2383     }
2384
2385 protected:
2386     virtual ~IConcatenationLayer() noexcept = default;
2387     apiv::VConcatenationLayer* mImpl;
2388 };
2389
2397 class IDeconvolutionLayer : public ILayer
2398 {
2399 public:
2411     TRT_DEPRECATED void setKernelSize(DimsHW kernelSize) noexcept
2412     {
2413         mImpl->setKernelSize(kernelSize);
2414     }
2415
2423     TRT_DEPRECATED DimsHW getKernelSize() const noexcept
2424     {
2425         return mImpl->getKernelSize();
2426     }
2427
2435     void setNbOutputMaps(int32_t nbOutputMaps) noexcept
2436     {
2437         mImpl->setNbOutputMaps(nbOutputMaps);
2438     }
2439
2445     int32_t getNbOutputMaps() const noexcept
2446     {
2447         return mImpl->getNbOutputMaps();
2448     }
2449
2461     TRT_DEPRECATED void setStride(DimsHW stride) noexcept
2462     {
2463         mImpl->setStride(stride);
2464     }
2465
2473     TRT_DEPRECATED DimsHW getStride() const noexcept
2474     {
2475         return mImpl->getStride();
2476     }
2477
2493     TRT_DEPRECATED void setPadding(DimsHW padding) noexcept
2494     {
2495         mImpl->setPadding(padding);
2496     }
2497
2507     TRT_DEPRECATED DimsHW getPadding() const noexcept
2508     {
2509         return mImpl->getPadding();
2510     }
2511
2527     void setNbGroups(int32_t nbGroups) noexcept
2528     {
2529         mImpl->setNbGroups(nbGroups);
2530     }

```

```
2531
2537 int32_t getNbGroups() const noexcept
2538 {
2539     return mImpl->getNbGroups();
2540 }
2541
2551 void setKernelWeights(Weights weights) noexcept
2552 {
2553     mImpl->setKernelWeights(weights);
2554 }
2555
2561 Weights getKernelWeights() const noexcept
2562 {
2563     return mImpl->getKernelWeights();
2564 }
2565
2576 void setBiasWeights(Weights weights) noexcept
2577 {
2578     mImpl->setBiasWeights(weights);
2579 }
2580
2586 Weights getBiasWeights() const noexcept
2587 {
2588     return mImpl->getBiasWeights();
2589 }
2590
2604 void setPrePadding(Dims padding) noexcept
2605 {
2606     mImpl->setPrePadding(padding);
2607 }
2608
2614 Dims getPrePadding() const noexcept
2615 {
2616     return mImpl->getPrePadding();
2617 }
2618
2632 void setPostPadding(Dims padding) noexcept
2633 {
2634     mImpl->setPostPadding(padding);
2635 }
2636
2642 Dims getPostPadding() const noexcept
2643 {
2644     return mImpl->getPostPadding();
2645 }
2646
2656 void setPaddingMode(PaddingMode paddingMode) noexcept
2657 {
2658     mImpl->setPaddingMode(paddingMode);
2659 }
2660
2668 PaddingMode getPaddingMode() const noexcept
2669 {
2670     return mImpl->getPaddingMode();
2671 }
2672
2683 void setKernelSizeNd(Dims kernelSize) noexcept
2684 {
2685     mImpl->setKernelSizeNd(kernelSize);
2686 }
2687
2693 Dims getKernelSizeNd() const noexcept
2694 {
2695     return mImpl->getKernelSizeNd();
2696 }
2697
2710 void setStrideNd(Dims stride) noexcept
2711 {
2712     mImpl->setStrideNd(stride);
2713 }
2714
2720 Dims getStrideNd() const noexcept
2721 {
2722     return mImpl->getStrideNd();
2723 }
2724
2738 void setPaddingNd(Dims padding) noexcept
2739 {
2740     mImpl->setPaddingNd(padding);
2741 }
2742
```

```

2750     Dims getPaddingNd() const noexcept
2751     {
2752         return mImpl->getPaddingNd();
2753     }
2754
2776     using ILayer::setInput;
2777
2784     void setDilationNd(Dims dilation) noexcept
2785     {
2786         mImpl->setDilationNd(dilation);
2787     }
2788
2794     Dims getDilationNd() const noexcept
2795     {
2796         return mImpl->getDilationNd();
2797     }
2798
2799 protected:
2800     virtual ~IDEconvolutionLayer() noexcept = default;
2801     apiv::VDeconvolutionLayer* mImpl;
2802 };
2803
2817 enum class ElementWiseOperation : int32_t
2818 {
2819     kSUM = 0,
2820     kPROD = 1,
2821     kMAX = 2,
2822     kMIN = 3,
2823     kSUB = 4,
2824     kDIV = 5,
2825     kPOW = 6,
2826     kFLOOR_DIV = 7,
2827     kAND = 8,
2828     kOR = 9,
2829     kXOR = 10,
2830     kEQUAL = 11,
2831     kGREATER = 12,
2832     kLESS = 13
2833 };
2834
2835 namespace impl
2836 {
2842     template <>
2843     struct EnumMaxImpl<ElementWiseOperation>
2844     {
2845         static constexpr int32_t kVALUE = 14;
2846     };
2847 } // namespace impl
2848
2868 class IElementWiseLayer : public ILayer
2869 {
2870 public:
2880     void setOperation(ElementWiseOperation op) noexcept
2881     {
2882         return mImpl->setOperation(op);
2883     }
2884
2892     ElementWiseOperation getOperation() const noexcept
2893     {
2894         return mImpl->getOperation();
2895     }
2896
2897 protected:
2898     apiv::VElementWiseLayer* mImpl;
2899     virtual ~IElementWiseLayer() noexcept = default;
2900 };
2901
2907 enum class GatherMode : int32_t
2908 {
2909     kDEFAULT = 0,
2910     kELEMENT = 1,
2911     kND = 2
2912 };
2913
2919 template <>
2920 constexpr inline int32_t EnumMax<GatherMode>() noexcept
2921 {
2922     return 3;
2923 }
2924
3003 class IGatherLayer : public ILayer

```

```

3004 {
3005 public:
3015     void setGatherAxis(int32_t axis) noexcept
3016     {
3017         mImpl->setGatherAxis(axis);
3018     }
3019
3026     int32_t getGatherAxis() const noexcept
3027     {
3028         return mImpl->getGatherAxis();
3029     }
3030
3047     void setNbElementWiseDims(int32_t elementWiseDims) noexcept
3048     {
3049         mImpl->setNbElementWiseDims(elementWiseDims);
3050     }
3051
3057     int32_t getNbElementWiseDims() const noexcept
3058     {
3059         return mImpl->getNbElementWiseDims();
3060     }
3061
3067     void setMode(GatherMode mode) noexcept
3068     {
3069         mImpl->setMode(mode);
3070     }
3071
3077     GatherMode getMode() const noexcept
3078     {
3079         return mImpl->getMode();
3080     }
3081
3082 protected:
3083     apiv::VGatherLayer* mImpl;
3084     virtual ~IGatherLayer() noexcept = default;
3085 };
3086
3166 enum class RNNOperation : int32_t
3167 {
3168     kRELU = 0,
3169     kTANH = 1,
3170     kLSTM = 2,
3171     kGRU = 3
3172 };
3173
3179 template <>
3180 constexpr inline int32_t EnumMax<RNNOperation>() noexcept
3181 {
3182     return 4;
3183 }
3184
3192 enum class RNNDirection : int32_t
3193 {
3194     kUNIDIRECTION = 0,
3195     kBIDIRECTION = 1
3196 };
3197
3203 template <>
3204 constexpr inline int32_t EnumMax<RNNDirection>() noexcept
3205 {
3206     return 2;
3207 }
3208
3224 enum class RNNInputMode : int32_t
3225 {
3226     kLINEAR = 0,
3227     kSKIP = 1
3228 };
3229
3235 template <>
3236 constexpr inline int32_t EnumMax<RNNInputMode>() noexcept
3237 {
3238     return 2;
3239 }
3240
3248 enum class RNNGateType : int32_t
3249 {
3250     kINPUT = 0,
3251     kOUTPUT = 1,
3252     kFORGET = 2,
3253     kUPDATE = 3,

```

```

3254     kRESET = 4,
3255     kCELL = 5,
3256     kHIDDEN = 6
3257 };
3258
3264 template <>
3265 constexpr inline int32_t EnumMax<RNNGateType>() noexcept
3266 {
3267     return 7;
3268 }
3269
3281 class TRT_DEPRECATED IRNNv2Layer : public ILayer
3282 {
3283 public:
3284     int32_t getLayerCount() const noexcept
3285     {
3286         return mImpl->getLayerCount();
3287     }
3288     int32_t getHiddenSize() const noexcept
3289     {
3290         return mImpl->getHiddenSize();
3291     }
3292     int32_t getMaxSeqLength() const noexcept
3293     {
3294         return mImpl->getMaxSeqLength();
3295     }
3296     int32_t getDataLength() const noexcept
3297     {
3298         return mImpl->getDataLength();
3299     }
3300
3315     void setSequenceLengths(ITensor& seqLengths) noexcept
3316     {
3317         return mImpl->setSequenceLengths(seqLengths);
3318     }
3319
3327     ITensor* getSequenceLengths() const noexcept
3328     {
3329         return mImpl->getSequenceLengths();
3330     }
3331
3337     void setOperation(RNNOperation op) noexcept
3338     {
3339         mImpl->setOperation(op);
3340     }
3341
3347     RNNOperation getOperation() const noexcept
3348     {
3349         return mImpl->getOperation();
3350     }
3351
3357     void setInputMode(RNNInputMode op) noexcept
3358     {
3359         mImpl->setInputMode(op);
3360     }
3361
3367     RNNInputMode getInputMode() const noexcept
3368     {
3369         return mImpl->getInputMode();
3370     }
3371
3382     void setDirection(RNNDirection op) noexcept
3383     {
3384         mImpl->setDirection(op);
3385     }
3386
3392     RNNDirection getDirection() const noexcept
3393     {
3394         return mImpl->getDirection();
3395     }
3396
3451     void setWeightsForGate(int32_t layerIndex, RNNGateType gate, bool isW, Weights weights) noexcept
3452     {
3453         mImpl->setWeightsForGate(layerIndex, gate, isW, weights);
3454     }
3455
3461     Weights getWeightsForGate(int32_t layerIndex, RNNGateType gate, bool isW) const noexcept
3462     {
3463         return mImpl->getWeightsForGate(layerIndex, gate, isW);
3464     }
3465

```

```

3486     void setBiasForGate(int32_t layerIndex, RNNGateType gate, bool isW, Weights bias) noexcept
3487     {
3488         mImpl->setBiasForGate(layerIndex, gate, isW, bias);
3489     }
3490
3496     Weights getBiasForGate(int32_t layerIndex, RNNGateType gate, bool isW) const noexcept
3497     {
3498         return mImpl->getBiasForGate(layerIndex, gate, isW);
3499     }
3500
3513     void setHiddenState(ITensor& hidden) noexcept
3514     {
3515         mImpl->setHiddenState(hidden);
3516     }
3517
3523     ITensor* getHiddenState() const noexcept
3524     {
3525         return mImpl->getHiddenState();
3526     }
3527
3542     void setCellState(ITensor& cell) noexcept
3543     {
3544         mImpl->setCellState(cell);
3545     }
3546
3552     ITensor* getCellState() const noexcept
3553     {
3554         return mImpl->getCellState();
3555     }
3556
3557 protected:
3558     apiv::VRNNv2Layer* mImpl;
3559     virtual ~IRNNv2Layer() noexcept = default;
3560 };
3561
3571 class IPluginV2Layer : public ILayer
3572 {
3573 public:
3579     IPluginV2& getPlugin() noexcept
3580     {
3581         return mImpl->getPlugin();
3582     }
3583
3584 protected:
3585     apiv::VPluginV2Layer* mImpl;
3586     virtual ~IPluginV2Layer() noexcept = default;
3587 };
3588
3602 enum class UnaryOperation : int32_t
3603 {
3604     kEXP = 0,
3605     kLOG = 1,
3606     kSQRT = 2,
3607     kRECIP = 3,
3608     kABS = 4,
3609     kNEG = 5,
3610     kSIN = 6,
3611     kCOS = 7,
3612     kTAN = 8,
3613     kSINH = 9,
3614     kCOSH = 10,
3615     kASIN = 11,
3616     kACOS = 12,
3617     kATAN = 13,
3618     kASINH = 14,
3619     kACOSH = 15,
3620     kATANH = 16,
3621     kCEIL = 17,
3622     kFLOOR = 18,
3623     kERF = 19,
3624     kNOT = 20,
3625     kSIGN = 21,
3626     kROUND = 22
3627 };
3628
3634 template <>
3635 constexpr inline int32_t EnumMax<UnaryOperation>() noexcept
3636 {
3637     return 23;
3638 }
3639

```



```

3647 class IUnaryLayer : public ILayer
3648 {
3649 public:
3650     void setOperation(UnaryOperation op) noexcept
3651     {
3652         mImpl->setOperation(op);
3653     }
3654     UnaryOperation getOperation() const noexcept
3655     {
3656         return mImpl->getOperation();
3657     }
3658 protected:
3659     apiv::VUnaryLayer* mImpl;
3660     virtual ~IUnaryLayer() noexcept = default;
3661 };
3662 enum class ReduceOperation : int32_t
3663 {
3664     kSUM = 0,
3665     kPROD = 1,
3666     kMAX = 2,
3667     kMIN = 3,
3668     kAVG = 4
3669 };
3670 template <>
3671 constexpr inline int32_t EnumMax<ReduceOperation>() noexcept
3672 {
3673     return 5;
3674 }
3675 class IReduceLayer : public ILayer
3676 {
3677 public:
3678     void setOperation(ReduceOperation op) noexcept
3679     {
3680         mImpl->setOperation(op);
3681     }
3682     ReduceOperation getOperation() const noexcept
3683     {
3684         return mImpl->getOperation();
3685     }
3686     void setReduceAxes(uint32_t reduceAxes) noexcept
3687     {
3688         mImpl->setReduceAxes(reduceAxes);
3689     }
3690     uint32_t getReduceAxes() const noexcept
3691     {
3692         return mImpl->getReduceAxes();
3693     }
3694     void setKeepDimensions(bool keepDimensions) noexcept
3695     {
3696         mImpl->setKeepDimensions(keepDimensions);
3697     }
3698     bool getKeepDimensions() const noexcept
3699     {
3700         return mImpl->getKeepDimensions();
3701     }
3702 protected:
3703     apiv::VReduceLayer* mImpl;
3704     virtual ~IReduceLayer() noexcept = default;
3705 };
3706 class IPaddingLayer : public ILayer
3707 {
3708 public:
3709     TRT_DEPRECATED void setPrePadding(DimsHW padding) noexcept
3710     {
3711         mImpl->setPrePadding(padding);
3712     }
3713     TRT_DEPRECATED DimsHW getPrePadding() const noexcept
3714     {

```

```

3826     return mImpl->getPrePadding();
3827 }
3828
3838 TRT_DEPRECATED void setPostPadding(DimsHW padding) noexcept
3839 {
3840     mImpl->setPostPadding(padding);
3841 }
3842
3850 TRT_DEPRECATED DimsHW getPostPadding() const noexcept
3851 {
3852     return mImpl->getPostPadding();
3853 }
3854
3864 void setPrePaddingNd(Dims padding) noexcept
3865 {
3866     mImpl->setPrePaddingNd(padding);
3867 }
3868
3876 Dims getPrePaddingNd() const noexcept
3877 {
3878     return mImpl->getPrePaddingNd();
3879 }
3880
3890 void setPostPaddingNd(Dims padding) noexcept
3891 {
3892     mImpl->setPostPaddingNd(padding);
3893 }
3894
3902 Dims getPostPaddingNd() const noexcept
3903 {
3904     return mImpl->getPostPaddingNd();
3905 }
3906
3907 protected:
3908     apiv::VPaddingLayer* mImpl;
3909     virtual ~IPaddingLayer() noexcept = default;
3910 };
3911
3912 struct Permutation
3913 {
3920     int32_t order[Dims::MAX_DIMS];
3921 };
3922
3935 class IShuffleLayer : public ILayer
3936 {
3937 public:
3947     void setFirstTranspose(Permutation permutation) noexcept
3948     {
3949         mImpl->setFirstTranspose(permutation);
3950     }
3951
3959     Permutation getFirstTranspose() const noexcept
3960     {
3961         return mImpl->getFirstTranspose();
3962     }
3963
3984     void setReshapeDimensions(Dims dimensions) noexcept
3985     {
3986         mImpl->setReshapeDimensions(dimensions);
3987     }
3988
3997     Dims getReshapeDimensions() const noexcept
3998     {
3999         return mImpl->getReshapeDimensions();
4000     }
4001
4007     //
4030     using ILayer::setInput;
4031
4044     void setSecondTranspose(Permutation permutation) noexcept
4045     {
4046         mImpl->setSecondTranspose(permutation);
4047     }
4048
4056     Permutation getSecondTranspose() const noexcept
4057     {
4058         return mImpl->getSecondTranspose();
4059     }
4060
4072     void setZeroIsPlaceholder(bool zeroIsPlaceholder) noexcept
4073     {

```

```

4074         return mImpl->setZeroIsPlaceholder(zeroIsPlaceholder);
4075     }
4076
4085     bool getZeroIsPlaceholder() const noexcept
4086     {
4087         return mImpl->getZeroIsPlaceholder();
4088     }
4089
4090 protected:
4091     apiv::VShuffleLayer* mImpl;
4092     virtual ~IShuffleLayer() noexcept = default;
4093 };
4094
4100 enum class SliceMode : int32_t
4101 {
4102     kDEFAULT = 0,
4103     kWRAP = 1,
4104     kCLAMP = 2,
4105     kFILL = 3,
4106     kREFLECT = 4,
4109 };
4110
4116 template <>
4117 constexpr inline int32_t EnumMax<SliceMode>() noexcept
4118 {
4119     return 5;
4120 }
4121
4164 class ISliceLayer : public ILayer
4165 {
4166 public:
4176     void setStart(Dims start) noexcept
4177     {
4178         mImpl->setStart(start);
4179     }
4180
4191     Dims getStart() const noexcept
4192     {
4193         return mImpl->getStart();
4194     }
4195
4205     void setSize(Dims size) noexcept
4206     {
4207         return mImpl->setSize(size);
4208     }
4209
4220     Dims getSize() const noexcept
4221     {
4222         return mImpl->getSize();
4223     }
4224
4234     void setStride(Dims stride) noexcept
4235     {
4236         mImpl->setStride(stride);
4237     }
4238
4249     Dims getStride() const noexcept
4250     {
4251         return mImpl->getStride();
4252     }
4253
4259     void setMode(SliceMode mode) noexcept
4260     {
4261         mImpl->setMode(mode);
4262     }
4263
4269     SliceMode getMode() const noexcept
4270     {
4271         return mImpl->getMode();
4272     }
4273
4295     using ILayer::setInput;
4296
4297 protected:
4298     apiv::VSliceLayer* mImpl;
4299     virtual ~ISliceLayer() noexcept = default;
4300 };
4301
4314 class IShapeLayer : public ILayer
4315 {
4316 protected:

```

```

4317     apiv::VShapeLayer* mImpl;
4318     virtual ~IShapeLayer() noexcept = default;
4319 };
4320
4321 enum class TopKOperation : int32_t
4322 {
4323     kMAX = 0,
4324     kMIN = 1,
4325 };
4326
4327 template <>
4328 constexpr inline int32_t EnumMax<TopKOperation>() noexcept
4329 {
4330     return 2;
4331 }
4332
4333 class ITopKLayer : public ILayer
4334 {
4335 public:
4336     void setOperation(TopKOperation op) noexcept
4337     {
4338         mImpl->setOperation(op);
4339     }
4340
4341     TopKOperation getOperation() const noexcept
4342     {
4343         return mImpl->getOperation();
4344     }
4345
4346     void setK(int32_t k) noexcept
4347     {
4348         mImpl->setK(k);
4349     }
4350
4351     int32_t getK() const noexcept
4352     {
4353         return mImpl->getK();
4354     }
4355
4356     void setReduceAxes(uint32_t reduceAxes) noexcept
4357     {
4358         mImpl->setReduceAxes(reduceAxes);
4359     }
4360
4361     uint32_t getReduceAxes() const noexcept
4362     {
4363         return mImpl->getReduceAxes();
4364     }
4365
4366 protected:
4367     apiv::VTopKLayer* mImpl;
4368     virtual ~ITopKLayer() noexcept = default;
4369 };
4370
4371 enum class MatrixOperation : int32_t
4372 {
4373     kNONE,
4374     kTRANSPOSE,
4375     kVECTOR
4376 };
4377
4378 template <>
4379 constexpr inline int32_t EnumMax<MatrixOperation>() noexcept
4380 {
4381     return 3;
4382 }
4383
4384 class IMatrixMultiplyLayer : public ILayer
4385 {
4386 public:
4387     void setOperation(int32_t index, MatrixOperation op) noexcept
4388     {
4389         mImpl->setOperation(index, op);
4390     }
4391
4392     MatrixOperation getOperation(int32_t index) const noexcept
4393     {
4394         return mImpl->getOperation(index);
4395     }
4396 }

```

```

4510
4511 protected:
4512     apiv::VMatrixMultiplyLayer* mImpl;
4513     virtual ~IMatrixMultiplyLayer() noexcept = default;
4514 };
4515
4530 class IRaggedSoftMaxLayer : public ILayer
4531 {
4532 protected:
4533     apiv::VRaggedSoftMaxLayer* mImpl;
4534     virtual ~IRaggedSoftMaxLayer() noexcept = default;
4535 };
4536
4560 class IIdentityLayer : public ILayer
4561 {
4562 protected:
4563     apiv::VIdentityLayer* mImpl;
4564     virtual ~IIdentityLayer() noexcept = default;
4565 };
4566
4575 class IConstantLayer : public ILayer
4576 {
4577 public:
4578     void setWeights(Weights weights) noexcept
4579     {
4580         mImpl->setWeights(weights);
4581     }
4582
4597     Weights getWeights() const noexcept
4598     {
4599         return mImpl->getWeights();
4600     }
4601
4609     void setDimensions(Dims dimensions) noexcept
4610     {
4611         mImpl->setDimensions(dimensions);
4612     }
4613
4621     Dims getDimensions() const noexcept
4622     {
4623         return mImpl->getDimensions();
4624     }
4625
4626 protected:
4627     apiv::VConstantLayer* mImpl;
4628     virtual ~IConstantLayer() noexcept = default;
4629 };
4630
4640 class IParametricReLULayer : public ILayer
4641 {
4642 protected:
4643     apiv::VParametricReLULayer* mImpl;
4644     virtual ~IParametricReLULayer() noexcept = default;
4645 };
4646
4652 enum class ResizeMode : int32_t
4653 {
4654     kNEAREST = 0,
4655     kLINEAR = 1
4656 };
4657
4658 namespace impl
4659 {
4665 template <>
4666 struct EnumMaxImpl<ResizeMode>
4667 {
4668     static constexpr int32_t kVALUE = 2;
4669 };
4670 } // namespace impl
4671
4679 enum class ResizeCoordinateTransformation : int32_t
4680 {
4693     kALIGN_CORNERS = 0,
4694
4701     kASYMMETRIC = 1,
4702
4709     kHALF_PIXEL = 2,
4710 };
4711
4712 namespace impl
4713 {

```

```

4719 template <>
4720 struct EnumMaxImpl<ResizeCoordinateTransformation>
4721 {
4722     static constexpr int32_t kVALUE = 3;
4723 };
4724 } // namespace impl
4725
4733 enum class ResizeSelector : int32_t
4734 {
4735     kFORMULA = 0,
4736     kUPPER = 1,
4737 };
4742 namespace impl
4743 {
4744     template <>
4745     struct EnumMaxImpl<ResizeSelector>
4746     {
4747         static constexpr int32_t kVALUE = 2;
4748     };
4749 } // namespace impl
4750
4763 enum class ResizeRoundMode : int32_t
4764 {
4765     kHALF_UP = 0,
4766     kHALF_DOWN = 1,
4767     kFLOOR = 2,
4768     kCEIL = 3,
4769 };
4778 namespace impl
4779 {
4780     template <>
4781     struct EnumMaxImpl<ResizeRoundMode>
4782     {
4783         static constexpr int32_t kVALUE = 4;
4784     };
4785 } // namespace impl
4791
4828 class IResizeLayer : public ILayer
4829 {
4830 public:
4831     void setOutputDimensions(Dims dimensions) noexcept
4832     {
4833         return mImpl->setOutputDimensions(dimensions);
4834     }
4835     Dims getOutputDimensions() const noexcept
4836     {
4837         return mImpl->getOutputDimensions();
4838     }
4839     void setScales(float const* scales, int32_t nbScales) noexcept
4840     {
4841         mImpl->setScales(scales, nbScales);
4842     }
4843     int32_t getScales(int32_t size, float* scales) const noexcept
4844     {
4845         return mImpl->getScales(size, scales);
4846     }
4847     void setResizeMode(ResizeMode resizeMode) noexcept
4848     {
4849         mImpl->setResizeMode(resizeMode);
4850     }
4851     ResizeMode getResizeMode() const noexcept
4852     {
4853         return mImpl->getResizeMode();
4854     }
4855     TRT_DEPRECATED void setAlignCorners(bool alignCorners) noexcept
4856     {
4857         mImpl->setAlignCorners(alignCorners);
4858     }
4859 }

```

```

4958     TRT_DEPRECATED bool getAlignCorners() const noexcept
4959     {
4960         return mImpl->getAlignCorners();
4961     }
4962
4982     using ILayer::setInput;
4983
4993     void setCoordinateTransformation(ResizeCoordinateTransformation coordTransform) noexcept
4994     {
4995         mImpl->setCoordinateTransformation(coordTransform);
4996     }
4997
5003     ResizeCoordinateTransformation getCoordinateTransformation() const noexcept
5004     {
5005         return mImpl->getCoordinateTransformation();
5006     }
5007
5018     void setSelectorForSinglePixel(ResizeSelector selector) noexcept
5019     {
5020         mImpl->setSelectorForSinglePixel(selector);
5021     }
5022
5028     ResizeSelector getSelectorForSinglePixel() const noexcept
5029     {
5030         return mImpl->getSelectorForSinglePixel();
5031     }
5032
5042     void setNearestRounding(ResizeRoundMode value) noexcept
5043     {
5044         mImpl->setNearestRounding(value);
5045     }
5046
5052     ResizeRoundMode getNearestRounding() const noexcept
5053     {
5054         return mImpl->getNearestRounding();
5055     }
5056
5057 protected:
5058     virtual ~IResizeLayer() noexcept = default;
5059     apiv::VResizeLayer* mImpl;
5060 };
5061
5063 enum class LoopOutput : int32_t
5064 {
5066     kLAST_VALUE = 0,
5067
5069     kCONCATENATE = 1,
5070
5072     kREVERSE = 2
5073 };
5074
5080 template <>
5081 constexpr inline int32_t EnumMax<LoopOutput>() noexcept
5082 {
5083     return 3;
5084 }
5085
5087 enum class TripLimit : int32_t
5088 {
5089
5090     kCOUNT = 0,
5091     kWHILE = 1
5092 };
5093
5099 template <>
5100 constexpr inline int32_t EnumMax<TripLimit>() noexcept
5101 {
5102     return 2;
5103 }
5104
5105 class ILoop;
5106
5107 class ILoopBoundaryLayer : public ILayer
5108 {
5109 public:
5111     ILoop* getLoop() const noexcept
5112     {
5113         return mBoundary->getLoop();
5114     }
5115
5116 protected:

```

```

5117     virtual ~ILoopBoundaryLayer() noexcept = default;
5118     apiv::VLoopBoundaryLayer* mBoundary;
5119 };
5120
5121 class IIfConditionalBoundaryLayer : public ILayer
5122 {
5123 public:
5124     IIfConditional* getConditional() const noexcept
5125     {
5126         return mBoundary->getConditional();
5127     }
5128
5129 protected:
5130     virtual ~IIfConditionalBoundaryLayer() noexcept = default;
5131     apiv::VConditionalBoundaryLayer* mBoundary;
5132 };
5133
5134 class IConditionLayer : public IIfConditionalBoundaryLayer
5135 {
5136 public:
5137     virtual ~IConditionLayer() noexcept = default;
5138     apiv::VConditionLayer* mImpl;
5139 };
5140
5141 class IIfConditionalOutputLayer : public IIfConditionalBoundaryLayer
5142 {
5143 public:
5144     virtual ~IIfConditionalOutputLayer() noexcept = default;
5145     apiv::VConditionalOutputLayer* mImpl;
5146 };
5147
5148 class IIfConditionalInputLayer : public IIfConditionalBoundaryLayer
5149 {
5150 public:
5151     virtual ~IIfConditionalInputLayer() noexcept = default;
5152     apiv::VConditionalInputLayer* mImpl;
5153 };
5154
5155 class IIfConditional : public INoCopy
5156 {
5157 public:
5158     IConditionLayer* setCondition(ITensor& condition) noexcept
5159     {
5160         return mImpl->setCondition(condition);
5161     }
5162
5163     IIfConditionalOutputLayer* addOutput(ITensor& trueSubgraphOutput, ITensor& falseSubgraphOutput)
5164     noexcept
5165     {
5166         return mImpl->addOutput(trueSubgraphOutput, falseSubgraphOutput);
5167     }
5168
5169     IIfConditionalInputLayer* addInput(ITensor& input) noexcept
5170     {
5171         return mImpl->addInput(input);
5172     }
5173
5174     void setName(char const* name) noexcept
5175     {
5176         mImpl->setName(name);
5177     }
5178
5179     char const* getName() const noexcept
5180     {
5181         return mImpl->getName();
5182     }
5183
5184 protected:
5185     virtual ~IIfConditional() noexcept = default;
5186     apiv::VIfConditional* mImpl;
5187 };
5188
5189 class IRecurrenceLayer : public ILoopBoundaryLayer
5190 {
5191 public:
5192     //
5193     using ILayer::setInput;

```



```

5292
5293 protected:
5294     virtual ~IRecurrenceLayer() noexcept = default;
5295     apiv::VRecurrenceLayer* mImpl;
5296 };
5297
5315 class ILoopOutputLayer : public ILoopBoundaryLayer
5316 {
5317 public:
5318     LoopOutput getLoopOutput() const noexcept
5319     {
5320         return mImpl->getLoopOutput();
5321     }
5322
5335     void setAxis(int32_t axis) noexcept
5336     {
5337         mImpl->setAxis(axis);
5338     }
5339
5341     int32_t getAxis() const noexcept
5342     {
5343         return mImpl->getAxis();
5344     }
5345
5351     //
5366     using ILayer::setInput;
5367
5368 protected:
5369     virtual ~ILoopOutputLayer() noexcept = default;
5370     apiv::VLoopOutputLayer* mImpl;
5371 };
5372
5373 class ITripLimitLayer : public ILoopBoundaryLayer
5374 {
5375 public:
5376     TripLimit getTripLimit() const noexcept
5377     {
5378         return mImpl->getTripLimit();
5379     }
5380
5381 protected:
5382     virtual ~ITripLimitLayer() noexcept = default;
5383     apiv::VTripLimitLayer* mImpl;
5384 };
5385
5386 class IIteratorLayer : public ILoopBoundaryLayer
5387 {
5388 public:
5390     void setAxis(int32_t axis) noexcept
5391     {
5392         mImpl->setAxis(axis);
5393     }
5394
5396     int32_t getAxis() const noexcept
5397     {
5398         return mImpl->getAxis();
5399     }
5400
5406     void setReverse(bool reverse) noexcept
5407     {
5408         mImpl->setReverse(reverse);
5409     }
5410
5412     bool getReverse() const noexcept
5413     {
5414         return mImpl->getReverse();
5415     }
5416
5417 protected:
5418     virtual ~IIteratorLayer() noexcept = default;
5419     apiv::VIteratorLayer* mImpl;
5420 };
5421
5427 class ILoop : public INoCopy
5428 {
5429 public:
5436     IRecurrenceLayer* addRecurrence(ITensor& initialValue) noexcept
5437     {
5438         return mImpl->addRecurrence(initialValue);
5439     }
5440

```

```

5457     ITripLimitLayer* addTripLimit(ITensor& tensor, TripLimit limit) noexcept
5458     {
5459         return mImpl->addTripLimit(tensor, limit);
5460     }
5461
5470     IIteratorLayer* addIterator(ITensor& tensor, int32_t axis = 0, bool reverse = false) noexcept
5471     {
5472         return mImpl->addIterator(tensor, axis, reverse);
5473     }
5474
5482     ILoopOutputLayer* addLoopOutput(ITensor& tensor, LoopOutput outputKind, int32_t axis = 0) noexcept
5483     {
5484         return mImpl->addLoopOutput(tensor, outputKind, axis);
5485     }
5486
5495     void setName(char const* name) noexcept
5496     {
5497         mImpl->setName(name);
5498     }
5499
5505     char const* getName() const noexcept
5506     {
5507         return mImpl->getName();
5508     }
5509
5510 protected:
5511     virtual ~ILoop() noexcept = default;
5512     apiv::VLoop* mImpl;
5513 };
5514
5518 class ISelectLayer : public ILayer
5519 {
5520 protected:
5521     virtual ~ISelectLayer() noexcept = default;
5522     apiv::VSelectLayer* mImpl;
5523 };
5524
5539 class IAssertionLayer : public ILayer
5540 {
5541 public:
5550     void setMessage(char const* message) noexcept
5551     {
5552         mImpl->setMessage(message);
5553     }
5554
5560     char const* getMessage() const noexcept
5561     {
5562         return mImpl->getMessage();
5563     }
5564
5565 protected:
5566     virtual ~IAssertionLayer() noexcept = default;
5567
5568     apiv::VAssertionLayer* mImpl;
5569 };
5570
5578 enum class FillOperation : int32_t
5579 {
5580     kLINSPLACE = 0,
5581     kRANDOM.UNIFORM = 1
5582 };
5583
5589 template <>
5590 constexpr inline int32_t EnumMax<FillOperation>() noexcept
5591 {
5592     return 2;
5593 }
5594
5620 class IFillLayer : public ILayer
5621 {
5622 public:
5631     //
5632     void setDimensions(Dims dimensions) noexcept
5633     {
5634         mImpl->setDimensions(dimensions);
5635     }
5636
5647     Dims getDimensions() const noexcept
5648     {
5649         return mImpl->getDimensions();
5650     }

```

```

5651
5652 void setOperation(FillOperation op) noexcept
5653 {
5654     mImpl->setOperation(op);
5655 }
5656
5657 FillOperation getOperation() const noexcept
5658 {
5659     return mImpl->getOperation();
5660 }
5661
5662 //
5663 void setAlpha(double alpha) noexcept
5664 {
5665     mImpl->setAlpha(alpha);
5666 }
5667
5668 double getAlpha() const noexcept
5669 {
5670     return mImpl->getAlpha();
5671 }
5672
5673 void setBeta(double beta) noexcept
5674 {
5675     mImpl->setBeta(beta);
5676 }
5677
5678 double getBeta() const noexcept
5679 {
5680     return mImpl->getBeta();
5681 }
5682
5683 using ILayer::setInput;
5684
5685 protected:
5686     virtual ~IFillLayer() noexcept = default;
5687     apiv::VFillLayer* mImpl;
5688 };
5689
5690 class IQuantizeLayer : public ILayer
5691 {
5692 public:
5693     int32_t getAxis() const noexcept
5694     {
5695         return mImpl->getAxis();
5696     }
5697     void setAxis(int32_t axis) noexcept
5698     {
5699         mImpl->setAxis(axis);
5700     }
5701
5702 protected:
5703     virtual ~IQuantizeLayer() noexcept = default;
5704     apiv::VQuantizeLayer* mImpl;
5705 };
5706
5707 class IDequantizeLayer : public ILayer
5708 {
5709 public:
5710     int32_t getAxis() const noexcept
5711     {
5712         return mImpl->getAxis();
5713     }
5714     void setAxis(int32_t axis) noexcept
5715     {
5716         mImpl->setAxis(axis);
5717     }
5718
5719 protected:
5720     virtual ~IDequantizeLayer() noexcept = default;
5721     apiv::VDequantizeLayer* mImpl;
5722 };
5723
5724 class IEinsumLayer : public ILayer
5725 {
5726 public:
5727     bool setEquation(char const* equation) noexcept
5728     {
5729         return mImpl->setEquation(equation);
5730     }
5731 }
5732
5733

```

```

6004     char const* getEquation() const noexcept
6005     {
6006         return mImpl->getEquation();
6007     }
6008
6009 protected:
6010     virtual ~IEinsumLayer() noexcept = default;
6011     apiv::VEinsumLayer* mImpl;
6012 };
6013
6014 enum class ScatterMode : int32_t
6015 {
6016     kELEMENT = 0,
6017     kND = 1,
6018 };
6019
6020 template <>
6021 constexpr inline int32_t EnumMax<ScatterMode>() noexcept
6022 {
6023     return 2;
6024 }
6025
6026 class IScatterLayer : public ILayer
6027 {
6028 public:
6029     void setMode(ScatterMode mode) noexcept
6030     {
6031         mImpl->setMode(mode);
6032     }
6033
6034     ScatterMode getMode() const noexcept
6035     {
6036         return mImpl->getMode();
6037     }
6038
6039     void setAxis(int32_t axis) noexcept
6040     {
6041         mImpl->setAxis(axis);
6042     }
6043
6044     int32_t getAxis() const noexcept
6045     {
6046         return mImpl->getAxis();
6047     }
6048
6049 protected:
6050     apiv::VScatterLayer* mImpl;
6051     virtual ~IScatterLayer() noexcept = default;
6052 }; // class IScatterLayer
6053
6054 class INetworkDefinition : public INoCopy
6055 {
6056 public:
6057     virtual ~INetworkDefinition() noexcept = default;
6058
6059     ITensor* addInput(char const* name, DataType type, Dims dimensions) noexcept
6060     {
6061         return mImpl->addInput(name, type, dimensions);
6062     }
6063
6064     void markOutput(ITensor& tensor) noexcept
6065     {
6066         mImpl->markOutput(tensor);
6067     }
6068
6069     TRT_DEPRECATED IConvolutionLayer* addConvolution(
6070         ITensor& input, int32_t nbOutputMaps, DimsHW kernelSize, Weights kernelWeights, Weights
6071         biasWeights) noexcept
6072     {
6073         return mImpl->addConvolution(input, nbOutputMaps, kernelSize, kernelWeights, biasWeights);
6074     }
6075
6076     TRT_DEPRECATED IFullyConnectedLayer* addFullyConnected(
6077         ITensor& input, int32_t nbOutputs, Weights kernelWeights, Weights biasWeights) noexcept
6078     {
6079         return mImpl->addFullyConnected(input, nbOutputs, kernelWeights, biasWeights);
6080     }
6081
6082     IActivationLayer* addActivation(ITensor& input, ActivationType type) noexcept
6083     {
6084         return mImpl->addActivation(input, type);
6085     }

```

```

6280     }
6281
6296     TRT_DEPRECATED IPoolingLayer* addPooling(ITensor& input, PoolingType type, DimsHW windowSize) noexcept
6297     {
6298         return mImpl->addPooling(input, type, windowSize);
6299     }
6300
6315     ILRNLayer* addLRN(ITensor& input, int32_t window, float alpha, float beta, float k) noexcept
6316     {
6317         return mImpl->addLRN(input, window, alpha, beta, k);
6318     }
6319
6342     IScaleLayer* addScale(ITensor& input, ScaleMode mode, Weights shift, Weights scale, Weights power)
noexcept
6343     {
6344         return mImpl->addScale(input, mode, shift, scale, power);
6345     }
6346
6355     ISoftMaxLayer* addSoftMax(ITensor& input) noexcept
6356     {
6357         return mImpl->addSoftMax(input);
6358     }
6359
6372     IConcatenationLayer* addConcatenation(ITensor* const* inputs, int32_t nbInputs) noexcept
6373     {
6374         return mImpl->addConcatenation(inputs, nbInputs);
6375     }
6376
6395     TRT_DEPRECATED IDeconvolutionLayer* addDeconvolution(
6396         ITensor& input, int32_t nbOutputMaps, DimsHW kernelSize, Weights kernelWeights, Weights
biasWeights) noexcept
6397     {
6398         return mImpl->addDeconvolution(input, nbOutputMaps, kernelSize, kernelWeights, biasWeights);
6399     }
6400
6423     IElementWiseLayer* addElementWise(ITensor& input1, ITensor& input2, ElementWiseOperation op) noexcept
6424     {
6425         return mImpl->addElementWise(input1, input2, op);
6426     }
6427
6445     IUnaryLayer* addUnary(ITensor& input, UnaryOperation operation) noexcept
6446     {
6447         return mImpl->addUnary(input, operation);
6448     }
6449
6462     TRT_DEPRECATED IPaddingLayer* addPadding(ITensor& input, DimsHW prePadding, DimsHW postPadding)
noexcept
6463     {
6464         return mImpl->addPadding(input, prePadding, postPadding);
6465     }
6466
6476     IShuffleLayer* addShuffle(ITensor& input) noexcept
6477     {
6478         return mImpl->addShuffle(input);
6479     }
6480
6488     int32_t getNbLayers() const noexcept
6489     {
6490         return mImpl->getNbLayers();
6491     }
6492
6502     ILayer* getLayer(int32_t index) const noexcept
6503     {
6504         return mImpl->getLayer(index);
6505     }
6506
6514     int32_t getNbInputs() const noexcept
6515     {
6516         return mImpl->getNbInputs();
6517     }
6518
6530     ITensor* getInput(int32_t index) const noexcept
6531     {
6532         return mImpl->getInput(index);
6533     }
6534
6544     int32_t getNbOutputs() const noexcept
6545     {
6546         return mImpl->getNbOutputs();
6547     }
6548

```

```

6560     ITensor* getOutput(int32_t index) const noexcept
6561     {
6562         return mImpl->getOutput(index);
6563     }
6564
6572     TRT_DEPRECATED void destroy() noexcept
6573     {
6574         delete this;
6575     }
6576
6599     IReduceLayer* addReduce(
6600         ITensor& input, ReduceOperation operation, uint32_t reduceAxes, bool keepDimensions) noexcept
6601     {
6602         return mImpl->addReduce(input, operation, reduceAxes, keepDimensions);
6603     }
6604
6633     ITopKLayer* addTopK(ITensor& input, TopKOperation op, int32_t k, uint32_t reduceAxes) noexcept
6634     {
6635         return mImpl->addTopK(input, op, k, reduceAxes);
6636     }
6637
6649     IGatherLayer* addGather(ITensor& data, ITensor& indices, int32_t axis) noexcept
6650     {
6651         return mImpl->addGather(data, indices, axis);
6652     }
6653
6665     IGatherLayer* addGatherV2(ITensor& data, ITensor& indices, GatherMode mode)
6666     {
6667         return mImpl->addGatherV2(data, indices, mode);
6668     }
6669
6683     IRaggedSoftMaxLayer* addRaggedSoftMax(ITensor& input, ITensor& bounds) noexcept
6684     {
6685         return mImpl->addRaggedSoftMax(input, bounds);
6686     }
6687
6704     IMatrixMultiplyLayer* addMatrixMultiply(
6705         ITensor& input0, MatrixOperation op0, ITensor& input1, MatrixOperation op1) noexcept
6706     {
6707         return mImpl->addMatrixMultiply(input0, op0, input1, op1);
6708     }
6709
6730     IConstantLayer* addConstant(Dims dimensions, Weights weights) noexcept
6731     {
6732         return mImpl->addConstant(dimensions, weights);
6733     }
6734
6799     TRT_DEPRECATED IRNNv2Layer* addRNNv2(
6800         ITensor& input, int32_t layerCount, int32_t hiddenSize, int32_t maxSeqLen, RNNOperation op) noexcept
6801     {
6802         return mImpl->addRNNv2(input, layerCount, hiddenSize, maxSeqLen, op);
6803     }
6804
6814     IIdentityLayer* addIdentity(ITensor& input) noexcept
6815     {
6816         return mImpl->addIdentity(input);
6817     }
6818
6829     void removeTensor(ITensor& tensor) noexcept
6830     {
6831         mImpl->removeTensor(tensor);
6832     }
6833
6841     void unmarkOutput(ITensor& tensor) noexcept
6842     {
6843         mImpl->unmarkOutput(tensor);
6844     }
6845
6860     IPluginV2Layer* addPluginV2(ITensor* const* inputs, int32_t nbInputs, IPluginV2& plugin) noexcept
6861     {
6862         return mImpl->addPluginV2(inputs, nbInputs, plugin);
6863     }
6864
6879     ISliceLayer* addSlice(ITensor& input, Dims start, Dims size, Dims stride) noexcept
6880     {
6881         return mImpl->addSlice(input, start, size, stride);
6882     }
6883
6901     void setName(char const* name) noexcept
6902     {
6903         mImpl->setName(name);

```

```

6904     }
6905
6915     char const* getName() const noexcept
6916     {
6917         return mImpl->getName();
6918     }
6919
6933     IShapeLayer* addShape(ITensor& input) noexcept
6934     {
6935         return mImpl->addShape(input);
6936     }
6937
6951     bool hasImplicitBatchDimension() const noexcept
6952     {
6953         return mImpl->hasImplicitBatchDimension();
6954     }
6955
6969     bool markOutputForShapes(ITensor& tensor) noexcept
6970     {
6971         return mImpl->markOutputForShapes(tensor);
6972     }
6973
6981     bool unmarkOutputForShapes(ITensor& tensor) noexcept
6982     {
6983         return mImpl->unmarkOutputForShapes(tensor);
6984     }
6985
6999     IParametricReLULayer* addParametricReLU(ITensor& input, ITensor& slope) noexcept
7000     {
7001         return mImpl->addParametricReLU(input, slope);
7002     }
7003
7021     IConvolutionLayer* addConvolutionNd(
7022         ITensor& input, int32_t nbOutputMaps, Dims kernelSize, Weights kernelWeights, Weights biasWeights)
7023     noexcept
7024     {
7025         return mImpl->addConvolutionNd(input, nbOutputMaps, kernelSize, kernelWeights, biasWeights);
7026     }
7041     IPoolingLayer* addPoolingNd(ITensor& input, PoolingType type, Dims windowSize) noexcept
7042     {
7043         return mImpl->addPoolingNd(input, type, windowSize);
7044     }
7045
7060     //
7063     IDeconvolutionLayer* addDeconvolutionNd(
7064         ITensor& input, int32_t nbOutputMaps, Dims kernelSize, Weights kernelWeights, Weights biasWeights)
7065     noexcept
7066     {
7067         return mImpl->addDeconvolutionNd(input, nbOutputMaps, kernelSize, kernelWeights, biasWeights);
7068     }
7099     IScaleLayer* addScaleNd(
7100         ITensor& input, ScaleMode mode, Weights shift, Weights scale, Weights power, int32_t channelAxis)
7101     noexcept
7102     {
7103         return mImpl->addScaleNd(input, mode, shift, scale, power, channelAxis);
7104     }
7115     IResizeLayer* addResize(ITensor& input) noexcept
7116     {
7117         return mImpl->addResize(input);
7118     }
7119
7132     TRT_DEPRECATED bool hasExplicitPrecision() const noexcept
7133     {
7134         return mImpl->hasExplicitPrecision();
7135     }
7136
7148     ILoop* addLoop() noexcept
7149     {
7150         return mImpl->addLoop();
7151     }
7152
7188     ISelectLayer* addSelect(ITensor& condition, ITensor& thenInput, ITensor& elseInput) noexcept
7189     {
7190         return mImpl->addSelect(condition, thenInput, elseInput);
7191     }
7192
7205     IAssertionLayer* addAssertion(ITensor& condition, char const* message) noexcept
7206     {

```

```

7207     return mImpl->addAssertion(condition, message);
7208 }
7209
7223 IFillLayer* addFill(Dims dimensions, FillOperation op) noexcept
7224 {
7225     return mImpl->addFill(dimensions, op);
7226 }
7227
7240 TRT_DEPRECATED IPaddingLayer* addPaddingNd(ITensor& input, Dims prePadding, Dims postPadding) noexcept
7241 {
7242     return mImpl->addPaddingNd(input, prePadding, postPadding);
7243 }
7244
7260 bool setWeightsName(Weights weights, char const* name) noexcept
7261 {
7262     return mImpl->setWeightsName(weights, name);
7263 }
7264
7276 //
7279 void setErrorRecorder(IErrorRecorder* recorder) noexcept
7280 {
7281     mImpl->setErrorRecorder(recorder);
7282 }
7283
7294 IErrorRecorder* getErrorRecorder() const noexcept
7295 {
7296     return mImpl->getErrorRecorder();
7297 }
7298
7313 IDequantizeLayer* addDequantize(ITensor& input, ITensor& scale) noexcept
7314 {
7315     return mImpl->addDequantize(input, scale);
7316 }
7317
7333 IScatterLayer* addScatter(ITensor& data, ITensor& indices, ITensor& updates, ScatterMode mode) noexcept
7334 {
7335     return mImpl->addScatter(data, indices, updates, mode);
7336 }
7337
7352 IQuantizeLayer* addQuantize(ITensor& input, ITensor& scale) noexcept
7353 {
7354     return mImpl->addQuantize(input, scale);
7355 }
7356
7367 IIfConditional* addIfConditional() noexcept
7368 {
7369     return mImpl->addIfConditional();
7370 }
7371
7381 IEinsumLayer* addEinsum(ITensor* const* inputs, int32_t nbInputs, char const* equation) noexcept
7382 {
7383     return mImpl->addEinsum(inputs, nbInputs, equation);
7384 }
7385
7386 protected:
7387     apiv::VNetworkDefinition* mImpl;
7388 };
7389
7395 enum class CalibrationAlgoType : int32_t
7396 {
7397     kLEGACY.CALIBRATION = 0,
7398     kENTROPY.CALIBRATION = 1,
7399     kENTROPY.CALIBRATION_2 = 2,
7400     kMINMAX.CALIBRATION = 3,
7401 };
7402
7408 template <>
7409 constexpr inline int32_t EnumMax<CalibrationAlgoType>() noexcept
7410 {
7411     return 4;
7412 }
7413
7425 class IInt8Calibrator
7426 {
7427 public:
7433     virtual int32_t getBatchSize() const noexcept = 0;
7434
7448     virtual bool getBatch(void* bindings[], char const* names[], int32_t nbBindings) noexcept = 0;
7449
7464     virtual void const* readCalibrationCache(std::size_t& length) noexcept = 0;
7465

```



```

7474     virtual void writeCalibrationCache(void const* ptr, std::size_t length) noexcept = 0;
7475
7481     virtual CalibrationAlgoType getAlgorithm() noexcept = 0;
7482
7483     virtual ~IInt8Calibrator() noexcept = default;
7484 };
7485
7490 class IInt8EntropyCalibrator : public IInt8Calibrator
7491 {
7492 public:
7496     CalibrationAlgoType getAlgorithm() noexcept override
7497     {
7498         return CalibrationAlgoType::kENTROPY_CALIBRATION;
7499     }
7500
7501     virtual ~IInt8EntropyCalibrator() noexcept = default;
7502 };
7503
7508 class IInt8EntropyCalibrator2 : public IInt8Calibrator
7509 {
7510 public:
7514     CalibrationAlgoType getAlgorithm() noexcept override
7515     {
7516         return CalibrationAlgoType::kENTROPY_CALIBRATION_2;
7517     }
7518
7519     virtual ~IInt8EntropyCalibrator2() noexcept = default;
7520 };
7521
7525 class IInt8MinMaxCalibrator : public IInt8Calibrator
7526 {
7527 public:
7531     CalibrationAlgoType getAlgorithm() noexcept override
7532     {
7533         return CalibrationAlgoType::kMINMAX_CALIBRATION;
7534     }
7535
7536     virtual ~IInt8MinMaxCalibrator() noexcept = default;
7537 };
7538
7543 class IInt8LegacyCalibrator : public IInt8Calibrator
7544 {
7545 public:
7549     CalibrationAlgoType getAlgorithm() noexcept override
7550     {
7551         return CalibrationAlgoType::kLEGACY_CALIBRATION;
7552     }
7553
7560     virtual double getQuantile() const noexcept = 0;
7561
7568     virtual double getRegressionCutoff() const noexcept = 0;
7569
7582     virtual void const* readHistogramCache(std::size_t& length) noexcept = 0;
7583
7592     virtual void writeHistogramCache(void const* ptr, std::size_t length) noexcept = 0;
7593
7594     virtual ~IInt8LegacyCalibrator() noexcept = default;
7595 };
7596
7607 class IAlgorithmIOInfo : public INoCopy
7608 {
7609 public:
7613     TensorFormat getTensorFormat() const noexcept
7614     {
7615         return mImpl->getTensorFormat();
7616     }
7617
7621     DataType getDataType() const noexcept
7622     {
7623         return mImpl->getDataType();
7624     }
7625
7629     Dims getStrides() const noexcept
7630     {
7631         return mImpl->getStrides();
7632     }
7633
7634 protected:
7635     virtual ~IAlgorithmIOInfo() noexcept = default;
7636     apiv::VAlgorithmIOInfo* mImpl;
7637 };

```

```

7638
7650 class IAlgorithmVariant : public INoCopy
7651 {
7652 public:
7656     int64_t getImplementation() const noexcept
7657     {
7658         return mImpl->getImplementation();
7659     }
7660
7664     int64_t getTactic() const noexcept
7665     {
7666         return mImpl->getTactic();
7667     }
7668
7669 protected:
7670     virtual ~IAlgorithmVariant() noexcept = default;
7671     apiv::VAlgorithmVariant* mImpl;
7672 };
7673
7682 class IAlgorithmContext : public INoCopy
7683 {
7684 public:
7689     char const* getName() const noexcept
7690     {
7691         return mImpl->getName();
7692     }
7693
7700     Dims getDimensions(int32_t index, OptProfileSelector select) const noexcept
7701     {
7702         return mImpl->getDimensions(index, select);
7703     }
7704
7708     int32_t getNbInputs() const noexcept
7709     {
7710         return mImpl->getNbInputs();
7711     }
7712
7716     int32_t getNbOutputs() const noexcept
7717     {
7718         return mImpl->getNbOutputs();
7719     }
7720
7721 protected:
7722     virtual ~IAlgorithmContext() noexcept = default;
7723     apiv::VAlgorithmContext* mImpl;
7724 };
7725
7735 class IAlgorithm : public INoCopy
7736 {
7737 public:
7748     TRT_DEPRECATED IAlgorithmIOInfo const& getAlgorithmIOInfo(int32_t index) const noexcept
7749     {
7750         return mImpl->getAlgorithmIOInfo(index);
7751     }
7752
7756     IAlgorithmVariant const& getAlgorithmVariant() const noexcept
7757     {
7758         return mImpl->getAlgorithmVariant();
7759     }
7760
7764     float getTimingMSec() const noexcept
7765     {
7766         return mImpl->getTimingMSec();
7767     }
7768
7772     std::size_t getWorkspaceSize() const noexcept
7773     {
7774         return mImpl->getWorkspaceSize();
7775     }
7776
7785     IAlgorithmIOInfo const* getAlgorithmIOInfoByIndex(int32_t index) const noexcept
7786     {
7787         return mImpl->getAlgorithmIOInfoByIndex(index);
7788     }
7789
7790 protected:
7791     virtual ~IAlgorithm() noexcept = default;
7792     apiv::VAlgorithm* mImpl;
7793 }; // IAlgorithm
7794
7803 class IAlgorithmSelector

```

```

7804 {
7805 public:
7820     virtual int32_t selectAlgorithms(IAlgorithmContext const& context, IAlgorithm const* const* choices,
7821         int32_t nbChoices, int32_t* selection) noexcept = 0;
7832     virtual void reportAlgorithms(IAlgorithmContext const* const* algoContexts, IAlgorithm const* const*
7833         algoChoices,
7834         int32_t nbAlgorithms) noexcept = 0;
7835     virtual ~IAlgorithmSelector() noexcept = default;
7836 };
7837
7844 using QuantizationFlags = uint32_t;
7845
7853 enum class QuantizationFlag : int32_t
7854 {
7858     kCALIBRATE_BEFORE_FUSION = 0
7859 };
7860
7866 template <>
7867 constexpr inline int32_t EnumMax<QuantizationFlag>() noexcept
7868 {
7869     return 1;
7870 }
7871
7878 using BuilderFlags = uint32_t;
7879
7887 enum class BuilderFlag : int32_t
7888 {
7889     kFP16 = 0,
7890     kINT8 = 1,
7891     kDEBUG = 2,
7892     kGPU_FALLBACK = 3,
7893
7904     kSTRICT_TYPES TRT_DEPRECATED_ENUM = 4,
7905
7906     kREFIT = 5,
7907     kDISABLE_TIMING_CACHE = 6,
7908
7912     kTF32 = 7,
7913
7915     kSPARSE_WEIGHTS = 8,
7916
7923     kSAFETY_SCOPE = 9,
7924
7926     kOBEY_PRECISION_CONSTRAINTS = 10,
7927
7930     kPREFER_PRECISION_CONSTRAINTS = 11,
7931
7935     kDIRECT_IO = 12,
7936
7938     kREJECT_EMPTY_ALGORITHMS = 13
7939 };
7940
7946 template <>
7947 constexpr inline int32_t EnumMax<BuilderFlag>() noexcept
7948 {
7949     return 14;
7950 }
7951
7962 class ITimingCache : public INoCopy
7963 {
7964 public:
7965     virtual ~ITimingCache() noexcept = default;
7966
7976     nvinfer1::IHostMemory* serialize() const noexcept
7977     {
7978         return mImpl->serialize();
7979     }
7980
8000     bool combine(ITimingCache const& inputCache, bool ignoreMismatch) noexcept
8001     {
8002         return mImpl->combine(inputCache, ignoreMismatch);
8003     }
8004
8010     bool reset() noexcept
8011     {
8012         return mImpl->reset();
8013     }
8014
8015 protected:
8016     apiv::VTimingCache* mImpl;

```

```

8017 };
8018
8026 enum class MemoryPoolType : int32_t
8027 {
8035     kWORKSPACE = 0,
8036
8043     kDLA_MANAGED_SRAM = 1,
8044
8050     kDLA_LOCAL_DRAM = 2,
8051
8057     kDLA_GLOBAL_DRAM = 3,
8058 };
8059
8065 template <>
8066 constexpr inline int32_t EnumMax<MemoryPoolType>() noexcept
8067 {
8068     return 4;
8069 }
8070
8078 class IBuilderConfig : public INoCopy
8079 {
8080 public:
8081     virtual ~IBuilderConfig() noexcept = default;
8082
8095     TRT_DEPRECATED virtual void setMinTimingIterations(int32_t minTiming) noexcept
8096     {
8097         mImpl->setMinTimingIterations(minTiming);
8098     }
8099
8109     TRT_DEPRECATED virtual int32_t getMinTimingIterations() const noexcept
8110     {
8111         return mImpl->getMinTimingIterations();
8112     }
8113
8122     virtual void setAvgTimingIterations(int32_t avgTiming) noexcept
8123     {
8124         mImpl->setAvgTimingIterations(avgTiming);
8125     }
8126
8134     int32_t getAvgTimingIterations() const noexcept
8135     {
8136         return mImpl->getAvgTimingIterations();
8137     }
8138
8147     void setEngineCapability(EngineCapability capability) noexcept
8148     {
8149         mImpl->setEngineCapability(capability);
8150     }
8151
8159     EngineCapability getEngineCapability() const noexcept
8160     {
8161         return mImpl->getEngineCapability();
8162     }
8163
8169     void setInt8Calibrator(IInt8Calibrator* calibrator) noexcept
8170     {
8171         mImpl->setInt8Calibrator(calibrator);
8172     }
8173
8177     IInt8Calibrator* getInt8Calibrator() const noexcept
8178     {
8179         return mImpl->getInt8Calibrator();
8180     }
8181
8192     TRT_DEPRECATED void setMaxWorkspaceSize(std::size_t workspaceSize) noexcept
8193     {
8194         mImpl->setMaxWorkspaceSize(workspaceSize);
8195     }
8196
8209     TRT_DEPRECATED std::size_t getMaxWorkspaceSize() const noexcept
8210     {
8211         return mImpl->getMaxWorkspaceSize();
8212     }
8213
8226     void setFlags(BuilderFlags builderFlags) noexcept
8227     {
8228         mImpl->setFlags(builderFlags);
8229     }
8230
8238     BuilderFlags getFlags() const noexcept
8239     {

```

```

8240     return mImpl->getFlags();
8241 }
8242
8250 void clearFlag(BuilderFlag builderFlag) noexcept
8251 {
8252     mImpl->clearFlag(builderFlag);
8253 }
8254
8262 void setFlag(BuilderFlag builderFlag) noexcept
8263 {
8264     mImpl->setFlag(builderFlag);
8265 }
8266
8274 bool getFlag(BuilderFlag builderFlag) const noexcept
8275 {
8276     return mImpl->getFlag(builderFlag);
8277 }
8278
8289 void setDeviceType(ILayer const* layer, DeviceType deviceType) noexcept
8290 {
8291     mImpl->setDeviceType(layer, deviceType);
8292 }
8293
8298 DeviceType getDeviceType(ILayer const* layer) const noexcept
8299 {
8300     return mImpl->getDeviceType(layer);
8301 }
8302
8308 bool isDeviceTypeSet(ILayer const* layer) const noexcept
8309 {
8310     return mImpl->isDeviceTypeSet(layer);
8311 }
8312
8318 void resetDeviceType(ILayer const* layer) noexcept
8319 {
8320     mImpl->resetDeviceType(layer);
8321 }
8322
8327 bool canRunOnDLA(ILayer const* layer) const noexcept
8328 {
8329     return mImpl->canRunOnDLA(layer);
8330 }
8331
8342 void setDLACore(int32_t dlaCore) noexcept
8343 {
8344     mImpl->setDLACore(dlaCore);
8345 }
8346
8351 int32_t getDLACore() const noexcept
8352 {
8353     return mImpl->getDLACore();
8354 }
8355
8361 void setDefaultDeviceType(DeviceType deviceType) noexcept
8362 {
8363     mImpl->setDefaultDeviceType(deviceType);
8364 }
8365
8371 DeviceType getDefaultDeviceType() const noexcept
8372 {
8373     return mImpl->getDefaultDeviceType();
8374 }
8375
8381 void reset() noexcept
8382 {
8383     mImpl->reset();
8384 }
8385
8395 TRT_DEPRECATED void destroy() noexcept
8396 {
8397     delete this;
8398 }
8399
8407 void setProfileStream(const cudaStream_t stream) noexcept
8408 {
8409     return mImpl->setProfileStream(stream);
8410 }
8411
8419 cudaStream_t getProfileStream() const noexcept
8420 {
8421     return mImpl->getProfileStream();

```

```

8422     }
8423
8435     int32_t addOptimizationProfile(IOptimizationProfile const* profile) noexcept
8436     {
8437         return mImpl->addOptimizationProfile(profile);
8438     }
8439
8448     int32_t getNbOptimizationProfiles() const noexcept
8449     {
8450         return mImpl->getNbOptimizationProfiles();
8451     }
8452
8460     void setProfilingVerbosity(ProfilingVerbosity verbosity) noexcept
8461     {
8462         mImpl->setProfilingVerbosity(verbosity);
8463     }
8464
8473     ProfilingVerbosity getProfilingVerbosity() const noexcept
8474     {
8475         return mImpl->getProfilingVerbosity();
8476     }
8477
8482     void setAlgorithmSelector(IAgorithmSelector* selector) noexcept
8483     {
8484         mImpl->setAlgorithmSelector(selector);
8485     }
8486
8490     IAgorithmSelector* getAlgorithmSelector() const noexcept
8491     {
8492         return mImpl->getAlgorithmSelector();
8493     }
8494
8505     bool setCalibrationProfile(IOptimizationProfile const* profile) noexcept
8506     {
8507         return mImpl->setCalibrationProfile(profile);
8508     }
8509
8515     IOptimizationProfile const* getCalibrationProfile() noexcept
8516     {
8517         return mImpl->getCalibrationProfile();
8518     }
8519
8532     void setQuantizationFlags(QuantizationFlags flags) noexcept
8533     {
8534         mImpl->setQuantizationFlags(flags);
8535     }
8536
8544     QuantizationFlags getQuantizationFlags() const noexcept
8545     {
8546         return mImpl->getQuantizationFlags();
8547     }
8548
8556     void clearQuantizationFlag(QuantizationFlag flag) noexcept
8557     {
8558         mImpl->clearQuantizationFlag(flag);
8559     }
8560
8568     void setQuantizationFlag(QuantizationFlag flag) noexcept
8569     {
8570         mImpl->setQuantizationFlag(flag);
8571     }
8572
8580     bool getQuantizationFlag(QuantizationFlag flag) const noexcept
8581     {
8582         return mImpl->getQuantizationFlag(flag);
8583     }
8584
8605     bool setTacticSources(TacticSources tacticSources) noexcept
8606     {
8607         return mImpl->setTacticSources(tacticSources);
8608     }
8609
8620     TacticSources getTacticSources() const noexcept
8621     {
8622         return mImpl->getTacticSources();
8623     }
8624
8639     nvinfer1::ITimingCache* createTimingCache(void const* blob, std::size_t size) const noexcept
8640     {
8641         return mImpl->createTimingCache(blob, size);
8642     }

```

```

8643
8662     bool setTimingCache(ITimingCache const& cache, bool ignoreMismatch) noexcept
8663     {
8664         return mImpl->setTimingCache(cache, ignoreMismatch);
8665     }
8666
8672     nvinfer1::ITimingCache const* getTimingCache() const noexcept
8673     {
8674         return mImpl->getTimingCache();
8675     }
8676
8704     void setMemoryPoolLimit(MemoryPoolType pool, std::size_t poolSize) noexcept
8705     {
8706         mImpl->setMemoryPoolLimit(pool, poolSize);
8707     }
8708
8723     std::size_t getMemoryPoolLimit(MemoryPoolType pool) const noexcept
8724     {
8725         return mImpl->getMemoryPoolLimit(pool);
8726     }
8727
8728 protected:
8729     apiv::VBuilderConfig* mImpl;
8730 };
8731
8738 using NetworkDefinitionCreationFlags = uint32_t;
8739
8748 enum class NetworkDefinitionCreationFlag : int32_t
8749 {
8755     kEXPLICIT_BATCH = 0,
8756
8759     kEXPLICIT_PRECISION TRT_DEPRECATED_ENUM = 1,
8760 };
8761
8767 template <>
8768 constexpr inline int32_t EnumMax<NetworkDefinitionCreationFlag>() noexcept
8769 {
8770     return 2;
8771 }
8772
8780 class IBuilder : public INoCopy
8781 {
8782 public:
8783     virtual ~IBuilder() noexcept = default;
8784
8795     TRT_DEPRECATED void setMaxBatchSize(int32_t batchSize) noexcept
8796     {
8797         mImpl->setMaxBatchSize(batchSize);
8798     }
8799
8810     TRT_DEPRECATED int32_t getMaxBatchSize() const noexcept
8811     {
8812         return mImpl->getMaxBatchSize();
8813     }
8814
8818     bool platformHasFastFp16() const noexcept
8819     {
8820         return mImpl->platformHasFastFp16();
8821     }
8822
8826     bool platformHasFastInt8() const noexcept
8827     {
8828         return mImpl->platformHasFastInt8();
8829     }
8830
8838     TRT_DEPRECATED void destroy() noexcept
8839     {
8840         delete this;
8841     }
8842
8850     int32_t getMaxDLABatchSize() const noexcept
8851     {
8852         return mImpl->getMaxDLABatchSize();
8853     }
8854
8858     int32_t getNbDLACores() const noexcept
8859     {
8860         return mImpl->getNbDLACores();
8861     }
8862
8874     void setGpuAllocator(IGpuAllocator* allocator) noexcept

```

```

8875     {
8876         mImpl->setGpuAllocator(allocator);
8877     }
8878
8884     nvinfer1::IBuilderConfig* createBuilderConfig() noexcept
8885     {
8886         return mImpl->createBuilderConfig();
8887     }
8888
8899     TRT_DEPRECATED nvinfer1::ICudaEngine* buildEngineWithConfig(
8900         INetworkDefinition& network, IBuilderConfig& config) noexcept
8901     {
8902         return mImpl->buildEngineWithConfig(network, config);
8903     }
8904
8917     nvinfer1::INetworkDefinition* createNetworkV2(NetworkDefinitionCreationFlags flags) noexcept
8918     {
8919         return mImpl->createNetworkV2(flags);
8920     }
8921
8931     nvinfer1::IOptimizationProfile* createOptimizationProfile() noexcept
8932     {
8933         return mImpl->createOptimizationProfile();
8934     }
8935
8947     //
8950     void setErrorRecorder(IErrorRecorder* recorder) noexcept
8951     {
8952         mImpl->setErrorRecorder(recorder);
8953     }
8954
8965     IErrorRecorder* getErrorRecorder() const noexcept
8966     {
8967         return mImpl->getErrorRecorder();
8968     }
8969
8973     void reset() noexcept
8974     {
8975         mImpl->reset();
8976     }
8977
8981     bool platformHasTf32() const noexcept
8982     {
8983         return mImpl->platformHasTf32();
8984     }
8985
9000     nvinfer1::IHostMemory* buildSerializedNetwork(INetworkDefinition& network, IBuilderConfig& config)
noexcept
9001     {
9002         return mImpl->buildSerializedNetwork(network, config);
9003     }
9004
9024     bool isNetworkSupported(INetworkDefinition const& network, IBuilderConfig const& config) const noexcept
9025     {
9026         return mImpl->isNetworkSupported(network, config);
9027     }
9028
9034     ILogger* getLogger() const noexcept
9035     {
9036         return mImpl->getLogger();
9037     }
9038
9048     bool setMaxThreads(int32_t maxThreads) noexcept
9049     {
9050         return mImpl->setMaxThreads(maxThreads);
9051     }
9052
9062     int32_t getMaxThreads() const noexcept
9063     {
9064         return mImpl->getMaxThreads();
9065     }
9066
9067 protected:
9068     apiv::VBuilder* mImpl;
9069 };
9070
9071 } // namespace nvinfer1
9072
9077 extern "C" TENSORRTAPI void* createInferBuilder_INTERNAL(void* logger, int32_t version) noexcept;
9078
9079 namespace nvinfer1

```



```

9080 {
9081 namespace
9082 {
9083
9091 inline IBuilder* createInferBuilder(ILogger& logger) noexcept
9092 {
9093     return static_cast<IBuilder*>(createInferBuilder_INTERNAL(&logger, NV_TENSORRT_VERSION));
9094 }
9095
9096 } // namespace
9097
9108 extern "C" TENSORRTAPI nvinfer1::IPluginRegistry* getBuilderPluginRegistry(
9109     nvinfer1::EngineCapability capability) noexcept;
9110
9111 } // namespace nvinfer1
9112
9113 #endif // NV_INFER_H

```

10.5 NvInferConsistency.h File Reference

```

#include "NvInferConsistencyImpl.h"
#include "NvInferRuntimeCommon.h"

```

Classes

- class [nvinfer1::consistency::IConsistencyChecker](#)
Validates a serialized engine blob.
- class [nvinfer1::consistency::IPluginChecker](#)
Consistency Checker plugin class for user implemented Plugins.

Namespaces

- namespace [nvinfer1](#)
The TensorRT API version 1 namespace.
- namespace [nvinfer1::consistency](#)

Functions

- void * [createConsistencyChecker_INTERNAL](#) (void *logger, void const *blob, size_t size, int32_t version)
Internal C entry point for creating IConsistencyChecker.

10.5.1 Function Documentation

10.5.1.1 createConsistencyChecker_INTERNAL()

```
void * createConsistencyChecker_INTERNAL (
    void * logger,
    void const * blob,
    size_t size,
    int32_t version )
```

Internal C entry point for creating IConsistencyChecker.

10.6 NvInferConsistency.h

[Go to the documentation of this file.](#)

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_INFERENCE_CONSISTENCY_H
14 #define NV_INFERENCE_CONSISTENCY_H
15
16 #include "NvInferConsistencyImpl.h"
17 #include "NvInferRuntimeCommon.h"
18
19
20 namespace nvinfer1
21 {
22
23 namespace consistency
24 {
25
26 class IConsistencyChecker
27 {
28 public:
29     //
30     bool validate() const noexcept
31     {
32         return mImpl->validate();
33     }
34
35     virtual ~IConsistencyChecker() = default;
36
37 protected:
38     apiv::VConsistencyChecker* mImpl;
39     IConsistencyChecker() = default;
40     IConsistencyChecker(IConsistencyChecker const& other) = delete;
41     IConsistencyChecker& operator=(IConsistencyChecker const& other) = delete;
42     IConsistencyChecker(IConsistencyChecker&& other) = delete;
43     IConsistencyChecker& operator=(IConsistencyChecker&& other) = delete;
44 };
45
46 class IPluginChecker : public IPluginCreator
47 {
48 public:
49     virtual bool validate(char const* name, void const* serialData, size_t serialLength, PluginTensorDesc
50         const* in,
51         size_t nbInputs, PluginTensorDesc const* out, size_t nbOutputs, int64_t workspaceSize) const noexcept
52         = 0;
53
54     IPluginChecker() = default;
55     virtual ~IPluginChecker() override = default;
56
57 protected:
```

```

99     IPluginChecker(IPluginChecker const&) = default;
100     IPluginChecker(IPluginChecker&&) = default;
101     IPluginChecker& operator=(IPluginChecker const&) &= default;
102     IPluginChecker& operator=(IPluginChecker&&) &= default;
103 };
104
105 } // namespace consistency
106
107 } // namespace nvinfer1
108
109 extern "C" TENSORRTAPI void* createConsistencyChecker_INTERNAL(void* logger, void const* blob, size_t size,
110     int32_t version);
111
112 namespace nvinfer1
113 {
114
115     namespace consistency
116     {
117
118         namespace // anonymous
119         {
120
121             inline IConsistencyChecker* createConsistencyChecker(ILogger& logger, void const* blob, size_t size)
122             {
123                 return static_cast<IConsistencyChecker*>(
124                     createConsistencyChecker_INTERNAL(&logger, blob, size, NV_TENSORRT_VERSION));
125             }
126
127         } // namespace
128     } // namespace consistency
129 } // namespace nvinfer1
130
131 #endif // NV_INFER_CONSISTENCY_H

```

10.7 NvInferLegacyDims.h File Reference

```
#include "NvInferRuntimeCommon.h"
```

Classes

- class [nvinfer1::Dims2](#)
Descriptor for two-dimensional data.
- class [nvinfer1::DimsHW](#)
Descriptor for two-dimensional spatial data.
- class [nvinfer1::Dims3](#)
Descriptor for three-dimensional data.
- class [nvinfer1::Dims4](#)
Descriptor for four-dimensional data.

Namespaces

- namespace [nvinfer1](#)
The TensorRT API version 1 namespace.

10.7.1 Detailed Description

This file contains declarations of legacy dimensions types which use channel semantics in their names, and declarations on which those types rely.

10.8 NvInferLegacyDims.h

[Go to the documentation of this file.](#)

```

1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_INFER_LEGACY_DIMS_H
14 #define NV_INFER_LEGACY_DIMS_H
15
16 #include "NvInferRuntimeCommon.h"
17
18
19
20
21
22
23
24
25
26
27
28
29
30 namespace nvinfer1
31 {
32     class Dims2 : public Dims
33     {
34     public:
35         Dims2()
36             : Dims{2, {}}
37         {
38         }
39
40         Dims2(int32_t d0, int32_t d1)
41             : Dims{2, {d0, d1}}
42         {
43         }
44     };
45
46     class DimsHW : public Dims2
47     {
48     public:
49         DimsHW()
50             : Dims2()
51         {
52         }
53
54         DimsHW(int32_t height, int32_t width)
55             : Dims2(height, width)
56         {
57         }
58
59         int32_t& h()
60         {
61             return d[0];
62         }
63
64         int32_t h() const
65         {
66             return d[0];
67         }
68
69         int32_t& w()
70         {
71             return d[1];
72         }
73
74         int32_t w() const
75         {
76             return d[1];
77         }
78     };
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122

```

```

123     }
124 };
125
130 class Dims3 : public Dims
131 {
132 public:
133     Dims3()
134         : Dims{3, {}}
135     {
136     }
137
138     Dims3(int32_t d0, int32_t d1, int32_t d2)
139         : Dims{3, {d0, d1, d2}}
140     {
141     }
142 };
143
144 class Dims4 : public Dims
145 {
146 public:
147     Dims4()
148         : Dims{4, {}}
149     {
150     }
151
152     Dims4(int32_t d0, int32_t d1, int32_t d2, int32_t d3)
153         : Dims{4, {d0, d1, d2, d3}}
154     {
155     }
156 };
157
158 } // namespace nvinfer1
159 #endif // NV_INFERR_LEGACY_DIMS_H

```

10.9 NvInferPlugin.h File Reference

```

#include "NvInfer.h"
#include "NvInferPluginUtils.h"

```

Functions

- `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".
- `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".
- `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".
- `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".

- `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".
- `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".*
- `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".
- `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.
- `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.
- `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 $y = scale * (x - mean) / \sqrt{variance + epsilon} + bias$ 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()

```
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".

10.9.2.2 createBatchedNMSPlugin()

```
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)`.

10.9.2.3 createInstanceNormalizationPlugin()

```
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.

10.9.2.4 createNMSPlugin()

```
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".

10.9.2.5 createNormalizePlugin()

```
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 (acrossSpatial is true) or nearby spatial locations (within channel in which case acrossSpatial is false).
<i>channelShared</i>	Whether the scale weight(s) is shared across channels.
<i>eps</i>	Epsilon for not dividing by zero.

10.9.2.6 createPriorBoxPlugin()

```
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".

10.9.2.7 createRegionPlugin()

```
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".

10.9.2.8 createReorgPlugin()

```
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.
---------------	--

10.9.2.9 createRPNROIPlugin()

```
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.

Parameters

<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.

10.9.2.10 createSplitPlugin()

```
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.

10.9.2.11 initLibNvInferPlugins()

```
bool initLibNvInferPlugins (
    void * logger,
    char const * libNamespace )
```

Initialize and register all the existing TensorRT plugins to the Plugin Registry with an optional namespace. The plugin library author should ensure that this function name is unique to the library. This function should be called once before accessing the Plugin Registry.

Parameters

<i>logger</i>	Logger object to print plugin registration information
<i>libNamespace</i>	Namespace used to register all the plugins in this library

10.10 NvInferPlugin.h

[Go to the documentation of this file.](#)

```

1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993–2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_INFERR_PLUGIN_H
14 #define NV_INFERR_PLUGIN_H
15
16 #include "NvInfer.h"
17 #include "NvInferPluginUtils.h"
18
19 extern "C"
20 {
21     TENSORRTAPI nvinfer1::IPluginV2* createRPNOIPlugin(int32_t featureStride, int32_t preNmsTop, int32_t
nmsMaxOut,
22         float iouThreshold, float minBoxSize, float spatialScale, nvinfer1::DimsHW pooling,
23         nvinfer1::Weights anchorRatios, nvinfer1::Weights anchorScales);
24
25     TENSORRTAPI nvinfer1::IPluginV2* createNormalizePlugin(
26         nvinfer1::Weights const* scales, bool acrossSpatial, bool channelShared, float eps);
27
28     TENSORRTAPI nvinfer1::IPluginV2* createPriorBoxPlugin(nvinfer1::plugin::PriorBoxParameters param);
29
30     TENSORRTAPI nvinfer1::IPluginV2* createAnchorGeneratorPlugin(
31         nvinfer1::plugin::GridAnchorParameters* param, int32_t numLayers);
32
33     TENSORRTAPI nvinfer1::IPluginV2* createNMSPlugin(nvinfer1::plugin::DetectionOutputParameters param);
34
35     TENSORRTAPI nvinfer1::IPluginV2* createReorgPlugin(int32_t stride);
36
37     TENSORRTAPI nvinfer1::IPluginV2* createRegionPlugin(nvinfer1::plugin::RegionParameters params);
38
39     TENSORRTAPI nvinfer1::IPluginV2* createBatchedNMSPlugin(nvinfer1::plugin::NMSParameters param);
40
41     TENSORRTAPI nvinfer1::IPluginV2* createSplitPlugin(int32_t axis, int32_t* outputLengths, int32_t
noutput);
42
43     TENSORRTAPI nvinfer1::IPluginV2* createInstanceNormalizationPlugin(
44         float epsilon, nvinfer1::Weights scaleWeights, nvinfer1::Weights biasWeights);
45
46     TENSORRTAPI bool initLibNvInferPlugins(void* logger, char const* libNamespace);
47 } // extern "C"
48
49 #endif // NV_INFERR_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 x W). [GridAnchorParameters](#) defines a set of parameters for creating the plugin layer for all feature maps. It contains:
- struct [nvinfer1::plugin::DetectionOutputParameters](#)
The DetectionOutput plugin layer generates the detection output based on location and confidence predictions by doing non maximum suppression. This plugin first decodes the bounding boxes based on the anchors generated. It then performs non_max_suppression on the decoded bounding boxes. [DetectionOutputParameters](#) defines a set of parameters for creating the DetectionOutput plugin layer. It contains:
- struct [nvinfer1::plugin::softmaxTree](#)
When performing yolo9000, [softmaxTree](#) is helping to do softmax on confidence scores, for element to get the precise classification through word-tree structured classification definition.
- struct [nvinfer1::plugin::RegionParameters](#)
The Region plugin layer performs region proposal calculation: generate 5 bounding boxes per cell (for yolo9000, generate 3 bounding boxes per cell). For each box, calculating its probabilities of objects detections from 80 pre-defined classifications (yolo9000 has 9418 pre-defined classifications, and these 9418 items are organized as work-tree structure). [RegionParameters](#) defines a set of parameters for creating the Region plugin layer.
- struct [nvinfer1::plugin::NMSParameters](#)
The [NMSParameters](#) are used by the BatchedNMSPlugin for performing the non_max_suppression operation over boxes for object detection networks.

Namespaces

- namespace [nvinfer1](#)
The TensorRT API version 1 namespace.
- namespace [nvinfer1::plugin](#)

Enumerations

- enum class [nvinfer1::plugin::CodeTypeSSD](#) : int32_t { [nvinfer1::plugin::CORNER](#) = 0 , [nvinfer1::plugin::CENTER_SIZE](#) = 1 , [nvinfer1::plugin::CORNER_SIZE](#) = 2 , [nvinfer1::plugin::TF_CENTER](#) = 3 }
- The type of encoding used for decoding the bounding boxes and loc_data.*

10.11.1 Detailed Description

This is the API for the Nvidia provided TensorRT plugin utilities. It lists all the parameters utilized by the TensorRT plugins.

10.12 NvInferPluginUtils.h

[Go to the documentation of this file.](#)

```

1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993–2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_INFERR_PLUGIN_UTILS_H
14 #define NV_INFERR_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         struct PriorBoxParameters
41         {
42             float *minSize, *maxSize, *aspectRatios;
43             int32_t numMinSize, numMaxSize, numAspectRatios;
44             bool flip;
45             bool clip;
46             float variance[4];
47             int32_t imgH, imgW;
48             float stepH, stepW;
49             float offset;
50         };
51
52         struct RPROIParams
53         {
54             int32_t poolingH;
55             int32_t poolingW;
56             int32_t featureStride;
57             int32_t preNmsTop;
58             int32_t nmsMaxOut;
59             int32_t anchorsRatioCount;
60             int32_t anchorsScaleCount;
61             float iouThreshold;
62             float minBoxSize;
63             float spatialScale;
64         };
65
66         struct GridAnchorParameters
67         {
68             float minSize, maxSize;
69             float* aspectRatios;
70             int32_t numAspectRatios, H, W;
71             float variance[4];
72         };
73
74         enum class CodeTypeSSD : int32_t
75         {
76             CORNER = 0,
77             CENTER_SIZE = 1,
78             CORNER_SIZE = 2,
79             TF_CENTER = 3
80         };
81
82         struct DetectionOutputParameters
83         {
84             bool shareLocation, varianceEncodedInTarget;
85             int32_t backgroundLabelId, numClasses, topK, keepTopK;
86             float confidenceThreshold, nmsThreshold;
87         };
88     }
89 }

```

```

158     CodeTypeSSD codeType;
159     int32_t inputOrder[3];
160     bool confSigmoid;
161     bool isNormalized;
162     bool isBatchAgnostic{true};
163 };
164
165 struct softmaxTree
166 {
167     int32_t* leaf;
168     int32_t n;
169     int32_t* parent;
170     int32_t* child;
171     int32_t* group;
172     char** name;
173
174     int32_t groups;
175     int32_t* groupSize;
176     int32_t* groupOffset;
177 };
178
179 struct RegionParameters
180 {
181     int32_t num;
182     int32_t coords;
183     int32_t classes;
184     softmaxTree* smTree;
185 };
186
187 struct NMSParameters
188 {
189     bool shareLocation;
190     int32_t backgroundLabelId, numClasses, topK, keepTopK;
191     float scoreThreshold, iouThreshold;
192     bool isNormalized;
193 };
194
195 } // namespace plugin
196 } // namespace nvinfer1
197
198 #endif // NV_INFERR_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::IExecutionContext](#)
 - Context for executing inference using an engine, with functionally unsafe features.*
- class [nvinfer1::IEngineInspector](#)
 - An engine inspector which prints out the layer information of an engine or an execution context.*
- class [nvinfer1::PluginRegistrar< T >](#)
 - Register the plugin creator to the registry The static registry object will be instantiated when the plugin library is loaded. This static object will register all creators available in the library to the registry.*

Namespaces

- namespace [nvinfer1](#)
 - The TensorRT API version 1 namespace.*
- namespace [nvinfer1::impl](#)

Macros

- `#define REGISTER_TENSORRT_PLUGIN(name) static nvinfer1::PluginRegistrar<name> plugin← Registrar##name {}`

Typedefs

- using [nvinfer1::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::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 }
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< OptProfileSelector > () noexcept`
Number of different values of OptProfileSelector enum.
- template<> constexpr `int32_t nvinfer1::EnumMax< TacticSource > () noexcept`
Maximum number of tactic sources in TacticSource enum.

- `template<> constexpr int32_t nvinfer1::EnumMax< ProfilingVerbosity > () noexcept`
Maximum number of profile verbosity levels in ProfilingVerbosity enum.
- `template<> constexpr int32_t nvinfer1::EnumMax< LayerInformationFormat > () noexcept`
- `nvinfer1::IPluginRegistry * getPluginRegistry () noexcept`
Return the plugin registry.
- `nvinfer1::ILogger * getLogger () noexcept`
Return the logger object.

10.13.1 Detailed Description

This is the top-level API file for TensorRT extended runtime library.

10.13.2 Macro Definition Documentation

10.13.2.1 REGISTER_TENSORRT_PLUGIN

```
#define REGISTER_TENSORRT_PLUGIN(  
    name ) static nvinfer1::PluginRegistrar<name> pluginRegistrar##name {}
```

10.13.3 Function Documentation

10.13.3.1 getLogger()

```
nvinfer1::ILogger * getLogger ( ) [noexcept]
```

Return the logger object.

Note

the global logger is used only by standalone functions which have no associated builder, runtime or refitter.

10.13.3.2 getPluginRegistry()

```
nvinfer1::IPluginRegistry * getPluginRegistry ( ) [noexcept]
```

Return the plugin registry.

10.14 NvInferRuntime.h

[Go to the documentation of this file.](#)

```

1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_INFERRUNTIME_H
14 #define NV_INFERRUNTIME_H
15
16
17
18
19
20
21
22 #include "NvInferImpl.h"
23 #include "NvInferRuntimeCommon.h"
24
25 namespace nvinfer1
26 {
27
28 class IExecutionContext;
29 class ICudaEngine;
30 class IPluginFactory;
31 class IEngineInspector;
32
33
34
35
36
37
38
39
40
41
42 class INoCopy
43 {
44 protected:
45     INoCopy() = default;
46     virtual ~INoCopy() = default;
47     INoCopy(INoCopy const& other) = delete;
48     INoCopy& operator=(INoCopy const& other) = delete;
49     INoCopy(INoCopy&& other) = delete;
50     INoCopy& operator=(INoCopy&& other) = delete;
51 };
52
53
54
55
56
57
58 enum class EngineCapability : int32_t
59 {
60     kSTANDARD = 0,
61
62     kDEFAULT TRT_DEPRECATED_ENUM = kSTANDARD,
63
64     kSAFETY = 1,
65
66     kSAFE_GPU TRT_DEPRECATED_ENUM = kSAFETY,
67
68     kDLA_STANDALONE = 2,
69
70     kSAFE_DLA TRT_DEPRECATED_ENUM = kDLA_STANDALONE,
71 };
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99 };
100
101 namespace impl
102 {
103
104 template <>
105 struct EnumMaxImpl<EngineCapability>
106 {
107     static constexpr int32_t kVALUE = 3;
108 };
109 } // namespace impl
110
111
112
113
114
115 class Weights
116 {
117 public:
118     DataType type;
119     void const* values;
120     int64_t count;
121 };
122
123
124
125 class IHostMemory : public INoCopy
126 {
127 public:
128     virtual ~IHostMemory() noexcept = default;

```

```

147
149 void* data() const noexcept
150 {
151     return mImpl->data();
152 }
153
155 std::size_t size() const noexcept
156 {
157     return mImpl->size();
158 }
159
161 DataType type() const noexcept
162 {
163     return mImpl->type();
164 }
172 TRT_DEPRECATED void destroy() noexcept
173 {
174     delete this;
175 }
176
177 protected:
178     apiv::VHostMemory* mImpl;
179 };
180
191 enum class DimensionOperation : int32_t
192 {
193     kSUM = 0,
194     kPROD = 1,
195     kMAX = 2,
196     kMIN = 3,
197     kSUB = 4,
198     kEQUAL = 5,
199     kLESS = 6,
200     kFLOOR_DIV = 7,
201     kCEIL_DIV = 8
202 };
203
205 template <>
206 constexpr inline int32_t EnumMax<DimensionOperation>() noexcept
207 {
208     return 9;
209 }
210
215 enum class TensorLocation : int32_t
216 {
217     kDEVICE = 0,
218     kHOST = 1,
219 };
220
221 namespace impl
222 {
224 template <>
225 struct EnumMaxImpl<TensorLocation>
226 {
227     static constexpr int32_t kVALUE = 2;
228 };
229 } // namespace impl
230
243 class IDimensionExpr : public INoCopy
244 {
245 public:
247     bool isConstant() const noexcept
248     {
249         return mImpl->isConstant();
250     }
251
254     int32_t getConstantValue() const noexcept
255     {
256         return mImpl->getConstantValue();
257     }
258
259 protected:
260     apiv::VDimensionExpr* mImpl;
261     virtual ~IDimensionExpr() noexcept = default;
262 };
263
281 class IExprBuilder : public INoCopy
282 {
283 public:
285     IDimensionExpr const* constant(int32_t value) noexcept
286     {

```

```

287     return mImpl->constant(value);
288 }
289
292 IDimensionExpr const* operation(
293     DimensionOperation op, IDimensionExpr const& first, IDimensionExpr const& second) noexcept
294 {
295     return mImpl->operation(op, first, second);
296 }
297
298 protected:
299     apiv::VExprBuilder* mImpl;
300     virtual ~IExprBuilder() noexcept = default;
301 };
302
308 class DimsExprs
309 {
310 public:
311     int32_t nbDims;
312     IDimensionExpr const* d[Dims::MAX_DIMS];
313 };
314
320 struct DynamicPluginTensorDesc
321 {
322     PluginTensorDesc desc;
323
324     Dims min;
325
326     Dims max;
327
328     Dims max;
329 };
330
331
350 class IPluginV2DynamicExt : public nvinfer1::IPluginV2Ext
351 {
352 public:
353     IPluginV2DynamicExt* clone() const noexcept override = 0;
354
379     virtual DimsExprs getOutputDimensions(
380         int32_t outputIndex, DimsExprs const* inputs, int32_t nbInputs, IExprBuilder& exprBuilder) noexcept =
381         0;
382
385     static constexpr int32_t kFORMAT_COMBINATION_LIMIT = 100;
386
419     virtual bool supportsFormatCombination(
420         int32_t pos, PluginTensorDesc const* inOut, int32_t nbInputs, int32_t nbOutputs) noexcept = 0;
421
460     virtual void configurePlugin(DynamicPluginTensorDesc const* in, int32_t nbInputs,
461         DynamicPluginTensorDesc const* out, int32_t nbOutputs) noexcept = 0;
462
472     virtual size_t getWorkspaceSize(PluginTensorDesc const* inputs, int32_t nbInputs, PluginTensorDesc const*
473         outputs,
474         int32_t nbOutputs) const noexcept = 0;
475
487     virtual int32_t enqueue(PluginTensorDesc const* inputDesc, PluginTensorDesc const* outputDesc,
488         void const* const* inputs, void* const* outputs, void* workspace, cudaStream_t stream) noexcept = 0;
489
490 protected:
498     int32_t getTensorRTVersion() const noexcept override
499     {
500         return (static_cast<int32_t>(PluginVersion::kV2_DYNAMIC_EXT) << 24 | (NV_TENSORRT_VERSION & 0xFFFF));
501     }
502
503     virtual ~IPluginV2DynamicExt() noexcept {}
504
505 private:
506     // Following are obsolete base class methods, and must not be implemented or used.
507
508     void configurePlugin(Dims const*, int32_t, Dims const*, int32_t, DataType const*, DataType const*, bool
509         const*,
510         bool const*, PluginFormat, int32_t) noexcept override final
511     {
512     }
513
514     bool supportsFormat(DataType, PluginFormat) const noexcept override final
515     {
516         return false;
517     }
518
519     Dims getOutputDimensions(int32_t, Dims const*, int32_t) noexcept override final
520     {
521         return Dims{-1, {}};
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 class IRuntime : public INoCopy
580 {
581 public:
582     virtual ~IRuntime() noexcept = default;
583
584     TRT_DEPRECATED nvinfer1::ICudaEngine* deserializeCudaEngine(
585         void const* blob, std::size_t size, IPluginFactory* pluginFactory) noexcept
586     {
587         return mImpl->deserializeCudaEngine(blob, size, nullptr);
588     }
589
590     void setDLACore(int32_t dlaCore) noexcept
591     {
592         mImpl->setDLACore(dlaCore);
593     }
594
595     int32_t getDLACore() const noexcept
596     {
597         return mImpl->getDLACore();
598     }
599
600     int32_t getNbDLACores() const noexcept
601     {
602         return mImpl->getNbDLACores();
603     }
604 };

```

```

673     }
674
682     TRT_DEPRECATED void destroy() noexcept
683     {
684         delete this;
685     }
686
696     void setGpuAllocator(IGpuAllocator* allocator) noexcept
697     {
698         mImpl->setGpuAllocator(allocator);
699     }
700
712     //
715     void setErrorRecorder(IErrorRecorder* recorder) noexcept
716     {
717         mImpl->setErrorRecorder(recorder);
718     }
719
730     IErrorRecorder* getErrorRecorder() const noexcept
731     {
732         return mImpl->getErrorRecorder();
733     }
734
745     ICudaEngine* deserializeCudaEngine(void const* blob, std::size_t size) noexcept
746     {
747         return mImpl->deserializeCudaEngine(blob, size, nullptr);
748     }
749
755     ILogger* getLogger() const noexcept
756     {
757         return mImpl->getLogger();
758     }
759
769     bool setMaxThreads(int32_t maxThreads) noexcept
770     {
771         return mImpl->setMaxThreads(maxThreads);
772     }
773
783     int32_t getMaxThreads() const noexcept
784     {
785         return mImpl->getMaxThreads();
786     }
787
788 protected:
789     apiv::VRuntime* mImpl;
790 };
791
799 class IRewriter : public INoCopy
800 {
801 public:
802     virtual ~IRewriter() noexcept = default;
803
814     bool setWeights(char const* layerName, WeightsRole role, Weights weights) noexcept
815     {
816         return mImpl->setWeights(layerName, role, weights);
817     }
818
829     bool refitCudaEngine() noexcept
830     {
831         return mImpl->refitCudaEngine();
832     }
833
850     int32_t getMissing(int32_t size, char const** layerNames, WeightsRole* roles) noexcept
851     {
852         return mImpl->getMissing(size, layerNames, roles);
853     }
854
867     int32_t getAll(int32_t size, char const** layerNames, WeightsRole* roles) noexcept
868     {
869         return mImpl->getAll(size, layerNames, roles);
870     }
871
877     TRT_DEPRECATED void destroy() noexcept
878     {
879         delete this;
880     }
881
894     bool setDynamicRange(char const* tensorName, float min, float max) noexcept
895     {
896         return mImpl->setDynamicRange(tensorName, min, max);
897     }

```

```

898
906     float getDynamicRangeMin(char const* tensorName) const noexcept
907     {
908         return mImpl->getDynamicRangeMin(tensorName);
909     }
910
918     float getDynamicRangeMax(char const* tensorName) const noexcept
919     {
920         return mImpl->getDynamicRangeMax(tensorName);
921     }
922
934     int32_t getTensorsWithDynamicRange(int32_t size, char const** tensorNames) const noexcept
935     {
936         return mImpl->getTensorsWithDynamicRange(size, tensorNames);
937     }
938
950     //
953     void setErrorRecorder(IErrorRecorder* recorder) noexcept
954     {
955         mImpl->setErrorRecorder(recorder);
956     }
957
968     IErrorRecorder* getErrorRecorder() const noexcept
969     {
970         return mImpl->getErrorRecorder();
971     }
972
986     bool setNamedWeights(char const* name, Weights weights) noexcept
987     {
988         return mImpl->setNamedWeights(name, weights);
989     }
990
1006     int32_t getMissingWeights(int32_t size, char const** weightsNames) noexcept
1007     {
1008         return mImpl->getMissingWeights(size, weightsNames);
1009     }
1010
1022     int32_t getAllWeights(int32_t size, char const** weightsNames) noexcept
1023     {
1024         return mImpl->getAllWeights(size, weightsNames);
1025     }
1026
1032     ILogger* getLogger() const noexcept
1033     {
1034         return mImpl->getLogger();
1035     }
1036
1046     bool setMaxThreads(int32_t maxThreads) noexcept
1047     {
1048         return mImpl->setMaxThreads(maxThreads);
1049     }
1050
1060     int32_t getMaxThreads() const noexcept
1061     {
1062         return mImpl->getMaxThreads();
1063     }
1064
1065 protected:
1066     apiv::VRefitter* mImpl;
1067 };
1068
1079 enum class OptProfileSelector : int32_t
1080 {
1081     kMIN = 0,
1082     kOPT = 1,
1083     kMAX = 2
1084 };
1085
1091 template <>
1092 constexpr inline int32_t EnumMax<OptProfileSelector>() noexcept
1093 {
1094     return 3;
1095 }
1096
1119 class IOptimizationProfile : public INoCopy
1120 {
1121 public:
1147     bool setDimensions(char const* inputName, OptProfileSelector select, Dims dims) noexcept
1148     {
1149         return mImpl->setDimensions(inputName, select, dims);
1150     }

```

```

1151
1157     Dims getDimensions(char const* inputName, OptProfileSelector select) const noexcept
1158     {
1159         return mImpl->getDimensions(inputName, select);
1160     }
1161
1200     bool setShapeValues(
1201         char const* inputName, OptProfileSelector select, int32_t const* values, int32_t nbValues) noexcept
1202     {
1203         return mImpl->setShapeValues(inputName, select, values, nbValues);
1204     }
1205
1212     int32_t getNbShapeValues(char const* inputName) const noexcept
1213     {
1214         return mImpl->getNbShapeValues(inputName);
1215     }
1216
1222     int32_t const* getShapeValues(char const* inputName, OptProfileSelector select) const noexcept
1223     {
1224         return mImpl->getShapeValues(inputName, select);
1225     }
1226
1240     bool setExtraMemoryTarget(float target) noexcept
1241     {
1242         return mImpl->setExtraMemoryTarget(target);
1243     }
1244
1248     float getExtraMemoryTarget() const noexcept
1249     {
1250         return mImpl->getExtraMemoryTarget();
1251     }
1252
1264     bool isValid() const noexcept
1265     {
1266         return mImpl->isValid();
1267     }
1268
1269 protected:
1270     apiv::VOptimizationProfile* mImpl;
1271     virtual ~IOptimizationProfile() noexcept = default;
1272 };
1273
1281 enum class TacticSource : int32_t
1282 {
1283     kCUBLAS = 0,
1284     kCUBLAS_LT = 1,
1285     kCUDNN = 2,
1286
1287     kEDGE_MASK_CONVOLUTIONS = 3
1288 };
1289
1292 template <>
1293 constexpr inline int32_t EnumMax<TacticSource>() noexcept
1294 {
1295     return 4;
1296 }
1297
1298 using TacticSources = uint32_t;
1305
1316 enum class ProfilingVerbosity : int32_t
1317 {
1318     kLAYER_NAMES_ONLY = 0,
1319     kNONE = 1,
1320     kDETAILED = 2,
1321
1322     kDEFAULT_TRT_DEPRECATED_ENUM = kLAYER_NAMES_ONLY,
1323     kVERBOSE_TRT_DEPRECATED_ENUM = kDETAILED
1324 };
1325
1327 template <>
1328 constexpr inline int32_t EnumMax<ProfilingVerbosity>() noexcept
1329 {
1330     return 3;
1331 }
1332
1333
1334 class ICudaEngine : public INoCopy
1335 {
1336 public:
1337     virtual ~ICudaEngine() noexcept = default;
1338
1339     int32_t getNbBindings() const noexcept

```



```

1358     {
1359         return mImpl->getNbBindings();
1360     }
1361
1379     int32_t getBindingIndex(char const* name) const noexcept
1380     {
1381         return mImpl->getBindingIndex(name);
1382     }
1383
1399     char const* getBindingName(int32_t bindingIndex) const noexcept
1400     {
1401         return mImpl->getBindingName(bindingIndex);
1402     }
1403
1412     bool bindingIsInput(int32_t bindingIndex) const noexcept
1413     {
1414         return mImpl->bindingIsInput(bindingIndex);
1415     }
1416
1437     Dims getBindingDimensions(int32_t bindingIndex) const noexcept
1438     {
1439         return mImpl->getBindingDimensions(bindingIndex);
1440     }
1441
1450     DataType getBindingDataType(int32_t bindingIndex) const noexcept
1451     {
1452         return mImpl->getBindingDataType(bindingIndex);
1453     }
1454
1466     TRT_DEPRECATED int32_t getMaxBatchSize() const noexcept
1467     {
1468         return mImpl->getMaxBatchSize();
1469     }
1470
1480     int32_t getNbLayers() const noexcept
1481     {
1482         return mImpl->getNbLayers();
1483     }
1484
1494     IHostMemory* serialize() const noexcept
1495     {
1496         return mImpl->serialize();
1497     }
1498
1510     IExecutionContext* createExecutionContext() noexcept
1511     {
1512         return mImpl->createExecutionContext();
1513     }
1514
1522     TRT_DEPRECATED void destroy() noexcept
1523     {
1524         delete this;
1525     }
1526
1537     TensorLocation getLocation(int32_t bindingIndex) const noexcept
1538     {
1539         return mImpl->getLocation(bindingIndex);
1540     }
1541
1546     IExecutionContext* createExecutionContextWithoutDeviceMemory() noexcept
1547     {
1548         return mImpl->createExecutionContextWithoutDeviceMemory();
1549     }
1550
1556     size_t getDeviceMemorySize() const noexcept
1557     {
1558         return mImpl->getDeviceMemorySize();
1559     }
1560
1566     bool isRefittable() const noexcept
1567     {
1568         return mImpl->isRefittable();
1569     }
1570
1580     int32_t getBindingBytesPerComponent(int32_t bindingIndex) const noexcept
1581     {
1582         return mImpl->getBindingBytesPerComponent(bindingIndex);
1583     }
1584
1594     int32_t getBindingComponentsPerElement(int32_t bindingIndex) const noexcept
1595     {

```

```

1596     return mImpl->getBindingComponentsPerElement(bindingIndex);
1597 }
1598
1604 TensorFormat getBindingFormat(int32_t bindingIndex) const noexcept
1605 {
1606     return mImpl->getBindingFormat(bindingIndex);
1607 }
1608
1623 char const* getBindingFormatDesc(int32_t bindingIndex) const noexcept
1624 {
1625     return mImpl->getBindingFormatDesc(bindingIndex);
1626 }
1627
1635 int32_t getBindingVectorizedDim(int32_t bindingIndex) const noexcept
1636 {
1637     return mImpl->getBindingVectorizedDim(bindingIndex);
1638 }
1639
1650 char const* getName() const noexcept
1651 {
1652     return mImpl->getName();
1653 }
1654
1661 int32_t getNbOptimizationProfiles() const noexcept
1662 {
1663     return mImpl->getNbOptimizationProfiles();
1664 }
1665
1688 Dims getProfileDimensions(int32_t bindingIndex, int32_t profileIndex, OptProfileSelector select) const
noexcept
1689 {
1690     return mImpl->getProfileDimensions(bindingIndex, profileIndex, select);
1691 }
1692
1714 int32_t const* getProfileShapeValues(
1715     int32_t profileIndex, int32_t inputIndex, OptProfileSelector select) const noexcept
1716 {
1717     return mImpl->getProfileShapeValues(profileIndex, inputIndex, select);
1718 }
1719
1751 bool isShapeBinding(int32_t bindingIndex) const noexcept
1752 {
1753     return mImpl->isShapeBinding(bindingIndex);
1754 }
1755
1765 bool isExecutionBinding(int32_t bindingIndex) const noexcept
1766 {
1767     return mImpl->isExecutionBinding(bindingIndex);
1768 }
1769
1780 EngineCapability getEngineCapability() const noexcept
1781 {
1782     return mImpl->getEngineCapability();
1783 }
1784
1795 //
1798 void setErrorRecorder(IErrorRecorder* recorder) noexcept
1799 {
1800     return mImpl->setErrorRecorder(recorder);
1801 }
1802
1813 IErrorRecorder* getErrorRecorder() const noexcept
1814 {
1815     return mImpl->getErrorRecorder();
1816 }
1817
1832 bool hasImplicitBatchDimension() const noexcept
1833 {
1834     return mImpl->hasImplicitBatchDimension();
1835 }
1836
1847 TacticSources getTacticSources() const noexcept
1848 {
1849     return mImpl->getTacticSources();
1850 }
1851
1858 ProfilingVerbosity getProfilingVerbosity() const noexcept
1859 {
1860     return mImpl->getProfilingVerbosity();
1861 }
1862

```

```

1868     IEngineInspector* createEngineInspector() const noexcept
1869     {
1870         return mImpl->createEngineInspector();
1871     }
1872
1873 protected:
1874     apiv::VCudaEngine* mImpl;
1875 };
1876
1887 class IExecutionContext : public INoCopy
1888 {
1889 public:
1890     virtual ~IExecutionContext() noexcept = default;
1891
1914     TRT_DEPRECATED bool execute(int32_t batchSize, void* const* bindings) noexcept
1915     {
1916         return mImpl->execute(batchSize, bindings);
1917     }
1918
1948     TRT_DEPRECATED bool enqueue(
1949         int32_t batchSize, void* const* bindings, cudaStream_t stream, cudaEvent_t* inputConsumed) noexcept
1950     {
1951         return mImpl->enqueue(batchSize, bindings, stream, inputConsumed);
1952     }
1953
1962     void setDebugSync(bool sync) noexcept
1963     {
1964         mImpl->setDebugSync(sync);
1965     }
1966
1972     bool getDebugSync() const noexcept
1973     {
1974         return mImpl->getDebugSync();
1975     }
1976
1982     void setProfiler(IProfiler* profiler) noexcept
1983     {
1984         mImpl->setProfiler(profiler);
1985     }
1986
1992     IProfiler* getProfiler() const noexcept
1993     {
1994         return mImpl->getProfiler();
1995     }
1996
2002     ICudaEngine const& getEngine() const noexcept
2003     {
2004         return mImpl->getEngine();
2005     }
2006
2014     TRT_DEPRECATED void destroy() noexcept
2015     {
2016         delete this;
2017     }
2018
2026     void setName(char const* name) noexcept
2027     {
2028         mImpl->setName(name);
2029     }
2030
2036     char const* getName() const noexcept
2037     {
2038         return mImpl->getName();
2039     }
2040
2052     void setDeviceMemory(void* memory) noexcept
2053     {
2054         mImpl->setDeviceMemory(memory);
2055     }
2056
2073     Dims getStrides(int32_t bindingIndex) const noexcept
2074     {
2075         return mImpl->getStrides(bindingIndex);
2076     }
2077
2078 public:
2115     TRT_DEPRECATED
2116     bool setOptimizationProfile(int32_t profileIndex) noexcept
2117     {
2118         return mImpl->setOptimizationProfile(profileIndex);
2119     }

```

```

2120
2128     int32_t getOptimizationProfile() const noexcept
2129     {
2130         return mImpl->getOptimizationProfile();
2131     }
2132
2165     bool setBindingDimensions(int32_t bindingIndex, Dims dimensions) noexcept
2166     {
2167         return mImpl->setBindingDimensions(bindingIndex, dimensions);
2168     }
2169
2195     Dims getBindingDimensions(int32_t bindingIndex) const noexcept
2196     {
2197         return mImpl->getBindingDimensions(bindingIndex);
2198     }
2199
2225     bool setInputShapeBinding(int32_t bindingIndex, int32_t const* data) noexcept
2226     {
2227         return mImpl->setInputShapeBinding(bindingIndex, data);
2228     }
2229
2247     bool getShapeBinding(int32_t bindingIndex, int32_t* data) const noexcept
2248     {
2249         return mImpl->getShapeBinding(bindingIndex, data);
2250     }
2251
2262     bool allInputDimensionsSpecified() const noexcept
2263     {
2264         return mImpl->allInputDimensionsSpecified();
2265     }
2266
2276     bool allInputShapesSpecified() const noexcept
2277     {
2278         return mImpl->allInputShapesSpecified();
2279     }
2280
2281
2293     //
2296     void setErrorRecorder(IErrorRecorder* recorder) noexcept
2297     {
2298         mImpl->setErrorRecorder(recorder);
2299     }
2300
2311     IErrorRecorder* getErrorRecorder() const noexcept
2312     {
2313         return mImpl->getErrorRecorder();
2314     }
2315
2328     bool executeV2(void* const* bindings) noexcept
2329     {
2330         return mImpl->executeV2(bindings);
2331     }
2332
2356     bool enqueueV2(void* const* bindings, cudaStream_t stream, cudaEvent_t* inputConsumed) noexcept
2357     {
2358         return mImpl->enqueueV2(bindings, stream, inputConsumed);
2359     }
2360
2404     bool setOptimizationProfileAsync(int32_t profileIndex, cudaStream_t stream) noexcept
2405     {
2406         return mImpl->setOptimizationProfileAsync(profileIndex, stream);
2407     }
2408
2419     void setEnqueueEmitsProfile(bool enqueueEmitsProfile) noexcept
2420     {
2421         mImpl->setEnqueueEmitsProfile(enqueueEmitsProfile);
2422     }
2423
2430     bool getEnqueueEmitsProfile() const noexcept
2431     {
2432         return mImpl->getEnqueueEmitsProfile();
2433     }
2434
2459     bool reportToProfiler() const noexcept
2460     {
2461         return mImpl->reportToProfiler();
2462     }
2463
2464 protected:
2465     apiv::VExecutionContext* mImpl;
2466 }; // class IExecutionContext

```

```

2467
2475 enum class LayerInformationFormat : int32_t
2476 {
2477     kONELINE = 0,
2478     kJSON = 1,
2479 };
2480
2483 template <>
2484 constexpr inline int32_t EnumMax<LayerInformationFormat>() noexcept
2485 {
2486     return 2;
2487 }
2488
2504 class IEngineInspector : public INoCopy
2505 {
2506 public:
2507     virtual ~IEngineInspector() noexcept = default;
2508
2521     bool setExecutionContext(IExecutionContext const* context) noexcept
2522     {
2523         return mImpl->setExecutionContext(context);
2524     }
2525
2533     IExecutionContext const* getExecutionContext() const noexcept
2534     {
2535         return mImpl->getExecutionContext();
2536     }
2537
2558     AsciiChar const* getLayerInformation(int32_t layerIndex, LayerInformationFormat format) const noexcept
2559     {
2560         return mImpl->getLayerInformation(layerIndex, format);
2561     }
2562
2583     AsciiChar const* getEngineInformation(LayerInformationFormat format) const noexcept
2584     {
2585         return mImpl->getEngineInformation(format);
2586     }
2587
2599     //
2602     void setErrorRecorder(IErrorRecorder* recorder) noexcept
2603     {
2604         mImpl->setErrorRecorder(recorder);
2605     }
2606
2617     IErrorRecorder* getErrorRecorder() const noexcept
2618     {
2619         return mImpl->getErrorRecorder();
2620     }
2621
2622 protected:
2623     apiv::VEngineInspector* mImpl;
2624 }; // class IEngineInspector
2625
2626 } // namespace nvinfer1
2627
2632 extern "C" TENSORRTAPI void* createInferRuntime_INTERNAL(void* logger, int32_t version) noexcept;
2633
2638 extern "C" TENSORRTAPI void* createInferRefitter_INTERNAL(void* engine, void* logger, int32_t version)
2639     noexcept;
2643 extern "C" TENSORRTAPI nvinfer1::IPluginRegistry* getPluginRegistry() noexcept;
2644
2650 extern "C" TENSORRTAPI nvinfer1::ILogger* getLogger() noexcept;
2651
2652 namespace nvinfer1
2653 {
2654     namespace // unnamed namespace avoids linkage surprises when linking objects built with different versions
2655         of this // header.
2656     {
2662         inline IRuntime* createInferRuntime(ILogger& logger) noexcept
2663         {
2664             return static_cast<IRuntime*>(createInferRuntime_INTERNAL(&logger, NV_TENSORRT_VERSION));
2665         }
2666
2672         inline IRefitter* createInferRefitter(ICudaEngine& engine, ILogger& logger) noexcept
2673         {
2674             return static_cast<IRefitter*>(createInferRefitter_INTERNAL(&engine, &logger, NV_TENSORRT_VERSION));
2675         }
2676     }
2677 } // namespace

```

```

2678
2690 template <typename T>
2691 class PluginRegistrar
2692 {
2693 public:
2694     PluginRegistrar()
2695     {
2696         getPluginRegistry()->registerCreator(instance, "");
2697     }
2698
2699 private:
2701     T instance{};
2702 };
2703
2704 } // namespace nvinfer1
2705
2706 #define REGISTER_TENSORRT_PLUGIN(name)
2707     static nvinfer1::PluginRegistrar<name> pluginRegistrar##name {}
2708 #endif // NV_INFERENCE_RUNTIME_H

```

10.15 NvInferRuntimeCommon.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::PluginTensorDesc](#)
Fields that a plugin might see for an input or output.
- class [nvinfer1::IPluginV2](#)
Plugin class for user-implemented layers.
- class [nvinfer1::IPluginV2Ext](#)
Plugin class for user-implemented layers.
- class [nvinfer1::IPluginV2IOExt](#)
Plugin class for user-implemented layers.
- class [nvinfer1::PluginField](#)
Structure containing plugin attribute field names and associated data This information can be parsed to decode necessary plugin metadata.
- struct [nvinfer1::PluginFieldCollection](#)
Plugin field collection struct.
- class [nvinfer1::IPluginCreator](#)
Plugin creator class for user implemented layers.
- class [nvinfer1::IPluginRegistry](#)

Single registration point for all plugins in an application. It is used to find plugin implementations during engine deserialization. Internally, the plugin registry is considered to be a singleton so all plugins in an application are part of the same global registry. Note that the plugin registry is only supported for plugins of type *IPluginV2* and should also have a corresponding *IPluginCreator* implementation.

- struct `nvinfer1::impl::EnumMaxImpl< AllocatorFlag >`
Maximum number of elements in `AllocatorFlag` enum.
- class `nvinfer1::IGpuAllocator`
Application-implemented class for controlling allocation on the GPU.
- class `nvinfer1::ILogger`
Application-implemented logging interface for the builder, refitter and runtime.
- struct `nvinfer1::impl::EnumMaxImpl< ILogger::Severity >`
Maximum number of elements in `ILogger::Severity` enum.
- struct `nvinfer1::impl::EnumMaxImpl< ErrorCode >`
Maximum number of elements in `ErrorCode` enum.
- class `nvinfer1::IErrorRecorder`
Reference counted application-implemented error reporting interface for `TensorRT` objects.

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`
`AsciiChar` is the type used by `TensorRT` to represent valid ASCII characters.
- using `nvinfer1::Dims` = `Dims32`
- using `nvinfer1::PluginFormat` = `TensorFormat`
`PluginFormat` is reserved for backward compatibility.
- using `nvinfer1::AllocatorFlags` = `uint32_t`

Enumerations

- enum class `nvinfer1::DataType` : `int32_t` {
`nvinfer1::kFLOAT` = 0 , `nvinfer1::kHALF` = 1 , `nvinfer1::kINT8` = 2 , `nvinfer1::kINT32` = 3 ,
`nvinfer1::kBOOL` = 4 }
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::kDHW8` = 6 , `nvinfer1::kCDHW32` = 7 ,
`nvinfer1::kHWC` = 8 , `nvinfer1::kDLA_LINEAR` = 9 , `nvinfer1::kDLA_HWC4` = 10 , `nvinfer1::kHWC16` = 11 }
Format of the input/output tensors.
- enum class `nvinfer1::PluginVersion` : `uint8_t` { `nvinfer1::kV2` = 0 , `nvinfer1::kV2_EXT` = 1 , `nvinfer1::kV2_IOEXT` = 2 , `nvinfer1::kV2_DYNAMICEXT` = 3 }
- enum class `nvinfer1::PluginFieldType` : `int32_t` {
`nvinfer1::kFLOAT16` = 0 , `nvinfer1::kFLOAT32` = 1 , `nvinfer1::kFLOAT64` = 2 , `nvinfer1::kINT8` = 3 ,
`nvinfer1::kINT16` = 4 , `nvinfer1::kINT32` = 5 , `nvinfer1::kCHAR` = 6 , `nvinfer1::kDIMS` = 7 ,
`nvinfer1::kUNKNOWN` = 8 }
- enum class `nvinfer1::AllocatorFlag` : `int32_t` { `nvinfer1::kRESIZABLE` = 0 }
- enum class `nvinfer1::ErrorCode` : `int32_t` {
`nvinfer1::kSUCCESS` = 0 , `nvinfer1::kUNSPECIFIED_ERROR` = 1 , `nvinfer1::kINTERNAL_ERROR` = 2 ,
`nvinfer1::kINVALID_ARGUMENT` = 3 ,
`nvinfer1::kINVALID_CONFIG` = 4 , `nvinfer1::kFAILED_ALLOCATION` = 5 , `nvinfer1::kFAILED_INITIALIZATION` = 6 ,
`nvinfer1::kFAILED_EXECUTION` = 7 ,
`nvinfer1::kFAILED_COMPUTATION` = 8 , `nvinfer1::kINVALID_STATE` = 9 , `nvinfer1::kUNSUPPORTED_STATE` = 10 }
Error codes that can be returned by TensorRT during execution.

Functions

- template<typename T >
`constexpr int32_t nvinfer1::EnumMax ()` noexcept
Maximum number of elements in an enumeration type.
- `int32_t getInferLibVersion ()` noexcept
Return the library version number.

10.15.1 Detailed Description

This is the top-level API file for TensorRT core runtime library.

10.15.2 Macro Definition Documentation

10.15.2.1 NV_TENSORRT_VERSION

```
#define NV_TENSORRT_VERSION nvinfer1::kNV_TENSORRT_VERSION_IMPL
```


10.15.2.2 TENSORRTAPI

```
#define TENSORRTAPI
```

10.15.2.3 TRT_DEPRECATED

```
#define TRT_DEPRECATED __attribute__((deprecated))
```

10.15.2.4 TRT_DEPRECATED_API

```
#define TRT_DEPRECATED_API __attribute__((deprecated, visibility("default")))
```

10.15.2.5 TRT_DEPRECATED_ENUM

```
#define TRT_DEPRECATED_ENUM
```

10.15.2.6 TRTNOEXCEPT

```
#define TRTNOEXCEPT
```

10.15.3 Function Documentation

10.15.3.1 getInferLibVersion()

```
int32_t getInferLibVersion ( ) [noexcept]
```

Return the library version number.

The format is as for TENSORRT_VERSION: (TENSORRT_MAJOR * 1000) + (TENSORRT_MINOR * 100) + TENSOR_PATCH.

10.16 NvInferRuntimeCommon.h

[Go to the documentation of this file.](#)

```

1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993–2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_INFERRUNTIMECOMMON_H
14 #define NV_INFERRUNTIMECOMMON_H
15
16 #include "NvInferVersion.h"
17 #include <cstddef>
18 #include <stdint>
19 #include <cuda.runtime.api.h>
20
21 // Items that are marked as deprecated will be removed in a future release.
22 #if __cplusplus >= 201402L
23 #define TRT_DEPRECATED [[deprecated]]
24 #if __GNUC__ < 6
25 #define TRT_DEPRECATED_ENUM
26 #else
27 #define TRT_DEPRECATED_ENUM TRT_DEPRECATED
28 #endif
29 #ifdef _MSC_VER
30 #define TRT_DEPRECATED_API __declspec(dllexport)
31 #else
32 #define TRT_DEPRECATED_API [[deprecated]] __attribute__((visibility("default")))
33 #endif
34 #else
35 #ifdef _MSC_VER
36 #define TRT_DEPRECATED
37 #define TRT_DEPRECATED_ENUM
38 #define TRT_DEPRECATED_API __declspec(dllexport)
39 #else
40 #define TRT_DEPRECATED __attribute__((deprecated))
41 #define TRT_DEPRECATED_ENUM
42 #define TRT_DEPRECATED_API __attribute__((deprecated, visibility("default")))
43 #endif
44 #endif
45
46 // Defines which symbols are exported
47 #ifdef TENSORRT_BUILD_LIB
48 #ifdef _MSC_VER
49 #define TENSORRTAPI __declspec(dllexport)
50 #else
51 #define TENSORRTAPI __attribute__((visibility("default")))
52 #endif
53 #else
54 #define TENSORRTAPI
55 #endif
56 #define TRTNOEXCEPT
57
58 // forward declare some CUDA types to avoid an include dependency
59
60 extern "C"
61 {
62     struct cublasContext;
63     struct cudnnContext;
64 }
65
66 #define NV_TENSORRT_VERSION nvinfer1::kNV_TENSORRT_VERSION_IMPL
67 namespace nvinfer1
68 {
69     static constexpr int32_t kNV_TENSORRT_VERSION_IMPL
70     = (NV_TENSORRT_MAJOR * 1000) + (NV_TENSORRT_MINOR * 100) + NV_TENSORRT_PATCH; // major, minor, patch
71
72     using char_t = char;
73     using AsciiChar = char_t;
74
75     class IErrorRecorder;
76
77 
```

```

93 class IGpuAllocator;
94
95 namespace impl
96 {
97     template <typename T>
98     struct EnumMaxImpl;
99 } // namespace impl
100
101
102 template <typename T>
103 constexpr int32_t EnumMax() noexcept
104 {
105     return impl::EnumMaxImpl<T>::kVALUE;
106 }
107
108
109 enum class DataType : int32_t
110 {
111     kFLOAT = 0,
112     kHALF = 1,
113     kINT8 = 2,
114     kINT32 = 3,
115     kBOOL = 4
116 };
117
118 namespace impl
119 {
120     template <>
121     struct EnumMaxImpl<DataType>
122     {
123         // Declaration of kVALUE that represents maximum number of elements in DataType enum
124         static constexpr int32_t kVALUE = 5;
125     };
126 } // namespace impl
127
128 class Dims32
129 {
130 public:
131     static constexpr int32_t MAX_DIMS{8};
132     int32_t nbDims;
133     int32_t d[MAX_DIMS];
134 };
135
136 using Dims = Dims32;
137
138 enum class TensorFormat : int32_t
139 {
140     kLINEAR = 0,
141     kCHW2 = 1,
142     kHWC8 = 2,
143     kCHW4 = 3,
144     kCHW16 = 4,
145     kCHW32 = 5,
146     kDHWC8 = 6,
147     kCDHW32 = 7,
148     kHWC = 8,
149     kDLA.LINEAR = 9,
150     kDLA.HWC4 = 10,
151     kHWC16 = 11
152 };
153
154 using PluginFormat = TensorFormat;
155
156 namespace impl
157 {
158     template <>
159     struct EnumMaxImpl<TensorFormat>
160     {

```

```

317     static constexpr int32_t kVALUE = 12;
318 };
319 } // namespace impl
320
321 struct PluginTensorDesc
322 {
323     Dims dims;
324     DataType type;
325     TensorFormat format;
326     float scale;
327 };
328
329 enum class PluginVersion : uint8_t
330 {
331     kV2 = 0,
332     kV2_EXT = 1,
333     kV2_IOEXT = 2,
334     kV2_DYNAMICEXT = 3,
335 };
336
337 class IPluginV2
338 {
339 public:
340     virtual int32_t getTensorRTVersion() const noexcept
341     {
342         return NV_TENSORRT_VERSION;
343     }
344
345     virtual AsciiChar const* getPluginType() const noexcept = 0;
346
347     virtual AsciiChar const* getPluginVersion() const noexcept = 0;
348
349     virtual int32_t getNbOutputs() const noexcept = 0;
350
351     virtual Dims getOutputDimensions(int32_t index, Dims const* inputs, int32_t nbInputDims) noexcept = 0;
352
353     virtual bool supportsFormat(DataType type, PluginFormat format) const noexcept = 0;
354
355     virtual void configureWithFormat(Dims const* inputDims, int32_t nbInputs, Dims const* outputDims, int32_t
nbOutputs,
356         DataType type, PluginFormat format, int32_t maxBatchSize) noexcept
357         = 0;
358
359     virtual int32_t initialize() noexcept = 0;
360
361     virtual void terminate() noexcept = 0;
362
363     virtual size_t getWorkspaceSize(int32_t maxBatchSize) const noexcept = 0;
364
365     virtual int32_t enqueue(int32_t batchSize, void const* const* inputs, void* const* outputs, void*
workspace,
366         cudaStream_t stream) noexcept
367         = 0;
368
369     virtual size_t getSerializationSize() const noexcept = 0;
370
371     virtual void serialize(void* buffer) const noexcept = 0;
372
373     virtual void destroy() noexcept = 0;
374
375     virtual IPluginV2* clone() const noexcept = 0;
376
377     virtual void setPluginNamespace(AsciiChar const* pluginNamespace) noexcept = 0;
378
379     virtual AsciiChar const* getPluginNamespace() const noexcept = 0;
380
381     // @cond SuppressDoxyWarnings
382     IPluginV2() = default;
383     virtual ~IPluginV2() noexcept = default;
384 // @endcond
385
386 protected:
387 // @cond SuppressDoxyWarnings
388     IPluginV2(IPluginV2 const&) = default;
389     IPluginV2(IPluginV2&&) = default;
390     IPluginV2& operator=(IPluginV2 const&) & = default;
391     IPluginV2& operator=(IPluginV2&&) & = default;
392 // @endcond
393 };
394
395 class IPluginV2Ext : public IPluginV2

```

```

680 {
681 public:
697     virtual nvinfer1::DataType getOutputDataType(
698         int32_t index, nvinfer1::DataType const* inputTypes, int32_t nbInputs) const noexcept
699         = 0;
700
716     virtual bool isOutputBroadcastAcrossBatch(
717         int32_t outputIndex, bool const* inputIsBroadcasted, int32_t nbInputs) const noexcept
718         = 0;
719
738     virtual bool canBroadcastInputAcrossBatch(int32_t inputIndex) const noexcept = 0;
739
774     virtual void configurePlugin(Dims const* inputDims, int32_t nbInputs, Dims const* outputDims, int32_t
nbOutputs,
775         DataType const* inputTypes, DataType const* outputTypes, bool const* inputIsBroadcast,
776         bool const* outputIsBroadcast, PluginFormat floatFormat, int32_t maxBatchSize) noexcept
777         = 0;
778
779     IPluginV2Ext() = default;
780     ~IPluginV2Ext() override = default;
781
805     virtual void attachToContext(
806         cudnnContext* /*cudnn*/, cublasContext* /*cublas*/, IAllocator* /*allocator*/) noexcept
807     {
808     }
809
823     virtual void detachFromContext() noexcept {}
824
837     IPluginV2Ext* clone() const noexcept override = 0;
838
839 protected:
840     // @cond SuppressDoxyWarnings
841     IPluginV2Ext(IPluginV2Ext const&) = default;
842     IPluginV2Ext(IPluginV2Ext&&) = default;
843     IPluginV2Ext& operator=(IPluginV2Ext const&) & = default;
844     IPluginV2Ext& operator=(IPluginV2Ext&&) & = default;
845 // @endcond
846
858     int32_t getTensorRTVersion() const noexcept override
859     {
860         return static_cast<int32_t>((static_cast<uint32_t>(PluginVersion::kV2_EXT) << 24U)
861             | (static_cast<uint32_t>(NV_TENSORRT_VERSION) & 0xFFFFFU));
862     }
863
867     void configureWithFormat(Dims const* /*inputDims*/, int32_t /*nbInputs*/, Dims const* /*outputDims*/,
868         int32_t /*nbOutputs*/, DataType /*type*/, PluginFormat /*format*/, int32_t /*maxBatchSize*/) noexcept
override
869     {
870     }
871 };
872
882 class IPluginV2IOExt : public IPluginV2Ext
883 {
884 public:
902     virtual void configurePlugin(
903         PluginTensorDesc const* in, int32_t nbInput, PluginTensorDesc const* out, int32_t nbOutput) noexcept
904         = 0;
905
943     virtual bool supportsFormatCombination(
944         int32_t pos, PluginTensorDesc const* inOut, int32_t nbInputs, int32_t nbOutputs) const noexcept
945         = 0;
946
947     // @cond SuppressDoxyWarnings
948     IPluginV2IOExt() = default;
949     ~IPluginV2IOExt() override = default;
950 // @endcond
951
952 protected:
953 // @cond SuppressDoxyWarnings
954     IPluginV2IOExt(IPluginV2IOExt const&) = default;
955     IPluginV2IOExt(IPluginV2IOExt&&) = default;
956     IPluginV2IOExt& operator=(IPluginV2IOExt const&) & = default;
957     IPluginV2IOExt& operator=(IPluginV2IOExt&&) & = default;
958 // @endcond
959
971     int32_t getTensorRTVersion() const noexcept override
972     {
973         return static_cast<int32_t>((static_cast<uint32_t>(PluginVersion::kV2_IOEXT) << 24U)
974             | (static_cast<uint32_t>(NV_TENSORRT_VERSION) & 0xFFFFFU));
975     }
976

```

```

977 private:
978     // Following are obsolete base class methods, and must not be implemented or used.
979
980     void configurePlugin(Dims const*, int32_t, Dims const*, int32_t, DataType const*, DataType const*, bool
const*,
981         bool const*, PluginFormat, int32_t) noexcept final
982     {
983     }
984
985     bool supportsFormat(DataType, PluginFormat) const noexcept final
986     {
987         return false;
988     }
989 };
990
991
992
993
994
995
996 enum class PluginFieldType : int32_t
997 {
998     kFLOAT16 = 0,
999     kFLOAT32 = 1,
1000     kFLOAT64 = 2,
1001     kINT8 = 3,
1002     kINT16 = 4,
1003     kINT32 = 5,
1004     kCHAR = 6,
1005     kDIMS = 7,
1006     kUNKNOWN = 8
1007 };
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025 class PluginField
1026 {
1027 public:
1028     AsciiChar const* name;
1029     void const* data;
1030     PluginFieldType type;
1031     int32_t length;
1032
1033     PluginField(AsciiChar const* const name_ = nullptr, void const* const data_ = nullptr,
1034         PluginFieldType const type_ = PluginFieldType::kUNKNOWN, int32_t const length_ = 0) noexcept
1035         : name(name_)
1036         , data(data_)
1037         , type(type_)
1038         , length(length_)
1039     {
1040     }
1041 };
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057 struct PluginFieldCollection
1058 {
1059     int32_t nbFields;
1060     PluginField const* fields;
1061 };
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073 class IPluginCreator
1074 {
1075 public:
1076     virtual int32_t getTensorRTVersion() const noexcept
1077     {
1078         return NV_TENSORRT_VERSION;
1079     }
1080
1081     virtual AsciiChar const* getPluginName() const noexcept = 0;
1082
1083     virtual AsciiChar const* getPluginVersion() const noexcept = 0;
1084
1085     virtual PluginFieldCollection const* getFieldNames() noexcept = 0;
1086
1087     virtual IPluginV2* createPlugin(AsciiChar const* name, PluginFieldCollection const* fc) noexcept = 0;
1088
1089     virtual IPluginV2* deserializePlugin(AsciiChar const* name, void const* serialData, size_t
serialLength) noexcept
1090     = 0;
1091
1092     virtual void setPluginNamespace(AsciiChar const* pluginNamespace) noexcept = 0;
1093
1094     virtual AsciiChar const* getPluginNamespace() const noexcept = 0;
1095
1096     IPluginCreator() = default;
1097     virtual ~IPluginCreator() = default;
1098 };

```

```

1182 protected:
1183 // @cond SuppressDoxyWarnings
1184     IPluginCreator(IPluginCreator const&) = default;
1185     IPluginCreator(IPluginCreator&&) = default;
1186     IPluginCreator& operator=(IPluginCreator const&) & = default;
1187     IPluginCreator& operator=(IPluginCreator&&) & = default;
1188 // @endcond
1189 };
1190
1208
1209 class IPluginRegistry
1210 {
1211 public:
1223     virtual bool registerCreator(IPluginCreator& creator, AsciiChar const* const pluginNamespace) noexcept
1224         = 0;
1233     virtual IPluginCreator* const* getPluginCreatorList(int32_t* const numCreators) const noexcept = 0;
1234
1246     virtual IPluginCreator* getPluginCreator(AsciiChar const* const pluginName, AsciiChar const* const
1247 pluginVersion,
1248     AsciiChar const* const pluginNamespace = "") noexcept
1249         = 0;
1250     // @cond SuppressDoxyWarnings
1251     IPluginRegistry() = default;
1252     IPluginRegistry(IPluginRegistry const&) = delete;
1253     IPluginRegistry(IPluginRegistry&&) = delete;
1254     IPluginRegistry& operator=(IPluginRegistry const&) & = delete;
1255     IPluginRegistry& operator=(IPluginRegistry&&) & = delete;
1256 // @endcond
1257
1258 protected:
1259     virtual ~IPluginRegistry() noexcept = default;
1260
1261 public:
1271     //
1278     virtual void setErrorRecorder(IErrorRecorder* const recorder) noexcept = 0;
1279
1295     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
1296
1312     virtual bool deregisterCreator(IPluginCreator const& creator) noexcept = 0;
1313 };
1314
1315 enum class AllocatorFlag : int32_t
1316 {
1317     kRESIZABLE = 0,
1318 };
1319
1320 namespace impl
1321 {
1322     template <>
1323     struct EnumMaxImpl<AllocatorFlag>
1324     {
1325     {
1326         static constexpr int32_t kVALUE = 1;
1327     };
1328 } // namespace impl
1329
1330 using AllocatorFlags = uint32_t;
1331
1337 class IGPUAllocator
1338 {
1339 public:
1361     virtual void* allocate(uint64_t const size, uint64_t const alignment, AllocatorFlags const flags)
1362         noexcept = 0;
1381     TRT_DEPRECATED virtual void free(void* const memory) noexcept = 0;
1382
1387     virtual ~IGPUAllocator() = default;
1388     IGPUAllocator() = default;
1389
1423     virtual void* reallocate(void* /*baseAddr*/, uint64_t /*alignment*/, uint64_t /*newSize*/) noexcept
1424     {
1425         return nullptr;
1426     }
1427
1448     virtual bool deallocate(void* const memory) noexcept
1449     {
1450         this->free(memory);
1451         return true;
1452     }
1453

```

```

1454 protected:
1455 // @cond SuppressDoxyWarnings
1456     I_gpuAllocator(I_gpuAllocator const&) = default;
1457     I_gpuAllocator(I_gpuAllocator&&) = default;
1458     I_gpuAllocator& operator=(I_gpuAllocator const&) & = default;
1459     I_gpuAllocator& operator=(I_gpuAllocator&&) & = default;
1460 // @endcond
1461 };
1462
1463 class ILogger
1464 {
1465 public:
1466     enum class Severity : int32_t
1467     {
1468         kINTERNAL_ERROR = 0,
1469         kERROR = 1,
1470         kWARNING = 2,
1471         kINFO = 3,
1472         kVERBOSE = 4,
1473     };
1474
1475     virtual void log(Severity severity, AsciiChar const* msg) noexcept = 0;
1476
1477     ILogger() = default;
1478     virtual ~ILogger() = default;
1479
1480 protected:
1481 // @cond SuppressDoxyWarnings
1482     ILogger(ILogger const&) = default;
1483     ILogger(ILogger&&) = default;
1484     ILogger& operator=(ILogger const&) & = default;
1485     ILogger& operator=(ILogger&&) & = default;
1486 // @endcond
1487 };
1488
1489 namespace impl
1490 {
1491     template <>
1492     struct EnumMaxImpl<ILogger::Severity>
1493     {
1494         static constexpr int32_t kVALUE = 5;
1495     };
1496 } // namespace impl
1497
1498 enum class ErrorCode : int32_t
1499 {
1500     kSUCCESS = 0,
1501
1502     kUNSPECIFIED_ERROR = 1,
1503
1504     kINTERNAL_ERROR = 2,
1505
1506     kINVALID_ARGUMENT = 3,
1507
1508     kINVALID_CONFIG = 4,
1509
1510     kFAILED_ALLOCATION = 5,
1511
1512     kFAILED_INITIALIZATION = 6,
1513
1514     kFAILED_EXECUTION = 7,
1515
1516     kFAILED_COMPUTATION = 8,
1517
1518     kINVALID_STATE = 9,
1519
1520     kUNSUPPORTED_STATE = 10,
1521 };
1522
1523 namespace impl
1524 {
1525     template <>
1526     struct EnumMaxImpl<ErrorCode>
1527     {
1528         static constexpr int32_t kVALUE = 11;
1529     };
1530 } // namespace impl
1531
1532 class IErrorRecorder
1533 {

```



```

1666 public:
1670     using ErrorDesc = char const*;
1671
1675     static constexpr size_t kMAX_DESC_LENGTH{127U};
1676
1680     using RefCount = int32_t;
1681
1682     IErrorRecorder() = default;
1683     virtual ~IErrorRecorder() noexcept = default;
1684
1685     // Public API used to retrieve information from the error recorder.
1686
1705     virtual int32_t getNbErrors() const noexcept = 0;
1706
1724     virtual ErrorCode getErrorCode(int32_t errorIdx) const noexcept = 0;
1725
1745     virtual ErrorDesc getErrorDesc(int32_t errorIdx) const noexcept = 0;
1746
1761     virtual bool hasOverflowed() const noexcept = 0;
1762
1777     virtual void clear() noexcept = 0;
1778
1779     // API used by TensorRT to report Error information to the application.
1780
1801     virtual bool reportError(ErrorCode val, ErrorDesc desc) noexcept = 0;
1802
1819     virtual RefCount incRefCount() noexcept = 0;
1820
1837     virtual RefCount decRefCount() noexcept = 0;
1838
1839 protected:
1840     // @cond SuppressDoxyWarnings
1841     IErrorRecorder(IErrorRecorder const&) = default;
1842     IErrorRecorder(IErrorRecorder&&) = default;
1843     IErrorRecorder& operator=(IErrorRecorder const&) & = default;
1844     IErrorRecorder& operator=(IErrorRecorder&&) & = default;
1845     // @endcond
1846 }; // class IErrorRecorder
1847 } // namespace nvinfer1
1848
1854 extern "C" TENSORRTAPI int32_t getInferLibVersion() noexcept;
1855
1856 #endif // NV_INFER_RUNTIME_COMMON_H

```

10.17 NvInferSafeRuntime.h File Reference

```

#include "NvInferRuntimeCommon.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::FloatingPointErrorInformation](#)
Space to record information about floating point runtime errors.
- class [nvinfer1::safe::IExecutionContext](#)
Functionally safe context for executing inference using an engine.
- class [nvinfer1::safe::PluginRegistrar< T >](#)
Register the plugin creator to the registry The static registry object will be instantiated when the plugin library is loaded. This static object will register all creators available in the library to the registry.

Namespaces

- namespace `nvinfer1`
The TensorRT API version 1 namespace.
- namespace `nvinfer1::safe`
The safety subset of TensorRT's API version 1 namespace.

Macros

- `#define REGISTER_SAFE_TENSORRT_PLUGIN(name) static nvinfer1::safe::PluginRegistrar<name> pluginRegistrar##name {}`

Functions

- `IRuntime * nvinfer1::safe::createInferRuntime (ILogger &logger) noexcept`
Create an instance of an `safe::IRuntime` class.
- `nvinfer1::IPluginRegistry * nvinfer1::safe::getSafePluginRegistry () noexcept`
Return the safe plugin registry.

10.17.1 Detailed Description

The functionality in this file is only supported in NVIDIA Drive(R) products.

10.17.2 Macro Definition Documentation

10.17.2.1 REGISTER_SAFE_TENSORRT_PLUGIN

```
#define REGISTER_SAFE_TENSORRT_PLUGIN(  
    name ) static nvinfer1::safe::PluginRegistrar<name> pluginRegistrar##name {}
```

10.18 NvInferSafeRuntime.h

[Go to the documentation of this file.](#)

```

1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993–2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_INFERSAFE_RUNTIME_H
14 #define NV_INFERSAFE_RUNTIME_H
15
16 #include "NvInferRuntimeCommon.h"
17 #include <cstdlib>
18 #include <stdint>
19
20
21
22
23
24
25
26
27
28
29
30 namespace nvinfer1
31 {
32     namespace safe
33     {
34         class ICudaEngine;
35         class IExecutionContext;
36
37         class IRuntime
38         {
39         public:
40             virtual ICudaEngine* deserializeCudaEngine(void const* const blob, std::size_t const size) noexcept = 0;
41
42             virtual void setGpuAllocator(IGpuAllocator* const allocator) noexcept = 0;
43
44             //
45             virtual void setErrorRecorder(IErrorRecorder* const recorder) noexcept = 0;
46
47             virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
48
49             IRuntime() = default;
50             virtual ~IRuntime() noexcept = default;
51             IRuntime(IRuntime const&) = delete;
52             IRuntime(IRuntime&&) = delete;
53             IRuntime& operator=(IRuntime const&) & = delete;
54             IRuntime& operator=(IRuntime&&) & = delete;
55         };
56
57         class ICudaEngine
58         {
59         public:
60             virtual std::int32_t getNbBindings() const noexcept = 0;
61
62             virtual std::int32_t getBindingIndex(AsciiChar const* const name) const noexcept = 0;
63
64             virtual AsciiChar const* getBindingName(std::int32_t const bindingIndex) const noexcept = 0;
65
66             virtual bool bindingIsInput(std::int32_t const bindingIndex) const noexcept = 0;
67
68             virtual Dims getBindingDimensions(std::int32_t const bindingIndex) const noexcept = 0;
69
70             virtual DataType getBindingDataType(std::int32_t const bindingIndex) const noexcept = 0;
71
72             virtual IExecutionContext* createExecutionContext() noexcept = 0;
73
74             virtual IExecutionContext* createExecutionContextWithoutDeviceMemory() noexcept = 0;
75
76             virtual size_t getDeviceMemorySize() const noexcept = 0;
77
78             virtual std::int32_t getBindingBytesPerComponent(std::int32_t const bindingIndex) const noexcept = 0;
79
80             virtual std::int32_t getBindingComponentsPerElement(std::int32_t const bindingIndex) const noexcept = 0;
81
82             virtual TensorFormat getBindingFormat(std::int32_t const bindingIndex) const noexcept = 0;
83
84             virtual std::int32_t getBindingVectorizedDim(std::int32_t const bindingIndex) const noexcept = 0;
85
86         };
87     }
88 }

```

```

343     virtual AsciiChar const* getName() const noexcept = 0;
344
354     //
361     virtual void setErrorRecorder(IErrorRecorder* const recorder) noexcept = 0;
362
378     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
379
380     ICudaEngine() = default;
381     virtual ~ICudaEngine() noexcept = default;
382     ICudaEngine(ICudaEngine const&) = delete;
383     ICudaEngine(ICudaEngine&&) = delete;
384     ICudaEngine& operator=(ICudaEngine const&) & = delete;
385     ICudaEngine& operator=(ICudaEngine&&) & = delete;
386 };
387
394 struct FloatingPointErrorInformation
395 {
397     int32_t nbNanErrors;
399     int32_t nbInfErrors;
400 };
401
415 class IExecutionContext
416 {
417 public:
427     virtual ICudaEngine const& getEngine() const noexcept = 0;
428
443     virtual void setName(AsciiChar const* const name) noexcept = 0;
444
454     virtual AsciiChar const* getName() const noexcept = 0;
455
471     virtual void setDeviceMemory(void* const memory) noexcept = 0;
472
482     virtual Dims getStrides(std::int32_t const bindingIndex) const noexcept = 0;
483
493     //
500     virtual void setErrorRecorder(IErrorRecorder* const recorder) noexcept = 0;
501
516     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
517
537     virtual bool enqueueV2(
538         void* const* const bindings, cudaStream_t const stream, cudaEvent_t* const inputConsumed) noexcept
539         = 0;
540
541     IExecutionContext() = default;
542     virtual ~IExecutionContext() noexcept = default;
543     IExecutionContext(IExecutionContext const&) = delete;
544     IExecutionContext(IExecutionContext&&) = delete;
545     IExecutionContext& operator=(IExecutionContext const&) & = delete;
546     IExecutionContext& operator=(IExecutionContext&&) & = delete;
547
567     virtual void setErrorBuffer(FloatingPointErrorInformation* const buffer) noexcept = 0;
568
580     virtual FloatingPointErrorInformation* getErrorBuffer() const noexcept = 0;
581 };
582
592 IRuntime* createInferRuntime(ILogger& logger) noexcept;
593
601 extern "C" TENSORRTAPI nvinfer1::IPluginRegistry* getSafePluginRegistry() noexcept;
602
614 template <typename T>
615 class PluginRegistrar
616 {
617 public:
618     PluginRegistrar()
619     {
620         getSafePluginRegistry()->registerCreator(instance, "");
621     }
622
623 private:
624     T instance{};
625 };
626 // namespace safe
627 // namespace nvinfer1
628 #define REGISTER_SAFE_TENSORRT_PLUGIN(name)
629     static nvinfer1::safe::PluginRegistrar<name> pluginRegistrar##name {}
630 #endif // NV.INFER.SAFE.RUNTIME.H

```

10.19 NvInferVersion.h File Reference

Macros

- #define `NV_TENSORRT_MAJOR` 8
TensorRT major version.
- #define `NV_TENSORRT_MINOR` 4
TensorRT minor version.
- #define `NV_TENSORRT_PATCH` 11
TensorRT patch version.
- #define `NV_TENSORRT_BUILD` 4
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` 4
Shared object library minor version number.
- #define `NV_TENSORRT_SONAME_PATCH` 11
Shared object library patch version number.

10.19.1 Detailed Description

Defines the TensorRT version

10.19.2 Macro Definition Documentation

10.19.2.1 NV_TENSORRT_BUILD

```
#define NV_TENSORRT_BUILD 4
```

TensorRT build number.

10.19.2.2 NV_TENSORRT_LWS_MAJOR

```
#define NV_TENSORRT_LWS_MAJOR 0
```

TensorRT LWS major version.

10.19.2.3 NV_TENSORRT_LWS_MINOR

```
#define NV_TENSORRT_LWS_MINOR 0
```

TensorRT LWS minor version.

10.19.2.4 NV_TENSORRT_LWS_PATCH

```
#define NV_TENSORRT_LWS_PATCH 0
```

TensorRT LWS patch version.

10.19.2.5 NV_TENSORRT_MAJOR

```
#define NV_TENSORRT_MAJOR 8
```

TensorRT major version.

10.19.2.6 NV_TENSORRT_MINOR

```
#define NV_TENSORRT_MINOR 4
```

TensorRT minor version.

10.19.2.7 NV_TENSORRT_PATCH

```
#define NV_TENSORRT_PATCH 11
```

TensorRT patch version.

10.19.2.8 NV_TENSORRT_SONAME_MAJOR

```
#define NV_TENSORRT_SONAME_MAJOR 8
```

Shared object library major version number.

10.19.2.9 NV_TENSORRT_SONAME_MINOR

```
#define NV_TENSORRT_SONAME_MINOR 4
```

Shared object library minor version number.

10.19.2.10 NV_TENSORRT_SONAME_PATCH

```
#define NV_TENSORRT_SONAME_PATCH 11
```

Shared object library patch version number.

10.20 NvInferVersion.h

[Go to the documentation of this file.](#)

```

1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13
14
15
16
17
18
19 #ifndef NV_INFERENCE_VERSION_H
20 #define NV_INFERENCE_VERSION_H
21
22 #define NV_TENSORRT_MAJOR 8
23 #define NV_TENSORRT_MINOR 4
24 #define NV_TENSORRT_PATCH 11
25 #define NV_TENSORRT_BUILD 4
26
27 #define NV_TENSORRT_LWS_MAJOR 0
28 #define NV_TENSORRT_LWS_MINOR 0
29 #define NV_TENSORRT_LWS_PATCH 0
30
31 #define NV_TENSORRT_SONAME_MAJOR 8
32 #define NV_TENSORRT_SONAME_MINOR 4
33 #define NV_TENSORRT_SONAME_PATCH 11
34
35 #endif // NV_INFERENCE_VERSION_H

```

10.21 NvOnnxConfig.h File Reference

```
#include "NvInfer.h"
```

Classes

- class [nvonnxparser::IOnnxConfig](#)
Configuration Manager Class.

Namespaces

- namespace [nvonnxparser](#)
The TensorRT ONNX parser API namespace.

Functions

- [IOnnxConfig * nvonnxparser::createONNXConfig \(\)](#)

10.21.1 Detailed Description

This is the API file for the Configuration Manager for ONNX Parser for Nvidia TensorRT.

10.22 NvOnnxConfig.h

[Go to the documentation of this file.](#)

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993–2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_OnnxConfig_H
14 #define NV_OnnxConfig_H
15
16 #include "NvInfer.h"
17
18 namespace nvonnxparser
19 {
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41 class IOnnxConfig
42 {
43 public:
44     virtual ~IOnnxConfig() noexcept = default;
45     typedef int32_t Verbosity;
46
47
48
49
50
```



```

59     virtual void setModelDtype(const nvinfer1::DataType) noexcept = 0;
60
68     virtual nvinfer1::DataType getModelDtype() const noexcept = 0;
69
77     virtual char const* getModelFileName() const noexcept = 0;
78
90     virtual void setModelFileName(char const* onnxFilename) noexcept = 0;
91
99     virtual Verbosity getVerbosityLevel() const noexcept = 0;
100
108    virtual void addVerbosity() noexcept = 0;
109
115    virtual void reduceVerbosity() noexcept = 0;
116
122    virtual void setVerbosityLevel(Verbosity) noexcept = 0;
123
132    virtual char const* getTextFileName() const noexcept = 0;
133
145    virtual void setTextFileName(char const* textFileName) noexcept = 0;
146
155    virtual char const* getFullTextFileName() const noexcept = 0;
156
168    virtual void setFullTextFileName(char const* fullTextFileName) noexcept = 0;
169
177    virtual bool getPrintLayerInfo() const noexcept = 0;
178
184    virtual void setPrintLayerInfo(bool) noexcept = 0;
185
193    TRT_DEPRECATED virtual void destroy() noexcept = 0;
194
195 }; // class IOnnxConfig
196
197 TENSORRTAPI IOnnxConfig* createONNXConfig();
198
199 } // namespace nvonnxparser
200
201 #endif

```

10.23 NvUffParser.h File Reference

```
#include "NvInfer.h"
```

Classes

- class [nvuffparser::FieldMap](#)
An array of field params used as a layer parameter for plugin layers.
- struct [nvuffparser::FieldCollection](#)
- class [nvuffparser::IUffParser](#)
Class used for parsing models described using the UFF format.

Namespaces

- namespace [nvuffparser](#)
The TensorRT UFF parser API namespace.

Macros

- `#define` [UFF_REQUIRED_VERSION_MAJOR](#) 0
- `#define` [UFF_REQUIRED_VERSION_MINOR](#) 6
- `#define` [UFF_REQUIRED_VERSION_PATCH](#) 9

Enumerations

- enum class `nvuffparser::UffInputOrder` : `int32_t` { `nvuffparser::kNCHW` = 0 , `nvuffparser::kNHWC` = 1 , `nvuffparser::kNC` = 2 }

The different possible supported input order.

- enum class `nvuffparser::FieldType` : `int32_t` { `nvuffparser::kFLOAT` = 0 , `nvuffparser::kINT32` = 1 , `nvuffparser::kCHAR` = 2 , `nvuffparser::kDIMS` = 4 , `nvuffparser::kDATATYPE` = 5 , `nvuffparser::kUNKNOWN` = 6 }

The possible field types for custom layer.

Functions

- `IUffParser * nvuffparser::createUffParser ()` noexcept
Creates a `IUffParser` object.
- `void nvuffparser::shutdownProtobufLibrary (void)` noexcept
Shuts down protocol buffers library.

10.23.1 Detailed Description

This is the API for the UFF Parser

10.23.2 Macro Definition Documentation

10.23.2.1 UFF_REQUIRED_VERSION_MAJOR

```
#define UFF_REQUIRED_VERSION_MAJOR 0
```

10.23.2.2 UFF_REQUIRED_VERSION_MINOR

```
#define UFF_REQUIRED_VERSION_MINOR 6
```

10.23.2.3 UFF_REQUIRED_VERSION_PATCH

```
#define UFF_REQUIRED_VERSION_PATCH 9
```

10.24 NvUffParser.h

[Go to the documentation of this file.](#)

```

1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993–2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_UFF_PARSER_H
14 #define NV_UFF_PARSER_H
15
16 #include "NvInfer.h"
17
18
19
20
21
22
23
24 // Current supported Universal Framework Format (UFF) version for the parser.
25 #define UFF_REQUIRED_VERSION_MAJOR 0
26 #define UFF_REQUIRED_VERSION_MINOR 6
27 #define UFF_REQUIRED_VERSION_PATCH 9
28
29
30 namespace nvuffparser
31 {
32
33
34 enum class UffInputOrder : int32_t
35 {
36     kNCHW = 0,
37     kNHWC = 1,
38     kNC = 2
39 };
40
41 enum class FieldType : int32_t
42 {
43     kFLOAT = 0,
44     kINT32 = 1,
45     kCHAR = 2,
46     kDIMS = 4,
47     kDATATYPE = 5,
48     kUNKNOWN = 6
49 };
50
51 class TENSORRTAPI FieldMap
52 {
53 public:
54     char const* name;
55     void const* data;
56     FieldType type = FieldType::kUNKNOWN;
57     int32_t length = 1;
58
59     FieldMap(char const* name, void const* data, const FieldType type, int32_t length = 1);
60 };
61
62 struct FieldCollection
63 {
64     int32_t nbFields;
65     FieldMap const* fields;
66 };
67
68 class IUffParser
69 {
70 public:
71     virtual bool registerInput(char const* inputName, nvinfer1::Dims inputDims, UffInputOrder inputOrder)
72         noexcept = 0;
73
74     virtual bool registerOutput(char const* outputName) noexcept = 0;
75
76     virtual bool parse(char const* file, nvinfer1::INetworkDefinition& network,
77         nvinfer1::DataType weightsType = nvinfer1::DataType::kFLOAT) noexcept = 0;
78
79     virtual bool parseBuffer(char const* buffer, std::size_t size, nvinfer1::INetworkDefinition& network,
80         nvinfer1::DataType weightsType = nvinfer1::DataType::kFLOAT) noexcept = 0;
81
82     TRT_DEPRECATED virtual void destroy() noexcept = 0;
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139

```

```

140
144     virtual int32_t  getUffRequiredVersionMajor() noexcept = 0;
145
149     virtual int32_t  getUffRequiredVersionMinor() noexcept = 0;
150
154     virtual int32_t  getUffRequiredVersionPatch() noexcept = 0;
155
159     virtual void    setPluginNamespace(char const* libNamespace) noexcept = 0;
160
161     virtual ~IUffParser() noexcept = default;
162
163 public:
164     //
165     virtual void    setErrorRecorder(nvinfer1::IErrorRecorder* recorder) noexcept = 0;
166
167     virtual nvinfer1::IErrorRecorder* getErrorRecorder() const noexcept = 0;
168 };
169
170 TENSORRTAPI IUffParser* createUffParser() noexcept;
171
172 TENSORRTAPI void shutdownProtobufLibrary(void) noexcept;
173 } // namespace nvuffparser
174
175 extern "C" TENSORRTAPI void* createNvUffParser_INTERNAL() noexcept;
176
177 #endif /* !NV_UFF_PARSER_H */

```

10.25 NvUtils.h File Reference

```
#include "NvInfer.h"
```

Namespaces

- namespace [nvinfer1](#)
The TensorRT API version 1 namespace.
- namespace [nvinfer1::utils](#)

Functions

- **TRT_DEPRECATED** bool [nvinfer1::utils::reshapeWeights](#) (Weights const &input, int32_t const *shape, int32_t const *shapeOrder, void *data, int32_t nbDims) noexcept
Reformat the input weights of the given shape based on the new order of dimensions.
- **TRT_DEPRECATED** bool [nvinfer1::utils::reorderSubBuffers](#) (void *input, int32_t const *order, int32_t num, int32_t size) noexcept
Takes an input stream and re-orders num chunks of the data given the size and order.
- **TRT_DEPRECATED** bool [nvinfer1::utils::transposeSubBuffers](#) (void *input, DataType type, int32_t num, int32_t height, int32_t width) noexcept
*Transpose num sub-buffers of height * width.*

10.25.1 Detailed Description

This file includes various utility functions

10.26 NvUtils.h

[Go to the documentation of this file.](#)

```

1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2022 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: LicenseRef-NvidiaProprietary
4  *
5  * NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
6  * property and proprietary rights in and to this material, related
7  * documentation and any modifications thereto. Any use, reproduction,
8  * disclosure or distribution of this material and related documentation
9  * without an express license agreement from NVIDIA CORPORATION or
10 * its affiliates is strictly prohibited.
11 */
12
13 #ifndef NV_UTILS_H
14 #define NV_UTILS_H
15
16 #include "NvInfer.h"
17
18
19
20
21
22
23
24 namespace nvinfer1
25 {
26 namespace utils
27 {
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75 TRT_DEPRECATED TENSORRTAPI bool reshapeWeights(
76     Weights const& input, int32_t const* shape, int32_t const* shapeOrder, void* data, int32_t nbDims)
77     noexcept;
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103 TRT_DEPRECATED TENSORRTAPI bool reorderSubBuffers(
104     void* input, int32_t const* order, int32_t num, int32_t size) noexcept;
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123 TRT_DEPRECATED TENSORRTAPI bool transposeSubBuffers(
124     void* input, DataType type, int32_t num, int32_t height, int32_t width) noexcept;
125
126
127
128
129
130
131
132
133
134 } // namespace utils
135 } // namespace nvinfer1
136 #endif // NV_UTILS_H

```

10.27 NvOnnxParser.h File Reference

```

#include "NvInfer.h"
#include <stddef.h>
#include <vector>

```

Classes

- class [nvonnxparser::IParserError](#)
an object containing information about an error
- class [nvonnxparser::IParser](#)
an object for parsing ONNX models into a TensorRT network definition

Namespaces

- namespace [nvonnxparser](#)
The TensorRT ONNX parser API namespace.

Macros

- #define [NV_ONNX_PARSER_MAJOR](#) 0
- #define [NV_ONNX_PARSER_MINOR](#) 1
- #define [NV_ONNX_PARSER_PATCH](#) 0

Typedefs

- typedef std::pair< std::vector< size_t >, bool > [SubGraph_t](#)
The data structure containing the parsing capability of a set of nodes in an ONNX graph.
- typedef std::vector< [SubGraph_t](#) > [SubGraphCollection_t](#)
The data structure containing all SubGraph_t partitioned out of an ONNX graph.

Enumerations

- enum class [nvonnxparser::ErrorCode](#) : int {
[nvonnxparser::kSUCCESS](#) = 0 , [nvonnxparser::kINTERNAL_ERROR](#) = 1 , [nvonnxparser::kMEM_ALLOC_FAILED](#) = 2 , [nvonnxparser::kMODEL_DESERIALIZE_FAILED](#) = 3 ,
[nvonnxparser::kINVALID_VALUE](#) = 4 , [nvonnxparser::kINVALID_GRAPH](#) = 5 , [nvonnxparser::kINVALID_NODE](#) = 6 , [nvonnxparser::kUNSUPPORTED_GRAPH](#) = 7 ,
[nvonnxparser::kUNSUPPORTED_NODE](#) = 8 }
the type of parser error

Functions

- template<typename T >
int32_t [nvonnxparser::EnumMax](#) ()
- template<> int32_t [nvonnxparser::EnumMax](#)< [ErrorCode](#) > ()
- [TENSORRTAPI](#) void * [createNvOnnxParser_INTERNAL](#) (void *network, void *logger, int version)
- [TENSORRTAPI](#) int [getNvOnnxParserVersion](#) ()

10.27.1 Detailed Description

This is the API for the ONNX Parser

10.27.2 Macro Definition Documentation

10.27.2.1 NV_ONNX_PARSER_MAJOR

```
#define NV_ONNX_PARSER_MAJOR 0
```

10.27.2.2 NV_ONNX_PARSER_MINOR

```
#define NV_ONNX_PARSER_MINOR 1
```

10.27.2.3 NV_ONNX_PARSER_PATCH

```
#define NV_ONNX_PARSER_PATCH 0
```

10.27.3 Typedef Documentation

10.27.3.1 SubGraph_t

[SubGraph_t](#)

The data structure containing the parsing capability of a set of nodes in an ONNX graph.

10.27.3.2 SubGraphCollection_t

[SubGraphCollection_t](#)

The data structure containing all SubGraph_t partitioned out of an ONNX graph.

10.27.4 Function Documentation

10.27.4.1 createNvOnnxParser_INTERNAL()

```
TENSORRTAPI void * createNvOnnxParser_INTERNAL (  
    void * network,  
    void * logger,  
    int version )
```

10.27.4.2 getNvOnnxParserVersion()

```
TENSORRTAPI int getNvOnnxParserVersion ( )
```

10.28 NvOnnxParser.h

[Go to the documentation of this file.](#)

```
1 /*
2  * Copyright (c) 1993-2022, NVIDIA CORPORATION. All rights reserved.
3  *
4  * Permission is hereby granted, free of charge, to any person obtaining a
5  * copy of this software and associated documentation files (the "Software"),
6  * to deal in the Software without restriction, including without limitation
7  * the rights to use, copy, modify, merge, publish, distribute, sublicense,
8  * and/or sell copies of the Software, and to permit persons to whom the
9  * Software is furnished to do so, subject to the following conditions:
10 *
11 * The above copyright notice and this permission notice shall be included in
12 * all copies or substantial portions of the Software.
13 *
14 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
17 * THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
18 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
19 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
20 * DEALINGS IN THE SOFTWARE.
21 */
22
23 #ifndef NV_ONNX_PARSER_H
24 #define NV_ONNX_PARSER_H
25
26 #include "NvInfer.h"
27 #include <stddef.h>
28 #include <vector>
29
30
31
32
33
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 const int NV_ONNX_PARSER_VERSION = ((NV_ONNX_PARSER_MAJOR * 10000) + (NV_ONNX_PARSER_MINOR * 100) +
41 NV_ONNX_PARSER_PATCH);
42
43
44
45
46
47 typedef std::pair<std::vector<size_t>, bool> SubGraph_t;
48
49
50
51
52
53
54 typedef std::vector<SubGraph_t> SubGraphCollection_t;
55
56
57
58
59
60
61 namespace nvonnxparser
62 {
63
64
65
66
67
68
69
70
71 template <typename T>
72 inline int32_t EnumMax();
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```



```

94 class IParserError
95 {
96 public:
97     virtual ErrorCode code() const = 0;
102     virtual const char* desc() const = 0;
105     virtual const char* file() const = 0;
108     virtual int line() const = 0;
111     virtual const char* func() const = 0;
114     virtual int node() const = 0;
115
116 protected:
117     virtual ~IParserError() {}
118 };
119
124 class IParser
125 {
126 public:
140     virtual bool parse(void const* serialized_onnx_model,
141                       size_t serialized_onnx_model_size,
142                       const char* model_path = nullptr)
143         = 0;
144
154     virtual bool parseFromFile(const char* onnx_model_file, int verbosity) = 0;
155
167     virtual bool supportsModel(void const* serialized_onnx_model,
168                                size_t serialized_onnx_model_size,
169                                SubGraphCollection_t& sub_graph_collection,
170                                const char* model_path = nullptr)
171         = 0;
172
182     virtual bool parseWithWeightDescriptors(
183         void const* serialized_onnx_model, size_t serialized_onnx_model_size)
184         = 0;
185
194     virtual bool supportsOperator(const char* op_name) const = 0;
199     TRT_DEPRECATED virtual void destroy() = 0;
205     virtual int getNbErrors() const = 0;
210     virtual IParserError const* getError(int index) const = 0;
215     virtual void clearErrors() = 0;
216
217     virtual ~IParser() noexcept = default;
218 };
219
220 } // namespace nvonnxparser
221
222 extern "C" TENSORRTAPI void* createNvOnnxParser_INTERNAL(void* network, void* logger, int version);
223 extern "C" TENSORRTAPI int getNvOnnxParserVersion();
224
225 namespace nvonnxparser
226 {
227
228     namespace
229     {
230
245     inline IParser* createParser(nvinfer1::INetworkDefinition& network, nvinfer1::ILogger& logger)
246     {
247         return static_cast<IParser*>(createNvOnnxParser_INTERNAL(&network, &logger, NV_ONNX_PARSER_VERSION));
248     }
249
250 } // namespace
251
252 } // namespace nvonnxparser
253
254 #endif // NV_ONNX_PARSER_H

```

Index

- ~IActivationLayer
 - nvinfer1::IActivationLayer, [114](#)
- ~IAlgorithm
 - nvinfer1::IAlgorithm, [118](#)
- ~IAlgorithmContext
 - nvinfer1::IAlgorithmContext, [121](#)
- ~IAlgorithmIOInfo
 - nvinfer1::IAlgorithmIOInfo, [123](#)
- ~IAlgorithmSelector
 - nvinfer1::IAlgorithmSelector, [125](#)
- ~IAlgorithmVariant
 - nvinfer1::IAlgorithmVariant, [127](#)
- ~IAssertionLayer
 - nvinfer1::IAssertionLayer, [129](#)
- ~IBinaryProtoBlob
 - nvcaffeparser1::IBinaryProtoBlob, [131](#)
- ~IBlobNameToTensor
 - nvcaffeparser1::IBlobNameToTensor, [133](#)
- ~IBuilder
 - nvinfer1::IBuilder, [135](#)
- ~IBuilderConfig
 - nvinfer1::IBuilderConfig, [146](#)
- ~ICaffeParser
 - nvcaffeparser1::ICaffeParser, [165](#)
- ~IConcatenationLayer
 - nvinfer1::IConcatenationLayer, [170](#)
- ~IConditionLayer
 - nvinfer1::IConditionLayer, [172](#)
- ~IConsistencyChecker
 - nvinfer1::consistency::IConsistencyChecker, [173](#)
- ~IConstantLayer
 - nvinfer1::IConstantLayer, [176](#)
- ~IConvolutionLayer
 - nvinfer1::IConvolutionLayer, [180](#)
- ~ICudaEngine
 - nvinfer1::ICudaEngine, [192](#)
 - nvinfer1::safe::ICudaEngine, [207](#)
- ~IDeconvolutionLayer
 - nvinfer1::IDeconvolutionLayer, [219](#)
- ~IDequantizeLayer
 - nvinfer1::IDequantizeLayer, [230](#)
- ~IDimensionExpr
 - nvinfer1::IDimensionExpr, [232](#)
- ~IEinsumLayer
 - nvinfer1::IEinsumLayer, [235](#)
- ~IElementWiseLayer
 - nvinfer1::IElementWiseLayer, [237](#)
- ~IEngineInspector
 - nvinfer1::IEngineInspector, [239](#)
- ~IErrorRecorder
 - nvinfer1::IErrorRecorder, [244](#)
- ~IExecutionContext
 - nvinfer1::IExecutionContext, [251](#)
 - nvinfer1::safe::IExecutionContext, [267](#)
- ~IExprBuilder
 - nvinfer1::IExprBuilder, [274](#)
- ~IFillLayer
 - nvinfer1::IFillLayer, [277](#)
- ~IFullyConnectedLayer
 - nvinfer1::IFullyConnectedLayer, [283](#)
- ~IGatherLayer
 - nvinfer1::IGatherLayer, [288](#)
- ~IGpuAllocator
 - nvinfer1::IGpuAllocator, [291](#)
- ~IHostMemory
 - nvinfer1::IHostMemory, [296](#)
- ~IIDentityLayer
 - nvinfer1::IIDentityLayer, [298](#)
- ~IIIfConditional
 - nvinfer1::IIIfConditional, [300](#)
- ~IIIfConditionalBoundaryLayer
 - nvinfer1::IIIfConditionalBoundaryLayer, [303](#)
- ~IIIfConditionalInputLayer
 - nvinfer1::IIIfConditionalInputLayer, [304](#)
- ~IIIfConditionalOutputLayer
 - nvinfer1::IIIfConditionalOutputLayer, [306](#)
- ~IIInt8Calibrator
 - nvinfer1::IIInt8Calibrator, [307](#)
- ~IIInt8EntropyCalibrator
 - nvinfer1::IIInt8EntropyCalibrator, [310](#)
- ~IIInt8EntropyCalibrator2
 - nvinfer1::IIInt8EntropyCalibrator2, [311](#)
- ~IIInt8LegacyCalibrator
 - nvinfer1::IIInt8LegacyCalibrator, [312](#)
- ~IIInt8MinMaxCalibrator
 - nvinfer1::IIInt8MinMaxCalibrator, [314](#)
- ~IIteratorLayer
 - nvinfer1::IIteratorLayer, [316](#)
- ~ILRNLayer
 - nvinfer1::ILRNLayer, [336](#)

- ~ILayer
 - nvinfer1::ILayer, 319
- ~ILogger
 - nvinfer1::ILogger, 327
- ~ILoop
 - nvinfer1::ILoop, 328
- ~ILoopBoundaryLayer
 - nvinfer1::ILoopBoundaryLayer, 331
- ~ILoopOutputLayer
 - nvinfer1::ILoopOutputLayer, 333
- ~IMatrixMultiplyLayer
 - nvinfer1::IMatrixMultiplyLayer, 341
- ~INetworkDefinition
 - nvinfer1::INetworkDefinition, 346
- ~INoCopy
 - nvinfer1::INoCopy, 385
- ~IOnnxConfig
 - nvonnxparser::IOnnxConfig, 387
- ~IOptimizationProfile
 - nvinfer1::IOptimizationProfile, 394
- ~IPaddingLayer
 - nvinfer1::IPaddingLayer, 399
- ~IParametricReLULayer
 - nvinfer1::IParametricReLULayer, 403
- ~IParser
 - nvonnxparser::IParser, 405
- ~IParserError
 - nvonnxparser::IParserError, 409
- ~IPluginChecker
 - nvinfer1::consistency::IPluginChecker, 412
- ~IPluginCreator
 - nvinfer1::IPluginCreator, 414
- ~IPluginFactoryV2
 - nvcaffeparser1::IPluginFactoryV2, 419
- ~IPluginRegistry
 - nvinfer1::IPluginRegistry, 421
- ~IPluginV2DynamicExt
 - nvinfer1::IPluginV2DynamicExt, 436
- ~IPluginV2Ext
 - nvinfer1::IPluginV2Ext, 442
- ~IPluginV2Layer
 - nvinfer1::IPluginV2Layer, 451
- ~IPoolingLayer
 - nvinfer1::IPoolingLayer, 454
- ~IProfiler
 - nvinfer1::IProfiler, 462
- ~IQuantizeLayer
 - nvinfer1::IQuantizeLayer, 465
- ~IRNNv2Layer
 - nvinfer1::IRNNv2Layer, 491
- ~IRaggedSoftMaxLayer
 - nvinfer1::IRaggedSoftMaxLayer, 467
- ~IRecurrenceLayer
 - nvinfer1::IRecurrenceLayer, 469
- ~IReduceLayer
 - nvinfer1::IReduceLayer, 471
- ~IREfitter
 - nvinfer1::IREfitter, 474
- ~IResizeLayer
 - nvinfer1::IResizeLayer, 483
- ~IRuntime
 - nvinfer1::IRuntime, 499
 - nvinfer1::safe::IRuntime, 505
- ~IScaleLayer
 - nvinfer1::IScaleLayer, 510
- ~IScatterLayer
 - nvinfer1::IScatterLayer, 516
- ~ISelectLayer
 - nvinfer1::ISelectLayer, 518
- ~IShapeLayer
 - nvinfer1::IShapeLayer, 519
- ~IShuffleLayer
 - nvinfer1::IShuffleLayer, 521
- ~ISliceLayer
 - nvinfer1::ISliceLayer, 527
- ~ISoftMaxLayer
 - nvinfer1::ISoftMaxLayer, 532
- ~ITensor
 - nvinfer1::ITensor, 535
- ~ITimingCache
 - nvinfer1::ITimingCache, 544
- ~ITopKLayer
 - nvinfer1::ITopKLayer, 547
- ~ITripLimitLayer
 - nvinfer1::ITripLimitLayer, 550
- ~IUffParser
 - nvuffparser::IUffParser, 551
- ~IUnaryLayer
 - nvinfer1::IUnaryLayer, 556
- ActivationType
 - nvinfer1, 37
- addActivation
 - nvinfer1::INetworkDefinition, 346
- addAssertion
 - nvinfer1::INetworkDefinition, 347
- addConcatenation
 - nvinfer1::INetworkDefinition, 347
- addConstant
 - nvinfer1::INetworkDefinition, 348
- addConvolution
 - nvinfer1::INetworkDefinition, 348
- addConvolutionNd
 - nvinfer1::INetworkDefinition, 349
- addDeconvolution
 - nvinfer1::INetworkDefinition, 350
- addDeconvolutionNd
 - nvinfer1::INetworkDefinition, 351

- addDequantize
 - nvinfer1::INetworkDefinition, 352
- addEinsum
 - nvinfer1::INetworkDefinition, 352
- addElementWise
 - nvinfer1::INetworkDefinition, 353
- addFill
 - nvinfer1::INetworkDefinition, 353
- addFullyConnected
 - nvinfer1::INetworkDefinition, 354
- addGather
 - nvinfer1::INetworkDefinition, 355
- addGatherV2
 - nvinfer1::INetworkDefinition, 355
- addIdentity
 - nvinfer1::INetworkDefinition, 356
- addIfConditional
 - nvinfer1::INetworkDefinition, 356
- addInput
 - nvinfer1::IIfConditional, 300
 - nvinfer1::INetworkDefinition, 357
- addIterator
 - nvinfer1::ILoop, 329
- addLoop
 - nvinfer1::INetworkDefinition, 358
- addLoopOutput
 - nvinfer1::ILoop, 329
- addLRN
 - nvinfer1::INetworkDefinition, 358
- addMatrixMultiply
 - nvinfer1::INetworkDefinition, 359
- addOptimizationProfile
 - nvinfer1::IBuilderConfig, 146
- addOutput
 - nvinfer1::IIfConditional, 301
- addPadding
 - nvinfer1::INetworkDefinition, 359
- addPaddingNd
 - nvinfer1::INetworkDefinition, 360
- addParametricReLU
 - nvinfer1::INetworkDefinition, 361
- addPluginV2
 - nvinfer1::INetworkDefinition, 361
- addPooling
 - nvinfer1::INetworkDefinition, 362
- addPoolingNd
 - nvinfer1::INetworkDefinition, 363
- addQuantize
 - nvinfer1::INetworkDefinition, 363
- addRaggedSoftMax
 - nvinfer1::INetworkDefinition, 364
- addRecurrence
 - nvinfer1::ILoop, 329
- addReduce
 - nvinfer1::INetworkDefinition, 364
- addResize
 - nvinfer1::INetworkDefinition, 365
- addRNNv2
 - nvinfer1::INetworkDefinition, 366
- addScale
 - nvinfer1::INetworkDefinition, 367
- addScaleNd
 - nvinfer1::INetworkDefinition, 368
- addScatter
 - nvinfer1::INetworkDefinition, 369
- addSelect
 - nvinfer1::INetworkDefinition, 370
- addShape
 - nvinfer1::INetworkDefinition, 370
- addShuffle
 - nvinfer1::INetworkDefinition, 372
- addSlice
 - nvinfer1::INetworkDefinition, 372
- addSoftMax
 - nvinfer1::INetworkDefinition, 373
- addTopK
 - nvinfer1::INetworkDefinition, 373
- addTripLimit
 - nvinfer1::ILoop, 329
- addUnary
 - nvinfer1::INetworkDefinition, 374
- addVerbosity
 - nvonnxparser::IOnnxConfig, 387
- allInputDimensionsSpecified
 - nvinfer1::IExecutionContext, 251
- allInputShapesSpecified
 - nvinfer1::IExecutionContext, 252
- allocate
 - nvinfer1::IGpuAllocator, 292
- AllocatorFlag
 - nvinfer1, 38
- AllocatorFlags
 - nvinfer1, 35
- anchorsRatioCount
 - nvinfer1::plugin::RPROIParams, 572
- anchorsScaleCount
 - nvinfer1::plugin::RPROIParams, 572
- AsciiChar
 - nvinfer1, 35
- aspectRatios
 - nvinfer1::plugin::GridAnchorParameters, 112
 - nvinfer1::plugin::PriorBoxParameters, 567
- attachToContext
 - nvinfer1::IPluginV2Ext, 442
- backgroundLabelId
 - nvinfer1::plugin::DetectionOutputParameters, 84
 - nvinfer1::plugin::NMSPParameters, 558

- bindingIsInput
 - nvinfer1::ICudaEngine, 193
 - nvinfer1::safe::ICudaEngine, 208
- buildEngineWithConfig
 - nvinfer1::IBuilder, 135
- BuilderFlag
 - nvinfer1, 38
- BuilderFlags
 - nvinfer1, 35
- buildSerializedNetwork
 - nvinfer1::IBuilder, 136
- CalibrationAlgoType
 - nvinfer1, 39
- canBroadcastInputAcrossBatch
 - nvinfer1::IPluginV2Ext, 443
- canRunOnDLA
 - nvinfer1::IBuilderConfig, 147
- CENTER_SIZE
 - nvinfer1::plugin, 74
- char_t
 - nvinfer1, 35
- child
 - nvinfer1::plugin::softmaxTree, 574
- classes
 - nvinfer1::plugin::RegionParameters, 570
- clear
 - nvinfer1::IErrorRecorder, 245
- clearErrors
 - nvonnxparser::IParser, 405
- clearFlag
 - nvinfer1::IBuilderConfig, 147
- clearQuantizationFlag
 - nvinfer1::IBuilderConfig, 147
- clip
 - nvinfer1::plugin::PriorBoxParameters, 567
- clone
 - nvinfer1::IPluginV2, 425
 - nvinfer1::IPluginV2DynamicExt, 436
 - nvinfer1::IPluginV2Ext, 443
- code
 - nvonnxparser::IParserError, 409
- codeType
 - nvinfer1::plugin::DetectionOutputParameters, 84
- CodeTypeSSD
 - nvinfer1::plugin, 73
- combine
 - nvinfer1::ITimingCache, 544
- confidenceThreshold
 - nvinfer1::plugin::DetectionOutputParameters, 84
- configurePlugin
 - nvinfer1::IPluginV2DynamicExt, 436
 - nvinfer1::IPluginV2Ext, 444
 - nvinfer1::IPluginV2IOExt, 448
- configureWithFormat
 - nvinfer1::IPluginV2, 426
 - nvinfer1::IPluginV2Ext, 445
- confSigmoid
 - nvinfer1::plugin::DetectionOutputParameters, 84
- constant
 - nvinfer1::IExprBuilder, 274
- coords
 - nvinfer1::plugin::RegionParameters, 571
- CORNER
 - nvinfer1::plugin, 74
- CORNER_SIZE
 - nvinfer1::plugin, 74
- count
 - nvinfer1::Weights, 576
- createAnchorGeneratorPlugin
 - NvInferPlugin.h, 631
- createBatchedNMSPlugin
 - NvInferPlugin.h, 632
- createBuilderConfig
 - nvinfer1::IBuilder, 136
- createCaffeParser
 - nvcaffeparser1, 25
- createConsistencyChecker_INTERNAL
 - NvInferConsistency.h, 626
- createEngineInspector
 - nvinfer1::ICudaEngine, 193
- createExecutionContext
 - nvinfer1::ICudaEngine, 193
 - nvinfer1::safe::ICudaEngine, 208
- createExecutionContextWithoutDeviceMemory
 - nvinfer1::ICudaEngine, 194
 - nvinfer1::safe::ICudaEngine, 209
- createInferRuntime
 - nvinfer1::safe, 74
- createInstanceNormalizationPlugin
 - NvInferPlugin.h, 632
- createNetworkV2
 - nvinfer1::IBuilder, 137
- createNMSPlugin
 - NvInferPlugin.h, 633
- createNormalizePlugin
 - NvInferPlugin.h, 633
- createNvOnnxParser_INTERNAL
 - NvOnnxParser.h, 680
- createONNXConfig
 - nvonnxparser, 80
- createOptimizationProfile
 - nvinfer1::IBuilder, 137
- createPlugin
 - nvcaffeparser1::IPluginFactoryV2, 419
 - nvinfer1::IPluginCreator, 415
- createPriorBoxPlugin
 - NvInferPlugin.h, 633

- createRegionPlugin
 - NvInferPlugin.h, 633
- createReorgPlugin
 - NvInferPlugin.h, 634
- createRPNROIPlugin
 - NvInferPlugin.h, 634
- createSplitPlugin
 - NvInferPlugin.h, 635
- createTimingCache
 - nvinfer1::IBuilderConfig, 147
- createUffParser
 - nvuffparser, 82
- d
 - nvinfer1::Dims32, 90
 - nvinfer1::DimsExprs, 92
- data
 - nvinfer1::IHostMemory, 296
 - nvinfer1::plugin::Quadruple, 569
 - nvinfer1::PluginField, 560
 - nvuffparser::FieldMap, 109
- DataType
 - nvinfer1, 40
- deallocate
 - nvinfer1::IGpuAllocator, 292
- decRefCount
 - nvinfer1::IErrorRecorder, 245
- deregisterCreator
 - nvinfer1::IPluginRegistry, 421
- desc
 - nvinfer1::DynamicPluginTensorDesc, 96
 - nvonnxparser::IParserError, 409
- deserializeCudaEngine
 - nvinfer1::IRuntime, 500
 - nvinfer1::safe::IRuntime, 506
- deserializePlugin
 - nvinfer1::IPluginCreator, 415
- destroy
 - nvcaffeparser1::IBinaryProtoBlob, 131
 - nvcaffeparser1::ICaffeParser, 165
 - nvinfer1::IBuilder, 137
 - nvinfer1::IBuilderConfig, 148
 - nvinfer1::ICudaEngine, 194
 - nvinfer1::IExecutionContext, 252
 - nvinfer1::IHostMemory, 296
 - nvinfer1::INetworkDefinition, 375
 - nvinfer1::IPluginV2, 427
 - nvinfer1::IRefitter, 474
 - nvinfer1::IRuntime, 501
 - nvonnxparser::IOnnxConfig, 388
 - nvonnxparser::IParser, 405
 - nvuffparser::IUffParser, 552
- detachFromContext
 - nvinfer1::IPluginV2Ext, 445
- DeviceType
 - nvinfer1, 40
- DimensionOperation
 - nvinfer1, 40
- Dims, 86
 - nvinfer1, 36
- dims
 - nvinfer1::PluginTensorDesc, 565
- Dims2
 - nvinfer1::Dims2, 87
- Dims3
 - nvinfer1::Dims3, 88, 89
- Dims4
 - nvinfer1::Dims4, 91
- DimsHW
 - nvinfer1::DimsHW, 93, 94
- dynamicRangeIsSet
 - nvinfer1::ITensor, 535
- ElementWiseOperation
 - nvinfer1, 41
- EngineCapability
 - nvinfer1, 42
- enqueue
 - nvinfer1::IExecutionContext, 253
 - nvinfer1::IPluginV2, 427
 - nvinfer1::IPluginV2DynamicExt, 438
- enqueueV2
 - nvinfer1::IExecutionContext, 253
 - nvinfer1::safe::IExecutionContext, 267
- EnumMax
 - nvinfer1, 64
 - nvonnxparser, 80
- EnumMax< BuilderFlag >
 - nvinfer1, 64
- EnumMax< CalibrationAlgoType >
 - nvinfer1, 65
- EnumMax< DeviceType >
 - nvinfer1, 65
- EnumMax< DimensionOperation >
 - nvinfer1, 65
- EnumMax< ErrorCode >
 - nvonnxparser, 80
- EnumMax< FillOperation >
 - nvinfer1, 65
- EnumMax< GatherMode >
 - nvinfer1, 66
- EnumMax< LayerInformationFormat >
 - nvinfer1, 66
- EnumMax< LayerType >
 - nvinfer1, 66
- EnumMax< LoopOutput >
 - nvinfer1, 66
- EnumMax< MatrixOperation >

- nvinfer1, [67](#)
- EnumMax< MemoryPoolType >
 - nvinfer1, [67](#)
- EnumMax< NetworkDefinitionCreationFlag >
 - nvinfer1, [67](#)
- EnumMax< OptProfileSelector >
 - nvinfer1, [67](#)
- EnumMax< ProfilingVerbosity >
 - nvinfer1, [68](#)
- EnumMax< QuantizationFlag >
 - nvinfer1, [68](#)
- EnumMax< ReduceOperation >
 - nvinfer1, [68](#)
- EnumMax< RNNDirection >
 - nvinfer1, [68](#)
- EnumMax< RNNGateType >
 - nvinfer1, [69](#)
- EnumMax< RNNInputMode >
 - nvinfer1, [69](#)
- EnumMax< RNNOperation >
 - nvinfer1, [69](#)
- EnumMax< ScaleMode >
 - nvinfer1, [69](#)
- EnumMax< ScatterMode >
 - nvinfer1, [70](#)
- EnumMax< SliceMode >
 - nvinfer1, [70](#)
- EnumMax< TacticSource >
 - nvinfer1, [70](#)
- EnumMax< TopKOperation >
 - nvinfer1, [70](#)
- EnumMax< TripLimit >
 - nvinfer1, [71](#)
- EnumMax< UnaryOperation >
 - nvinfer1, [71](#)
- EnumMax< WeightsRole >
 - nvinfer1, [71](#)
- ErrorCode
 - nvinfer1, [43](#)
 - nvonnxparser, [79](#)
- ErrorDesc
 - nvinfer1::IErrorRecorder, [244](#)
- execute
 - nvinfer1::IExecutionContext, [254](#)
- executeV2
 - nvinfer1::IExecutionContext, [255](#)
- featureStride
 - nvinfer1::plugin::RPROIParams, [572](#)
- FieldMap
 - nvuffparser::FieldMap, [109](#)
- fields
 - nvinfer1::PluginFieldCollection, [562](#)
 - nvuffparser::FieldCollection, [108](#)
- FieldType
 - nvuffparser, [81](#)
- file
 - nvonnxparser::IParserError, [410](#)
- FillOperation
 - nvinfer1, [44](#)
- find
 - nvcaffeparser1::IBlobNameToTensor, [133](#)
- flip
 - nvinfer1::plugin::PriorBoxParameters, [567](#)
- format
 - nvinfer1::PluginTensorDesc, [565](#)
- free
 - nvinfer1::IGpuAllocator, [293](#)
- func
 - nvonnxparser::IParserError, [410](#)
- GatherMode
 - nvinfer1, [44](#)
- getActivationType
 - nvinfer1::IActivationLayer, [114](#)
- getAlgorithm
 - nvinfer1::IInt8Calibrator, [307](#)
 - nvinfer1::IInt8EntropyCalibrator, [310](#)
 - nvinfer1::IInt8EntropyCalibrator2, [311](#)
 - nvinfer1::IInt8LegacyCalibrator, [312](#)
 - nvinfer1::IInt8MinMaxCalibrator, [315](#)
- getAlgorithmIOInfo
 - nvinfer1::IAlgorithm, [118](#)
- getAlgorithmIOInfoByIndex
 - nvinfer1::IAlgorithm, [118](#)
- getAlgorithmSelector
 - nvinfer1::IBuilderConfig, [148](#)
- getAlgorithmVariant
 - nvinfer1::IAlgorithm, [119](#)
- getAlignCorners
 - nvinfer1::IResizeLayer, [484](#)
- getAll
 - nvinfer1::IRefitter, [475](#)
- getAllowedFormats
 - nvinfer1::ITensor, [536](#)
- getAllWeights
 - nvinfer1::IRefitter, [475](#)
- getAlpha
 - nvinfer1::IActivationLayer, [114](#)
 - nvinfer1::IFillLayer, [277](#)
 - nvinfer1::ILRNLayr, [337](#)
- getAverageCountExcludesPadding
 - nvinfer1::IPoolingLayer, [454](#)
- getAvgTimingIterations
 - nvinfer1::IBuilderConfig, [149](#)
- getAxes
 - nvinfer1::ISoftMaxLayer, [532](#)
- getAxis

- nvinfer1::IConcatenationLayer, 170
 - nvinfer1::IDequantizeLayer, 230
 - nvinfer1::IIteratorLayer, 316
 - nvinfer1::ILoopOutputLayer, 334
 - nvinfer1::IQuantizeLayer, 466
 - nvinfer1::IScatterLayer, 516
- getBatch
 - nvinfer1::Int8Calibrator, 307
- getBatchSize
 - nvinfer1::Int8Calibrator, 308
- getBeta
 - nvinfer1::IActivationLayer, 115
 - nvinfer1::IFillLayer, 277
 - nvinfer1::ILRNLayer, 337
- getBiasForGate
 - nvinfer1::IRNNv2Layer, 491
- getBiasWeights
 - nvinfer1::IConvolutionLayer, 180
 - nvinfer1::IDeconvolutionLayer, 219
 - nvinfer1::IFullyConnectedLayer, 283
- getBindingBytesPerComponent
 - nvinfer1::ICudaEngine, 194
 - nvinfer1::safe::ICudaEngine, 209
- getBindingComponentsPerElement
 - nvinfer1::ICudaEngine, 195
 - nvinfer1::safe::ICudaEngine, 210
- getBindingDataType
 - nvinfer1::ICudaEngine, 195
 - nvinfer1::safe::ICudaEngine, 210
- getBindingDimensions
 - nvinfer1::ICudaEngine, 196
 - nvinfer1::IExecutionContext, 256
 - nvinfer1::safe::ICudaEngine, 211
- getBindingFormat
 - nvinfer1::ICudaEngine, 196
 - nvinfer1::safe::ICudaEngine, 211
- getBindingFormatDesc
 - nvinfer1::ICudaEngine, 197
- getBindingIndex
 - nvinfer1::ICudaEngine, 197
 - nvinfer1::safe::ICudaEngine, 212
- getBindingName
 - nvinfer1::ICudaEngine, 197
 - nvinfer1::safe::ICudaEngine, 213
- getBindingVectorizedDim
 - nvinfer1::ICudaEngine, 198
 - nvinfer1::safe::ICudaEngine, 213
- getBlendFactor
 - nvinfer1::IPoolingLayer, 454
- getBroadcastAcrossBatch
 - nvinfer1::ITensor, 536
- getBuilderPluginRegistry
 - nvinfer1, 71
- getCalibrationProfile
 - nvinfer1::IBuilderConfig, 149
- getCellState
 - nvinfer1::IRNNv2Layer, 492
- getChannelAxis
 - nvinfer1::IScaleLayer, 511
- getConditional
 - nvinfer1::IIfConditionalBoundaryLayer, 303
- getConstantValue
 - nvinfer1::IDimensionExpr, 232
- getCoordinateTransformation
 - nvinfer1::IResizeLayer, 484
- getData
 - nvcaffeparser1::IBinaryProtoBlob, 131
- getDataLength
 - nvinfer1::IRNNv2Layer, 492
- getDataType
 - nvcaffeparser1::IBinaryProtoBlob, 132
 - nvinfer1::IAlgorithmIOInfo, 123
- getDebugSync
 - nvinfer1::IExecutionContext, 256
- getDefaultDeviceType
 - nvinfer1::IBuilderConfig, 149
- getDeviceMemorySize
 - nvinfer1::ICudaEngine, 198
 - nvinfer1::safe::ICudaEngine, 214
- getDeviceType
 - nvinfer1::IBuilderConfig, 149
- getDilation
 - nvinfer1::IConvolutionLayer, 180
- getDilationNd
 - nvinfer1::IConvolutionLayer, 181
 - nvinfer1::IDeconvolutionLayer, 219
- getDimensions
 - nvcaffeparser1::IBinaryProtoBlob, 132
 - nvinfer1::IAlgorithmContext, 121
 - nvinfer1::IConstantLayer, 176
 - nvinfer1::IFillLayer, 278
 - nvinfer1::IOptimizationProfile, 394
 - nvinfer1::ITensor, 536
- getDirection
 - nvinfer1::IRNNv2Layer, 492
- getDLACore
 - nvinfer1::IBuilderConfig, 150
 - nvinfer1::IRuntime, 501
- getDynamicRangeMax
 - nvinfer1::IRefitter, 476
 - nvinfer1::ITensor, 537
- getDynamicRangeMin
 - nvinfer1::IRefitter, 476
 - nvinfer1::ITensor, 537
- getEngine
 - nvinfer1::IExecutionContext, 256
 - nvinfer1::safe::IExecutionContext, 268
- getEngineCapability

- nvinfer1::IBuilderConfig, 150
 - nvinfer1::ICudaEngine, 199
- getEngineInformation
 - nvinfer1::IEngineInspector, 239
- getEnqueueEmitsProfile
 - nvinfer1::IExecutionContext, 257
- getEquation
 - nvinfer1::IEinsumLayer, 235
- getError
 - nvonnxparser::IParser, 405
- getErrorBuffer
 - nvinfer1::safe::IExecutionContext, 268
- getErrorCode
 - nvinfer1::IErrorRecorder, 245
- getErrorDesc
 - nvinfer1::IErrorRecorder, 246
- getErrorRecorder
 - nvcaffeparser1::ICaffeParser, 165
 - nvinfer1::IBuilder, 138
 - nvinfer1::ICudaEngine, 199
 - nvinfer1::IEngineInspector, 240
 - nvinfer1::IExecutionContext, 257
 - nvinfer1::INetworkDefinition, 375
 - nvinfer1::IPluginRegistry, 421
 - nvinfer1::IRefitter, 476
 - nvinfer1::IRuntime, 501
 - nvinfer1::safe::ICudaEngine, 214
 - nvinfer1::safe::IExecutionContext, 269
 - nvinfer1::safe::IRuntime, 507
 - nvuffparser::IUffParser, 552
- getExecutionContext
 - nvinfer1::IEngineInspector, 240
- getExtraMemoryTarget
 - nvinfer1::IOptimizationProfile, 394
- getFieldNames
 - nvinfer1::IPluginCreator, 415
- getFirstTranspose
 - nvinfer1::IShuffleLayer, 521
- getFlag
 - nvinfer1::IBuilderConfig, 150
- getFlags
 - nvinfer1::IBuilderConfig, 151
- getFullTextFileName
 - nvonnxparser::IOnnxConfig, 388
- getGatherAxis
 - nvinfer1::IGatherLayer, 289
- getHiddenSize
 - nvinfer1::IRNNv2Layer, 492
- getHiddenState
 - nvinfer1::IRNNv2Layer, 493
- getImplementation
 - nvinfer1::IAlgorithmVariant, 127
- getInferLibVersion
 - NvInferRuntimeCommon.h, 658
- getInput
 - nvinfer1::ILayer, 319
 - nvinfer1::INetworkDefinition, 376
- getInputMode
 - nvinfer1::IRNNv2Layer, 493
- getInt8Calibrator
 - nvinfer1::IBuilderConfig, 151
- getK
 - nvinfer1::ILRNLayer, 337
 - nvinfer1::ITopKLayer, 547
- getKeepDimensions
 - nvinfer1::IReduceLayer, 471
- getKernelSize
 - nvinfer1::IConvolutionLayer, 181
 - nvinfer1::IDeconvolutionLayer, 219
- getKernelSizeNd
 - nvinfer1::IConvolutionLayer, 181
 - nvinfer1::IDeconvolutionLayer, 220
- getKernelWeights
 - nvinfer1::IConvolutionLayer, 182
 - nvinfer1::IDeconvolutionLayer, 220
 - nvinfer1::IFullyConnectedLayer, 283
- getLayer
 - nvinfer1::INetworkDefinition, 376
- getLayerCount
 - nvinfer1::IRNNv2Layer, 493
- getLayerInformation
 - nvinfer1::IEngineInspector, 241
- getLocation
 - nvinfer1::ICudaEngine, 199
 - nvinfer1::ITensor, 537
- getLogger
 - nvinfer1::IBuilder, 138
 - nvinfer1::IRefitter, 476
 - nvinfer1::IRuntime, 501
 - NvInferRuntime.h, 642
- getLoop
 - nvinfer1::ILoopBoundaryLayer, 332
- getLoopOutput
 - nvinfer1::ILoopOutputLayer, 334
- getMaxBatchSize
 - nvinfer1::IBuilder, 138
 - nvinfer1::ICudaEngine, 200
- getMaxDLABatchSize
 - nvinfer1::IBuilder, 139
- getMaxSeqLength
 - nvinfer1::IRNNv2Layer, 493
- getMaxThreads
 - nvinfer1::IBuilder, 139
 - nvinfer1::IRefitter, 477
 - nvinfer1::IRuntime, 502
- getMaxWorkspaceSize
 - nvinfer1::IBuilderConfig, 151
- getMemoryPoolLimit

- nvinfer1::IBuilderConfig, 152
- getMessage
 - nvinfer1::IAssertionLayer, 129
- getMinTimingIterations
 - nvinfer1::IBuilderConfig, 152
- getMissing
 - nvinfer1::IRefitter, 477
- getMissingWeights
 - nvinfer1::IRefitter, 478
- getMode
 - nvinfer1::IGatherLayer, 289
 - nvinfer1::IScaleLayer, 511
 - nvinfer1::IScatterLayer, 516
 - nvinfer1::ISliceLayer, 527
- getModelDtype
 - nvonnxparser::IOnnxConfig, 388
- getModelFileName
 - nvonnxparser::IOnnxConfig, 389
- getName
 - nvinfer1::IAlgorithmContext, 121
 - nvinfer1::ICudaEngine, 200
 - nvinfer1::IExecutionContext, 257
 - nvinfer1::IIfConditional, 301
 - nvinfer1::ILayer, 320
 - nvinfer1::ILoop, 330
 - nvinfer1::INetworkDefinition, 377
 - nvinfer1::ITensor, 537
 - nvinfer1::safe::ICudaEngine, 214
 - nvinfer1::safe::IExecutionContext, 269
- getNbBindings
 - nvinfer1::ICudaEngine, 200
 - nvinfer1::safe::ICudaEngine, 215
- getNbDLACores
 - nvinfer1::IBuilder, 139
 - nvinfer1::IRuntime, 502
- getNbElementWiseDims
 - nvinfer1::IGatherLayer, 289
- getNbErrors
 - nvinfer1::IErrorRecorder, 247
 - nvonnxparser::IParser, 406
- getNbGroups
 - nvinfer1::IConvolutionLayer, 182
 - nvinfer1::IDeconvolutionLayer, 220
- getNbInputs
 - nvinfer1::IAlgorithmContext, 121
 - nvinfer1::ILayer, 320
 - nvinfer1::INetworkDefinition, 377
- getNbLayers
 - nvinfer1::ICudaEngine, 201
 - nvinfer1::INetworkDefinition, 377
- getNbOptimizationProfiles
 - nvinfer1::IBuilderConfig, 153
 - nvinfer1::ICudaEngine, 201
- getNbOutputChannels
 - nvinfer1::IFullyConnectedLayer, 283
- getNbOutputMaps
 - nvinfer1::IConvolutionLayer, 182
 - nvinfer1::IDeconvolutionLayer, 220
- getNbOutputs
 - nvinfer1::IAlgorithmContext, 121
 - nvinfer1::ILayer, 320
 - nvinfer1::INetworkDefinition, 378
 - nvinfer1::IPluginV2, 428
- getNbShapeValues
 - nvinfer1::IOptimizationProfile, 395
- getNearestRounding
 - nvinfer1::IResizeLayer, 484
- getNvOnnxParserVersion
 - NvOnnxParser.h, 680
- getOperation
 - nvinfer1::IElementWiseLayer, 237
 - nvinfer1::IFillLayer, 278
 - nvinfer1::IMatrixMultiplyLayer, 341
 - nvinfer1::IReduceLayer, 471
 - nvinfer1::IRNNv2Layer, 493
 - nvinfer1::ITopKLayer, 547
 - nvinfer1::UnaryLayer, 556
- getOptimizationProfile
 - nvinfer1::IExecutionContext, 258
- getOutput
 - nvinfer1::ILayer, 320
 - nvinfer1::INetworkDefinition, 378
- getOutputDataType
 - nvinfer1::IPluginV2Ext, 445
- getOutputDimensions
 - nvinfer1::IPluginV2, 428
 - nvinfer1::IPluginV2DynamicExt, 438
 - nvinfer1::IResizeLayer, 484
- getOutputType
 - nvinfer1::ILayer, 320
- getPadding
 - nvinfer1::IConvolutionLayer, 182
 - nvinfer1::IDeconvolutionLayer, 221
 - nvinfer1::IPoolingLayer, 454
- getPaddingMode
 - nvinfer1::IConvolutionLayer, 183
 - nvinfer1::IDeconvolutionLayer, 221
 - nvinfer1::IPoolingLayer, 455
- getPaddingNd
 - nvinfer1::IConvolutionLayer, 183
 - nvinfer1::IDeconvolutionLayer, 221
 - nvinfer1::IPoolingLayer, 455
- getPlugin
 - nvinfer1::IPluginV2Layer, 451
- getPluginCreator
 - nvinfer1::IPluginRegistry, 422
- getPluginCreatorList
 - nvinfer1::IPluginRegistry, 422

- getPluginName
 - nvinfer1::IPluginCreator, 416
- getPluginNamespace
 - nvinfer1::IPluginCreator, 416
 - nvinfer1::IPluginV2, 429
- getPluginRegistry
 - NvInferRuntime.h, 642
- getPluginType
 - nvinfer1::IPluginV2, 429
- getPluginVersion
 - nvinfer1::IPluginCreator, 417
 - nvinfer1::IPluginV2, 430
- getPoolingType
 - nvinfer1::IPoolingLayer, 455
- getPostPadding
 - nvinfer1::IConvolutionLayer, 183
 - nvinfer1::IDeconvolutionLayer, 222
 - nvinfer1::IPaddingLayer, 400
 - nvinfer1::IPoolingLayer, 456
- getPostPaddingNd
 - nvinfer1::IPaddingLayer, 400
- getPower
 - nvinfer1::IScaleLayer, 511
- getPrecision
 - nvinfer1::ILayer, 321
- getPrePadding
 - nvinfer1::IConvolutionLayer, 184
 - nvinfer1::IDeconvolutionLayer, 222
 - nvinfer1::IPaddingLayer, 400
 - nvinfer1::IPoolingLayer, 456
- getPrePaddingNd
 - nvinfer1::IPaddingLayer, 400
- getPrintLayerInfo
 - nvonnxparser::IOnnxConfig, 389
- getProfileDimensions
 - nvinfer1::ICudaEngine, 201
- getProfiler
 - nvinfer1::IExecutionContext, 258
- getProfileShapeValues
 - nvinfer1::ICudaEngine, 202
- getProfileStream
 - nvinfer1::IBuilderConfig, 153
- getProfilingVerbosity
 - nvinfer1::IBuilderConfig, 153
 - nvinfer1::ICudaEngine, 203
- getQuantile
 - nvinfer1::IInt8LegacyCalibrator, 312
- getQuantizationFlag
 - nvinfer1::IBuilderConfig, 154
- getQuantizationFlags
 - nvinfer1::IBuilderConfig, 154
- getReduceAxes
 - nvinfer1::IReduceLayer, 471
 - nvinfer1::ITopKLayer, 548
- getRegressionCutoff
 - nvinfer1::IInt8LegacyCalibrator, 313
- getReshapeDimensions
 - nvinfer1::IShuffleLayer, 521
- getResizeMode
 - nvinfer1::IResizeLayer, 485
- getReverse
 - nvinfer1::IIteratorLayer, 316
- getSafePluginRegistry
 - nvinfer1::safe, 75
- getScale
 - nvinfer1::IScaleLayer, 511
- getScales
 - nvinfer1::IResizeLayer, 485
- getSecondTranspose
 - nvinfer1::IShuffleLayer, 522
- getSelectorForSinglePixel
 - nvinfer1::IResizeLayer, 485
- getSequenceLengths
 - nvinfer1::IRNNv2Layer, 494
- getSerializationSize
 - nvinfer1::IPluginV2, 430
- getShapeBinding
 - nvinfer1::IExecutionContext, 258
- getShapeValues
 - nvinfer1::IOptimizationProfile, 395
- getShift
 - nvinfer1::IScaleLayer, 512
- getSize
 - nvinfer1::ISliceLayer, 527
- getStart
 - nvinfer1::ISliceLayer, 528
- getStride
 - nvinfer1::IConvolutionLayer, 184
 - nvinfer1::IDeconvolutionLayer, 222
 - nvinfer1::IPoolingLayer, 456
 - nvinfer1::ISliceLayer, 528
- getStrideNd
 - nvinfer1::IConvolutionLayer, 184
 - nvinfer1::IDeconvolutionLayer, 222
 - nvinfer1::IPoolingLayer, 456
- getStrides
 - nvinfer1::IAlgorithmIOInfo, 123
 - nvinfer1::IExecutionContext, 259
 - nvinfer1::safe::IExecutionContext, 270
- getTactic
 - nvinfer1::IAlgorithmVariant, 128
- getTacticSources
 - nvinfer1::IBuilderConfig, 154
 - nvinfer1::ICudaEngine, 203
- getTensorFormat
 - nvinfer1::IAlgorithmIOInfo, 124
- getTensorRTVersion
 - nvinfer1::IPluginCreator, 417

- nvinfer1::IPluginV2, 430
 - nvinfer1::IPluginV2DynamicExt, 439
 - nvinfer1::IPluginV2Ext, 446
 - nvinfer1::IPluginV2IOExt, 449
- getTensorsWithDynamicRange
 - nvinfer1::IRefitter, 478
- getTextFileName
 - nvonnxparser::IOnnxConfig, 389
- getTimingCache
 - nvinfer1::IBuilderConfig, 155
- getTimingMSec
 - nvinfer1::IAlgorithm, 119
- getTripLimit
 - nvinfer1::ITripLimitLayer, 550
- getType
 - nvinfer1::ILayer, 321
 - nvinfer1::ITensor, 538
- getUffRequiredVersionMajor
 - nvuffparser::IUffParser, 552
- getUffRequiredVersionMinor
 - nvuffparser::IUffParser, 552
- getUffRequiredVersionPatch
 - nvuffparser::IUffParser, 552
- getVerbosityLevel
 - nvonnxparser::IOnnxConfig, 390
- getWeights
 - nvinfer1::IConstantLayer, 176
- getWeightsForGate
 - nvinfer1::IRNNv2Layer, 494
- getWindowSize
 - nvinfer1::ILRNLayer, 337
 - nvinfer1::IPoolingLayer, 457
- getWindowSizeNd
 - nvinfer1::IPoolingLayer, 457
- getWorkspaceSize
 - nvinfer1::IAlgorithm, 119
 - nvinfer1::IPluginV2, 431
 - nvinfer1::IPluginV2DynamicExt, 439
- getZeroIsPlaceholder
 - nvinfer1::IShuffleLayer, 522
- group
 - nvinfer1::plugin::softmaxTree, 574
- groupOffset
 - nvinfer1::plugin::softmaxTree, 574
- groups
 - nvinfer1::plugin::softmaxTree, 574
- groupSize
 - nvinfer1::plugin::softmaxTree, 574
- H
 - nvinfer1::plugin::GridAnchorParameters, 112
- h
 - nvinfer1::DimsHW, 94
- hasExplicitPrecision
 - nvinfer1::INetworkDefinition, 379
- hasImplicitBatchDimension
 - nvinfer1::ICudaEngine, 203
 - nvinfer1::INetworkDefinition, 379
- hasOverflowed
 - nvinfer1::IErrorRecorder, 247
- IConsistencyChecker
 - nvinfer1::consistency::IConsistencyChecker, 174
- ICudaEngine
 - nvinfer1::safe::ICudaEngine, 207, 208
- IErrorRecorder
 - nvinfer1::IErrorRecorder, 244
- IExecutionContext
 - nvinfer1::safe::IExecutionContext, 267
- IGpuAllocator
 - nvinfer1::IGpuAllocator, 291
- ILogger
 - nvinfer1::ILogger, 327
- imgH
 - nvinfer1::plugin::PriorBoxParameters, 567
- imgW
 - nvinfer1::plugin::PriorBoxParameters, 567
- incRefCount
 - nvinfer1::IErrorRecorder, 248
- initialize
 - nvinfer1::IPluginV2, 431
- initLibNvInferPlugins
 - NvInferPlugin.h, 635
- INoCopy
 - nvinfer1::INoCopy, 385
- inputOrder
 - nvinfer1::plugin::DetectionOutputParameters, 85
- iouThreshold
 - nvinfer1::plugin::NMSPParameters, 558
 - nvinfer1::plugin::RPROIParams, 572
- IPluginChecker
 - nvinfer1::consistency::IPluginChecker, 411, 412
- IPluginCreator
 - nvinfer1::IPluginCreator, 414
- IPluginV2Ext
 - nvinfer1::IPluginV2Ext, 442
- IRuntime
 - nvinfer1::safe::IRuntime, 505, 506
- isBatchAgnostic
 - nvinfer1::plugin::DetectionOutputParameters, 85
- isConstant
 - nvinfer1::IDimensionExpr, 233
- isDeviceTypeSet
 - nvinfer1::IBuilderConfig, 155
- isExecutionBinding
 - nvinfer1::ICudaEngine, 203
- isExecutionTensor
 - nvinfer1::ITensor, 538

- isNetworkInput
 - nvinfer1::ITensor, 538
- isNetworkOutput
 - nvinfer1::ITensor, 539
- isNetworkSupported
 - nvinfer1::IBuilder, 140
- isNormalized
 - nvinfer1::plugin::DetectionOutputParameters, 85
 - nvinfer1::plugin::NMSParameters, 558
- isOutputBroadcastAcrossBatch
 - nvinfer1::IPluginV2Ext, 446
- isPluginV2
 - nvcaffeparser1::IPluginFactoryV2, 419
- isRefittable
 - nvinfer1::ICudaEngine, 204
- isShapeBinding
 - nvinfer1::ICudaEngine, 204
- isShapeTensor
 - nvinfer1::ITensor, 539
- isValid
 - nvinfer1::IOptimizationProfile, 395

- kABS
 - nvinfer1, 63
- kACOS
 - nvinfer1, 63
- kACOSH
 - nvinfer1, 64
- kACTIVATION
 - nvinfer1, 45
- kALIGN_CORNERS
 - nvinfer1, 55
- kAND
 - nvinfer1, 42
- kANY
 - nvinfer1, 64
- kASIN
 - nvinfer1, 63
- kASINH
 - nvinfer1, 64
- kASSERTION
 - nvinfer1, 46
- kASYMMETRIC
 - nvinfer1, 55
- kATAN
 - nvinfer1, 64
- kATANH
 - nvinfer1, 64
- kAVERAGE
 - nvinfer1, 53
- kAVG
 - nvinfer1, 54
- kBIAS
 - nvinfer1, 64

- kBIDIRECTION
 - nvinfer1, 56
- kBOOL
 - nvinfer1, 40
- kCAFFE_ROUND_DOWN
 - nvinfer1, 52
- kCAFFE_ROUND_UP
 - nvinfer1, 52
- kCALIBRATE_BEFORE_FUSION
 - nvinfer1, 54
- kCDHW32
 - nvinfer1, 61
- kCEIL
 - nvinfer1, 56, 64
- kCEIL_DIV
 - nvinfer1, 41
- kCELL
 - nvinfer1, 57
- kCHANNEL
 - nvinfer1, 59
- kCHAR
 - nvinfer1, 52
 - nvuffparser, 81
- kCHW16
 - nvinfer1, 61
- kCHW2
 - nvinfer1, 61
- kCHW32
 - nvinfer1, 61
- kCHW4
 - nvinfer1, 61
- kCLAMP
 - nvinfer1, 60
- kCLIP
 - nvinfer1, 38
- kCONCATENATE
 - nvinfer1, 47
- kCONCATENATION
 - nvinfer1, 46
- kCONDITION
 - nvinfer1, 46
- kCONDITIONAL_INPUT
 - nvinfer1, 46
- kCONDITIONAL_OUTPUT
 - nvinfer1, 46
- kCONSTANT
 - nvinfer1, 46, 64
- kCONVOLUTION
 - nvinfer1, 45
- kCOS
 - nvinfer1, 63
- kCOSH
 - nvinfer1, 63
- kCOUNT

- nvinfer1, 63
- kCUBLAS
 - nvinfer1, 60
- kCUBLAS_LT
 - nvinfer1, 60
- kCUDNN
 - nvinfer1, 60
- kDATATYPE
 - nvuffparser, 81
- kDEBUG
 - nvinfer1, 38
- kDECONVOLUTION
 - nvinfer1, 46
- kDEFAULT
 - nvinfer1, 42, 45, 53, 60
- kDEQUANTIZE
 - nvinfer1, 46
- kDETAILED
 - nvinfer1, 53
- kDEVICE
 - nvinfer1, 62
- kDHWCS
 - nvinfer1, 61
- kDIMS
 - nvinfer1, 52
 - nvuffparser, 81
- kDIRECT_IO
 - nvinfer1, 39
- kDISABLE_TIMING_CACHE
 - nvinfer1, 39
- kDIV
 - nvinfer1, 41
- kDLA
 - nvinfer1, 40
- kDLA_GLOBAL_DRAM
 - nvinfer1, 48
- kDLA_HWC4
 - nvinfer1, 62
- kDLA_LINEAR
 - nvinfer1, 62
- kDLA_LOCAL_DRAM
 - nvinfer1, 48
- kDLA_MANAGED_SRAM
 - nvinfer1, 48
- kDLA_STANDALONE
 - nvinfer1, 42
- kEDGE_MASK_CONVOLUTIONS
 - nvinfer1, 60
- keepTopK
 - nvinfer1::plugin::DetectionOutputParameters, 85
 - nvinfer1::plugin::NMSParameters, 558
- KEINSUM
 - nvinfer1, 46
- kELEMENT
 - nvinfer1, 45, 59
- kELEMENTWISE
 - nvinfer1, 46, 59
- kELU
 - nvinfer1, 38
- kENTROPY_CALIBRATION
 - nvinfer1, 40
- kENTROPY_CALIBRATION_2
 - nvinfer1, 40
- kEQUAL
 - nvinfer1, 41, 42
- kERF
 - nvinfer1, 64
- kERROR
 - nvinfer1::ILogger, 326
- kEXP
 - nvinfer1, 63
- kEXPLICIT_BATCH
 - nvinfer1, 48
- kEXPLICIT_PRECISION
 - nvinfer1, 48
- kEXPLICIT_ROUND_DOWN
 - nvinfer1, 52
- kEXPLICIT_ROUND_UP
 - nvinfer1, 52
- kFAILED_ALLOCATION
 - nvinfer1, 43
- kFAILED_COMPUTATION
 - nvinfer1, 43
- kFAILED_EXECUTION
 - nvinfer1, 43
- kFAILED_INITIALIZATION
 - nvinfer1, 43
- kFILL
 - nvinfer1, 46, 60
- kFLOAT
 - nvinfer1, 40
 - nvuffparser, 81
- kFLOAT16
 - nvinfer1, 52
- kFLOAT32
 - nvinfer1, 52
- kFLOAT64
 - nvinfer1, 52
- kFLOOR
 - nvinfer1, 56, 64
- kFLOOR_DIV
 - nvinfer1, 41, 42
- kFORGET
 - nvinfer1, 57
- kFORMAT_COMBINATION_LIMIT
 - nvinfer1::IPluginV2DynamicExt, 440
- kFORMULA
 - nvinfer1, 56

- kFP16
 - nvinfer1, 38
- kFULLY_CONNECTED
 - nvinfer1, 45
- kGATHER
 - nvinfer1, 46
- kGPU
 - nvinfer1, 40
- kGPU_FALLBACK
 - nvinfer1, 39
- kGREATER
 - nvinfer1, 42
- kGRU
 - nvinfer1, 58
- kHALF
 - nvinfer1, 40
- kHALF_DOWN
 - nvinfer1, 56
- kHALF_PIXEL
 - nvinfer1, 55
- kHALF_UP
 - nvinfer1, 56
- kHARD_SIGMOID
 - nvinfer1, 38
- kHIDDEN
 - nvinfer1, 57
- kHOST
 - nvinfer1, 62
- kHWC
 - nvinfer1, 61
- kHWC16
 - nvinfer1, 62
- kHWC8
 - nvinfer1, 61
- kIDENTITY
 - nvinfer1, 46
- kINFO
 - nvinfer1::ILogger, 326
- kINPUT
 - nvinfer1, 57
- kINT16
 - nvinfer1, 52
- kINT32
 - nvinfer1, 40, 52
 - nvuffparser, 81
- kINT8
 - nvinfer1, 38, 40, 52
- kINTERNAL_ERROR
 - nvinfer1, 43
 - nvinfer1::ILogger, 326
 - nvonnxparser, 79
- kINVALID_ARGUMENT
 - nvinfer1, 43
- kINVALID_CONFIG
 - nvinfer1, 43
- kINVALID_GRAPH
 - nvonnxparser, 79
- kINVALID_NODE
 - nvonnxparser, 79
- kINVALID_STATE
 - nvinfer1, 43
- kINVALID_VALUE
 - nvonnxparser, 79
- kITERATOR
 - nvinfer1, 46
- kJSON
 - nvinfer1, 45
- kKERNEL
 - nvinfer1, 64
- kLAST_VALUE
 - nvinfer1, 47
- kLAYER_NAMES_ONLY
 - nvinfer1, 53
- kLEAKY_RELU
 - nvinfer1, 38
- kLEGACY_CALIBRATION
 - nvinfer1, 40
- kLESS
 - nvinfer1, 41, 42
- kLINEAR
 - nvinfer1, 55, 57, 61
- kLinspace
 - nvinfer1, 44
- kLOG
 - nvinfer1, 63
- kLOOP_OUTPUT
 - nvinfer1, 46
- kLRN
 - nvinfer1, 45
- kLSTM
 - nvinfer1, 58
- kMATRIX_MULTIPLY
 - nvinfer1, 46
- kMAX
 - nvinfer1, 41, 49, 53, 54, 62
- kMAX_AVERAGE_BLEND
 - nvinfer1, 53
- kMAX_DESC_LENGTH
 - nvinfer1::ILogger, 249
- kMEM_ALLOC_FAILED
 - nvonnxparser, 79
- kMIN
 - nvinfer1, 41, 49, 54, 62
- kMINMAX_CALIBRATION
 - nvinfer1, 40
- kMODEL_DESERIALIZE_FAILED
 - nvonnxparser, 79
- kNC

nvuffparser, 81

kNCHW
nvuffparser, 81

kND
nvinfer1, 45, 59

kNEAREST
nvinfer1, 55

kNEG
nvinfer1, 63

kNHWC
nvuffparser, 81

kNONE
nvinfer1, 47, 53

kNOT
nvinfer1, 64

kOBEY_PRECISION_CONSTRAINTS
nvinfer1, 39

kONELINE
nvinfer1, 45

kOPT
nvinfer1, 49

kOR
nvinfer1, 42

kOUTPUT
nvinfer1, 57

kPADDING
nvinfer1, 46

kPARAMETRIC_RELU
nvinfer1, 46

kPLUGIN
nvinfer1, 46

kPLUGIN_V2
nvinfer1, 46

kPOOLING
nvinfer1, 45

kPOW
nvinfer1, 42

kPREFER_PRECISION_CONSTRAINTS
nvinfer1, 39

kPROD
nvinfer1, 41, 54

kQUANTIZE
nvinfer1, 46

kRAGGED_SOFTMAX
nvinfer1, 46

kRANDOM_UNIFORM
nvinfer1, 44

kRECIP
nvinfer1, 63

kRECURRENCE
nvinfer1, 46

kREDUCE
nvinfer1, 46

kREFIT
nvinfer1, 39

kREFLECT
nvinfer1, 60

kREJECT_EMPTY_ALGORITHMS
nvinfer1, 39

kRELU
nvinfer1, 38, 58

kRESET
nvinfer1, 57

kRESIZABLE
nvinfer1, 38

kRESIZE
nvinfer1, 46

kREVERSE
nvinfer1, 47

kRNN_V2
nvinfer1, 46

kROUND
nvinfer1, 64

kSAFE_DLA
nvinfer1, 42

kSAFE_GPU
nvinfer1, 42

kSAFETY
nvinfer1, 42

kSAFETY_SCOPE
nvinfer1, 39

kSAME_LOWER
nvinfer1, 52

kSAME_UPPER
nvinfer1, 52

kSCALE
nvinfer1, 45, 64

kSCALED_TANH
nvinfer1, 38

kSCATTER
nvinfer1, 46

kSELECT
nvinfer1, 46

kSELU
nvinfer1, 38

kSHAPE
nvinfer1, 46

kSHIFT
nvinfer1, 64

kSHUFFLE
nvinfer1, 46

kSIGMOID
nvinfer1, 38

kSIGN
nvinfer1, 64

kSIN
nvinfer1, 63

kSINH

- nvinfer1, [63](#)
- kSKIP
 - nvinfer1, [57](#)
- kSLICE
 - nvinfer1, [46](#)
- kSOFTMAX
 - nvinfer1, [45](#)
- kSOFTPLUS
 - nvinfer1, [38](#)
- kSOFTSIGN
 - nvinfer1, [38](#)
- kSPARSE_WEIGHTS
 - nvinfer1, [39](#)
- kSQRT
 - nvinfer1, [63](#)
- kSTANDARD
 - nvinfer1, [42](#)
- kSTRICT_TYPES
 - nvinfer1, [39](#)
- kSUB
 - nvinfer1, [41](#)
- kSUCCESS
 - nvinfer1, [43](#)
 - nvonnxparser, [79](#)
- kSUM
 - nvinfer1, [41](#), [54](#)
- kTAN
 - nvinfer1, [63](#)
- kTANH
 - nvinfer1, [38](#), [58](#)
- kTF32
 - nvinfer1, [39](#)
- kTHRESHOLDED_RELU
 - nvinfer1, [38](#)
- kTOPK
 - nvinfer1, [46](#)
- kTRANSPOSE
 - nvinfer1, [47](#)
- kTRIP_LIMIT
 - nvinfer1, [46](#)
- kUNARY
 - nvinfer1, [46](#)
- kUNIDIRECTION
 - nvinfer1, [56](#)
- kUNIFORM
 - nvinfer1, [59](#)
- kUNKNOWN
 - nvinfer1, [52](#)
 - nvuffparser, [81](#)
- kUNSPECIFIED_ERROR
 - nvinfer1, [43](#)
- kUNSUPPORTED_GRAPH
 - nvonnxparser, [79](#)
- kUNSUPPORTED_NODE
 - nvonnxparser, [79](#)
- kUNSUPPORTED_STATE
 - nvinfer1, [44](#)
- kUPDATE
 - nvinfer1, [57](#)
- kUPPER
 - nvinfer1, [56](#)
- kV2
 - nvinfer1, [52](#)
- kV2_DYNAMICEXT
 - nvinfer1, [52](#)
- kV2_EXT
 - nvinfer1, [52](#)
- kV2_IOEXT
 - nvinfer1, [52](#)
- kVALUE
 - nvinfer1::impl::EnumMaxImpl< ActivationType >, [97](#)
 - nvinfer1::impl::EnumMaxImpl< AllocatorFlag >, [98](#)
 - nvinfer1::impl::EnumMaxImpl< DataType >, [98](#)
 - nvinfer1::impl::EnumMaxImpl< ElementWiseOperation >, [99](#)
 - nvinfer1::impl::EnumMaxImpl< EngineCapability >, [100](#)
 - nvinfer1::impl::EnumMaxImpl< ErrorCode >, [101](#)
 - nvinfer1::impl::EnumMaxImpl< ILogger::Severity >, [102](#)
 - nvinfer1::impl::EnumMaxImpl< PaddingMode >, [102](#)
 - nvinfer1::impl::EnumMaxImpl< PoolingType >, [103](#)
 - nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation >, [104](#)
 - nvinfer1::impl::EnumMaxImpl< ResizeMode >, [104](#)
 - nvinfer1::impl::EnumMaxImpl< ResizeRoundMode >, [105](#)
 - nvinfer1::impl::EnumMaxImpl< ResizeSelector >, [106](#)
 - nvinfer1::impl::EnumMaxImpl< TensorFormat >, [107](#)
 - nvinfer1::impl::EnumMaxImpl< TensorLocation >, [107](#)
- kVECTOR
 - nvinfer1, [47](#)
- kVERBOSE
 - nvinfer1, [53](#)
 - nvinfer1::ILogger, [326](#)
- kWARNING
 - nvinfer1::ILogger, [326](#)
- kWHILE
 - nvinfer1, [63](#)
- kWORKSPACE

- nvinfer1, 48
- kWRAP
 - nvinfer1, 60
- kXOR
 - nvinfer1, 42
- LayerInformationFormat
 - nvinfer1, 45
- LayerType
 - nvinfer1, 45
- leaf
 - nvinfer1::plugin::softmaxTree, 575
- length
 - nvinfer1::PluginField, 561
 - nvuffparser::FieldMap, 109
- line
 - nvonnxparser::IParserError, 410
- log
 - nvinfer1::ILogger, 327
- LoopOutput
 - nvinfer1, 46
- markOutput
 - nvinfer1::INetworkDefinition, 379
- markOutputForShapes
 - nvinfer1::INetworkDefinition, 380
- MatrixOperation
 - nvinfer1, 47
- max
 - nvinfer1::DynamicPluginTensorDesc, 96
- MAX_DIMS
 - nvinfer1::Dims32, 90
- maxSize
 - nvinfer1::plugin::GridAnchorParameters, 112
 - nvinfer1::plugin::PriorBoxParameters, 568
- mBoundary
 - nvinfer1::IIfConditionalBoundaryLayer, 303
 - nvinfer1::ILoopBoundaryLayer, 332
- MemoryPoolType
 - nvinfer1, 47
- mImpl
 - nvinfer1::consistency::IConsistencyChecker, 175
 - nvinfer1::IActivationLayer, 116
 - nvinfer1::IAlgorithm, 119
 - nvinfer1::IAlgorithmContext, 122
 - nvinfer1::IAlgorithmIOInfo, 124
 - nvinfer1::IAlgorithmVariant, 128
 - nvinfer1::IAssertionLayer, 130
 - nvinfer1::IBuilder, 143
 - nvinfer1::IBuilderConfig, 164
 - nvinfer1::IConcatenationLayer, 171
 - nvinfer1::IConditionLayer, 172
 - nvinfer1::IConstantLayer, 177
 - nvinfer1::IConvolutionLayer, 190
 - nvinfer1::ICudaEngine, 206
 - nvinfer1::IDeconvolutionLayer, 228
 - nvinfer1::IDequantizeLayer, 231
 - nvinfer1::IDimensionExpr, 233
 - nvinfer1::IEinsumLayer, 235
 - nvinfer1::IElementWiseLayer, 238
 - nvinfer1::IEngineInspector, 242
 - nvinfer1::IExecutionContext, 266
 - nvinfer1::IExprBuilder, 275
 - nvinfer1::IFillLayer, 281
 - nvinfer1::IFullyConnectedLayer, 285
 - nvinfer1::IGatherLayer, 291
 - nvinfer1::IHostMemory, 297
 - nvinfer1::IIdentityLayer, 298
 - nvinfer1::IIfConditional, 302
 - nvinfer1::IIfConditionalInputLayer, 305
 - nvinfer1::IIfConditionalOutputLayer, 306
 - nvinfer1::IIteratorLayer, 317
 - nvinfer1::ILoop, 330
 - nvinfer1::ILoopOutputLayer, 335
 - nvinfer1::ILRNLayer, 339
 - nvinfer1::IMatrixMultiplyLayer, 342
 - nvinfer1::INetworkDefinition, 383
 - nvinfer1::IOptimizationProfile, 398
 - nvinfer1::IPaddingLayer, 402
 - nvinfer1::IParametricReLULayer, 404
 - nvinfer1::IPluginV2Layer, 452
 - nvinfer1::IPoolingLayer, 462
 - nvinfer1::IQuantizeLayer, 466
 - nvinfer1::IRaggedSoftMaxLayer, 468
 - nvinfer1::IRecurrenceLayer, 469
 - nvinfer1::IReduceLayer, 473
 - nvinfer1::IRefitter, 481
 - nvinfer1::IResizeLayer, 489
 - nvinfer1::IRNNv2Layer, 498
 - nvinfer1::IRuntime, 504
 - nvinfer1::IScaleLayer, 514
 - nvinfer1::IScatterLayer, 517
 - nvinfer1::ISelectLayer, 518
 - nvinfer1::IShapeLayer, 519
 - nvinfer1::IShuffleLayer, 525
 - nvinfer1::ISliceLayer, 531
 - nvinfer1::ISoftMaxLayer, 533
 - nvinfer1::ITensor, 543
 - nvinfer1::ITimingCache, 546
 - nvinfer1::ITopKLayer, 549
 - nvinfer1::ITripLimitLayer, 550
 - nvinfer1::IUnaryLayer, 556
- min
 - nvinfer1::DynamicPluginTensorDesc, 96
- minBoxSize
 - nvinfer1::plugin::RPROIParams, 572
- minSize
 - nvinfer1::plugin::GridAnchorParameters, 112
 - nvinfer1::plugin::PriorBoxParameters, 568

mLayer
 nvinfer1::ILayer, 325

n
 nvinfer1::plugin::softmaxTree, 575

name
 nvinfer1::plugin::softmaxTree, 575
 nvinfer1::PluginField, 561
 nvuffparser::FieldMap, 109

nbDims
 nvinfer1::Dims32, 90
 nvinfer1::DimsExprs, 92

nbFields
 nvinfer1::PluginFieldCollection, 562
 nvuffparser::FieldCollection, 108

nbInfErrors
 nvinfer1::safe::FloatingPointErrorInformation, 110

nbNanErrors
 nvinfer1::safe::FloatingPointErrorInformation, 110

NetworkDefinitionCreationFlag
 nvinfer1, 48

NetworkDefinitionCreationFlags
 nvinfer1, 36

nmsMaxOut
 nvinfer1::plugin::RPROIParams, 573

nmsThreshold
 nvinfer1::plugin::DetectionOutputParameters, 85

node
 nvonnxparser::IParserError, 410

num
 nvinfer1::plugin::RegionParameters, 571

numAspectRatios
 nvinfer1::plugin::GridAnchorParameters, 112
 nvinfer1::plugin::PriorBoxParameters, 568

numClasses
 nvinfer1::plugin::DetectionOutputParameters, 85
 nvinfer1::plugin::NMSParameters, 558

numMaxSize
 nvinfer1::plugin::PriorBoxParameters, 568

numMinSize
 nvinfer1::plugin::PriorBoxParameters, 568

NV_ONNX_PARSER_MAJOR
 NvOnnxParser.h, 679

NV_ONNX_PARSER_MINOR
 NvOnnxParser.h, 679

NV_ONNX_PARSER_PATCH
 NvOnnxParser.h, 680

NV_TENSORRT_BUILD
 NvInferVersion.h, 670

NV_TENSORRT_LWS_MAJOR
 NvInferVersion.h, 670

NV_TENSORRT_LWS_MINOR
 NvInferVersion.h, 671

NV_TENSORRT_LWS_PATCH
 NvInferVersion.h, 671

NV_TENSORRT_MAJOR
 NvInferVersion.h, 671

NV_TENSORRT_MINOR
 NvInferVersion.h, 671

NV_TENSORRT_PATCH
 NvInferVersion.h, 671

NV_TENSORRT_SONAME_MAJOR
 NvInferVersion.h, 671

NV_TENSORRT_SONAME_MINOR
 NvInferVersion.h, 672

NV_TENSORRT_SONAME_PATCH
 NvInferVersion.h, 672

NV_TENSORRT_VERSION
 NvInferRuntimeCommon.h, 657

NvCaffeParser.h, 577, 578

nvcaffeparser1, 25
 createCaffeParser, 25
 shutdownProtobufLibrary, 26

nvcaffeparser1::IBinaryProtoBlob, 130
 ~IBinaryProtoBlob, 131
 destroy, 131
 getData, 131
 getDataType, 132
 getDimensions, 132

nvcaffeparser1::IBlobNameToTensor, 132
 ~IBlobNameToTensor, 133
 find, 133

nvcaffeparser1::ICaffeParser, 164
 ~ICaffeParser, 165
 destroy, 165
 getErrorRecorder, 165
 parse, 166
 parseBinaryProto, 166
 parseBuffers, 167
 setErrorRecorder, 168
 setPluginFactoryV2, 168
 setPluginNamespace, 168
 setProtobufBufferSize, 169

nvcaffeparser1::IPluginFactoryV2, 418
 ~IPluginFactoryV2, 419
 createPlugin, 419
 isPluginV2, 419

NvInfer.h, 579, 585

nvinfer1, 26
 ActivationType, 37
 AllocatorFlag, 38
 AllocatorFlags, 35
 AsciiChar, 35
 BuilderFlag, 38
 BuilderFlags, 35
 CalibrationAlgoType, 39
 char.t, 35
 DataType, 40

DeviceType, 40
DimensionOperation, 40
Dims, 36
ElementWiseOperation, 41
EngineCapability, 42
EnumMax, 64
EnumMax< BuilderFlag >, 64
EnumMax< CalibrationAlgoType >, 65
EnumMax< DeviceType >, 65
EnumMax< DimensionOperation >, 65
EnumMax< FillOperation >, 65
EnumMax< GatherMode >, 66
EnumMax< LayerInformationFormat >, 66
EnumMax< LayerType >, 66
EnumMax< LoopOutput >, 66
EnumMax< MatrixOperation >, 67
EnumMax< MemoryPoolType >, 67
EnumMax< NetworkDefinitionCreationFlag >, 67
EnumMax< OptProfileSelector >, 67
EnumMax< ProfilingVerbosity >, 68
EnumMax< QuantizationFlag >, 68
EnumMax< ReduceOperation >, 68
EnumMax< RNNDirection >, 68
EnumMax< RNNGateType >, 69
EnumMax< RNNInputMode >, 69
EnumMax< RNNOperation >, 69
EnumMax< ScaleMode >, 69
EnumMax< ScatterMode >, 70
EnumMax< SliceMode >, 70
EnumMax< TacticSource >, 70
EnumMax< TopKOperation >, 70
EnumMax< TripLimit >, 71
EnumMax< UnaryOperation >, 71
EnumMax< WeightsRole >, 71
ErrorCode, 43
FillOperation, 44
GatherMode, 44
getBuilderPluginRegistry, 71
kABS, 63
kACOS, 63
kACOSH, 64
kACTIVATION, 45
kALIGN_CORNERS, 55
kAND, 42
kANY, 64
kASIN, 63
kASINH, 64
kASSERTION, 46
kASYMMETRIC, 55
kATAN, 64
kATANH, 64
kAVERAGE, 53
kAVG, 54
kBIAS, 64
kBIDIRECTION, 56
kBOOL, 40
kCAFFE_ROUND_DOWN, 52
kCAFFE_ROUND_UP, 52
kCALIBRATE_BEFORE_FUSION, 54
kCDHW32, 61
kCEIL, 56, 64
kCEIL_DIV, 41
kCELL, 57
kCHANNEL, 59
kCHAR, 52
kCHW16, 61
kCHW2, 61
kCHW32, 61
kCHW4, 61
kCLAMP, 60
kCLIP, 38
kCONCATENATE, 47
kCONCATENATION, 46
kCONDITION, 46
kCONDITIONAL_INPUT, 46
kCONDITIONAL_OUTPUT, 46
kCONSTANT, 46, 64
kCONVOLUTION, 45
kCOS, 63
kCOSH, 63
kCOUNT, 63
kCUBLAS, 60
kCUBLAS_LT, 60
kCUDNN, 60
kDEBUG, 38
kDECONVOLUTION, 46
kDEFAULT, 42, 45, 53, 60
kDEQUANTIZE, 46
kDETAILED, 53
kDEVICE, 62
kDHWC8, 61
kDIMS, 52
kDIRECT_IO, 39
kDISABLE_TIMING_CACHE, 39
kDIV, 41
kDLA, 40
kDLA_GLOBAL_DRAM, 48
kDLA_HWC4, 62
kDLA_LINEAR, 62
kDLA_LOCAL_DRAM, 48
kDLA_MANAGED_SRAM, 48
kDLA_STANDALONE, 42
kEDGE_MASK_CONVOLUTIONS, 60
kEINSUM, 46
keLEMENT, 45, 59
keLEMENTWISE, 46, 59
kELU, 38
kENTROPY_CALIBRATION, 40

KENTROPY_CALIBRATION_2, 40
 KEQUAL, 41, 42
 KERF, 64
 KEXP, 63
 KEXPLICIT_BATCH, 48
 KEXPLICIT_PRECISION, 48
 KEXPLICIT_ROUND_DOWN, 52
 KEXPLICIT_ROUND_UP, 52
 KFAILED_ALLOCATION, 43
 KFAILED_COMPUTATION, 43
 KFAILED_EXECUTION, 43
 KFAILED_INITIALIZATION, 43
 KFILL, 46, 60
 KFLOAT, 40
 KFLOAT16, 52
 KFLOAT32, 52
 KFLOAT64, 52
 KFLOOR, 56, 64
 KFLOOR_DIV, 41, 42
 KFORGET, 57
 KFORMULA, 56
 KFP16, 38
 KFULLY_CONNECTED, 45
 KGATHER, 46
 KGPU, 40
 KGPU_FALLBACK, 39
 KGREATER, 42
 KGRU, 58
 KHALF, 40
 KHALF_DOWN, 56
 KHALF_PIXEL, 55
 KHALF_UP, 56
 KHARD_SIGMOID, 38
 KHIDDEN, 57
 KHOST, 62
 KHWC, 61
 KHWC16, 62
 KHWC8, 61
 KIDENTITY, 46
 KINPUT, 57
 KINT16, 52
 KINT32, 40, 52
 KINT8, 38, 40, 52
 KINTERNAL_ERROR, 43
 KINVALID_ARGUMENT, 43
 KINVALID_CONFIG, 43
 KINVALID_STATE, 43
 KITERATOR, 46
 KJSON, 45
 KKERNEL, 64
 KLAST_VALUE, 47
 KLAYER_NAMES_ONLY, 53
 KLEAKY_RELU, 38
 KLEGACY_CALIBRATION, 40
 KLESS, 41, 42
 KLINEAR, 55, 57, 61
 KLinspace, 44
 KLOG, 63
 KLOOP_OUTPUT, 46
 KLRN, 45
 KLSTM, 58
 KMATRIX_MULTIPLY, 46
 KMAX, 41, 49, 53, 54, 62
 KMAX_AVERAGE_BLEND, 53
 KMIN, 41, 49, 54, 62
 KMINMAX_CALIBRATION, 40
 KND, 45, 59
 KNEAREST, 55
 KNEG, 63
 KNONE, 47, 53
 KNOT, 64
 KOBAY_PRECISION_CONSTRAINTS, 39
 KONELINE, 45
 KOPT, 49
 KOR, 42
 KOUTPUT, 57
 KPADDING, 46
 KPARAMETRIC_RELU, 46
 KPLUGIN, 46
 KPLUGIN_V2, 46
 KPOOLING, 45
 KPOW, 42
 KPREFER_PRECISION_CONSTRAINTS, 39
 KPROD, 41, 54
 KQUANTIZE, 46
 KRAGGED_SOFTMAX, 46
 KRANDOM_UNIFORM, 44
 KRECIP, 63
 KRECURRENCE, 46
 KREDUCE, 46
 KREFIT, 39
 KREFLECT, 60
 KREJECT_EMPTY_ALGORITHMS, 39
 KRELU, 38, 58
 KRESET, 57
 KRESIZABLE, 38
 KRESIZE, 46
 KREVERSE, 47
 KRNN_V2, 46
 KROUND, 64
 KSAFE_DLA, 42
 KSAFE_GPU, 42
 KSAFETY, 42
 KSAFETY_SCOPE, 39
 KSAME_LOWER, 52
 KSAME_UPPER, 52
 KSCALE, 45, 64
 KSCALED_TANH, 38

- kSCATTER, 46
- kSELECT, 46
- kSELU, 38
- kSHAPE, 46
- kSHIFT, 64
- kSHUFFLE, 46
- kSIGMOID, 38
- kSIGN, 64
- kSIN, 63
- kSINH, 63
- kSKIP, 57
- kSLICE, 46
- kSOFTMAX, 45
- kSOFTPLUS, 38
- kSOFTSIGN, 38
- kSPARSE_WEIGHTS, 39
- kSQRT, 63
- kSTANDARD, 42
- kSTRICT_TYPES, 39
- kSUB, 41
- kSUCCESS, 43
- kSUM, 41, 54
- KTAN, 63
- KTANH, 38, 58
- kTF32, 39
- kTHRESHOLDED_RELU, 38
- kTOPK, 46
- kTRANSPOSE, 47
- kTRIP_LIMIT, 46
- kUNARY, 46
- kUNIDIRECTION, 56
- kUNIFORM, 59
- kUNKNOWN, 52
- kUNSPECIFIED_ERROR, 43
- kUNSUPPORTED_STATE, 44
- kUPDATE, 57
- kUPPER, 56
- kV2, 52
- kV2_DYNAMICEXT, 52
- kV2_EXT, 52
- kV2_IOEXT, 52
- kVECTOR, 47
- kVERBOSE, 53
- kWHILE, 63
- kWORKSPACE, 48
- kWRAP, 60
- kXOR, 42
- LayerInformationFormat, 45
- LayerType, 45
- LoopOutput, 46
- MatrixOperation, 47
- MemoryPoolType, 47
- NetworkDefinitionCreationFlag, 48
- NetworkDefinitionCreationFlags, 36
- OptProfileSelector, 48
- PaddingMode, 49
- PluginFieldType, 52
- PluginFormat, 36
- PluginVersion, 52
- PoolingType, 53
- ProfilingVerbosity, 53
- QuantizationFlag, 53
- QuantizationFlags, 36
- ReduceOperation, 54
- ResizeCoordinateTransformation, 54
- ResizeMode, 55
- ResizeRoundMode, 55
- ResizeSelector, 56
- RNNDirection, 56
- RNNGateType, 56
- RNNInputMode, 57
- RNNOperation, 57
- ScaleMode, 59
- ScatterMode, 59
- SliceMode, 59
- TacticSource, 60
- TacticSources, 37
- TensorFormat, 60
- TensorFormats, 37
- TensorLocation, 62
- TopKOperation, 62
- TripLimit, 63
- UnaryOperation, 63
- WeightsRole, 64
- nvInfer1::consistency, 72
- nvInfer1::consistency::IConsistencyChecker, 173
 - ~IConsistencyChecker, 173
 - IConsistencyChecker, 174
 - mImpl, 175
 - operator=, 174
 - validate, 174
- nvInfer1::consistency::IPluginChecker, 411
 - ~IPluginChecker, 412
 - IPluginChecker, 411, 412
 - operator=, 412
 - validate, 412
- nvInfer1::Dims2, 87
 - Dims2, 87
- nvInfer1::Dims3, 88
 - Dims3, 88, 89
- nvInfer1::Dims32, 89
 - d, 90
 - MAX_DIMS, 90
 - nbDims, 90
- nvInfer1::Dims4, 90
 - Dims4, 91
- nvInfer1::DimsExprs, 92
 - d, 92

- nbDims, 92
- nvinfer1::DimsHW, 93
 - DimsHW, 93, 94
 - h, 94
 - w, 94, 95
- nvinfer1::DynamicPluginTensorDesc, 95
 - desc, 96
 - max, 96
 - min, 96
- nvinfer1::IActivationLayer, 113
 - ~IActivationLayer, 114
 - getActivationType, 114
 - getAlpha, 114
 - getBeta, 115
 - mImpl, 116
 - setActivationType, 115
 - setAlpha, 115
 - setBeta, 116
- nvinfer1::IAlgorithm, 117
 - ~IAlgorithm, 118
 - getAlgorithmIOInfo, 118
 - getAlgorithmIOInfoByIndex, 118
 - getAlgorithmVariant, 119
 - getTimingMSec, 119
 - getWorkspaceSize, 119
 - mImpl, 119
- nvinfer1::IAlgorithmContext, 120
 - ~IAlgorithmContext, 121
 - getDimensions, 121
 - getName, 121
 - getNbInputs, 121
 - getNbOutputs, 121
 - mImpl, 122
- nvinfer1::IAlgorithmIOInfo, 122
 - ~IAlgorithmIOInfo, 123
 - getDataType, 123
 - getStrides, 123
 - getTensorFormat, 124
 - mImpl, 124
- nvinfer1::IAlgorithmSelector, 124
 - ~IAlgorithmSelector, 125
 - reportAlgorithms, 125
 - selectAlgorithms, 125
- nvinfer1::IAlgorithmVariant, 126
 - ~IAlgorithmVariant, 127
 - getImplementation, 127
 - getTactic, 128
 - mImpl, 128
- nvinfer1::IAssertionLayer, 128
 - ~IAssertionLayer, 129
 - getMessage, 129
 - mImpl, 130
 - setMessage, 130
- nvinfer1::IBuilder, 133
 - ~IBuilder, 135
 - buildEngineWithConfig, 135
 - buildSerializedNetwork, 136
 - createBuilderConfig, 136
 - createNetworkV2, 137
 - createOptimizationProfile, 137
 - destroy, 137
 - getErrorRecorder, 138
 - getLogger, 138
 - getMaxBatchSize, 138
 - getMaxDLABatchSize, 139
 - getMaxThreads, 139
 - getNbDLACores, 139
 - isNetworkSupported, 140
 - mImpl, 143
 - platformHasFastFp16, 140
 - platformHasFastInt8, 140
 - platformHasTf32, 141
 - reset, 141
 - setErrorRecorder, 141
 - setGpuAllocator, 141
 - setMaxBatchSize, 142
 - setMaxThreads, 142
- nvinfer1::IBuilderConfig, 143
 - ~IBuilderConfig, 146
 - addOptimizationProfile, 146
 - canRunOnDLA, 147
 - clearFlag, 147
 - clearQuantizationFlag, 147
 - createTimingCache, 147
 - destroy, 148
 - getAlgorithmSelector, 148
 - getAvgTimingIterations, 149
 - getCalibrationProfile, 149
 - getDefaultDeviceType, 149
 - getDeviceType, 149
 - getDLACore, 150
 - getEngineCapability, 150
 - getFlag, 150
 - getFlags, 151
 - getInt8Calibrator, 151
 - getMaxWorkspaceSize, 151
 - getMemoryPoolLimit, 152
 - getMinTimingIterations, 152
 - getNbOptimizationProfiles, 153
 - getProfileStream, 153
 - getProfilingVerbosity, 153
 - getQuantizationFlag, 154
 - getQuantizationFlags, 154
 - getTacticSources, 154
 - getTimingCache, 155
 - isDeviceTypeSet, 155
 - mImpl, 164
 - reset, 155

- resetDeviceType, 156
- setAlgorithmSelector, 156
- setAvgTimingIterations, 156
- setCalibrationProfile, 156
- setDefaultDeviceType, 157
- setDeviceType, 157
- setDLACore, 158
- setEngineCapability, 158
- setFlag, 158
- setFlags, 159
- setInt8Calibrator, 159
- setMaxWorkspaceSize, 159
- setMemoryPoolLimit, 160
- setMinTimingIterations, 161
- setProfileStream, 161
- setProfilingVerbosity, 161
- setQuantizationFlag, 162
- setQuantizationFlags, 162
- setTacticSources, 162
- setTimingCache, 163
- nvInfer1::IConcatenationLayer, 169
 - ~IConcatenationLayer, 170
 - getAxis, 170
 - mImpl, 171
 - setAxis, 171
- nvInfer1::IConditionLayer, 172
 - ~IConditionLayer, 172
 - mImpl, 172
- nvInfer1::IConstantLayer, 175
 - ~IConstantLayer, 176
 - getDimensions, 176
 - getWeights, 176
 - mImpl, 177
 - setDimensions, 177
 - setWeights, 177
- nvInfer1::IConvolutionLayer, 178
 - ~IConvolutionLayer, 180
 - getBiasWeights, 180
 - getDilation, 180
 - getDilationNd, 181
 - getKernelSize, 181
 - getKernelSizeNd, 181
 - getKernelWeights, 182
 - getNbGroups, 182
 - getNbOutputMaps, 182
 - getPadding, 182
 - getPaddingMode, 183
 - getPaddingNd, 183
 - getPostPadding, 183
 - getPrePadding, 184
 - getStride, 184
 - getStrideNd, 184
 - mImpl, 190
 - setBiasWeights, 184
 - setDilation, 185
 - setDilationNd, 185
 - setInput, 185
 - setKernelSize, 186
 - setKernelSizeNd, 186
 - setKernelWeights, 186
 - setNbGroups, 187
 - setNbOutputMaps, 187
 - setPadding, 187
 - setPaddingMode, 188
 - setPaddingNd, 188
 - setPostPadding, 188
 - setPrePadding, 189
 - setStride, 189
 - setStrideNd, 189
- nvInfer1::ICudaEngine, 190
 - ~ICudaEngine, 192
 - bindingIsInput, 193
 - createEngineInspector, 193
 - createExecutionContext, 193
 - createExecutionContextWithoutDeviceMemory, 194
 - destroy, 194
 - getBindingBytesPerComponent, 194
 - getBindingComponentsPerElement, 195
 - getBindingDataType, 195
 - getBindingDimensions, 196
 - getBindingFormat, 196
 - getBindingFormatDesc, 197
 - getBindingIndex, 197
 - getBindingName, 197
 - getBindingVectorizedDim, 198
 - getDeviceMemorySize, 198
 - getEngineCapability, 199
 - getErrorRecorder, 199
 - getLocation, 199
 - getMaxBatchSize, 200
 - getName, 200
 - getNbBindings, 200
 - getNbLayers, 201
 - getNbOptimizationProfiles, 201
 - getProfileDimensions, 201
 - getProfileShapeValues, 202
 - getProfilingVerbosity, 203
 - getTacticSources, 203
 - hasImplicitBatchDimension, 203
 - isExecutionBinding, 203
 - isRefittable, 204
 - isShapeBinding, 204
 - mImpl, 206
 - serialize, 205
 - setErrorRecorder, 205
- nvInfer1::IDeconvolutionLayer, 217
 - ~IDeconvolutionLayer, 219
 - getBiasWeights, 219

- getDilationNd, 219
- getKernelSize, 219
- getKernelSizeNd, 220
- getKernelWeights, 220
- getNbGroups, 220
- getNbOutputMaps, 220
- getPadding, 221
- getPaddingMode, 221
- getPaddingNd, 221
- getPostPadding, 222
- getPrePadding, 222
- getStride, 222
- getStrideNd, 222
- mImpl, 228
- setBiasWeights, 223
- setDilationNd, 223
- setInput, 223
- setKernelSize, 224
- setKernelSizeNd, 224
- setKernelWeights, 225
- setNbGroups, 225
- setNbOutputMaps, 225
- setPadding, 226
- setPaddingMode, 226
- setPaddingNd, 226
- setPostPadding, 227
- setPrePadding, 227
- setStride, 227
- setStrideNd, 228
- nvinfer1::IDequantizeLayer, 229
 - ~IDequantizeLayer, 230
 - getAxis, 230
 - mImpl, 231
 - setAxis, 231
- nvinfer1::IDimensionExpr, 231
 - ~IDimensionExpr, 232
 - getConstantValue, 232
 - isConstant, 233
 - mImpl, 233
- nvinfer1::IEinsumLayer, 233
 - ~IEinsumLayer, 235
 - getEquation, 235
 - mImpl, 235
 - setEquation, 235
- nvinfer1::IElementWiseLayer, 236
 - ~IElementWiseLayer, 237
 - getOperation, 237
 - mImpl, 238
 - setOperation, 237
- nvinfer1::IEngineInspector, 238
 - ~IEngineInspector, 239
 - getEngineInformation, 239
 - getErrorRecorder, 240
 - getExecutionContext, 240
 - getLayerInformation, 241
 - mImpl, 242
 - setErrorRecorder, 241
 - setExecutionContext, 242
- nvinfer1::IErrorRecorder, 243
 - ~IErrorRecorder, 244
 - clear, 245
 - decRefCount, 245
 - ErrorDesc, 244
 - getErrorCode, 245
 - getErrorDesc, 246
 - getNbErrors, 247
 - hasOverflowed, 247
 - IErrorRecorder, 244
 - incRefCount, 248
 - kMAX_DESC_LENGTH, 249
 - RefCount, 244
 - reportError, 248
- nvinfer1::IExecutionContext, 249
 - ~IExecutionContext, 251
 - allInputDimensionsSpecified, 251
 - allInputShapesSpecified, 252
 - destroy, 252
 - enqueue, 253
 - enqueueV2, 253
 - execute, 254
 - executeV2, 255
 - getBindingDimensions, 256
 - getDebugSync, 256
 - getEngine, 256
 - getEnqueueEmitsProfile, 257
 - getErrorRecorder, 257
 - getName, 257
 - getOptimizationProfile, 258
 - getProfiler, 258
 - getShapeBinding, 258
 - getStrides, 259
 - mImpl, 266
 - reportToProfiler, 259
 - setBindingDimensions, 260
 - setDebugSync, 261
 - setDeviceMemory, 261
 - setEnqueueEmitsProfile, 261
 - setErrorRecorder, 262
 - setInputShapeBinding, 262
 - setName, 263
 - setOptimizationProfile, 263
 - setOptimizationProfileAsync, 264
 - setProfiler, 265
- nvinfer1::IExprBuilder, 273
 - ~IExprBuilder, 274
 - constant, 274
 - mImpl, 275
 - operation, 274

- nvinfer1::IFillLayer, 275
 - ~IFillLayer, 277
 - getAlpha, 277
 - getBeta, 277
 - getDimensions, 278
 - getOperation, 278
 - mImpl, 281
 - setAlpha, 278
 - setBeta, 279
 - setDimensions, 279
 - setInput, 280
 - setOperation, 280
- nvinfer1::IFullyConnectedLayer, 281
 - ~IFullyConnectedLayer, 283
 - getBiasWeights, 283
 - getKernelWeights, 283
 - getNbOutputChannels, 283
 - mImpl, 285
 - setBiasWeights, 284
 - setInput, 284
 - setKernelWeights, 285
 - setNbOutputChannels, 285
- nvinfer1::IGatherLayer, 286
 - ~IGatherLayer, 288
 - getGatherAxis, 289
 - getMode, 289
 - getNbElementWiseDims, 289
 - mImpl, 291
 - setGatherAxis, 289
 - setMode, 290
 - setNbElementWiseDims, 290
- nvinfer1::IGpuAllocator, 291
 - ~IGpuAllocator, 291
 - allocate, 292
 - deallocate, 292
 - free, 293
 - IGpuAllocator, 291
 - reallocate, 294
- nvinfer1::IHostMemory, 295
 - ~IHostMemory, 296
 - data, 296
 - destroy, 296
 - mImpl, 297
 - size, 297
 - type, 297
- nvinfer1::IIdentityLayer, 297
 - ~IIdentityLayer, 298
 - mImpl, 298
- nvinfer1::IIfConditional, 299
 - ~IIfConditional, 300
 - addInput, 300
 - addOutput, 301
 - getName, 301
 - mImpl, 302
 - setCondition, 301
 - setName, 302
- nvinfer1::IIfConditionalBoundaryLayer, 302
 - ~IIfConditionalBoundaryLayer, 303
 - getConditional, 303
 - mBoundary, 303
- nvinfer1::IIfConditionalInputLayer, 304
 - ~IIfConditionalInputLayer, 304
 - mImpl, 305
- nvinfer1::IIfConditionalOutputLayer, 305
 - ~IIfConditionalOutputLayer, 306
 - mImpl, 306
- nvinfer1::IInt8Calibrator, 306
 - ~IInt8Calibrator, 307
 - getAlgorithm, 307
 - getBatch, 307
 - getBatchSize, 308
 - readCalibrationCache, 308
 - writeCalibrationCache, 309
- nvinfer1::IInt8EntropyCalibrator, 309
 - ~IInt8EntropyCalibrator, 310
 - getAlgorithm, 310
- nvinfer1::IInt8EntropyCalibrator2, 310
 - ~IInt8EntropyCalibrator2, 311
 - getAlgorithm, 311
- nvinfer1::IInt8LegacyCalibrator, 311
 - ~IInt8LegacyCalibrator, 312
 - getAlgorithm, 312
 - getQuantile, 312
 - getRegressionCutoff, 313
 - readHistogramCache, 313
 - writeHistogramCache, 313
- nvinfer1::IInt8MinMaxCalibrator, 314
 - ~IInt8MinMaxCalibrator, 314
 - getAlgorithm, 315
- nvinfer1::IIteratorLayer, 315
 - ~IIteratorLayer, 316
 - getAxis, 316
 - getReverse, 316
 - mImpl, 317
 - setAxis, 316
 - setReverse, 316
- nvinfer1::ILayer, 317
 - ~ILayer, 319
 - getInput, 319
 - getName, 320
 - getNbInputs, 320
 - getNbOutputs, 320
 - getOutput, 320
 - getOutputType, 320
 - getPrecision, 321
 - getType, 321
 - mLayer, 325
 - outputTypesSet, 321

- precisionIsSet, 322
- resetOutputType, 322
- resetPrecision, 323
- setInput, 323
- setName, 323
- setOutputType, 323
- setPrecision, 324
- nvinfer1::ILogger, 325
 - ~ILogger, 327
 - ILogger, 327
 - kERROR, 326
 - kINFO, 326
 - kINTERNAL_ERROR, 326
 - kVERBOSE, 326
 - kWARNING, 326
 - log, 327
 - Severity, 326
- nvinfer1::ILoop, 328
 - ~ILoop, 328
 - addIterator, 329
 - addLoopOutput, 329
 - addRecurrence, 329
 - addTripLimit, 329
 - getName, 330
 - mImpl, 330
 - setName, 330
- nvinfer1::ILoopBoundaryLayer, 331
 - ~ILoopBoundaryLayer, 331
 - getLoop, 332
 - mBoundary, 332
- nvinfer1::ILoopOutputLayer, 332
 - ~ILoopOutputLayer, 333
 - getAxis, 334
 - getLoopOutput, 334
 - mImpl, 335
 - setAxis, 334
 - setInput, 334
- nvinfer1::ILRNLayer, 335
 - ~ILRNLayer, 336
 - getAlpha, 337
 - getBeta, 337
 - getK, 337
 - getWindowSize, 337
 - mImpl, 339
 - setAlpha, 338
 - setBeta, 338
 - setK, 338
 - setWindowSize, 339
- nvinfer1::IMatrixMultiplyLayer, 340
 - ~IMatrixMultiplyLayer, 341
 - getOperation, 341
 - mImpl, 342
 - setOperation, 341
- nvinfer1::impl, 72
- nvinfer1::impl::EnumMaxImpl< ActivationType >, 97
 - kVALUE, 97
- nvinfer1::impl::EnumMaxImpl< AllocatorFlag >, 97
 - kVALUE, 98
- nvinfer1::impl::EnumMaxImpl< DataType >, 98
 - kVALUE, 98
- nvinfer1::impl::EnumMaxImpl< ElementWiseOperation >, 99
 - kVALUE, 99
- nvinfer1::impl::EnumMaxImpl< EngineCapability >, 100
 - kVALUE, 100
- nvinfer1::impl::EnumMaxImpl< ErrorCode >, 100
 - kVALUE, 101
- nvinfer1::impl::EnumMaxImpl< ILogger::Severity >, 101
 - kVALUE, 102
- nvinfer1::impl::EnumMaxImpl< PaddingMode >, 102
 - kVALUE, 102
- nvinfer1::impl::EnumMaxImpl< PoolingType >, 103
 - kVALUE, 103
- nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation >, 103
 - kVALUE, 104
- nvinfer1::impl::EnumMaxImpl< ResizeMode >, 104
 - kVALUE, 104
- nvinfer1::impl::EnumMaxImpl< ResizeRoundMode >, 105
 - kVALUE, 105
- nvinfer1::impl::EnumMaxImpl< ResizeSelector >, 105
 - kVALUE, 106
- nvinfer1::impl::EnumMaxImpl< T >, 96
- nvinfer1::impl::EnumMaxImpl< TensorFormat >, 106
 - kVALUE, 107
- nvinfer1::impl::EnumMaxImpl< TensorLocation >, 107
 - kVALUE, 107
- nvinfer1::INetworkDefinition, 342
 - ~INetworkDefinition, 346
 - addActivation, 346
 - addAssertion, 347
 - addConcatenation, 347
 - addConstant, 348
 - addConvolution, 348
 - addConvolutionNd, 349
 - addDeconvolution, 350
 - addDeconvolutionNd, 351
 - addDequantize, 352
 - addEinsum, 352
 - addElementWise, 353
 - addFill, 353
 - addFullyConnected, 354
 - addGather, 355
 - addGatherV2, 355
 - addIdentity, 356

- addIfConditional, 356
- addInput, 357
- addLoop, 358
- addLRN, 358
- addMatrixMultiply, 359
- addPadding, 359
- addPaddingNd, 360
- addParametricReLU, 361
- addPluginV2, 361
- addPooling, 362
- addPoolingNd, 363
- addQuantize, 363
- addRaggedSoftMax, 364
- addReduce, 364
- addResize, 365
- addRNNv2, 366
- addScale, 367
- addScaleNd, 368
- addScatter, 369
- addSelect, 370
- addShape, 370
- addShuffle, 372
- addSlice, 372
- addSoftMax, 373
- addTopK, 373
- addUnary, 374
- destroy, 375
- getErrorRecorder, 375
- getInput, 376
- getLayer, 376
- getName, 377
- getNbInputs, 377
- getNbLayers, 377
- getNbOutputs, 378
- getOutput, 378
- hasExplicitPrecision, 379
- hasImplicitBatchDimension, 379
- markOutput, 379
- markOutputForShapes, 380
- mImpl, 383
- removeTensor, 380
- setErrorRecorder, 381
- setName, 381
- setWeightsName, 382
- unmarkOutput, 382
- unmarkOutputForShapes, 383
- nvinfer1::INoCopy, 384
 - ~INoCopy, 385
 - INoCopy, 385
 - operator=, 385
- nvinfer1::IOptimizationProfile, 393
 - ~IOptimizationProfile, 394
 - getDimensions, 394
 - getExtraMemoryTarget, 394
 - getNbShapeValues, 395
 - getShapeValues, 395
 - isValid, 395
 - mImpl, 398
 - setDimensions, 395
 - setExtraMemoryTarget, 396
 - setShapeValues, 397
- nvinfer1::IPaddingLayer, 398
 - ~IPaddingLayer, 399
 - getPostPadding, 400
 - getPostPaddingNd, 400
 - getPrePadding, 400
 - getPrePaddingNd, 400
 - mImpl, 402
 - setPostPadding, 401
 - setPostPaddingNd, 401
 - setPrePadding, 401
 - setPrePaddingNd, 402
- nvinfer1::IParametricReLULayer, 403
 - ~IParametricReLULayer, 403
 - mImpl, 404
- nvinfer1::IPluginCreator, 413
 - ~IPluginCreator, 414
 - createPlugin, 415
 - deserializePlugin, 415
 - getFieldNames, 415
 - getPluginName, 416
 - getPluginNamespace, 416
 - getPluginVersion, 417
 - getTensorRTVersion, 417
 - IPluginCreator, 414
 - setPluginNamespace, 417
- nvinfer1::IPluginRegistry, 420
 - ~IPluginRegistry, 421
 - deregisterCreator, 421
 - getErrorRecorder, 421
 - getPluginCreator, 422
 - getPluginCreatorList, 422
 - registerCreator, 423
 - setErrorRecorder, 423
- nvinfer1::IPluginV2, 424
 - clone, 425
 - configureWithFormat, 426
 - destroy, 427
 - enqueue, 427
 - getNbOutputs, 428
 - getOutputDimensions, 428
 - getPluginNamespace, 429
 - getPluginType, 429
 - getPluginVersion, 430
 - getSerializationSize, 430
 - getTensorRTVersion, 430
 - getWorkspaceSize, 431
 - initialize, 431

- serialize, 432
- setPluginNamespace, 432
- supportsFormat, 433
- terminate, 434
- nvInfer1::IPluginV2DynamicExt, 435
 - ~IPluginV2DynamicExt, 436
 - clone, 436
 - configurePlugin, 436
 - enqueue, 438
 - getOutputDimensions, 438
 - getTensorRTVersion, 439
 - getWorkspaceSize, 439
 - kFORMAT_COMBINATION_LIMIT, 440
 - supportsFormatCombination, 439
- nvInfer1::IPluginV2Ext, 441
 - ~IPluginV2Ext, 442
 - attachToContext, 442
 - canBroadcastInputAcrossBatch, 443
 - clone, 443
 - configurePlugin, 444
 - configureWithFormat, 445
 - detachFromContext, 445
 - getOutputDataType, 445
 - getTensorRTVersion, 446
 - IPluginV2Ext, 442
 - isOutputBroadcastAcrossBatch, 446
- nvInfer1::IPluginV2IOExt, 447
 - configurePlugin, 448
 - getTensorRTVersion, 449
 - supportsFormatCombination, 449
- nvInfer1::IPluginV2Layer, 450
 - ~IPluginV2Layer, 451
 - getPlugin, 451
 - mImpl, 452
- nvInfer1::IPoolingLayer, 452
 - ~IPoolingLayer, 454
 - getAverageCountExcludesPadding, 454
 - getBlendFactor, 454
 - getPadding, 454
 - getPaddingMode, 455
 - getPaddingNd, 455
 - getPoolingType, 455
 - getPostPadding, 456
 - getPrePadding, 456
 - getStride, 456
 - getStrideNd, 456
 - getWindowSize, 457
 - getWindowSizeNd, 457
 - mImpl, 462
 - setAverageCountExcludesPadding, 457
 - setBlendFactor, 458
 - setPadding, 458
 - setPaddingMode, 458
 - setPaddingNd, 459
 - setPoolingType, 459
 - setPostPadding, 459
 - setPrePadding, 460
 - setStride, 460
 - setStrideNd, 460
 - setWindowSize, 461
 - setWindowSizeNd, 461
- nvInfer1::IProfiler, 462
 - ~IProfiler, 462
 - reportLayerTime, 463
- nvInfer1::IQuantizeLayer, 464
 - ~IQuantizeLayer, 465
 - getAxis, 466
 - mImpl, 466
 - setAxis, 466
- nvInfer1::IRaggedSoftMaxLayer, 467
 - ~IRaggedSoftMaxLayer, 467
 - mImpl, 468
- nvInfer1::IRecurrenceLayer, 468
 - ~IRecurrenceLayer, 469
 - mImpl, 469
 - setInput, 469
- nvInfer1::IReduceLayer, 470
 - ~IReduceLayer, 471
 - getKeepDimensions, 471
 - getOperation, 471
 - getReduceAxes, 471
 - mImpl, 473
 - setKeepDimensions, 472
 - setOperation, 472
 - setReduceAxes, 472
- nvInfer1::IRefitter, 473
 - ~IRefitter, 474
 - destroy, 474
 - getAll, 475
 - getAllWeights, 475
 - getDynamicRangeMax, 476
 - getDynamicRangeMin, 476
 - getErrorRecorder, 476
 - getLogger, 476
 - getMaxThreads, 477
 - getMissing, 477
 - getMissingWeights, 478
 - getTensorsWithDynamicRange, 478
 - mImpl, 481
 - refitCudaEngine, 479
 - setDynamicRange, 479
 - setErrorRecorder, 479
 - setMaxThreads, 480
 - setNamedWeights, 480
 - setWeights, 481
- nvInfer1::IResizeLayer, 481
 - ~IResizeLayer, 483
 - getAlignCorners, 484

- getCoordinateTransformation, 484
- getNearestRounding, 484
- getOutputDimensions, 484
- getResizeMode, 485
- getScales, 485
- getSelectorForSinglePixel, 485
- mImpl, 489
- setAlignCorners, 486
- setCoordinateTransformation, 486
- setInput, 486
- setNearestRounding, 487
- setOutputDimensions, 487
- setResizeMode, 488
- setScales, 488
- setSelectorForSinglePixel, 489
- nvinfer1::IRNNv2Layer, 490
 - ~IRNNv2Layer, 491
 - getBiasForGate, 491
 - getCellState, 492
 - getDataLength, 492
 - getDirection, 492
 - getHiddenSize, 492
 - getHiddenState, 493
 - getInputMode, 493
 - getLayerCount, 493
 - getMaxSeqLength, 493
 - getOperation, 493
 - getSequenceLengths, 494
 - getWeightsForGate, 494
 - mImpl, 498
 - setBiasForGate, 494
 - setCellState, 495
 - setDirection, 495
 - setHiddenState, 495
 - setInputMode, 496
 - setOperation, 496
 - setSequenceLengths, 496
 - setWeightsForGate, 497
- nvinfer1::IRuntime, 498
 - ~IRuntime, 499
 - deserializeCudaEngine, 500
 - destroy, 501
 - getDLACore, 501
 - getErrorRecorder, 501
 - getLogger, 501
 - getMaxThreads, 502
 - getNbDLACores, 502
 - mImpl, 504
 - setDLACore, 502
 - setErrorRecorder, 503
 - setGpuAllocator, 503
 - setMaxThreads, 504
- nvinfer1::IScaleLayer, 509
 - ~IScaleLayer, 510
 - getChannelAxis, 511
 - getMode, 511
 - getPower, 511
 - getScale, 511
 - getShift, 512
 - mImpl, 514
 - setChannelAxis, 512
 - setMode, 512
 - setPower, 513
 - setScale, 513
 - setShift, 513
- nvinfer1::IScatterLayer, 514
 - ~IScatterLayer, 516
 - getAxis, 516
 - getMode, 516
 - mImpl, 517
 - setAxis, 516
 - setMode, 516
- nvinfer1::ISelectLayer, 517
 - ~ISelectLayer, 518
 - mImpl, 518
- nvinfer1::IShapeLayer, 518
 - ~IShapeLayer, 519
 - mImpl, 519
- nvinfer1::IShuffleLayer, 520
 - ~IShuffleLayer, 521
 - getFirstTranspose, 521
 - getReshapeDimensions, 521
 - getSecondTranspose, 522
 - getZeroIsPlaceholder, 522
 - mImpl, 525
 - setFirstTranspose, 522
 - setInput, 523
 - setReshapeDimensions, 523
 - setSecondTranspose, 524
 - setZeroIsPlaceholder, 524
- nvinfer1::ISliceLayer, 525
 - ~ISliceLayer, 527
 - getMode, 527
 - getSize, 527
 - getStart, 528
 - getStride, 528
 - mImpl, 531
 - setInput, 528
 - setMode, 529
 - setSize, 529
 - setStart, 530
 - setStride, 530
- nvinfer1::ISoftMaxLayer, 531
 - ~ISoftMaxLayer, 532
 - getAxes, 532
 - mImpl, 533
 - setAxes, 532
- nvinfer1::ITensor, 533

- ~ITensor, 535
- dynamicRangeIsSet, 535
- getAllowedFormats, 536
- getBroadcastAcrossBatch, 536
- getDimensions, 536
- getDynamicRangeMax, 537
- getDynamicRangeMin, 537
- getLocation, 537
- getName, 537
- getType, 538
- isExecutionTensor, 538
- isNetworkInput, 538
- isNetworkOutput, 539
- isShapeTensor, 539
- mImpl, 543
- resetDynamicRange, 539
- setAllowedFormats, 540
- setBroadcastAcrossBatch, 540
- setDimensions, 541
- setDynamicRange, 541
- setLocation, 542
- setName, 542
- setType, 542
- nvinfer1::ITimingCache, 543
 - ~ITimingCache, 544
 - combine, 544
 - mImpl, 546
 - reset, 545
 - serialize, 545
- nvinfer1::ITopKLayer, 546
 - ~ITopKLayer, 547
 - getK, 547
 - getOperation, 547
 - getReduceAxes, 548
 - mImpl, 549
 - setK, 548
 - setOperation, 548
 - setReduceAxes, 548
- nvinfer1::ITripLimitLayer, 549
 - ~ITripLimitLayer, 550
 - getTripLimit, 550
 - mImpl, 550
- nvinfer1::IUnaryLayer, 555
 - ~IUnaryLayer, 556
 - getOperation, 556
 - mImpl, 556
 - setOperation, 556
- nvinfer1::Permutation, 559
 - order, 559
- nvinfer1::plugin, 73
 - CENTER_SIZE, 74
 - CodeTypeSSD, 73
 - CORNER, 74
 - CORNER_SIZE, 74
 - TF_CENTER, 74
- nvinfer1::plugin::DetectionOutputParameters, 83
 - backgroundLabelId, 84
 - codeType, 84
 - confidenceThreshold, 84
 - confSigmoid, 84
 - inputOrder, 85
 - isBatchAgnostic, 85
 - isNormalized, 85
 - keepTopK, 85
 - nmsThreshold, 85
 - numClasses, 85
 - shareLocation, 85
 - topK, 86
 - varianceEncodedInTarget, 86
- nvinfer1::plugin::GridAnchorParameters, 111
 - aspectRatios, 112
 - H, 112
 - maxSize, 112
 - minSize, 112
 - numAspectRatios, 112
 - variance, 112
 - W, 112
- nvinfer1::plugin::NMSPParameters, 557
 - backgroundLabelId, 558
 - iouThreshold, 558
 - isNormalized, 558
 - keepTopK, 558
 - numClasses, 558
 - scoreThreshold, 558
 - shareLocation, 558
 - topK, 559
- nvinfer1::plugin::PriorBoxParameters, 566
 - aspectRatios, 567
 - clip, 567
 - flip, 567
 - imgH, 567
 - imgW, 567
 - maxSize, 568
 - minSize, 568
 - numAspectRatios, 568
 - numMaxSize, 568
 - numMinSize, 568
 - offset, 568
 - stepH, 568
 - stepW, 569
 - variance, 569
- nvinfer1::plugin::Quadruple, 569
 - data, 569
- nvinfer1::plugin::RegionParameters, 570
 - classes, 570
 - coords, 571
 - num, 571
 - smTree, 571

- anchorsRatioCount, 572
 - anchorsScaleCount, 572
 - featureStride, 572
 - iouThreshold, 572
 - minBoxSize, 572
 - nmsMaxOut, 573
 - poolingH, 573
 - poolingW, 573
 - preNmsTop, 573
 - spatialScale, 573
- nvinfer1::plugin::softmaxTree, 573
 - child, 574
 - group, 574
 - groupOffset, 574
 - groups, 574
 - groupSize, 574
 - leaf, 575
 - n, 575
 - name, 575
 - parent, 575
- nvinfer1::PluginField, 560
 - data, 560
 - length, 561
 - name, 561
 - PluginField, 560
 - type, 561
- nvinfer1::PluginFieldCollection, 561
 - fields, 562
 - nbFields, 562
- nvinfer1::PluginRegistrar< T >, 562
 - PluginRegistrar, 563
- nvinfer1::PluginTensorDesc, 564
 - dims, 565
 - format, 565
 - scale, 565
 - type, 565
- nvinfer1::safe, 74
 - createInferRuntime, 74
 - getSafePluginRegistry, 75
- nvinfer1::safe::FloatingPointErrorInformation, 110
 - nbInfErrors, 110
 - nbNanErrors, 110
- nvinfer1::safe::ICudaEngine, 206
 - ~ICudaEngine, 207
 - bindingIsInput, 208
 - createExecutionContext, 208
 - createExecutionContextWithoutDeviceMemory, 209
 - getBindingBytesPerComponent, 209
 - getBindingComponentsPerElement, 210
 - getBindingDataType, 210
 - getBindingDimensions, 211
 - getBindingFormat, 211
 - getBindingIndex, 212
 - getBindingName, 213
 - getBindingVectorizedDim, 213
 - getDeviceMemorySize, 214
 - getErrorRecorder, 214
 - getName, 214
 - getNbBindings, 215
 - ICudaEngine, 207, 208
 - operator=, 215, 216
 - setErrorRecorder, 216
- nvinfer1::safe::IExecutionContext, 266
 - ~IExecutionContext, 267
 - enqueueV2, 267
 - getEngine, 268
 - getErrorBuffer, 268
 - getErrorRecorder, 269
 - getName, 269
 - getStrides, 270
 - IExecutionContext, 267
 - operator=, 270
 - setDeviceMemory, 271
 - setErrorBuffer, 271
 - setErrorRecorder, 272
 - setName, 272
- nvinfer1::safe::IRuntime, 504
 - ~IRuntime, 505
 - deserializeCudaEngine, 506
 - getErrorRecorder, 507
 - IRuntime, 505, 506
 - operator=, 507
 - setErrorRecorder, 508
 - setGpuAllocator, 508
- nvinfer1::safe::PluginRegistrar< T >, 563
 - PluginRegistrar, 564
- nvinfer1::utils, 75
 - reorderSubBuffers, 76
 - reshapeWeights, 77
 - transposeSubBuffers, 78
- nvinfer1::Weights, 575
 - count, 576
 - type, 576
 - values, 576
- NvInferConsistency.h, 626, 627
 - createConsistencyChecker_INTERNAL, 626
- NvInferLegacyDims.h, 628, 629
- NvInferPlugin.h, 630, 636
 - createAnchorGeneratorPlugin, 631
 - createBatchedNMSPlugin, 632
 - createInstanceNormalizationPlugin, 632
 - createNMSPlugin, 633
 - createNormalizePlugin, 633
 - createPriorBoxPlugin, 633
 - createRegionPlugin, 633
 - createReorgPlugin, 634
 - createRPNROIPlugin, 634

- createSplitPlugin, 635
- initLibNvInferPlugins, 635
- NvInferPluginUtils.h, 636, 638
- NvInferRuntime.h, 639, 643
 - getLogger, 642
 - getPluginRegistry, 642
 - REGISTER_TENSORRT_PLUGIN, 642
- NvInferRuntimeCommon.h, 655, 659
 - getInferLibVersion, 658
 - NV_TENSORRT_VERSION, 657
 - TENSORRTAPI, 657
 - TRT_DEPRECATED, 658
 - TRT_DEPRECATED_API, 658
 - TRT_DEPRECATED_ENUM, 658
 - TRTNOEXCEPT, 658
- NvInferSafeRuntime.h, 666, 668
 - REGISTER_SAFE_TENSORRT_PLUGIN, 667
- NvInferVersion.h, 670, 672
 - NV_TENSORRT_BUILD, 670
 - NV_TENSORRT_LWS_MAJOR, 670
 - NV_TENSORRT_LWS_MINOR, 671
 - NV_TENSORRT_LWS_PATCH, 671
 - NV_TENSORRT_MAJOR, 671
 - NV_TENSORRT_MINOR, 671
 - NV_TENSORRT_PATCH, 671
 - NV_TENSORRT_SONAME_MAJOR, 671
 - NV_TENSORRT_SONAME_MINOR, 672
 - NV_TENSORRT_SONAME_PATCH, 672
- NvOnnxConfig.h, 673
- nvonnxparser, 78
 - createONNXConfig, 80
 - EnumMax, 80
 - EnumMax< ErrorCode >, 80
 - ErrorCode, 79
 - kINTERNAL_ERROR, 79
 - kINVALID_GRAPH, 79
 - kINVALID_NODE, 79
 - kINVALID_VALUE, 79
 - kMEM_ALLOC_FAILED, 79
 - kMODEL_DESERIALIZE_FAILED, 79
 - kSUCCESS, 79
 - kUNSUPPORTED_GRAPH, 79
 - kUNSUPPORTED_NODE, 79
- NvOnnxParser.h, 678, 681
 - createNvOnnxParser_INTERNAL, 680
 - getNvOnnxParserVersion, 680
 - NV_ONNX_PARSER_MAJOR, 679
 - NV_ONNX_PARSER_MINOR, 679
 - NV_ONNX_PARSER_PATCH, 680
 - SubGraph.t, 680
 - SubGraphCollection.t, 680
- nvonnxparser::IOnnxConfig, 386
 - ~IOnnxConfig, 387
 - addVerbosity, 387
 - destroy, 388
 - getFullTextFileName, 388
 - getModelDtype, 388
 - getModelFileName, 389
 - getPrintLayerInfo, 389
 - getTextFileName, 389
 - getVerbosityLevel, 390
 - reduceVerbosity, 390
 - setFullTextFileName, 390
 - setModelDtype, 391
 - setModelFileName, 391
 - setPrintLayerInfo, 391
 - setTextFileName, 392
 - setVerbosityLevel, 392
 - Verbosity, 387
- nvonnxparser::IParser, 404
 - ~IParser, 405
 - clearErrors, 405
 - destroy, 405
 - getError, 405
 - getNbErrors, 406
 - parse, 406
 - parseFromFile, 407
 - parseWithWeightDescriptors, 407
 - supportsModel, 407
 - supportsOperator, 408
- nvonnxparser::IParserError, 408
 - ~IParserError, 409
 - code, 409
 - desc, 409
 - file, 410
 - func, 410
 - line, 410
 - node, 410
- nvuffparser, 80
 - createUffParser, 82
 - FieldType, 81
 - kCHAR, 81
 - kDATATYPE, 81
 - kDIMS, 81
 - kFLOAT, 81
 - kINT32, 81
 - kNC, 81
 - kNCHW, 81
 - kNHWC, 81
 - kUNKNOWN, 81
 - shutdownProtobufLibrary, 82
 - UffInputOrder, 81
- NvUffParser.h, 674, 676
 - UFF_REQUIRED_VERSION_MAJOR, 675
 - UFF_REQUIRED_VERSION_MINOR, 675
 - UFF_REQUIRED_VERSION_PATCH, 675
- nvuffparser::FieldCollection, 108
 - fields, 108

- nbFields, [108](#)
- nvuffparser::FieldMap, [108](#)
 - data, [109](#)
 - FieldMap, [109](#)
 - length, [109](#)
 - name, [109](#)
 - type, [109](#)
- nvuffparser::IUffParser, [550](#)
 - ~IUffParser, [551](#)
 - destroy, [552](#)
 - getErrorRecorder, [552](#)
 - getUffRequiredVersionMajor, [552](#)
 - getUffRequiredVersionMinor, [552](#)
 - getUffRequiredVersionPatch, [552](#)
 - parse, [553](#)
 - parseBuffer, [553](#)
 - registerInput, [553](#)
 - registerOutput, [554](#)
 - setErrorRecorder, [554](#)
 - setPluginNamespace, [555](#)
- NvUtils.h, [677](#), [678](#)
- offset
 - nvinfer1::plugin::PriorBoxParameters, [568](#)
- operation
 - nvinfer1::IExprBuilder, [274](#)
- operator=
 - nvinfer1::consistency::IConsistencyChecker, [174](#)
 - nvinfer1::consistency::IPluginChecker, [412](#)
 - nvinfer1::INoCopy, [385](#)
 - nvinfer1::safe::ICudaEngine, [215](#), [216](#)
 - nvinfer1::safe::IExecutionContext, [270](#)
 - nvinfer1::safe::IRuntime, [507](#)
- OptProfileSelector
 - nvinfer1, [48](#)
- order
 - nvinfer1::Permutation, [559](#)
- outputTypeIsSet
 - nvinfer1::ILayer, [321](#)
- PaddingMode
 - nvinfer1, [49](#)
- parent
 - nvinfer1::plugin::softmaxTree, [575](#)
- parse
 - nvcaffeparser1::ICaffeParser, [166](#)
 - nvonnxparser::IParser, [406](#)
 - nvuffparser::IUffParser, [553](#)
- parseBinaryProto
 - nvcaffeparser1::ICaffeParser, [166](#)
- parseBuffer
 - nvuffparser::IUffParser, [553](#)
- parseBuffers
 - nvcaffeparser1::ICaffeParser, [167](#)
- parseFromFile
 - nvonnxparser::IParser, [407](#)
- parseWithWeightDescriptors
 - nvonnxparser::IParser, [407](#)
- platformHasFastFp16
 - nvinfer1::IBuilder, [140](#)
- platformHasFastInt8
 - nvinfer1::IBuilder, [140](#)
- platformHasTf32
 - nvinfer1::IBuilder, [141](#)
- PluginField
 - nvinfer1::PluginField, [560](#)
- PluginFieldType
 - nvinfer1, [52](#)
- PluginFormat
 - nvinfer1, [36](#)
- PluginRegistrar
 - nvinfer1::PluginRegistrar< T >, [563](#)
 - nvinfer1::safe::PluginRegistrar< T >, [564](#)
- PluginVersion, [565](#)
 - nvinfer1, [52](#)
- poolingH
 - nvinfer1::plugin::RPROIParams, [573](#)
- PoolingType
 - nvinfer1, [53](#)
- poolingW
 - nvinfer1::plugin::RPROIParams, [573](#)
- precisionIsSet
 - nvinfer1::ILayer, [322](#)
- preNmsTop
 - nvinfer1::plugin::RPROIParams, [573](#)
- ProfilingVerbosity
 - nvinfer1, [53](#)
- QuantizationFlag
 - nvinfer1, [53](#)
- QuantizationFlags
 - nvinfer1, [36](#)
- readCalibrationCache
 - nvinfer1::IInt8Calibrator, [308](#)
- readHistogramCache
 - nvinfer1::IInt8LegacyCalibrator, [313](#)
- reallocate
 - nvinfer1::IGpuAllocator, [294](#)
- ReduceOperation
 - nvinfer1, [54](#)
- reduceVerbosity
 - nvonnxparser::IOnnxConfig, [390](#)
- RefCount
 - nvinfer1::IErrorRecorder, [244](#)
- refitCudaEngine
 - nvinfer1::IRefitter, [479](#)
- REGISTER_SAFE_TENSORRT_PLUGIN
 - NvInferSafeRuntime.h, [667](#)
- REGISTER_TENSORRT_PLUGIN

- NvInferRuntime.h, 642
- registerCreator
 - nvinfer1::IPluginRegistry, 423
- registerInput
 - nvuffparser::IUffParser, 553
- registerOutput
 - nvuffparser::IUffParser, 554
- removeTensor
 - nvinfer1::INetworkDefinition, 380
- reorderSubBuffers
 - nvinfer1::utils, 76
- reportAlgorithms
 - nvinfer1::IAlgorithmSelector, 125
- reportError
 - nvinfer1::IErrorRecorder, 248
- reportLayerTime
 - nvinfer1::IProfiler, 463
- reportToProfiler
 - nvinfer1::IExecutionContext, 259
- reset
 - nvinfer1::IBuilder, 141
 - nvinfer1::IBuilderConfig, 155
 - nvinfer1::ITimingCache, 545
- resetDeviceType
 - nvinfer1::IBuilderConfig, 156
- resetDynamicRange
 - nvinfer1::ITensor, 539
- resetOutputType
 - nvinfer1::ILayer, 322
- resetPrecision
 - nvinfer1::ILayer, 323
- reshapeWeights
 - nvinfer1::utils, 77
- ResizeCoordinateTransformation
 - nvinfer1, 54
- ResizeMode
 - nvinfer1, 55
- ResizeRoundMode
 - nvinfer1, 55
- ResizeSelector
 - nvinfer1, 56
- RNNDirection
 - nvinfer1, 56
- RNNGateType
 - nvinfer1, 56
- RNNInputMode
 - nvinfer1, 57
- RNNOperation
 - nvinfer1, 57
- scale
 - nvinfer1::PluginTensorDesc, 565
- ScaleMode
 - nvinfer1, 59
- ScatterMode
 - nvinfer1, 59
- scoreThreshold
 - nvinfer1::plugin::NMSPParameters, 558
- selectAlgorithms
 - nvinfer1::IAlgorithmSelector, 125
- serialize
 - nvinfer1::ICudaEngine, 205
 - nvinfer1::IPluginV2, 432
 - nvinfer1::ITimingCache, 545
- setActivationType
 - nvinfer1::IActivationLayer, 115
- setAlgorithmSelector
 - nvinfer1::IBuilderConfig, 156
- setAlignCorners
 - nvinfer1::IResizeLayer, 486
- setAllowedFormats
 - nvinfer1::ITensor, 540
- setAlpha
 - nvinfer1::IActivationLayer, 115
 - nvinfer1::IFillLayer, 278
 - nvinfer1::ILRNLayer, 338
- setAverageCountExcludesPadding
 - nvinfer1::IPoolingLayer, 457
- setAvgTimingIterations
 - nvinfer1::IBuilderConfig, 156
- setAxes
 - nvinfer1::ISoftMaxLayer, 532
- setAxis
 - nvinfer1::IConcatenationLayer, 171
 - nvinfer1::IDequantizeLayer, 231
 - nvinfer1::IIteratorLayer, 316
 - nvinfer1::ILoopOutputLayer, 334
 - nvinfer1::IQuantizeLayer, 466
 - nvinfer1::IScatterLayer, 516
- setBeta
 - nvinfer1::IActivationLayer, 116
 - nvinfer1::IFillLayer, 279
 - nvinfer1::ILRNLayer, 338
- setBiasForGate
 - nvinfer1::IRNNv2Layer, 494
- setBiasWeights
 - nvinfer1::IConvolutionLayer, 184
 - nvinfer1::IDeconvolutionLayer, 223
 - nvinfer1::IFullyConnectedLayer, 284
- setBindingDimensions
 - nvinfer1::IExecutionContext, 260
- setBlendFactor
 - nvinfer1::IPoolingLayer, 458
- setBroadcastAcrossBatch
 - nvinfer1::ITensor, 540
- setCalibrationProfile
 - nvinfer1::IBuilderConfig, 156
- setCellState

- nvinfer1::IRNNv2Layer, 495
- setChannelAxis
 - nvinfer1::IScaleLayer, 512
- setCondition
 - nvinfer1::IIfConditional, 301
- setCoordinateTransformation
 - nvinfer1::IResizeLayer, 486
- setDebugSync
 - nvinfer1::IExecutionContext, 261
- setDefaultDeviceType
 - nvinfer1::IBuilderConfig, 157
- setDeviceMemory
 - nvinfer1::IExecutionContext, 261
 - nvinfer1::safe::IExecutionContext, 271
- setDeviceType
 - nvinfer1::IBuilderConfig, 157
- setDilation
 - nvinfer1::IConvolutionLayer, 185
- setDilationNd
 - nvinfer1::IConvolutionLayer, 185
 - nvinfer1::IDeconvolutionLayer, 223
- setDimensions
 - nvinfer1::IConstantLayer, 177
 - nvinfer1::IFillLayer, 279
 - nvinfer1::IOptimizationProfile, 395
 - nvinfer1::ITensor, 541
- setDirection
 - nvinfer1::IRNNv2Layer, 495
- setDLACore
 - nvinfer1::IBuilderConfig, 158
 - nvinfer1::IRuntime, 502
- setDynamicRange
 - nvinfer1::IRefitter, 479
 - nvinfer1::ITensor, 541
- setEngineCapability
 - nvinfer1::IBuilderConfig, 158
- setEnqueueEmitsProfile
 - nvinfer1::IExecutionContext, 261
- setEquation
 - nvinfer1::IEinsumLayer, 235
- setErrorBuffer
 - nvinfer1::safe::IExecutionContext, 271
- setErrorRecorder
 - nvcaffeparser1::ICaffeParser, 168
 - nvinfer1::IBuilder, 141
 - nvinfer1::ICudaEngine, 205
 - nvinfer1::IEngineInspector, 241
 - nvinfer1::IExecutionContext, 262
 - nvinfer1::INetworkDefinition, 381
 - nvinfer1::IPluginRegistry, 423
 - nvinfer1::IRefitter, 479
 - nvinfer1::IRuntime, 503
 - nvinfer1::safe::ICudaEngine, 216
 - nvinfer1::safe::IExecutionContext, 272
 - nvinfer1::safe::IRuntime, 508
 - nvuffparser::IUffParser, 554
- setExecutionContext
 - nvinfer1::IEngineInspector, 242
- setExtraMemoryTarget
 - nvinfer1::IOptimizationProfile, 396
- setFirstTranspose
 - nvinfer1::IShuffleLayer, 522
- setFlag
 - nvinfer1::IBuilderConfig, 158
- setFlags
 - nvinfer1::IBuilderConfig, 159
- setFullTextFileName
 - nvonnxparser::IOnnxConfig, 390
- setGatherAxis
 - nvinfer1::IGatherLayer, 289
- setGpuAllocator
 - nvinfer1::IBuilder, 141
 - nvinfer1::IRuntime, 503
 - nvinfer1::safe::IRuntime, 508
- setHiddenState
 - nvinfer1::IRNNv2Layer, 495
- setInput
 - nvinfer1::IConvolutionLayer, 185
 - nvinfer1::IDeconvolutionLayer, 223
 - nvinfer1::IFillLayer, 280
 - nvinfer1::IFullyConnectedLayer, 284
 - nvinfer1::ILayer, 323
 - nvinfer1::ILoopOutputLayer, 334
 - nvinfer1::IRecurrenceLayer, 469
 - nvinfer1::IResizeLayer, 486
 - nvinfer1::IShuffleLayer, 523
 - nvinfer1::ISliceLayer, 528
- setInputMode
 - nvinfer1::IRNNv2Layer, 496
- setInputShapeBinding
 - nvinfer1::IExecutionContext, 262
- setInt8Calibrator
 - nvinfer1::IBuilderConfig, 159
- setK
 - nvinfer1::ILRNLayer, 338
 - nvinfer1::ITopKLayer, 548
- setKeepDimensions
 - nvinfer1::IReduceLayer, 472
- setKernelSize
 - nvinfer1::IConvolutionLayer, 186
 - nvinfer1::IDeconvolutionLayer, 224
- setKernelSizeNd
 - nvinfer1::IConvolutionLayer, 186
 - nvinfer1::IDeconvolutionLayer, 224
- setKernelWeights
 - nvinfer1::IConvolutionLayer, 186
 - nvinfer1::IDeconvolutionLayer, 225
 - nvinfer1::IFullyConnectedLayer, 285

- setLocation
 - nvinfer1::ITensor, 542
- setMaxBatchSize
 - nvinfer1::IBuilder, 142
- setMaxThreads
 - nvinfer1::IBuilder, 142
 - nvinfer1::IRefitter, 480
 - nvinfer1::IRuntime, 504
- setMaxWorkspaceSize
 - nvinfer1::IBuilderConfig, 159
- setMemoryPoolLimit
 - nvinfer1::IBuilderConfig, 160
- setMessage
 - nvinfer1::IAssertionLayer, 130
- setMinTimingIterations
 - nvinfer1::IBuilderConfig, 161
- setMode
 - nvinfer1::IGatherLayer, 290
 - nvinfer1::IScaleLayer, 512
 - nvinfer1::IScatterLayer, 516
 - nvinfer1::ISliceLayer, 529
- setModelDtype
 - nvonnxparser::IOnnxConfig, 391
- setModelFileName
 - nvonnxparser::IOnnxConfig, 391
- setName
 - nvinfer1::IExecutionContext, 263
 - nvinfer1::IIfConditional, 302
 - nvinfer1::ILayer, 323
 - nvinfer1::ILoop, 330
 - nvinfer1::INetworkDefinition, 381
 - nvinfer1::ITensor, 542
 - nvinfer1::safe::IExecutionContext, 272
- setNamedWeights
 - nvinfer1::IRefitter, 480
- setNbElementWiseDims
 - nvinfer1::IGatherLayer, 290
- setNbGroups
 - nvinfer1::IConvolutionLayer, 187
 - nvinfer1::IDeconvolutionLayer, 225
- setNbOutputChannels
 - nvinfer1::IFullyConnectedLayer, 285
- setNbOutputMaps
 - nvinfer1::IConvolutionLayer, 187
 - nvinfer1::IDeconvolutionLayer, 225
- setNearestRounding
 - nvinfer1::IResizeLayer, 487
- setOperation
 - nvinfer1::IElementWiseLayer, 237
 - nvinfer1::IFillLayer, 280
 - nvinfer1::IMatrixMultiplyLayer, 341
 - nvinfer1::IReduceLayer, 472
 - nvinfer1::IRNNv2Layer, 496
 - nvinfer1::ITopKLayer, 548
 - nvinfer1::UnaryLayer, 556
- setOptimizationProfile
 - nvinfer1::IExecutionContext, 263
- setOptimizationProfileAsync
 - nvinfer1::IExecutionContext, 264
- setOutputDimensions
 - nvinfer1::IResizeLayer, 487
- setOutputType
 - nvinfer1::ILayer, 323
- setPadding
 - nvinfer1::IConvolutionLayer, 187
 - nvinfer1::IDeconvolutionLayer, 226
 - nvinfer1::IPoolingLayer, 458
- setPaddingMode
 - nvinfer1::IConvolutionLayer, 188
 - nvinfer1::IDeconvolutionLayer, 226
 - nvinfer1::IPoolingLayer, 458
- setPaddingNd
 - nvinfer1::IConvolutionLayer, 188
 - nvinfer1::IDeconvolutionLayer, 226
 - nvinfer1::IPoolingLayer, 459
- setPluginFactoryV2
 - nvcaffeparser1::ICaffeParser, 168
- setPluginNamespace
 - nvcaffeparser1::ICaffeParser, 168
 - nvinfer1::IPluginCreator, 417
 - nvinfer1::IPluginV2, 432
 - nvuffparser::IUffParser, 555
- setPoolingType
 - nvinfer1::IPoolingLayer, 459
- setPostPadding
 - nvinfer1::IConvolutionLayer, 188
 - nvinfer1::IDeconvolutionLayer, 227
 - nvinfer1::IPaddingLayer, 401
 - nvinfer1::IPoolingLayer, 459
- setPostPaddingNd
 - nvinfer1::IPaddingLayer, 401
- setPower
 - nvinfer1::IScaleLayer, 513
- setPrecision
 - nvinfer1::ILayer, 324
- setPrePadding
 - nvinfer1::IConvolutionLayer, 189
 - nvinfer1::IDeconvolutionLayer, 227
 - nvinfer1::IPaddingLayer, 401
 - nvinfer1::IPoolingLayer, 460
- setPrePaddingNd
 - nvinfer1::IPaddingLayer, 402
- setPrintLayerInfo
 - nvonnxparser::IOnnxConfig, 391
- setProfiler
 - nvinfer1::IExecutionContext, 265
- setProfileStream
 - nvinfer1::IBuilderConfig, 161

- setProfilingVerbosity
 - nvinfer1::IBuilderConfig, 161
- setProtobufBufferSize
 - nvcaffeparser1::ICaffeParser, 169
- setQuantizationFlag
 - nvinfer1::IBuilderConfig, 162
- setQuantizationFlags
 - nvinfer1::IBuilderConfig, 162
- setReduceAxes
 - nvinfer1::IReduceLayer, 472
 - nvinfer1::ITopKLayer, 548
- setReshapeDimensions
 - nvinfer1::IShuffleLayer, 523
- setResizeMode
 - nvinfer1::IResizeLayer, 488
- setReverse
 - nvinfer1::IiteratorLayer, 316
- setScale
 - nvinfer1::IScaleLayer, 513
- setScales
 - nvinfer1::IResizeLayer, 488
- setSecondTranspose
 - nvinfer1::IShuffleLayer, 524
- setSelectorForSinglePixel
 - nvinfer1::IResizeLayer, 489
- setSequenceLengths
 - nvinfer1::IRNNv2Layer, 496
- setShapeValues
 - nvinfer1::IOptimizationProfile, 397
- setShift
 - nvinfer1::IScaleLayer, 513
- setSize
 - nvinfer1::ISliceLayer, 529
- setStart
 - nvinfer1::ISliceLayer, 530
- setStride
 - nvinfer1::IConvolutionLayer, 189
 - nvinfer1::IDeconvolutionLayer, 227
 - nvinfer1::IPoolingLayer, 460
 - nvinfer1::ISliceLayer, 530
- setStrideNd
 - nvinfer1::IConvolutionLayer, 189
 - nvinfer1::IDeconvolutionLayer, 228
 - nvinfer1::IPoolingLayer, 460
- setTacticSources
 - nvinfer1::IBuilderConfig, 162
- setTextFileName
 - nvonnxparser::IOnnxConfig, 392
- setTimingCache
 - nvinfer1::IBuilderConfig, 163
- setType
 - nvinfer1::ITensor, 542
- setVerbosityLevel
 - nvonnxparser::IOnnxConfig, 392
- setWeights
 - nvinfer1::IConstantLayer, 177
 - nvinfer1::IRefitter, 481
- setWeightsForGate
 - nvinfer1::IRNNv2Layer, 497
- setWeightsName
 - nvinfer1::INetworkDefinition, 382
- setWindowSize
 - nvinfer1::ILRNLayer, 339
 - nvinfer1::IPoolingLayer, 461
- setWindowSizeNd
 - nvinfer1::IPoolingLayer, 461
- setZeroIsPlaceholder
 - nvinfer1::IShuffleLayer, 524
- Severity
 - nvinfer1::ILogger, 326
- shareLocation
 - nvinfer1::plugin::DetectionOutputParameters, 85
 - nvinfer1::plugin::NMSPParameters, 558
- shutdownProtobufLibrary
 - nvcaffeparser1, 26
 - nvuffparser, 82
- size
 - nvinfer1::IHostMemory, 297
- SliceMode
 - nvinfer1, 59
- smTree
 - nvinfer1::plugin::RegionParameters, 571
- spatialScale
 - nvinfer1::plugin::RPROIPParams, 573
- stepH
 - nvinfer1::plugin::PriorBoxParameters, 568
- stepW
 - nvinfer1::plugin::PriorBoxParameters, 569
- SubGraph_t
 - NvOnnxParser.h, 680
- SubGraphCollection_t
 - NvOnnxParser.h, 680
- supportsFormat
 - nvinfer1::IPluginV2, 433
- supportsFormatCombination
 - nvinfer1::IPluginV2DynamicExt, 439
 - nvinfer1::IPluginV2IOExt, 449
- supportsModel
 - nvonnxparser::IParser, 407
- supportsOperator
 - nvonnxparser::IParser, 408
- TacticSource
 - nvinfer1, 60
- TacticSources
 - nvinfer1, 37
- TensorFormat
 - nvinfer1, 60

TensorFormats
 nvinfer1, 37

TensorLocation
 nvinfer1, 62

TENSORRTAPI
 NvInferRuntimeCommon.h, 657

terminate
 nvinfer1::IPluginV2, 434

TF_CENTER
 nvinfer1::plugin, 74

topK
 nvinfer1::plugin::DetectionOutputParameters, 86
 nvinfer1::plugin::NMSParameters, 559

TopKOperation
 nvinfer1, 62

transposeSubBuffers
 nvinfer1::utils, 78

TripLimit
 nvinfer1, 63

TRT_DEPRECATED
 NvInferRuntimeCommon.h, 658

TRT_DEPRECATED_API
 NvInferRuntimeCommon.h, 658

TRT_DEPRECATED_ENUM
 NvInferRuntimeCommon.h, 658

TRTNOEXCEPT
 NvInferRuntimeCommon.h, 658

type
 nvinfer1::IHostMemory, 297
 nvinfer1::PluginField, 561
 nvinfer1::PluginTensorDesc, 565
 nvinfer1::Weights, 576
 nvuffparser::FieldMap, 109

UFF_REQUIRED_VERSION_MAJOR
 NvUffParser.h, 675

UFF_REQUIRED_VERSION_MINOR
 NvUffParser.h, 675

UFF_REQUIRED_VERSION_PATCH
 NvUffParser.h, 675

UffInputOrder
 nvuffparser, 81

UnaryOperation
 nvinfer1, 63

unmarkOutput
 nvinfer1::INetworkDefinition, 382

unmarkOutputForShapes
 nvinfer1::INetworkDefinition, 383

validate
 nvinfer1::consistency::IConsistencyChecker, 174
 nvinfer1::consistency::IPluginChecker, 412

values
 nvinfer1::Weights, 576

variance
 nvinfer1::plugin::GridAnchorParameters, 112
 nvinfer1::plugin::PriorBoxParameters, 569

varianceEncodedInTarget
 nvinfer1::plugin::DetectionOutputParameters, 86

Verbosity
 nvonnxparser::IOnnxConfig, 387

W
 nvinfer1::plugin::GridAnchorParameters, 112

w
 nvinfer1::DimsHW, 94, 95

WeightsRole
 nvinfer1, 64

writeCalibrationCache
 nvinfer1::IInt8Calibrator, 309

writeHistogramCache
 nvinfer1::IInt8LegacyCalibrator, 313

NVIDIA TensorRT Standard Python API Documentation

Release 8.4.11

NVIDIA

Jun 07, 2022

TENSORRT PYTHON API REFERENCE

1	Getting Started with TensorRT	1
1.1	Installation	1
1.2	Samples	1
1.3	Installing PyCUDA	1
2	Core Concepts	3
2.1	TensorRT Workflow	3
2.2	Classes Overview	3
2.2.1	Logger	3
2.2.2	Parsers	3
2.2.3	Network	3
2.2.4	Builder	4
2.2.5	Engine and Context	4
3	Foundational Types	5
3.1	DataType	5
3.2	Weights	6
3.3	Dims	6
3.3.1	Volume	6
3.3.2	Dims	7
3.3.3	Dims2	7
3.3.4	DimsHW	7
3.3.5	Dims3	8
3.3.6	Dims4	8
3.4	IHostMemory	8
4	Core	9
4.1	Logger	9
4.2	Profiler	10
4.3	IOptimizationProfile	11
4.4	IBuilderConfig	13
4.5	Builder	19
4.5.1	NetworkDefinitionCreationFlag	19
4.5.2	Builder	19
4.6	ICudaEngine	21
4.7	IExecutionContext	26
4.8	Runtime	30
4.9	Refitter	31
4.10	IErrorRecorder	33
4.11	ITimingCache	36

4.12	GPU Allocator	36
4.12.1	AllocatorFlag	36
4.12.2	IGpuAllocator	36
4.13	EngineInspector	38
5	Network	39
5.1	INetworkDefinition	39
5.2	Layer Base Classes	51
5.2.1	ITensor	51
5.2.2	ILayer	54
5.3	Layers	56
5.3.1	PaddingMode	56
5.3.2	IConvolutionLayer	57
5.3.3	IFullyConnectedLayer	58
5.3.4	IActivationLayer	58
5.3.5	IPoolingLayer	59
5.3.6	ILRNLayer	60
5.3.7	IScaleLayer	60
5.3.8	ISoftMaxLayer	61
5.3.9	IConcatenationLayer	62
5.3.10	IDEconvolutionLayer	62
5.3.11	IElementWiseLayer	63
5.3.12	IGatherLayer	63
5.3.13	RNN Layers	64
5.3.13.1	IRNNv2Layer	66
5.3.14	IPluginV2Layer	68
5.3.15	IUnaryLayer	68
5.3.16	IReduceLayer	69
5.3.17	IPaddingLayer	69
5.3.18	IParametricReLULayer	70
5.3.19	ISelectLayer	70
5.3.20	IShuffleLayer	70
5.3.21	ISliceLayer	71
5.3.22	IShapeLayer	72
5.3.23	ITopKLayer	73
5.3.24	IMatrixMultiplyLayer	73
5.3.25	IRaggedSoftMaxLayer	74
5.3.26	IIdentityLayer	74
5.3.27	IConstantLayer	74
5.3.28	IResizeLayer	74
5.3.29	ILoop	76
5.3.29.1	ILoopBoundaryLayer	77
5.3.29.1.1	ITripLimitLayer	77
5.3.29.1.2	IRecurrenceLayer	77
5.3.29.1.3	IIteratorLayer	77
5.3.29.1.4	ILoopOutputLayer	78
5.3.30	IFillLayer	78
5.3.31	IQuantizeLayer	79
5.3.32	IDequantizeLayer	80
5.3.33	IScatterLayer	80
5.3.34	IIfConditional	80
5.3.35	IConditionLayer	81
5.3.36	IIfConditionalOutputLayer	81
5.3.37	IIfConditionalInputLayer	81

5.3.38	IEinsumLayer	82
5.3.39	IAssertionLayer	82
6	Plugin	83
6.1	IPluginCreator	83
6.2	IPluginRegistry	85
7	Int8	87
7.1	IInt8Calibrator	87
7.2	IInt8LegacyCalibrator	89
7.3	IInt8EntropyCalibrator	90
7.4	IInt8EntropyCalibrator2	92
7.5	IInt8MinMaxCalibrator	93
8	Algorithm Selector	95
9	UFF Parser	99
9.1	Fields	100
10	Caffe Parser	103
10.1	Plugins	104
11	Onnx Parser	105
12	UFF Converter	109
12.1	Conversion Tools	109
12.1.1	Tensorflow Modelstream to UFF	109
12.1.2	Tensorflow Frozen Protobuf Model to UFF	110
13	UFF Operators	111
13.1	Input	111
13.1.1	Supported Datatypes	111
13.2	Identity	111
13.2.1	Inputs	111
13.2.2	Supported Datatypes	111
13.3	Const	111
13.3.1	Supported Datatypes	112
13.4	Conv	112
13.4.1	Inputs	112
13.4.2	Attributes	112
13.4.3	Supported Datatypes	112
13.5	ConvTranspose	112
13.5.1	Inputs	112
13.5.2	Attributes	113
13.5.3	Supported Datatypes	113
13.6	Pool	113
13.6.1	Inputs	113
13.6.2	Attributes	113
13.6.3	Supported Datatypes	113
13.7	FullyConnected	113
13.7.1	Inputs	113
13.7.2	Supported Datatypes	114
13.8	LRN	114
13.8.1	Inputs	114
13.8.2	Attributes	114

13.8.3	Supported Datatypes	114
13.9	Binary	114
13.9.1	Inputs	114
13.9.2	Attributes	114
13.9.3	Supported Datatypes	115
13.10	Unary	115
13.10.1	Inputs	115
13.10.2	Attributes	115
13.10.3	Supported Datatypes	115
13.11	Reshape	115
13.11.1	Inputs	115
13.11.2	Supported Datatypes	116
13.12	ExpandDims	116
13.12.1	Inputs	116
13.12.2	Attributes	116
13.12.3	Supported Datatypes	116
13.13	ArgMax	116
13.13.1	Inputs	116
13.13.2	Attributes	116
13.13.3	Supported Datatypes	117
13.14	ArgMin	117
13.14.1	Inputs	117
13.14.2	Attributes	117
13.14.3	Supported Datatypes	117
13.15	Transpose	117
13.15.1	Inputs	117
13.15.2	Attributes	117
13.15.3	Supported Datatypes	117
13.16	Reduce	118
13.16.1	Inputs	118
13.16.2	Attributes	118
13.16.3	Supported Datatypes	118
13.17	Concat	118
13.17.1	Inputs	118
13.17.2	Attributes	118
13.17.3	Supported Datatypes	119
13.18	MarkOutput	119
13.18.1	Inputs	119
13.18.2	Supported Datatypes	119
13.19	Activation	119
13.19.1	Inputs	119
13.19.2	Attributes	119
13.19.3	Supported Datatypes	119
13.20	Softmax	119
13.20.1	Inputs	120
13.20.2	Attributes	120
13.20.3	Supported Datatypes	120
13.21	BatchNorm	120
13.21.1	Inputs	120
13.21.2	Attributes	120
13.21.3	Supported Datatypes	120
13.22	Shape	120
13.22.1	Inputs	121
13.22.2	Supported Datatypes	121

13.23	StridedSlice	121
13.23.1	Inputs	121
13.23.2	Attributes	121
13.23.3	Supported Datatypes	121
13.24	Stack	121
13.24.1	Inputs	122
13.24.2	Attributes	122
13.24.3	Supported Datatypes	122
13.25	Squeeze	122
13.26	Flatten	122
13.26.1	Inputs	122
13.26.2	Supported Datatypes	122
13.27	Pad	122
13.27.1	Inputs	122
13.27.2	Supported Datatypes	123
13.28	Gather	123
13.28.1	Inputs	123
13.28.2	Supported Datatypes	123
13.29	GatherV2	123
13.29.1	Inputs	123
13.29.2	Attributes	123
13.29.3	Supported Datatypes	123
14	Graph Surgeon	125
14.1	Node Creation	125
14.2	Static Graph	126
14.3	Dynamic Graph (Inherits from StaticGraph)	128
	Index	131

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.

itemsize `int` The size in bytes of this *DataType* .

Members:

FLOAT : Represents a 32-bit floating point number.

HALF : Represents a 16-bit floating point number.

INT8 : Represents an 8-bit integer.

INT32 : Represents a 32-bit integer.

BOOL : Represents a boolean.

TensorRT also exposes some short-hand, NumPy-style *DataType* aliases that can be used across the library:

Type	Alias
<code>tensorrt.DataType.FLOAT</code>	<code>tensorrt.float32</code>
<code>tensorrt.DataType.HALF</code>	<code>tensorrt.float16</code>
<code>tensorrt.DataType.INT32</code>	<code>tensorrt.int32</code>
<code>tensorrt.DataType.INT8</code>	<code>tensorrt.int8</code>
<code>tensorrt.DataType.BOOL</code>	<code>tensorrt.bool</code>

`tensorrt.nptype(trt_type)`

Returns the numpy-equivalent of a TensorRT *DataType* .

Parameters `trt_type` – The TensorRT data type to convert.

Returns The equivalent numpy type.

3.2 Weights

tensorrt.WeightsRole

How a layer uses particular Weights. The power weights of an `IScaleLayer` are omitted. Refitting those is not supported.

Members:

KERNEL : Kernel for `IConvolutionLayer` , `IDeconvolutionLayer` , or `IFullyConnectedLayer` .

BIAS : Bias for `IConvolutionLayer` , `IDeconvolutionLayer` , or `IFullyConnectedLayer` .

SHIFT : Shift part of `IScaleLayer` .

SCALE : Scale part of `IScaleLayer` .

CONSTANT : Weights for `IConstantLayer` .

ANY : Any other weights role.

class tensorrt.Weights(*args, **kwargs)

An array of weights used as a layer parameter. The weights are held by reference until the engine has been built - deep copies are not made automatically.

Variables

- **dtype** – `DataType` The type of the weights.
- **size** – `int` The number of weights in the array.
- **nbytes** – `int` Total bytes consumed by the elements of the weights buffer.

Overloaded function.

1. `__init__(self: tensorrt.tensorrt.Weights, type: tensorrt.tensorrt.DataType = <DataType.FLOAT: 0>) -> None`

Initializes an empty (0-length) Weights object with the specified type.

type A type to initialize the weights with. Default: `tensorrt.float32`

2. `__init__(self: tensorrt.tensorrt.Weights, a: numpy.ndarray) -> None`

a A numpy array whose values to use. No deep copies are made.

numpy(*self*: `tensorrt.tensorrt.Weights`) → `numpy.ndarray`

Create a numpy array using the underlying buffer of this weights object.

Returns A new numpy array that holds a reference to this weight object's buffer - no deep copy is made.

3.3 Dims

3.3.1 Volume

tensorrt.volume(iterable)

Computes the volume of an iterable.

Parameters **iterable** – Any python iterable, including a `Dims` object.

Returns The volume of the iterable. This will return 1 for empty iterables, as a scalar has an empty shape and the volume of a tensor with empty shape is 1.

3.3.2 Dims

class `tensorrt.Dims(*args, **kwargs)`

Structure to define the dimensions of a tensor. *Dims* and all derived classes behave like Python `tuple`s. Furthermore, the TensorRT API can implicitly convert Python iterables to *Dims* objects, so `tuple` or `list` can be used in place of this class.

Overloaded function.

1. `__init__(self: tensorrt.tensorrt.Dims) -> None`
2. `__init__(self: tensorrt.tensorrt.Dims, shape: List[int]) -> None`

property `MAX_DIMS`

The maximum number of dimensions supported by *Dims*.

3.3.3 Dims2

class `tensorrt.Dims2(*args, **kwargs)`

Structure to define 2D shape.

Overloaded function.

1. `__init__(self: tensorrt.tensorrt.Dims2) -> None`
2. `__init__(self: tensorrt.tensorrt.Dims2, dim0: int, dim1: int) -> None`
3. `__init__(self: tensorrt.tensorrt.Dims2, shape: List[int]) -> None`

3.3.4 DimsHW

class `tensorrt.DimsHW(*args, **kwargs)`

Structure to define 2D shape with height and width.

Variables

- **h** – `int` The first dimension (height).
- **w** – `int` The second dimension (width).

Overloaded function.

1. `__init__(self: tensorrt.tensorrt.DimsHW) -> None`
2. `__init__(self: tensorrt.tensorrt.DimsHW, h: int, w: int) -> None`
3. `__init__(self: tensorrt.tensorrt.DimsHW, shape: List[int]) -> None`

3.3.5 Dims3

class `tensorrt.Dims3(*args, **kwargs)`

Structure to define 3D shape.

Overloaded function.

1. `__init__(self: tensorrt.tensorrt.Dims3) -> None`
2. `__init__(self: tensorrt.tensorrt.Dims3, dim0: int, dim1: int, dim2: int) -> None`
3. `__init__(self: tensorrt.tensorrt.Dims3, shape: List[int]) -> None`

3.3.6 Dims4

class `tensorrt.Dims4(*args, **kwargs)`

Structure to define 4D tensor.

Overloaded function.

1. `__init__(self: tensorrt.tensorrt.Dims4) -> None`
2. `__init__(self: tensorrt.tensorrt.Dims4, dim0: int, dim1: int, dim2: int, dim3: int) -> None`
3. `__init__(self: tensorrt.tensorrt.Dims4, shape: List[int]) -> None`

3.4 IHostMemory

class `tensorrt.IHostMemory`

Handles library allocated memory that is accessible to the user.

The memory allocated via the host memory object is owned by the library and will be de-allocated when object is destroyed.

This class exposes a buffer interface using Python's buffer protocol.

Variables

- **dtype** – *DataType* The data type of this buffer.
- **nbytes** – *int* Total bytes consumed by the elements of the buffer.

`__del__(self: tensorrt.tensorrt.IHostMemory) → None`

`__exit__(exc_type, exc_value, traceback)`

Context managers are deprecated and have no effect. Objects are automatically freed when the reference count reaches 0.

`__init__(*args, **kwargs)`

4.1 Logger

class `tensorrt.ILogger`(*self*: `tensorrt.tensorrt.ILogger`) → None
Abstract base Logger class for the *Builder*, *ICudaEngine* and *Runtime* .

To implement a custom logger, ensure that you explicitly instantiate the base class in `__init__()` :

```
class MyLogger(trt.ILogger):
    def __init__(self):
        trt.ILogger.__init__(self)

    def log(self, severity, msg):
        ... # Your implementation here
```

Parameters `min_severity` – The initial minimum severity of this Logger.

Variables `min_severity` – `Logger.Severity` This minimum required severity of messages for the logger to log them.

The logger used to create an instance of `IBuilder`, `IRuntime` or `IRefitter` is used for all objects created through that interface. The logger should be valid until all objects created are released.

class `Severity`(*self*: `tensorrt.tensorrt.ILogger.Severity`, *value*: `int`) → None

Indicates the severity of a message. The values in this enum are also accessible in the `ILogger` directly. For example, `tensorrt.ILogger.INFO` corresponds to `tensorrt.ILogger.Severity.INFO` .

Members:

INTERNAL_ERROR : Represents an internal error. Execution is unrecoverable.

ERROR : Represents an application error.

WARNING : Represents an application error that TensorRT has recovered from or fallen back to a default.

INFO : Represents informational messages.

VERBOSE : Verbose messages with debugging information.

property name

log(*self*: `tensorrt.tensorrt.ILogger`, *severity*: `nvinfer1::ILogger::Severity`, *msg*: `str`) → None
Logs a message to `stderr` . This function must be overridden by a derived class.

Parameters

- **severity** – The severity of the message.
- **msg** – The log message.

class `tensorrt.Logger`(*self*: `tensorrt.tensorrt.Logger`, *min_severity*: `tensorrt.tensorrt.ILogger.Severity = <Severity.WARNING: 2>`) → None

Logger for the *Builder*, *ICudaEngine* and *Runtime* .

Parameters `min_severity` – The initial minimum severity of this Logger.

Variables `min_severity` – `Logger.Severity` This minimum required severity of messages for the logger to log them.

log(*self*: `tensorrt.tensorrt.Logger`, *severity*: `tensorrt.tensorrt.ILogger.Severity`, *msg*: `str`) → None

Logs a message to *stderr* .

Parameters

- **severity** – The severity of the message.
- **msg** – The log message.

4.2 Profiler

class `tensorrt.IProfiler`(*self*: `tensorrt.tensorrt.IProfiler`) → None

Abstract base Profiler class.

To implement a custom profiler, ensure that you explicitly instantiate the base class in `__init__()` :

```
class MyProfiler(trt.IProfiler):
    def __init__(self):
        trt.IProfiler.__init__(self)

    def report_layer_time(self, layer_name, ms):
        ... # Your implementation here
```

When this class is added to an *IExecutionContext*, the profiler will be called once per layer for each invocation of *IExecutionContext.execute_v2()* or *IExecutionContext.execute_async_v2()*.

It is not recommended to run inference with profiler enabled when the inference execution time is critical since the profiler may affect execution time negatively.

report_layer_time(*self*: `tensorrt.tensorrt.IProfiler`, *layer_name*: `str`, *ms*: `float`) → None

Reports time in milliseconds for each layer. This function must be overridden a derived class.

Parameters

- **layer_name** – The name of the layer, set when constructing the *INetworkDefinition* .
- **ms** – The time in milliseconds to execute the layer.

class `tensorrt.Profiler`(*self*: `tensorrt.tensorrt.Profiler`) → None

When this class is added to an *IExecutionContext*, the profiler will be called once per layer for each invocation of *IExecutionContext.execute_v2()* or *IExecutionContext.execute_async_v2()*.

It is not recommended to run inference with profiler enabled when the inference execution time is critical since the profiler may affect execution time negatively.

report_layer_time(*self*: `tensorrt.tensorrt.Profiler`, *layer_name*: `str`, *ms*: `float`) → None
 Prints time in milliseconds for each layer to stdout.

Parameters

- **layer_name** – The name of the layer, set when constructing the `INetworkDefinition`.
- **ms** – The time in milliseconds to execute the layer.

4.3 IOptimizationProfile

class `tensorrt.IOptimizationProfile`

Optimization profile for dynamic input dimensions and shape tensors.

When building an `ICudaEngine` from an `INetworkDefinition` that has dynamically resizable inputs (at least one input tensor has one or more of its dimensions specified as -1) or shape input tensors, users need to specify at least one optimization profile. Optimization profiles are numbered 0, 1, ...

The first optimization profile that has been defined (with index 0) will be used by the `ICudaEngine` whenever no optimization profile has been selected explicitly. If none of the inputs are dynamic, the default optimization profile will be generated automatically unless it is explicitly provided by the user (this is possible but not required in this case). If more than a single optimization profile is defined, users may set a target how much additional weight space should be maximally allocated to each additional profile (as a fraction of the maximum, unconstrained memory).

Users set optimum input tensor dimensions, as well as minimum and maximum input tensor dimensions. The builder selects the kernels that result in the lowest runtime for the optimum input tensor dimensions, and are valid for all input tensor sizes in the valid range between minimum and maximum dimensions. A runtime error will be raised if the input tensor dimensions fall outside the valid range for this profile. Likewise, users provide minimum, optimum, and maximum values for all shape tensor input values.

`IOptimizationProfile` implements `__nonzero__()` and `__bool__()` such that evaluating a profile as a bool (e.g. `if profile:`) will check whether the optimization profile can be passed to an `IBuilderConfig` object. This will perform partial validation, by e.g. checking that the maximum dimensions are at least as large as the optimum dimensions, and that the optimum dimensions are always at least as large as the minimum dimensions. Some validation steps require knowledge of the network definition and are deferred to engine build time.

Variables `extra_memory_target` – Additional memory that the builder should aim to maximally allocate for this profile, as a fraction of the memory it would use if the user did not impose any constraints on memory. This unconstrained case is the default; it corresponds to `extra_memory_target == 1.0`. If `extra_memory_target == 0.0`, the builder aims to create the new optimization profile without allocating any additional weight memory. Valid inputs lie between 0.0 and 1.0. This parameter is only a hint, and TensorRT does not guarantee that the `extra_memory_target` will be reached. This parameter is ignored for the first (default) optimization profile that is defined.

get_shape(*self*: `tensorrt.tensorrt.IOptimizationProfile`, *input*: `str`) → `List[tensorrt.tensorrt.Dims]`

Get the minimum/optimum/maximum dimensions for a dynamic input tensor. If the dimensions have not been previously set via `set_shape()`, return an invalid `Dims` with a length of -1.

Returns A `List[Dims]` of length 3, containing the minimum, optimum, and maximum shapes, in that order. If the shapes have not been set yet, an empty list is returned.

get_shape_input(*self*: `tensorrt.tensorrt.IOptimizationProfile`, *input*: `str`) → `List[List[int]]`

Get the minimum/optimum/maximum values for a shape input tensor.

Returns A `List[List[int]]` of length 3, containing the minimum, optimum, and maximum values, in that order. If the values have not been set yet, an empty list is returned.

set_shape(*self*: [tensorrt.tensorrt.IOptimizationProfile](#), *input*: *str*, *min*: [tensorrt.tensorrt.Dims](#), *opt*: [tensorrt.tensorrt.Dims](#), *max*: [tensorrt.tensorrt.Dims](#)) → None

Set the minimum/optimum/maximum dimensions for a dynamic input tensor.

This function must be called for any network input tensor that has dynamic dimensions. If *min*, *opt*, and *max* are the minimum, optimum, and maximum dimensions, and *real_shape* is the shape for this input tensor provided to the [INetworkDefinition](#), then the following conditions must hold:

- (1) $\text{len}(\text{min}) == \text{len}(\text{opt}) == \text{len}(\text{max}) == \text{len}(\text{real_shape})$
- (2) $0 \leq \text{min}[i] \leq \text{opt}[i] \leq \text{max}[i]$ for all *i*
- (3) if $\text{real_shape}[i] \neq -1$, then $\text{min}[i] == \text{opt}[i] == \text{max}[i] == \text{real_shape}[i]$

This function may (but need not be) called for an input tensor that does not have dynamic dimensions. In this case, all shapes must equal *real_shape*.

Parameters

- **input** – The name of the input tensor.
- **min** – The minimum dimensions for this input tensor.
- **opt** – The optimum dimensions for this input tensor.
- **max** – The maximum dimensions for this input tensor.

Raises `ValueError` if an inconsistency was detected. Note that inputs can be validated only partially; a full validation is performed at engine build time.

set_shape_input(*self*: [tensorrt.tensorrt.IOptimizationProfile](#), *input*: *str*, *min*: *List[int]*, *opt*: *List[int]*, *max*: *List[int]*) → None

Set the minimum/optimum/maximum values for a shape input tensor.

This function must be called for every input tensor *t* that is a shape tensor (*t.is_shape* == True). This implies that the datatype of *t* is `int32`, the rank is either 0 or 1, and the dimensions of *t* are fixed at network definition time. This function must NOT be called for any input tensor that is not a shape tensor.

If *min*, *opt*, and *max* are the minimum, optimum, and maximum values, it must be true that $\text{min}[i] \leq \text{opt}[i] \leq \text{max}[i]$ for all *i*.

Parameters

- **input** – The name of the input tensor.
- **min** – The minimum values for this shape tensor.
- **opt** – The optimum values for this shape tensor.
- **max** – The maximum values for this shape tensor.

Raises `ValueError` if an inconsistency was detected. Note that inputs can be validated only partially; a full validation is performed at engine build time.

4.4 IBuilderConfig

tensorrt.QuantizationFlag

List of valid flags for quantizing the network to int8.

Members:

CALIBRATE_BEFORE_FUSION : Run int8 calibration pass before layer fusion. Only valid for IInt8LegacyCalibrator and IInt8EntropyCalibrator. We always run int8 calibration pass before layer fusion for IInt8MinMaxCalibrator and IInt8EntropyCalibrator2. Disabled by default.

tensorrt.DeviceType

Device types that TensorRT can execute on

Members:

GPU : GPU device

DLA : DLA core

tensorrt.ProfilingVerbosity

Profiling verbosity in NVTX annotations and the engine inspector

Members:

LAYER_NAMES_ONLY : Print only the layer names. This is the default setting.

DETAILED : Print detailed layer information including layer names and layer parameters.

NONE : Do not print any layer information.

DEFAULT : [DEPRECATED] Same as **LAYER_NAMES_ONLY**.

VERBOSE : [DEPRECATED] Same as **DETAILED**.

tensorrt.TacticSource

Tactic sources that can provide tactics for TensorRT.

Members:

CUBLAS : Enables cuBLAS tactics. **NOTE:** Disabling this value will cause the cublas handle passed to plugins in attachToContext to be null.

CUBLAS_LT : Enables cuBLAS LT tactics

CUDNN : Enables cuDNN tactics

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.

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 sampleNvmedia for an example of integrating NvMediaDLA 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 `NvMediaDLA` 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.

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 `IAAlgorithmSelector` 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.

class tensorrt.IBuilderConfig

Variables

- **min_timing_iterations** – int [DEPRECATED] The number of minimization iterations used when timing layers. When timing layers, the builder minimizes over a set of average times for layer execution. This parameter controls the number of iterations used in minimization. By default the minimum number of iterations is 1.
- **avg_timing_iterations** – int The number of averaging iterations used when timing layers. When timing layers, the builder minimizes over a set of average times for layer execution. This parameter controls the number of iterations used in averaging. By default the number of averaging iterations is 1.
- **int8_calibrator** – *IInt8Calibrator* Int8 Calibration interface. The calibrator is to minimize the information loss during the INT8 quantization process.
- **max_workspace_size** – int [DEPRECATED] The maximum workspace size. The maximum GPU temporary memory which the engine can use at execution time.
- **flags** – int The build mode flags to turn on builder options for this network. The flags are listed in the `BuilderFlags` enum. The flags set configuration options to build the network. This should be in integer consisting of one or more *BuilderFlag*s, combined via binary OR. For example, `1 << BuilderFlag.FP16 | 1 << BuilderFlag.DEBUG`.
- **profile_stream** – int The handle for the CUDA stream that is used to profile this network.
- **num_optimization_profiles** – int The number of optimization profiles.
- **default_device_type** – *tensorrt.DeviceType* The default DeviceType to be used by the Builder.
- **DLA_core** – int The DLA core that the engine executes on. Must be between 0 and N-1 where N is the number of available DLA cores.
- **profiling_verbosity** – Profiling verbosity in NVTX annotations.
- **engine_capability** – The desired engine capability. See *EngineCapability* for details.

`__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_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_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. By default, CUBLAS, CUDNN, and `EDGE_MASK_CONVOLUTIONS` are always enabled. `CUBLAS_LT` is always enabled for x86 platforms and only enabled for non-x86 platforms when `CUDA >= 11.0`.

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`

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

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.
- **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.

__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 $H \text{ Stride} \% 32 == 0$ ”

Parameters `index` – The binding index.

get_binding_index(*self*: `tensorrt.tensorrt.ICudaEngine`, *name*: `str`) → `int`
 Retrieve the binding index for a named tensor.

You can also use engine’s `__getitem__()` with `engine[name]`. When invoked with a `str`, this will return the corresponding binding index.

`IExecutionContext.execute_async_v2()` and `IExecutionContext.execute_v2()` require an array of buffers. Engine bindings map from tensor names to indices in this array. Binding indices are assigned at `ICudaEngine` build time, and take values in the range `[0 ... n-1]` where `n` is the total number of inputs and outputs.

Parameters `name` – The tensor name.

Returns The binding index for the named tensor, or -1 if the name is not found.

get_binding_name(*self*: `tensorrt.tensorrt.ICudaEngine`, *index*: `int`) → `str`
 Retrieve the name corresponding to a binding index.

You can also use engine’s `__getitem__()` with `engine[index]`. When invoked with an `int`, this will return the corresponding binding name.

This is the reverse mapping to that provided by `get_binding_index()`.

Parameters `index` – The binding index.

Returns The name corresponding to the binding index.

get_binding_shape(**args*, ***kwargs*)
 Overloaded function.

1. `get_binding_shape(self: tensorrt.tensorrt.ICudaEngine, index: int) -> tensorrt.tensorrt.Dims`

Get the shape of a binding.

index The binding index.

Returns The shape of the binding if the index is in range, otherwise `Dims()`

2. `get_binding_shape(self: tensorrt.tensorrt.ICudaEngine, name: str) -> tensorrt.tensorrt.Dims`

Get the shape of a binding.

name The name of the tensor corresponding to an engine binding.

Returns The shape of the binding if the tensor is present, otherwise `Dims()`

get_binding_vectorized_dim(*self*: `tensorrt.tensorrt.ICudaEngine`, *index*: `int`) → `int`
 Return the dimension index that the buffer is vectorized.

Specifically -1 is returned if scalars per vector is 1.

Parameters `index` – The binding index.

get_location(**args*, ***kwargs*)
 Overloaded function.

1. `get_location(self: tensorrt.tensorrt.ICudaEngine, index: int) -> tensorrt.tensorrt.TensorLocation`

Get location of binding. This lets you know whether the binding should be a pointer to device or host memory.

index The binding index.

returns The location of the bound tensor with given index.

2. `get_location(self: tensorrt.tensorrt.ICudaEngine, name: str) -> tensorrt.tensorrt.TensorLocation`

Get location of binding. This lets you know whether the binding should be a pointer to device or host memory.

name The name of the tensor corresponding to an engine binding.

returns The location of the bound tensor with given index.

get_profile_shape(*args, **kwargs)

Overloaded function.

1. `get_profile_shape(self: tensorrt.tensorrt.ICudaEngine, profile_index: int, binding: int) -> List[tensorrt.tensorrt.Dims]`

Get the minimum/optimum/maximum dimensions for a particular binding under an optimization profile.

arg profile_index The index of the profile.

arg binding The binding index or name.

returns A `List[Dims]` of length 3, containing the minimum, optimum, and maximum shapes, in that order.

2. `get_profile_shape(self: tensorrt.tensorrt.ICudaEngine, profile_index: int, binding: str) -> List[tensorrt.tensorrt.Dims]`

Get the minimum/optimum/maximum dimensions for a particular binding under an optimization profile.

arg profile_index The index of the profile.

arg binding The binding index or name.

returns A `List[Dims]` of length 3, containing the minimum, optimum, and maximum shapes, in that order.

get_profile_shape_input(*args, **kwargs)

Overloaded function.

1. `get_profile_shape_input(self: tensorrt.tensorrt.ICudaEngine, profile_index: int, binding: int) -> List[List[int]]`

Get minimum/optimum/maximum values for an input shape binding under an optimization profile. If the specified binding is not an input shape binding, an exception is raised.

arg profile_index The index of the profile.

arg binding The binding index or name.

returns A `List[List[int]]` of length 3, containing the minimum, optimum, and maximum values, in that order. If the values have not been set yet, an empty list is returned.

2. `get_profile_shape_input(self: tensorrt.tensorrt.ICudaEngine, profile_index: int, binding: str) -> List[List[int]]`

Get minimum/optimum/maximum values for an input shape binding under an optimization profile. If the specified binding is not an input shape binding, an exception is raised.

arg profile_index The index of the profile.

arg binding The binding index or name.

returns A `List[List[int]]` of length 3, containing the minimum, optimum, and maximum values, in that order. If the values have not been set yet, an empty list is returned.

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.

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.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.
- **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.
- **error_recorder** – *IErrorRecorder* Application-implemented error reporting interface for TensorRT objects.

Ival enqueue_emits_profile bool Whether enqueue emits layer timing to the profiler. The default value is True. If set to False, enqueue will be asynchronous if there is a profiler attached. An extra method `IExecutionContext::report_to_profiler()` needs to be called to obtain the profiling data and report to the profiler attached.

`__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_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_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.

report_to_profiler(*self*: `tensorrt.tensorrt.IExecutionContext`) → `bool`

Calculate layer timing info for the current optimization profile in `IExecutionContext` and update the profiler after one iteration of inference launch.

If the `enqueue_emits_profiler` flag was set to true, the enqueue function will calculate layer timing implicitly if a profiler is provided. There is no need to call this function. If the `enqueue_emits_profiler` flag was set to false, the enqueue function will record the CUDA event timers if a profiler is provided. But it will not perform the layer timing calculation. This function needs to be called explicitly to calculate layer timing for the previous inference launch.

In the CUDA graph launch scenario, it will record the same set of CUDA events as in regular enqueue functions if the graph is captured from an `IExecutionContext` with profiler enabled. This function needs to be called after graph launch to report the layer timing info to the profiler.

Profiling CUDA graphs is only available from CUDA 11.1 onwards.

Returns `True` if the call succeeded, else `False` (e.g. profiler not provided, in CUDA graph capture mode, etc.)

set_binding_shape(*self*: `tensorrt.tensorrt.IExecutionContext`, *binding*: `int`, *shape*: `tensorrt.tensorrt.Dims`) → `bool`

Set the dynamic shape of a binding.

Requires the engine to be built without an implicit batch dimension. The binding must be an input tensor, and all dimensions must be compatible with the network definition (i.e. only the wildcard dimension -1 can be replaced with a new dimension > 0). Furthermore, the dimensions must be in the valid range for the currently selected optimization profile.

For all dynamic non-output bindings (which have at least one wildcard dimension of -1), this method needs to be called after setting `active_optimization_profile` before either `execute_async_v2()` or `execute_v2()` may be called. When all input shapes have been specified, `all_binding_shapes_specified` is set to `True`.

Parameters

- `binding` – The binding index.
- `shape` – The shape to set.

Returns False if an error occurs (e.g. specified binding is out of range for the currently selected optimization profile or specified shape is inconsistent with min-max range of the optimization profile), else True.

Note that the network can still be invalid for certain combinations of input shapes that lead to invalid output shapes. To confirm the correctness of the network input shapes, check whether the output binding has valid shape using `get_binding_shape()` on the output binding.

set_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_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.

4.8 Runtime

class `tensorrt.Runtime` (*self*: `tensorrt.tensorrt.Runtime`, *logger*: `tensorrt.tensorrt.ILogger`) → None

Allows a serialized `ICudaEngine` to be deserialized.

Variables

- **error_recorder** – `IErrorRecorder` Application-implemented error reporting interface for TensorRT objects.

- **gpu_allocator** – *IGpuAllocator* The GPU allocator to be used by the *Runtime* . All GPU memory acquired will use this allocator. If set to None, the default allocator will be used (Default: cudaMalloc/cudaFree).
- **DLA_core** – int The DLA core that the engine executes on. Must be between 0 and N-1 where N is the number of available DLA cores.
- **num_DLA_cores** – int The number of DLA engines available to this builder.
- **logger** – *ILogger* The logger provided when creating the refitter.
- **max_threads** – int The maximum thread that can be used by the *Runtime* .

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.

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** – *IErrorRecorder* Application-implemented error reporting interface for TensorRT objects.
- **logger** – *ILogger* The logger provided when creating the refitter.
- **max_threads** – int The maximum thread that can be used by the *Refitter* .

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.ErrorCodeTRT`

Error codes that can be returned by TensorRT during execution.

Members:

SUCCESS : Execution completed successfully.

UNSPECIFIED_ERROR : An error that does not fall into any other category. This error is included for forward compatibility.

INTERNAL_ERROR : A non-recoverable TensorRT error occurred.

INVALID_ARGUMENT : An argument passed to the function is invalid in isolation. This is a violation of the API contract.

INVALID_CONFIG : An error occurred when comparing the state of an argument relative to other arguments. For example, the dimensions for concat differ between two tensors outside of the channel dimension. This error is triggered when an argument is correct in isolation, but not relative to other arguments. This is to help to distinguish from the simple errors from the more complex errors. This is a violation of the API contract.

FAILED_ALLOCATION : An error occurred when performing an allocation of memory on the host or the device. A memory allocation error is normally fatal, but in the case where the application provided its own memory allocation routine, it is possible to increase the pool of available memory and resume execution.

FAILED_INITIALIZATION : One, or more, of the components that TensorRT relies on did not initialize correctly. This is a system setup issue.

FAILED_EXECUTION : An error occurred during execution that caused TensorRT to end prematurely, either an asynchronous error or other execution errors reported by CUDA/DLA. In a dynamic system, the data can be thrown away and the next frame can be processed or execution can be retried. This is either an execution error or a memory error.

FAILED_COMPUTATION : An error occurred during execution that caused the data to become corrupted, but execution finished. Examples of this error are NaN squashing or integer overflow. In a dynamic system, the data can be thrown away and the next frame can be processed or execution can be retried. This is either a data corruption error, an input error, or a range error.

INVALID_STATE : TensorRT was put into a bad state by incorrect sequence of function calls. An example of an invalid state is specifying a layer to be DLA only without GPU fallback, and that layer is not supported by DLA. This can occur in situations where a service is optimistically executing networks for multiple different configurations without checking proper error configurations, and instead throwing away bad configurations caught by TensorRT. This is a violation of the API contract, but can be recoverable.

Example of a recovery: GPU fallback is disabled and conv layer with large filter(63x63) is specified to run on DLA. This will fail due to DLA not supporting the large kernel size. This can be recovered by either turning on GPU fallback or setting the layer to run on the GPU.

UNSUPPORTED_STATE : An error occurred due to the network not being supported on the device due to constraints of the hardware or system. An example is running a unsafe layer in a safety certified context, or a resource requirement for the current network is greater than the capabilities of the target device. The network is otherwise correct, but the network and hardware combination is problematic. This can be recoverable. Examples: * Scratch space requests larger than available device memory and can be recovered by increasing allowed workspace size. * Tensor size exceeds the maximum element count and can be recovered by reducing the maximum batch size.

class `tensorrt.IErrorRecorder`(*self*: `tensorrt.tensorrt.IErrorRecorder`) → None
Reference counted application-implemented error reporting interface for TensorRT objects.

The error reporting mechanism is a user defined object that interacts with the internal state of the object that it is assigned to in order to determine information about abnormalities in execution. The error recorder gets both an error enum that is more descriptive than pass/fail and also a description that gives more detail on the exact failure modes. In the safety context, the error strings are all limited to 128 characters in length. The ErrorRecorder gets passed along to any class that is created from another class that has an ErrorRecorder assigned to it. For example, assigning an ErrorRecorder to an Builder allows all INetwork's, ILayer's, and ITensor's to use the same error recorder. For functions that have their own ErrorRecorder accessor functions. This allows registering a different error recorder or de-registering of the error recorder for that specific object.

The ErrorRecorder object implementation must be thread safe if the same ErrorRecorder is passed to different interface objects being executed in parallel in different threads. All locking and synchronization is pushed to the interface implementation and TensorRT does not hold any synchronization primitives when accessing the interface functions.

clear(*self*: `tensorrt.tensorrt.IErrorRecorder`) → None
Clear the error stack on the error recorder.

Removes all the tracked errors by the error recorder. This function must guarantee that after this function is called, and as long as no error occurs, `num_errors` will be zero.

get_error_code(*self*: `tensorrt.tensorrt.IErrorRecorder`, *arg0*: `int`) → `tensorrt.tensorrt.ErrorCodeTRT`
 Returns the ErrorCode enumeration.

The `error_idx` specifies what error code from 0 to `num_errors-1` that the application wants to analyze and return the error code enum.

Parameters `error_idx` – A 32bit integer that indexes into the error array.

Returns Returns the enum corresponding to `error_idx`.

get_error_desc(*self*: `tensorrt.tensorrt.IErrorRecorder`, *arg0*: `int`) → `str`
 Returns description of the error.

For the error specified by the `idx` value, return description of the error. In the safety context there is a constant length requirement to remove any dynamic memory allocations and the error message may be truncated. The format of the error description is “<EnumAsStr> - <Description>”.

Parameters `error_idx` – A 32bit integer that indexes into the error array.

Returns Returns description of the error.

has_overflowed(*self*: `tensorrt.tensorrt.IErrorRecorder`) → `bool`
 Determine if the error stack has overflowed.

In the case when the number of errors is large, this function is used to query if one or more errors have been dropped due to lack of storage capacity. This is especially important in the automotive safety case where the internal error handling mechanisms cannot allocate memory.

Returns True if errors have been dropped due to overflowing the error stack.

num_errors(*self*: `tensorrt.tensorrt.IErrorRecorder`) → `int`
 Return the number of errors

Determines the number of errors that occurred between the current point in execution and the last time that the `clear()` was executed. Due to the possibility of asynchronous errors occurring, a TensorRT API can return correct results, but still register errors with the Error Recorder. The value of `getNbErrors` must monotonically increase until `clear()` is called.

Returns Returns the number of errors detected, or 0 if there are no errors.

report_error(*self*: `tensorrt.tensorrt.IErrorRecorder`, *arg0*: `tensorrt.tensorrt.ErrorCodeTRT`, *arg1*: `str`) → `bool`

Clear the error stack on the error recorder.

Report an error to the user that has a given value and human readable description. The function returns false if processing can continue, which implies that the reported error is not fatal. This does not guarantee that processing continues, but provides a hint to TensorRT.

Parameters

- **val** – The error code enum that is being reported.
- **desc** – The description of the error.

Returns True if the error is determined to be fatal and processing of the current function must end.

4.11 ITimingCache

class `tensorrt.ITimingCache`

Class to handle tactic timing info collected from builder.

combine(*self*: `tensorrt.tensorrt.ITimingCache`, *input_cache*: `tensorrt.tensorrt.ITimingCache`,
ignore_mismatch: `bool`) → `bool`

Combine input timing cache into local instance.

Append entries in input cache to local cache. Conflicting entries will be skipped. The input cache must be generated by a TensorRT build of exact same version, otherwise combine will be skipped and return false. `bool(ignore_mismatch) == True` if combining a timing cache created from a different device.

Parameters

- **input_cache** – The input timing cache
- **ignore_mismatch** – Whether or not to allow cache verification header mismatch

Returns A `bool` indicating whether the combine operation is done successfully.

reset(*self*: `tensorrt.tensorrt.ITimingCache`) → `bool`

Empty the timing cache

Returns A `bool` indicating whether the reset operation is done successfully.

serialize(*self*: `tensorrt.tensorrt.ITimingCache`) → `tensorrt.tensorrt.IHostMemory`

Serialize a timing cache to a `IHostMemory` object.

Returns An `IHostMemory` object that contains a serialized timing cache.

4.12 GPU Allocator

4.12.1 AllocatorFlag

`tensorrt.AllocatorFlag`

Members:

`RESIZABLE` : TensorRT may call `realloc()` on this allocation

4.12.2 IGpuAllocator

class `tensorrt.IGpuAllocator`(*self*: `tensorrt.tensorrt.IGpuAllocator`) → `None`

Application-implemented class for controlling allocation on the GPU.

__init__(*self*: `tensorrt.tensorrt.IGpuAllocator`) → `None`

allocate(*self*: `tensorrt.tensorrt.IGpuAllocator`, *size*: `int`, *alignment*: `int`, *flags*: `int`) → `capsule`

A callback implemented by the application to handle acquisition of GPU memory. If an allocation request of size 0 is made, `None` should be returned.

If an allocation request cannot be satisfied, `None` should be returned.

Parameters

- **size** – The size of the memory required.

- **alignment** – The required alignment of memory. Alignment will be zero or a power of 2 not exceeding the alignment guaranteed by `cudaMalloc`. Thus this allocator can be safely implemented with `cudaMalloc/cudaFree`. An alignment value of zero indicates any alignment is acceptable.
- **flags** – Allocation flags. See [AllocatorFlag](#)

Returns The address of the allocated memory

deallocate(*self*: [tensorrt.tensorrt.IGpuAllocator](#), *memory*: *capsule*) → bool

A callback implemented by the application to handle release of GPU memory.

TensorRT may pass a 0 to this function if it was previously returned by `allocate()`.

Parameters **memory** – The memory address of the memory to release.

Returns True if the acquired memory is released successfully.

free(*self*: [tensorrt.tensorrt.IGpuAllocator](#), *memory*: *capsule*) → None

A callback implemented by the application to handle release of GPU memory.

TensorRT may pass a 0 to this function if it was previously returned by `allocate()`.

Parameters **memory** – The memory address of the memory to release.

realloc(*self*: [tensorrt.tensorrt.IGpuAllocator](#), *address*: *capsule*, *alignment*: *int*, *new_size*: *int*) → capsule

A callback implemented by the application to resize an existing allocation.

Only allocations which were allocated with `AllocatorFlag.RESIZABLE` will be resized.

Options are one of: - resize in place leaving `min(old_size, new_size)` bytes unchanged and return the original address - move `min(old_size, new_size)` bytes to a new location of sufficient size and return its address - return `nullptr`, to indicate that the request could not be fulfilled.

If `nullptr` is returned, TensorRT will assume that `resize()` is not implemented, and that the allocation at address is still valid.

This method is made available for use cases where delegating the resize strategy to the application provides an opportunity to improve memory management. One possible implementation is to allocate a large virtual device buffer and progressively commit physical memory with `cuMemMap`. `CU_MEM_ALLOC_GRANULARITY_RECOMMENDED` is suggested in this case.

TensorRT may call `realloc` to increase the buffer by relatively small amounts.

Parameters

- **address** – the address of the original allocation.
- **alignment** – The alignment used by the original allocation.
- **new_size** – The new memory size required.

Returns The address of the reallocated memory

4.13 EngineInspector

class `tensorrt.EngineInspector`

An engine inspector which prints out the layer information of an engine or an execution context. The engine or the context must be set before `get_layer_information()` or `get_engine_information()` can be called.

The amount of printed information depends on the profiling verbosity setting of the builder config when the engine is built. By default, the profiling verbosity is set to `ProfilingVerbosity.LAYER_NAMES_ONLY`, and only layer names will be printed. If the profiling verbosity is set to `ProfilingVerbosity.DETAILED`, layer names and layer parameters will be printed. If the profiling verbosity is set to `ProfilingVerbosity.NONE`, no layer information will be printed.

Variables

- **engine** – *ICudaEngine* Set or get the engine currently being inspected.
- **context** – *IExecutionContext* Set or get context currently being inspected.
- **error_recorder** – *IErrorRecorder* Application-implemented error reporting interface for TensorRT objects.

`__init__`(*args, **kwargs)

`get_engine_information`(self: `tensorrt.tensorrt.EngineInspector`, format: `tensorrt.tensorrt.LayerInformationFormat`) → str

Get a string describing the information about all the layers in the current engine or the execution context.

Parameters **format** – `LayerInformationFormat` The format the layer information should be printed in.

Returns A string describing the information about all the layers in the current engine or the execution context.

`get_layer_information`(self: `tensorrt.tensorrt.EngineInspector`, layer_index: int, format: `tensorrt.tensorrt.LayerInformationFormat`) → str

Get a string describing the information about a specific layer in the current engine or the execution context.

Parameters

- **layer_index** – The index of the layer. It must lie in `[0, engine.num_layers]`.
- **format** – `LayerInformationFormat` The format the layer information should be printed in.

Returns A string describing the information about a specific layer in the current engine or the execution context.

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_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_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_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_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 $\{N1, \dots, Np, S_{max}, H\}$:

$N1..Np$ 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 $2 \times \text{IRNNv2Layer.hidden_size}$)

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 $\{N1, \dots, Np, L, H\}$:

$N1..Np$ 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 $2 \times \text{IRNNv2Layer.hidden_size}$ 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`) \rightarrow `tensorrt.tensorrt.IScaleLayer`

Add a scale layer to the network. See `IScaleLayer` for more information.

Parameters

- **input** – The input tensor to the layer. This tensor is required to have a minimum of 3 dimensions.
- **mode** – The scaling mode.
- **shift** – The shift value.
- **scale** – The scale value.
- **power** – The power value.

If the weights are available, then the size of weights are dependent on the `ScaleMode`. For `UNIFORM`, the number of weights is equal to 1. For `CHANNEL`, the number of weights is equal to the channel dimension. For `ELEMENTWISE`, the number of weights is equal to the volume of the input.

Returns The new scale layer, or None if it could not be created.

add_scale_nd(*self*: *tensorrt.tensorrt.INetworkDefinition*, *input*: *tensorrt.tensorrt.ITensor*, *mode*: *tensorrt.tensorrt.ScaleMode*, *shift*: *tensorrt.tensorrt.Weights = None*, *scale*: *tensorrt.tensorrt.Weights = None*, *power*: *tensorrt.tensorrt.Weights = None*, *channel_axis*: *int*) → *tensorrt.tensorrt.IScaleLayer*

Add a multi-dimension scale layer to the network. See [IScaleLayer](#) for more information.

Parameters

- **input** – The input tensor to the layer. This tensor is required to have a minimum of 3 dimensions.
- **mode** – The scaling mode.
- **shift** – The shift value.
- **scale** – The scale value.
- **power** – The power value.
- **channel_axis** – The channel dimension axis.

If the weights are available, then the size of weights are dependent on the ScaleMode. For UNIFORM, the number of weights is equal to 1. For CHANNEL, the number of weights is equal to the channel dimension. For ELEMENTWISE, the number of weights is equal to the volume of the input.

Returns The new scale layer, or None if it could not be created.

add_scatter(*self*: *tensorrt.tensorrt.INetworkDefinition*, *data*: *tensorrt.tensorrt.ITensor*, *indices*: *tensorrt.tensorrt.ITensor*, *updates*: *tensorrt.tensorrt.ITensor*, *mode*: *tensorrt.tensorrt.ScatterMode*) → *tensorrt.tensorrt.IScatterLayer*

Add a scatter layer to the network. See [IScatterLayer](#) for more information.

Parameters

- **data** – The tensor to get default values from.
- **indices** – The tensor to get indices from to populate the output tensor.
- **updates** – The tensor to get values from to populate the output tensor.
- **mode** – operation mode see [IScatterLayer](#) for more info

Returns The new Scatter layer, or None if it could not be created.

add_select(*self*: *tensorrt.tensorrt.INetworkDefinition*, *condition*: *tensorrt.tensorrt.ITensor*, *then_input*: *tensorrt.tensorrt.ITensor*, *else_input*: *tensorrt.tensorrt.ITensor*) → *tensorrt.tensorrt.ISelectLayer*

Add a select layer. See [ISelectLayer](#) for more information.

Parameters

- **condition** – The condition tensor to the layer.
- **then_input** – The then input tensor to the layer.
- **else_input** – The else input tensor to the layer.

Returns The new select layer, or None if it could not be created.

add_shape(*self*: *tensorrt.tensorrt.INetworkDefinition*, *input*: *tensorrt.tensorrt.ITensor*) → *tensorrt.tensorrt.IShapeLayer*

Add a shape layer to the network. See [IShapeLayer](#) for more information.

Parameters **input** – The input tensor to the layer.

Returns The new shape layer, or None if it could not be created.

add_shuffle(*self*: `tensorrt.tensorrt.INetworkDefinition`, *input*: `tensorrt.tensorrt.ITensor`) → `tensorrt.tensorrt.IShuffleLayer`

Add a shuffle layer to the network. See [IShuffleLayer](#) for more information.

Parameters **input** – The input tensor to the layer.

Returns The new shuffle layer, or `None` if it could not be created.

add_slice(*self*: `tensorrt.tensorrt.INetworkDefinition`, *input*: `tensorrt.tensorrt.ITensor`, *start*: `tensorrt.tensorrt.Dims`, *shape*: `tensorrt.tensorrt.Dims`, *stride*: `tensorrt.tensorrt.Dims`) → `tensorrt.tensorrt.ISliceLayer`

Add a slice layer to the network. See [ISliceLayer](#) for more information.

Parameters

- **input** – The input tensor to the layer.
- **start** – The start offset.
- **shape** – The output shape.
- **stride** – The slicing stride. Positive, negative, zero stride values, and combinations of them in different dimensions are allowed.

Returns The new slice layer, or `None` if it could not be created.

add_softmax(*self*: `tensorrt.tensorrt.INetworkDefinition`, *input*: `tensorrt.tensorrt.ITensor`) → `tensorrt.tensorrt.ISoftMaxLayer`

Add a softmax layer to the network. See [ISoftMaxLayer](#) for more information.

Parameters **input** – The input tensor to the layer.

Returns The new softmax layer, or `None` if it could not be created.

add_topk(*self*: `tensorrt.tensorrt.INetworkDefinition`, *input*: `tensorrt.tensorrt.ITensor`, *op*: `tensorrt.tensorrt.TopKOperation`, *k*: `int`, *axes*: `int`) → `tensorrt.tensorrt.ITopKLayer`

Add a TopK layer to the network. See [ITopKLayer](#) for more information.

The TopK layer has two outputs of the same dimensions. The first contains data values, the second contains index positions for the values. Output values are sorted, largest first for operation `TopKOperation.MAX` and smallest first for operation `TopKOperation.MIN`.

Currently only values of K up to 3840 are supported.

Parameters

- **input** – The input tensor to the layer.
- **op** – Operation to perform.
- **k** – Number of elements to keep.
- **axes** – The reduction dimensions. The bit in position *i* of bitmask *axes* corresponds to explicit dimension *i* of the result. E.g., the least significant bit corresponds to the first explicit dimension and the next to least significant bit corresponds to the second explicit dimension. Currently *axes* must specify exactly one dimension, and it must be one of the last four dimensions.

Returns The new TopK layer, or `None` if it could not be created.

add_unary(*self*: `tensorrt.tensorrt.INetworkDefinition`, *input*: `tensorrt.tensorrt.ITensor`, *op*: `tensorrt.tensorrt.UnaryOperation`) → `tensorrt.tensorrt.IUnaryLayer`

Add a unary layer to the network. See [IUnaryLayer](#) for more information.

Parameters

- **input** – The input tensor to the layer.
- **op** – The operation to apply.

Returns The new unary layer, or None if it could not be created.

get_input(*self*: `tensorrt.tensorrt.INetworkDefinition`, *index*: `int`) → `tensorrt.tensorrt.ITensor`
 Get the input tensor specified by the given index.

Parameters **index** – The index of the input tensor.

Returns The tensor, or None if it is out of range.

get_layer(*self*: `tensorrt.tensorrt.INetworkDefinition`, *index*: `int`) → `tensorrt.tensorrt.ILayer`
 Get the layer specified by the given index.

Parameters **index** – The index of the layer.

Returns The layer, or None if it is out of range.

get_output(*self*: `tensorrt.tensorrt.INetworkDefinition`, *index*: `int`) → `tensorrt.tensorrt.ITensor`
 Get the output tensor specified by the given index.

Parameters **index** – The index of the output tensor.

Returns The tensor, or None if it is out of range.

mark_output(*self*: `tensorrt.tensorrt.INetworkDefinition`, *tensor*: `tensorrt.tensorrt.ITensor`) → None
 Mark a tensor as an output.

Parameters **tensor** – The tensor to mark.

mark_output_for_shapes(*self*: `tensorrt.tensorrt.INetworkDefinition`, *tensor*: `tensorrt.tensorrt.ITensor`) → bool

Enable tensor's value to be computed by `IExecutionContext.get_shape_binding()`.

Parameters **tensor** – The tensor to unmark as an output tensor. The tensor must be of type `int32` and have no more than one dimension.

Returns True if successful, False if tensor is already marked as an output.

remove_tensor(*self*: `tensorrt.tensorrt.INetworkDefinition`, *tensor*: `tensorrt.tensorrt.ITensor`) → None
 Remove a tensor from the network.

Parameters **tensor** – The tensor to remove

It is illegal to remove a tensor that is the input or output of a layer. If this method is called with such a tensor, a warning will be emitted on the log and the call will be ignored.

set_weights_name(*self*: `tensorrt.tensorrt.INetworkDefinition`, *weights*: `tensorrt.tensorrt.Weights`, *name*: `str`) → bool

Associate a name with all current uses of the given weights.

The name must be set after the Weights are used in the network. Lookup is associative. The name applies to all Weights with matching type, value pointer, and count. If Weights with a matching value pointer, but different type or count exists in the network, an error message is issued, the name is rejected, and return false. If the name has already been used for other weights, return false. None causes the weights to become unnamed, i.e. clears any previous name.

Parameters

- **weights** – The weights to be named.
- **name** – The name to associate with the weights.

Returns true on success.

unmark_output(*self*: tensorrt.tensorrt.INetworkDefinition, *tensor*: tensorrt.tensorrt.ITensor) → None
 Unmark a tensor as a network output.

Parameters **tensor** – The tensor to unmark as an output tensor.

unmark_output_for_shapes(*self*: tensorrt.tensorrt.INetworkDefinition, *tensor*: tensorrt.tensorrt.ITensor)
 → bool

Undo *mark_output_for_shapes()* .

Parameters **tensor** – The tensor to unmark as an output tensor.

Returns True if successful, False if tensor is not marked as an output.

5.2 Layer Base Classes

5.2.1 ITensor

tensorrt.TensorLocation

The physical location of the data.

Members:

DEVICE : Data is stored on the device.

HOST : Data is stored on the device.

tensorrt.TensorFormat

Format of the input/output tensors.

This enum is used by both plugins and network I/O tensors.

For more information about data formats, see the topic “Data Format Description” located in the TensorRT Developer Guide (<https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html>).

Members:

LINEAR : Row major linear format.

For a tensor with dimensions {N, C, H, W}, the W axis always has unit stride, and the stride of every other axis is at least the the product of of the next dimension times the next stride. the strides are the same as for a C array with dimensions [N][C][H][W].

CHW2 : Two wide channel vectorized row major format.

This format is bound to FP16. It is only available for dimensions >= 3.

For a tensor with dimensions {N, C, H, W}, the memory layout is equivalent to a C array with dimensions [N][(C+1)/2][H][W][2], with the tensor coordinates (n, c, h, w) mapping to array subscript [n][c/2][h][w][c%2].

HWC8 : Eight channel format where C is padded to a multiple of 8.

This format is bound to FP16. It is only available for dimensions >= 3.

For a tensor with dimensions {N, C, H, W}, the memory layout is equivalent to the array with dimensions [N][H][W][(C+7)/8*8], with the tensor coordinates (n, c, h, w) mapping to array subscript [n][h][w][c].

CHW4 : Four wide channel vectorized row major format. This format is bound to INT8. It is only available for dimensions >= 3.

For a tensor with dimensions $\{N, C, H, W\}$, the memory layout is equivalent to a C array with dimensions $[N][(C+3)/4][H][W][4]$, with the tensor coordinates (n, c, h, w) mapping to array subscript $[n][c/4][h][w][c\%4]$.

CHW16 : Sixteen wide channel vectorized row major format.

This format is bound to FP16. It is only available for dimensions ≥ 3 .

For a tensor with dimensions $\{N, C, H, W\}$, the memory layout is equivalent to a C array with dimensions $[N][(C+15)/16][H][W][16]$, with the tensor coordinates (n, c, h, w) mapping to array subscript $[n][c/16][h][w][c\%16]$.

CHW32 : Thirty-two wide channel vectorized row major format.

This format is only available for dimensions ≥ 3 .

For a tensor with dimensions $\{N, C, H, W\}$, the memory layout is equivalent to a C array with dimensions $[N][(C+31)/32][H][W][32]$, with the tensor coordinates (n, c, h, w) mapping to array subscript $[n][c/32][h][w][c\%32]$.

DHWC8 : Eight channel format where C is padded to a multiple of 8.

This format is bound to FP16, and it is only available for dimensions ≥ 4 .

For a tensor with dimensions $\{N, C, D, H, W\}$, the memory layout is equivalent to an array with dimensions $[N][D][H][W][(C+7)/8*8]$, with the tensor coordinates (n, c, d, h, w) mapping to array subscript $[n][d][h][w][c]$.

CDHW32 : Thirty-two wide channel vectorized row major format with 3 spatial dimensions.

This format is bound to FP16 and INT8. It is only available for dimensions ≥ 4 .

For a tensor with dimensions $\{N, C, D, H, W\}$, the memory layout is equivalent to a C array with dimensions $[N][(C+31)/32][D][H][W][32]$, with the tensor coordinates (n, d, c, h, w) mapping to array subscript $[n][c/32][d][h][w][c\%32]$.

HWC : Non-vectorized channel-last format. This format is bound to FP32 and is only available for dimensions ≥ 3 .

DLA_LINEAR : DLA planar format. Row major format. The stride for stepping along the H axis is rounded up to 64 bytes.

This format is bound to FP16/Int8 and is only available for dimensions ≥ 3 .

For a tensor with dimensions $\{N, C, H, W\}$, the memory layout is equivalent to a C array with dimensions $[N][C][H][\text{roundUp}(W, 64/\text{elementSize})]$ 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 $[n][h][w][c]$.

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(TensorFormats.CHW4) | 1 << int(TensorFormat.CHW32)`.

reset_dynamic_range(*self*: `tensorrt.tensorrt.ITensor`) → `None`

Undo the effect of setting the dynamic range.

set_dynamic_range(*self*: `tensorrt.tensorrt.ITensor`, *min*: `float`, *max*: `float`) → `bool`

Set dynamic range for the tensor. NOTE: It is suggested to use `tensor.dynamic_range = (min, max)` instead.

Parameters

- **min** – Minimum of the dynamic range.
- **max** – Maximum of the dyanmic range.

Returns `true` if succeed in setting range. Otherwise `false`.

5.2.2 ILayer

tensorrt.LayerType

Type of Layer

Members:

CONVOLUTION : Convolution layer
FULLY_CONNECTED : Fully connected layer
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
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

class `tensorrt.ILayer`

Base class for all layer classes in an *INetworkDefinition* .

Variables

- **name** – str The name of the layer.
- **type** – *LayerType* The type of the layer.
- **num_inputs** – int The number of inputs of the layer.
- **num_outputs** – int The number of outputs of the layer.
- **precision** – *DataType* The computation precision.
- **precision_is_set** – bool Whether the precision is set or not.

get_input(*self*: `tensorrt.tensorrt.ILayer`, *index*: int) → `tensorrt.tensorrt.ITensor`

Get the layer input corresponding to the given index.

Parameters **index** – The index of the input tensor.

Returns The input tensor, or None if the index is out of range.

get_output(*self*: `tensorrt.tensorrt.ILayer`, *index*: int) → `tensorrt.tensorrt.ITensor`

Get the layer output corresponding to the given index.

Parameters **index** – The index of the output tensor.

Returns The output tensor, or None if the index is out of range.

get_output_type(*self*: `tensorrt.tensorrt.ILayer`, *index*: int) → `tensorrt.tensorrt.DataType`

Get the output type of the layer.

Parameters **index** – The index of the output tensor.

Returns The output precision. Default : `DataType.FLOAT`.

output_type_is_set(*self*: `tensorrt.tensorrt.ILayer`, *index*: int) → bool

Whether the output type has been set for this layer.

Parameters **index** – The index of the output.

Returns Whether the output type has been explicitly set.

reset_output_type(*self*: `tensorrt.tensorrt.ILayer`, *index*: int) → None

Reset output type of this layer.

Parameters **index** – The index of the output.

reset_precision(*self*: `tensorrt.tensorrt.ILayer`) → None

Reset the computation precision of the layer.

set_input(*self*: `tensorrt.tensorrt.ILayer`, *index*: `int`, *tensor*: `tensorrt.tensorrt.ITensor`) → None

Set the layer input corresponding to the given index.

Parameters

- **index** – The index of the input tensor.
- **tensor** – The input tensor.

set_output_type(*self*: `tensorrt.tensorrt.ILayer`, *index*: `int`, *dtype*: `tensorrt.tensorrt.DataType`) → None

Constraint layer to generate output data with given type. Note that this method cannot be used to set the data type of the second output tensor of the topK layer. The data type of the second output tensor of the topK layer is always `int32`.

Parameters

- **index** – The index of the output tensor to set the type.
- **dtype** – `DataType` of the output.

5.3 Layers

5.3.1 PaddingMode

`tensorrt.PaddingMode`

Enumerates types of padding available in convolution, deconvolution and pooling layers. `Padding mode` takes precedence if both `padding_mode` and `pre_padding` are set.

EXPLICIT* corresponds to explicit padding.

SAME* implicitly calculates padding such that the output dimensions are the same as the input dimensions. For convolution and pooling, output dimensions are determined by `ceil(input dimensions, stride)`.

CAFFE* corresponds to symmetric padding.

Members:

EXPLICIT_ROUND_DOWN : Use explicit padding, rounding the output size down

EXPLICIT_ROUND_UP : Use explicit padding, rounding the output size up

SAME_UPPER : Use SAME padding, with `pre_padding <= post_padding`

SAME_LOWER : Use SAME padding, with `pre_padding >= post_padding`

CAFFE_ROUND_DOWN : Use CAFFE padding, rounding the output size down

CAFFE_ROUND_UP : Use CAFFE padding, rounding the output size up

5.3.2 IConvolutionLayer

class `tensorrt.IConvolutionLayer`

A convolution layer in an *INetworkDefinition* .

This layer performs a correlation operation between 3-dimensional filter with a 4-dimensional tensor to produce another 4-dimensional tensor.

An optional bias argument is supported, which adds a per-channel constant to each value in the output.

Variables

- **kernel_size** – *DimsHW* The HW kernel size of the convolution.
- **num_output_maps** – `int` The number of output maps for the convolution.
- **stride** – *DimsHW* The stride of the convolution. Default: (1, 1)
- **padding** – *DimsHW* The padding of the convolution. The input will be zero-padded by this number of elements in the height and width directions. If the padding is asymmetric, this value corresponds to the pre-padding. Default: (0, 0)
- **pre_padding** – *DimsHW* The pre-padding. The start of input will be zero-padded by this number of elements in the height and width directions. Default: (0, 0)
- **post_padding** – *DimsHW* The post-padding. The end of input will be zero-padded by this number of elements in the height and width directions. Default: (0, 0)
- **padding_mode** – *PaddingMode* The padding mode. Padding mode takes precedence if both `IConvolutionLayer.padding_mode` and either `IConvolutionLayer.pre_padding` or `IConvolutionLayer.post_padding` are set.
- **num_groups** – `int` The number of groups for a convolution. The input tensor channels are divided into this many groups, and a convolution is executed for each group, using a filter per group. The results of the group convolutions are concatenated to form the output. **Note** When using groups in int8 mode, the size of the groups (i.e. the channel count divided by the group count) must be a multiple of 4 for both input and output. Default: 1.
- **kernel** – *Weights* The kernel weights for the convolution. The weights are specified as a contiguous array in *GKCRS* order, where *G* is the number of groups, *K* the number of output feature maps, *C* the number of input channels, and *R* and *S* are the height and width of the filter.
- **bias** – *Weights* The bias weights for the convolution. Bias is optional. To omit bias, set this to an empty *Weights* object. The bias is applied per-channel, so the number of weights (if non-zero) must be equal to the number of output feature maps.
- **dilation** – *DimsHW* The dilation for a convolution. Default: (1, 1)
- **kernel_size_nd** – *Dims* The multi-dimension kernel size of the convolution.
- **stride_nd** – *Dims* The multi-dimension stride of the convolution. Default: (1, ..., 1)
- **padding_nd** – *Dims* The multi-dimension padding of the convolution. The input will be zero-padded by this number of elements in each dimension. If the padding is asymmetric, this value corresponds to the pre-padding. Default: (0, ..., 0)
- **dilation_nd** – *Dims* The multi-dimension dilation for the convolution. Default: (1, ..., 1)

5.3.3 IFullyConnectedLayer

class tensorrt.IFullyConnectedLayer

A fully connected layer in an *INetworkDefinition* .

This layer expects an input tensor of three or more non-batch dimensions. The input is automatically reshaped into an $M \times V$ tensor X , where V is a product of the last three dimensions and M is a product of the remaining dimensions (where the product over 0 dimensions is defined as 1). For example:

- If the input tensor has shape $\{C, H, W\}$, then the tensor is reshaped into $\{1, C * H * W\}$.
- If the input tensor has shape $\{P, C, H, W\}$, then the tensor is reshaped into $\{P, C * H * W\}$.

The layer then performs:

$$Y := \text{matmul}(X, W^T) + \text{bias}$$

Where X is the $M \times V$ tensor defined above, W is the $K \times V$ weight tensor of the layer, and bias is a row vector size K that is broadcasted to $M \times K$. K is the number of output channels, and configurable via `IFullyConnectedLayer.num_output_channels` . If bias is not specified, it is implicitly 0 .

The $M \times K$ result Y is then reshaped such that the last three dimensions are $\{K, 1, 1\}$ and the remaining dimensions match the dimensions of the input tensor. For example:

- If the input tensor has shape $\{C, H, W\}$, then the output tensor will have shape $\{K, 1, 1\}$.
- If the input tensor has shape $\{P, C, H, W\}$, then the output tensor will have shape $\{P, K, 1, 1\}$.

Variables

- **num_output_channels** – int The number of output channels K from the fully connected layer.
- **kernel** – *Weights* The kernel weights, given as a $K \times C$ matrix in row-major order.
- **bias** – *Weights* The bias weights. Bias is optional. To omit bias, set this to an empty *Weights* object.

5.3.4 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.

5.3.5 IPoolingLayer

tensorrt.PoolingType

The type of pooling to perform in a pooling layer.

Members:

MAX : Maximum over elements

AVERAGE : Average over elements. If the tensor is padded, the count includes the padding

MAX_AVERAGE_BLEND : Blending between the max pooling and average pooling: $(1 - blendFactor) * maxPool + blendFactor * avgPool$

class tensorrt.IPoolingLayer

A Pooling layer in an *INetworkDefinition* . The layer applies a reduction operation within a window over the input.

Variables

- **type** – *PoolingType* The type of pooling to be performed.
- **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: $max_{average}blendPool = (1 - blendFactor) * maxPool + blendFactor * avgPool$. `blend_factor` is a user value in [0,1] with the default value of 0.0. This value only applies for the `PoolingType.MAX_AVERAGE_BLEND` mode.

- **average_count_excludes_padding** – bool Whether average pooling uses as a denominator the overlap area between the window and the unpadded input. If this is not set, the denominator is the overlap between the pooling window and the padded input. Default: True
- **window_size_nd** – *Dims* The multi-dimension window size for pooling.
- **stride_nd** – *Dims* The multi-dimension stride for pooling. Default: (1, ..., 1)
- **padding_nd** – *Dims* The multi-dimension padding for pooling. Default: (0, ..., 0)

5.3.6 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].

5.3.7 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.

5.3.8 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.

5.3.9 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. 0 is the major axis (excluding the batch dimension). The default is the number of non-batch axes in the tensor minus three (e.g. for an NCHW input it would be 0), or 0 if there are fewer than 3 non-batch axes.

5.3.10 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)

5.3.11 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.

5.3.12 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.

5.3.13 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*
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

$$H[t] := \text{ReLU}(W[i].X[t] + R[i].H[t - 1] + Wb[i] + Rb[i])$$

TANH

$$H[t] := \text{tanh}(W[i].X[t] + R[i].H[t - 1] + Wb[i] + Rb[i])$$

LSTM

$$\begin{aligned} i[t] &:= \text{sigmoid}(W[i].X[t] + R[i].H[t - 1] + Wb[i] + Rb[i]) \\ f[t] &:= \text{sigmoid}(W[f].X[t] + R[f].H[t - 1] + Wb[f] + Rb[f]) \\ o[t] &:= \text{sigmoid}(W[o].X[t] + R[o].H[t - 1] + Wb[o] + Rb[o]) \\ c[t] &:= \text{tanh}(W[c].X[t] + R[c].H[t - 1] + Wb[c] + Rb[c]) \end{aligned}$$

$$\begin{aligned} C[t] &:= f[t] * C[t - 1] + i[t] * c[t] \\ H[t] &:= o[t] * \text{tanh}(C[t]) \end{aligned}$$

GRU

$$\begin{aligned} z[t] &:= \text{sigmoid}(W[z].X[t] + R[z].H[t - 1] + Wb[z] + Rb[z]) \\ r[t] &:= \text{sigmoid}(W[r].X[t] + R[r].H[t - 1] + Wb[r] + Rb[r]) \\ h[t] &:= \text{tanh}(W[h].X[t] + r[t] * (R[h].H[t - 1] + Rb[h]) + Wb[h]) \\ H[t] &:= (1 - z[t]) * h[t] + z[t] * H[t - 1] \end{aligned}$$

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

5.3.13.1 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 length of the data being processed by the RNN for use in computing other values.
- **seq_lengths** – *ITensor* Individual sequence lengths in the batch with the *ITensor* provided. The `seq_lengths ITensor` should be a $\{N_1, \dots, N_p\}$ tensor, where $N_1..N_p$ 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 2x 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 2x 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()`.

5.3.14 IPluginV2Layer

`class tensorrt.IPluginV2Layer`

A plugin layer in an *INetworkDefinition* .

Variables plugin – IPluginV2 The plugin for the layer.

5.3.15 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 float datatype.

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.

5.3.16 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.

5.3.17 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.

5.3.18 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.

5.3.19 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.

5.3.20 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.

5.3.21 ISliceLayer

class `tensorrt.ISliceLayer`

A slice layer in an *INetworkDefinition*.

The slice layer has two variants, static and dynamic. Static slice specifies the start, size, and stride dimensions at layer creation time via *Dims* and can use the `get/set` accessor functions of the *ISliceLayer*. Dynamic slice specifies one or more of start, size 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 $2 * Dims.MAX_DIMS$.

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.

5.3.22 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.

5.3.23 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.
- **axes** – *TopKOperation* The axes along which to reduce.

5.3.24 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.

5.3.25 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.

5.3.26 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.

5.3.27 IConstantLayer

class `tensorrt.IConstantLayer`

A constant layer in an *INetworkDefinition* .

Note: This layer does not support boolean types.

Variables

- **weights** – *Weights* The weights for the layer.
- **shape** – *Dims* The shape of the layer.

5.3.28 IResizeLayer

`tensorrt.ResizeMode`

Various modes of resize in the resize layer.

Members:

NEAREST : 1D, 2D, and 3D nearest neighbor resizing.

LINEAR : Can handle linear, bilinear, trilinear resizing.

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 $N > 0$.

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 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 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.

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.

5.3.29 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.IIteratorLayer`

Return layer that subscribes 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.

5.3.29.1 ILoopBoundaryLayer

class tensorrt.ILoopBoundaryLayer

Variables `loop` – *ILoop* associated with this boundary layer.

5.3.29.1.1 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*

5.3.29.1.2 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.

5.3.29.1.3 IIteratorLayer

class tensorrt.IIteratorLayer

Variables

- **axis** – The axis to iterate over
- **reverse** – For `reverse=false`, the layer is equivalent to `add_gather(tensor, I, 0)` where `I` is a scalar tensor containing the loop iteration number. For `reverse=true`, the layer is equivalent to `add_gather(tensor, M-1-I, 0)` where `M` is the trip count computed from `TripLimits` of kind `COUNT`. The default is `reverse=false`.

5.3.29.1.4 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.

5.3.30 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

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.

Parameters

- **index** – the index of the input to modify.
- **tensor** – the input tensor.

5.3.31 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.

5.3.32 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.

5.3.33 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.

5.3.34 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.

5.3.35 IConditionLayer

class `tensorrt.IConditionLayer`

Describes the boolean condition of an if-conditional.

5.3.36 IIfConditionalOutputLayer

class `tensorrt.IIfConditionalOutputLayer`

Describes kinds of if-conditional outputs.

5.3.37 IIfConditionalInputLayer

class `tensorrt.IIfConditionalInputLayer`

Describes kinds of if-conditional inputs.

5.3.38 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.

5.3.39 IAssertionLayer

class `tensorrt.IAssertionLayer`

An assertion layer in an *INetworkDefinition* .

This layer implements assertions. The input must be a boolean shape tensor. If any element of it is `False`, a build-time or run-time error occurs. Asserting equality of input dimensions may help the optimizer.

Variables `message` – `string` Message to print if the assertion fails.

6.1 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.

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.

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` .

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

tensorrt.CalibrationAlgoType

Version of calibration algorithm to use.

Members:

LEGACY_CALIBRATION

ENTROPY_CALIBRATION

ENTROPY_CALIBRATION_2

MINMAX_CALIBRATION

class tensorrt.IInt8Calibrator(*self*: tensorrt.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*: tensorrt.tensorrt.IInt8Calibrator) → tensorrt.tensorrt.CalibrationAlgoType

Get the algorithm used by this calibrator.

Returns The algorithm used by this calibrator.

get_batch(*self*: tensorrt.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.IInt8Calibrator`) → int

Get the batch size used for calibration batches.

Returns The batch size.

read_calibration_cache(*self*: `tensorrt.tensorrt.IInt8Calibrator`) → 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.IInt8Calibrator`, *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 fall-back option if the other calibrators yield poor results.

To implement a custom calibrator, ensure that you explicitly instantiate the base class in `__init__()` :

```
class MyCalibrator(trt.IInt8LegacyCalibrator):
    def __init__(self):
        trt.IInt8LegacyCalibrator.__init__(self)
```

Variables

- **quantile** – float The quantile (between 0 and 1) that will be used to select the region maximum when the quantile method is in use. See the user guide for more details on how the quantile is used.
- **regression_cutoff** – float The fraction (between 0 and 1) of the maximum used to define the regression cutoff when using regression to determine the region maximum. See the user guide for more details on how the regression cutoff is used

get_algorithm(*self*: `tensorrt.tensorrt.IInt8LegacyCalibrator`) → `tensorrt.tensorrt.CalibrationAlgoType`

Signals that this is the legacy calibrator.

Returns `CalibrationAlgoType.LEGACY_CALIBRATION`

get_batch(*self*: `tensorrt.tensorrt.IInt8LegacyCalibrator`, *names*: `List[str]`) → `List[int]`

Get a batch of input for calibration. The batch size of the input must match the batch size returned by `get_batch_size()`.

A possible implementation may look like this:

```
def get_batch(names):
    try:
        # Assume self.batches is a generator that provides batch data.
        data = next(self.batches)
        # Assume that self.device_input is a device buffer allocated by the
        ↪ constructor.
        cuda.memcpy_htod(self.device_input, data)
        return [int(self.device_input)]
    except StopIteration:
        # When we're out of batches, we return either [] or None.
        # This signals to TensorRT that there is no calibration data remaining.
        return None
```

Parameters *names* – The names of the network inputs for each object in the bindings array.

Returns A list of device memory pointers set to the memory containing each network input data, or an empty list if there are no more batches for calibration. You can allocate these device buffers with pycuda, for example, and then cast them to `int` to retrieve the pointer.

get_batch_size(*self*: `tensorrt.tensorrt.IInt8LegacyCalibrator`) → `int`

Get the batch size used for calibration batches.

Returns The batch size.

read_calibration_cache(*self*: `tensorrt.tensorrt.IInt8LegacyCalibrator`) → `buffer`

Load a calibration cache.

Calibration is potentially expensive, so it can be useful to generate the calibration data once, then use it on subsequent builds of the network. The cache includes the regression cutoff and quantile values used to generate it, and will not be used if these do not match the settings of the current calibrator. However, the network should also be recalibrated if its structure changes, or the input data set changes, and it is the responsibility of the application to ensure this.

Reading a cache is just like reading any other file in Python. For example, one possible implementation is:

```
def read_calibration_cache(self):
    # If there is a cache, use it instead of calibrating again. Otherwise,
    # implicitly return None.
    if os.path.exists(self.cache_file):
        with open(self.cache_file, "rb") as f:
            return f.read()
```

Returns A cache object or None if there is no data.

write_calibration_cache(*self*: `tensorrt.tensorrt.IInt8LegacyCalibrator`, *cache*: `buffer`) → `None`

Save a calibration cache.

Writing a cache is just like writing any other buffer in Python. For example, one possible implementation is:

```
def write_calibration_cache(self, cache):
    with open(self.cache_file, "wb") as f:
        f.write(cache)
```

Parameters `cache` – The calibration cache to write.

7.3 IInt8EntropyCalibrator

class `tensorrt.IInt8EntropyCalibrator`(*self*: `tensorrt.tensorrt.IInt8EntropyCalibrator`) → `None`

Extends the `IInt8Calibrator` class.

To implement a custom calibrator, ensure that you explicitly instantiate the base class in `__init__()` :

```
class MyCalibrator(trt.IInt8EntropyCalibrator):
    def __init__(self):
        trt.IInt8EntropyCalibrator.__init__(self)
```

This is the Legacy Entropy calibrator. It is less complicated than the legacy calibrator and produces better results.

get_algorithm(*self*: `tensorrt.tensorrt.IInt8EntropyCalibrator`) → `tensorrt.tensorrt.CalibrationAlgoType`

Signals that this is the entropy calibrator.

Returns `CalibrationAlgoType.ENTROPY_CALIBRATION`

get_batch(*self*: `tensorrt.tensorrt.IInt8EntropyCalibrator`, *names*: `List[str]`) → `List[int]`

Get a batch of input for calibration. The batch size of the input must match the batch size returned by `get_batch_size()`.

A possible implementation may look like this:

```

def get_batch(names):
    try:
        # Assume self.batches is a generator that provides batch data.
        data = next(self.batches)
        # Assume that self.device_input is a device buffer allocated by the
        ↪constructor.
        cuda.memcpy_htod(self.device_input, data)
        return [int(self.device_input)]
    except StopIteration:
        # When we're out of batches, we return either [] or None.
        # This signals to TensorRT that there is no calibration data remaining.
        return None
    
```

Parameters `names` – The names of the network inputs for each object in the bindings array.

Returns A list of device memory pointers set to the memory containing each network input data, or an empty list if there are no more batches for calibration. You can allocate these device buffers with `pycuda`, for example, and then cast them to `int` to retrieve the pointer.

get_batch_size(*self*: `tensorrt.tensorrt.IInt8EntropyCalibrator`) → `int`

Get the batch size used for calibration batches.

Returns The batch size.

read_calibration_cache(*self*: `tensorrt.tensorrt.IInt8EntropyCalibrator`) → `buffer`

Load a calibration cache.

Calibration is potentially expensive, so it can be useful to generate the calibration data once, then use it on subsequent builds of the network. The cache includes the regression cutoff and quantile values used to generate it, and will not be used if these do not match the settings of the current calibrator. However, the network should also be recalibrated if its structure changes, or the input data set changes, and it is the responsibility of the application to ensure this.

Reading a cache is just like reading any other file in Python. For example, one possible implementation is:

```

def read_calibration_cache(self):
    # If there is a cache, use it instead of calibrating again. Otherwise,
    ↪implicitly return None.
    if os.path.exists(self.cache_file):
        with open(self.cache_file, "rb") as f:
            return f.read()
    
```

Returns A cache object or `None` if there is no data.

write_calibration_cache(*self*: `tensorrt.tensorrt.IInt8EntropyCalibrator`, *cache*: `buffer`) → `None`

Save a calibration cache.

Writing a cache is just like writing any other buffer in Python. For example, one possible implementation is:

```

def write_calibration_cache(self, cache):
    with open(self.cache_file, "wb") as f:
        f.write(cache)
    
```

Parameters `cache` – The calibration cache to write.

7.4 IInt8EntropyCalibrator2

class `trt.IInt8EntropyCalibrator2`(*self*: `trt.IInt8EntropyCalibrator2`) → None
 Extends the `IInt8Calibrator` class.

To implement a custom calibrator, ensure that you explicitly instantiate the base class in `__init__()` :

```
class MyCalibrator(trt.IInt8EntropyCalibrator2):
    def __init__(self):
        trt.IInt8EntropyCalibrator2.__init__(self)
```

This is the preferred calibrator. This is the required calibrator for DLA, as it supports per activation tensor scaling.

get_algorithm(*self*: `trt.IInt8EntropyCalibrator2`) → `trt.CalibrationAlgoType`
 Signals that this is the entropy calibrator 2.

Returns `CalibrationAlgoType.ENTROPY_CALIBRATION_2`

get_batch(*self*: `trt.IInt8EntropyCalibrator2`, *names*: `List[str]`) → `List[int]`

Get a batch of input for calibration. The batch size of the input must match the batch size returned by `get_batch_size()`.

A possible implementation may look like this:

```
def get_batch(names):
    try:
        # Assume self.batches is a generator that provides batch data.
        data = next(self.batches)
        # Assume that self.device_input is a device buffer allocated by the
        ↪ constructor.
        cuda.memcpy_htod(self.device_input, data)
        return [int(self.device_input)]
    except StopIteration:
        # When we're out of batches, we return either [] or None.
        # This signals to TensorRT that there is no calibration data remaining.
        return None
```

Parameters *names* – The names of the network inputs for each object in the bindings array.

Returns A list of device memory pointers set to the memory containing each network input data, or an empty list if there are no more batches for calibration. You can allocate these device buffers with `pycuda`, for example, and then cast them to `int` to retrieve the pointer.

get_batch_size(*self*: `trt.IInt8EntropyCalibrator2`) → `int`
 Get the batch size used for calibration batches.

Returns The batch size.

read_calibration_cache(*self*: `trt.IInt8EntropyCalibrator2`) → `buffer`
 Load a calibration cache.

Calibration is potentially expensive, so it can be useful to generate the calibration data once, then use it on subsequent builds of the network. The cache includes the regression cutoff and quantile values used to generate it, and will not be used if these do not match the settings of the current calibrator. However, the network should also be recalibrated if its structure changes, or the input data set changes, and it is the responsibility of the application to ensure this.

Reading a cache is just like reading any other file in Python. For example, one possible implementation is:

```
def read_calibration_cache(self):
    # If there is a cache, use it instead of calibrating again. Otherwise,
    ↪ implicitly return None.
    if os.path.exists(self.cache_file):
        with open(self.cache_file, "rb") as f:
            return f.read()
```

Returns A cache object or None if there is no data.

write_calibration_cache(*self*: `tensorrt.tensorrt.IInt8EntropyCalibrator2`, *cache*: `buffer`) → None
Save a calibration cache.

Writing a cache is just like writing any other buffer in Python. For example, one possible implementation is:

```
def write_calibration_cache(self, cache):
    with open(self.cache_file, "wb") as f:
        f.write(cache)
```

Parameters `cache` – The calibration cache to write.

7.5 IInt8MinMaxCalibrator

class `tensorrt.IInt8MinMaxCalibrator`(*self*: `tensorrt.tensorrt.IInt8MinMaxCalibrator`) → None
Extends the `IInt8Calibrator` class.

To implement a custom calibrator, ensure that you explicitly instantiate the base class in `__init__()` :

```
class MyCalibrator(trt.IInt8MinMaxCalibrator):
    def __init__(self):
        trt.IInt8MinMaxCalibrator.__init__(self)
```

This is the preferred calibrator for NLP tasks for all backends. It supports per activation tensor scaling.

get_algorithm(*self*: `tensorrt.tensorrt.IInt8MinMaxCalibrator`) → `tensorrt.tensorrt.CalibrationAlgoType`
Signals that this is the minmax calibrator.

Returns `CalibrationAlgoType.MINMAX_CALIBRATION`

get_batch(*self*: `tensorrt.tensorrt.IInt8MinMaxCalibrator`, *names*: `List[str]`) → `List[int]`

Get a batch of input for calibration. The batch size of the input must match the batch size returned by `get_batch_size()`.

A possible implementation may look like this:

```
def get_batch(names):
    try:
        # Assume self.batches is a generator that provides batch data.
        data = next(self.batches)
        # Assume that self.device_input is a device buffer allocated by the
        ↪ constructor.
        cuda.memcpy_htod(self.device_input, data)
```

(continues on next page)

(continued from previous page)

```

    return [int(self.device_input)]
except StopIteration:
    # When we're out of batches, we return either [] or None.
    # This signals to TensorRT that there is no calibration data remaining.
    return None

```

Parameters *names* – The names of the network inputs for each object in the bindings array.

Returns A list of device memory pointers set to the memory containing each network input data, or an empty list if there are no more batches for calibration. You can allocate these device buffers with pycuda, for example, and then cast them to `int` to retrieve the pointer.

get_batch_size(*self*: `tensorrt.tensorrt.IInt8MinMaxCalibrator`) → `int`

Get the batch size used for calibration batches.

Returns The batch size.

read_calibration_cache(*self*: `tensorrt.tensorrt.IInt8MinMaxCalibrator`) → `buffer`

Load a calibration cache.

Calibration is potentially expensive, so it can be useful to generate the calibration data once, then use it on subsequent builds of the network. The cache includes the regression cutoff and quantile values used to generate it, and will not be used if these do not match the settings of the current calibrator. However, the network should also be recalibrated if its structure changes, or the input data set changes, and it is the responsibility of the application to ensure this.

Reading a cache is just like reading any other file in Python. For example, one possible implementation is:

```

def read_calibration_cache(self):
    # If there is a cache, use it instead of calibrating again. Otherwise,
    # implicitly return None.
    if os.path.exists(self.cache_file):
        with open(self.cache_file, "rb") as f:
            return f.read()

```

Returns A cache object or `None` if there is no data.

write_calibration_cache(*self*: `tensorrt.tensorrt.IInt8MinMaxCalibrator`, *cache*: `buffer`) → `None`

Save a calibration cache.

Writing a cache is just like writing any other buffer in Python. For example, one possible implementation is:

```

def write_calibration_cache(self, cache):
    with open(self.cache_file, "wb") as f:
        f.write(cache)

```

Parameters *cache* – The calibration cache to write.

ALGORITHM SELECTOR

class `tensorrt.IAlgorithmIOInfo`

This class carries information about input or output of the algorithm. `IAlgorithmIOInfo` for all the input and output along with `IAlgorithmVariant` denotes the variation of algorithm and can be used to select or reproduce an algorithm using `IAlgorithmSelector.select_algorithms()`.

Variables

- **tensor_format** – *TensorFormat* TensorFormat of the input/output of algorithm.
- **dtype** – *DataType* DataType of the input/output of algorithm.
- **strides** – *Dims* strides of the input/output tensor of algorithm.

`__init__`(*args, **kwargs)

class `tensorrt.IAlgorithmVariant`

provides a unique 128-bit identifier, which along with the input and output information denotes the variation of algorithm and can be used to select or reproduce an algorithm, using `IAlgorithmSelector.select_algorithms()` see `IAlgorithmIOInfo`, `IAlgorithm`, `IAlgorithmSelector.select_algorithms()` note A single implementation can have multiple tactics.

Variables

- **implementation** – `int` implementation of the algorithm.
- **tactic** – `int` tactic of the algorithm.

`__init__`(*args, **kwargs)

class `tensorrt.IAlgorithmContext`

Describes the context and requirements, that could be fulfilled by one or more instances of `IAlgorithm`. see `IAlgorithm`

Variables

- **name** – `str` name of the algorithm node.
- **num_inputs** – `int` number of inputs of the algorithm.
- **num_outputs** – `int` number of outputs of the algorithm.

`__init__`(*args, **kwargs)

`get_shape`(self: `tensorrt.tensorrt.IAlgorithmContext`, index: `int`) → `List[tensorrt.tensorrt.Dims]`

Get the minimum / optimum / maximum dimensions for a dynamic input tensor.

Parameters `index` – Index of the input or output of the algorithm. Incremental numbers assigned to indices of inputs and the outputs.

Returns A *List[Dims]* of length 3, containing the minimum, optimum, and maximum shapes, in that order. If the shapes have not been set yet, an empty list is returned.`

class `tensorrt.IAlgorithm`

Application-implemented interface for selecting and reporting the tactic selection of a layer. Tactic Selection is a step performed by the builder for deciding best algorithms for a layer.

Variables

- **algorithm_variant** – `IAAlgorithmVariant`& the algorithm variant.
- **timing_msec** – float The time in milliseconds to execute the algorithm.
- **workspace_size** – int The size of the GPU temporary memory in bytes which the algorithm uses at execution time.

`__init__(*args, **kwargs)`

`get_algorithm_io_info(self: tensorrt.tensorrt.IAlgorithm, index: int) → tensorrt.tensorrt.IAlgorithmIOInfo`

A single call for both inputs and outputs. Incremental numbers assigned to indices of inputs and the outputs.

Parameters `index` – Index of the input or output of the algorithm. Incremental numbers assigned to indices of inputs and the outputs.

Returns A `IAAlgorithmIOInfo`&

class `tensorrt.IAlgorithmSelector(self: tensorrt.tensorrt.IAlgorithmSelector) → None`

Interface implemented by application for selecting and reporting algorithms of a layer provided by the builder. note A layer in context of algorithm selection may be different from `ILayer` in `INetworkDefiniton`. For example, an algorithm might be implementing a conglomeration of multiple `ILayers` in `INetworkDefinition`.

To implement a custom algorithm selector, ensure that you explicitly instantiate the base class in `__init__()` :

```
class MyAlgoSelector(trt.IAlgorithmSelector):
    def __init__(self):
        trt.IAlgorithmSelector.__init__(self)
```

`__init__(self: tensorrt.tensorrt.IAlgorithmSelector) → None`

`report_algorithms(self: tensorrt.tensorrt.IAlgorithmSelector, contexts: List[tensorrt.tensorrt.IAlgorithmContext], choices: List[tensorrt.tensorrt.IAlgorithm]) → None`

Called by TensorRT to report choices it made.

Note: For a given optimization profile, this call comes after all calls to `select_algorithms`. `choices[i]` is the choice that TensorRT made for `algoContexts[i]`, for `i` in `[0, num_algorithms-1]`

For example, a possible implementation may look like this:

```
def report_algorithms(self, contexts, choices):
    # Prints the time of the chosen algorithm by TRT from the
    # selection list passed in by select_algorithms
    for choice in choices:
        print(choice.timing_msec)
```

Parameters

- **contexts** – The list of all algorithm contexts.

- **choices** – The list of algorithm choices made by TensorRT corresponding to each context.

select_algorithms(*self*: tensorrt.tensorrt.IAlgorithmSelector, *context*: tensorrt.tensorrt.IAlgorithmContext, *choices*: List[tensorrt.tensorrt.IAlgorithm]) → List[int]

Select Algorithms for a layer from the given list of algorithm choices.

Note: TRT uses its default algorithm selection to choose from the list returned by the user. If the returned list is empty, TRT’s default algorithm selection is used unless strict type constraints are set. The list of choices is valid only for this specific algorithm context.

For example, the simplest implementation looks like this:

```
def select_algorithms(self, context, choices):
    assert len(choices) > 0
    return list(range(len(choices)))
```

Parameters

- **context** – The context for which the algorithm choices are valid.
- **choices** – The list of algorithm choices to select for implementation of this layer.

Returns A List[int] indicating the indices from the choices vector that TensorRT should choose from.

UFF PARSER

tensorrt.UffInputOrder

The different possible supported input orders.

Members:

NCHW

NHWC

NC

class `tensorrt.UffParser`(*self*: `tensorrt.tensorrt.UffParser`) → None

This class is used for parsing models described using the UFF format.

Variables

- **uff_required_version_major** – int Version Major of the UFF.
- **uff_required_version_minor** – int Version Minor of the UFF.
- **uff_required_version_patch** – int Version Patch of the UFF.
- **plugin_namespace** – str The namespace used to lookup and create plugins in the network.
- **error_recorder** – *IErrorRecorder* Application-implemented error reporting interface for TensorRT objects.

__del__(*self*: `tensorrt.tensorrt.UffParser`) → None

__exit__(*exc_type*, *exc_value*, *traceback*)

Context managers are deprecated and have no effect. Objects are automatically freed when the reference count reaches 0.

__init__(*self*: `tensorrt.tensorrt.UffParser`) → None

parse(*self*: `tensorrt.tensorrt.UffParser`, *file*: str, *network*: `tensorrt.tensorrt.INetworkDefinition`, *weights_type*: `tensorrt.tensorrt.DataType = <DataType.FLOAT: 0>`) → bool

Parse a UFF file.

Parameters

- **file** – File name of the UFF file.
- **network** – Network in which the *UffParser* will fill the layers.
- **weights_type** – The type on which the weights will be transformed in.

Returns True if the UFF file is parsed without error.

parse_buffer(*self*: *tensorrt.tensorrt.UffParser*, *buffer*: *buffer*, *network*: *tensorrt.tensorrt.INetworkDefinition*, *weights_type*: *tensorrt.tensorrt.DataType* = <*DataType.FLOAT: 0*>) → bool

Parse a UFF buffer - useful if the file is already live in memory.

Parameters

- **buffer** – The UFF buffer.
- **network** – Network in which the UFFParser will fill the layers.
- **weights_type** – The type on which the weights will be transformed in.

Returns True if the UFF buffer is parsed without error.

register_input(*self*: *tensorrt.tensorrt.UffParser*, *name*: *str*, *shape*: *tensorrt.tensorrt.Dims*, *order*: *tensorrt.tensorrt.UffInputOrder* = <*UffInputOrder.NCHW: 0*>) → bool

Register an input name of a UFF network with the associated Dimensions.

Parameters

- **name** – Input name.
- **shape** – Input shape.
- **order** – Input order on which the framework input was originally.

Returns True if the name registers without error.

register_output(*self*: *tensorrt.tensorrt.UffParser*, *name*: *str*) → bool

Register an output name of a UFF network.

Parameters **output_name** – Output name.

Returns True if the name registers without error.

9.1 Fields

tensorrt.FieldType

The possible field types for the custom layer.

Members:

FLOAT

INT32

CHAR

DIMS

DATATYPE

UNKNOWN

class *tensorrt.FieldMap*(*self*: *tensorrt.tensorrt.FieldMap*, *name*: *str*, *data*: *capsule*, *type*: *tensorrt.tensorrt.FieldType*, *length*: *int* = 1) → None

This is a class containing an array of field params used as a layer parameter for plugin layers. The node fields are passed by the parser to the API through the plugin constructor. The implementation of the plugin should parse the contents of the *FieldMap* as part of the plugin constructor.

Variables

- **name** – str field param
- **data** – capsule field param

- **type** – *FieldType* field param
- **length** – int field param

class tensorrt.**FieldCollection**

This class contains an array of *FieldMap* s.

Variables

- **num_fields** – int The number of *FieldMap* s.
- **fields** – capsule The array of *FieldMap* s.

CAFFE PARSER

class `tensorrt.IBlobNameToTensor`

This class is used to store and query *ITensor* s after they have been extracted from a Caffe model using the *CaffeParser* .

find(*self*: `tensorrt.tensorrt.IBlobNameToTensor`, *name*: *str*) → `tensorrt.tensorrt.ITensor`

Given a blob name, this function returns an *ITensor* object.

Parameters *name* – Caffe blob name for which the user wants the corresponding *ITensor* .

Returns A *ITensor* object corresponding to the queried name. If no such *ITensor* exists, then an empty object is returned.

class `tensorrt.CaffeParser`(*self*: `tensorrt.tensorrt.CaffeParser`) → None

This class is used for parsing Caffe models. It allows users to export models trained using Caffe to TRT.

Variables

- **plugin_factory_v2** – *ICaffePluginFactoryV2* The *ICaffePluginFactory* used to create the user defined plugins.
- **plugin_namespace** – *str* The namespace used to lookup and create plugins in the network.
- **protobuf_buffer_size** – *int* The buffer size for the parsing and storage of the learned model.
- **error_recorder** – *IErrorRecorder* Application-implemented error reporting interface for TensorRT objects.

__del__(*self*: `tensorrt.tensorrt.CaffeParser`) → None

__exit__(*exc_type*, *exc_value*, *traceback*)

Context managers are deprecated and have no effect. Objects are automatically freed when the reference count reaches 0.

__init__(*self*: `tensorrt.tensorrt.CaffeParser`) → None

parse(*self*: `tensorrt.tensorrt.CaffeParser`, *deploy*: *str*, *model*: *str*, *network*: `tensorrt.tensorrt.INetworkDefinition`, *dtype*: `tensorrt.tensorrt.DataType`) → `tensorrt.tensorrt.IBlobNameToTensor`

Parse a prototxt file and a binaryproto Caffe model to extract network definition and weights associated with the network, respectively.

Parameters

- **deploy** – The plain text, prototxt file used to define the network definition.
- **model** – The binaryproto Caffe model that contains the weights associated with the network.

- **network** – Network in which the CaffeParser will fill the layers.
- **dtype** – The type to which the weights will be transformed.

Returns An *IBlobNameToTensor* object that contains the extracted data.

parse_binary_proto(*self: tensorrt.tensorrt.CaffeParser, filename: str*) → *numpy.ndarray*

Parse and extract data stored in binaryproto file. The binaryproto file contains data stored in a binary blob. *parse_binary_proto()* converts it to an *numpy.ndarray* object.

Parameters filename – Path to file containing binary proto.

Returns *numpy.ndarray* An array that contains the extracted data.

parse_buffer(*self: tensorrt.tensorrt.CaffeParser, deploy_buffer: buffer, model_buffer: buffer, network: tensorrt.tensorrt.INetworkDefinition, dtype: tensorrt.tensorrt.DataType*) → *tensorrt.tensorrt.IBlobNameToTensor*

Parse a prototxt file and a binaryproto Caffe model to extract network definition and weights associated with the network, respectively.

Parameters

- **deploy_buffer** – The memory buffer containing the plain text deploy prototxt used to define the network definition.
- **model_buffer** – The binaryproto Caffe memory buffer that contains the weights associated with the network.
- **network** – Network in which the CaffeParser will fill the layers.
- **dtype** – The type to which the weights will be transformed.

Returns An *IBlobNameToTensor* object that contains the extracted data.

tensorrt.shutdown_protobuf_library() → None

Shuts down protocol buffers library.

10.1 Plugins

class *tensorrt.ICaffePluginFactoryV2*

Plugin factory used to configure plugins.

create_plugin(*self: tensorrt.tensorrt.ICaffePluginFactoryV2, layer_name: str, weights: std::vector<nvinfer1::Weights, std::allocator<nvinfer1::Weights> >*) → *tensorrt.tensorrt.IPluginV2*

Creates a plugin.

arg layer_name Name of layer associated with the plugin.

arg weights Weights used for the layer.

Returns The newly created *IPluginV2* .

is_plugin_v2(*self: tensorrt.tensorrt.ICaffePluginFactoryV2, layer_name: str*) → bool

A user implemented function that determines if a layer configuration is provided by an *IPluginV2* .

Parameters layer_name – Name of the layer which the user wishes to validate.

Returns True if the the layer configuration is provided by an *IPluginV2* .

ONNX PARSER

class `tensorrt.OnnxParser`(*self*: `tensorrt.tensorrt.OnnxParser`, *network*: `tensorrt.tensorrt.INetworkDefinition`,
logger: `tensorrt.tensorrt.ILogger`) → None

This class is used for parsing ONNX models into a TensorRT network definition

Variables `num_errors` – int The number of errors that occurred during prior calls to `parse()`

Parameters

- **network** – The network definition to which the parser will write.
- **logger** – The logger to use.

`__del__`(*self*: `tensorrt.tensorrt.OnnxParser`) → None

`__exit__`(*exc_type*, *exc_value*, *traceback*)

Context managers are deprecated and have no effect. Objects are automatically freed when the reference count reaches 0.

`__init__`(*self*: `tensorrt.tensorrt.OnnxParser`, *network*: `tensorrt.tensorrt.INetworkDefinition`, *logger*:
`tensorrt.tensorrt.ILogger`) → None

Parameters

- **network** – The network definition to which the parser will write.
- **logger** – The logger to use.

`clear_errors`(*self*: `tensorrt.tensorrt.OnnxParser`) → None

Clear errors from prior calls to `parse()`

`get_error`(*self*: `tensorrt.tensorrt.OnnxParser`, *index*: int) → `nvonnxparser::IParserError`

Get an error that occurred during prior calls to `parse()`

Parameters `index` – Index of the error

`parse`(*self*: `tensorrt.tensorrt.OnnxParser`, *model*: `buffer`, *path*: `str = None`) → bool

Parse a serialized ONNX model into the TensorRT network.

Parameters

- **model** – The serialized ONNX model.
- **path** – The path to the model file. Only required if the model has externally stored weights.

Returns true if the model was parsed successfully

`parse_from_file`(*self*: `tensorrt.tensorrt.OnnxParser`, *model*: `str`) → bool

Parse an ONNX model from file into a TensorRT network.

Parameters `model` – The path to an ONNX model.

Returns true if the model was parsed successfully

parse_with_weight_descriptors(*self*: [tensorrt.tensorrt.OnnxParser](#), *model*: *buffer*) → bool

Parse a serialized ONNX model into the TensorRT network with consideration of user provided weights.

Parameters **model** – The serialized ONNX model.

Returns true if the model was parsed successfully

supports_model(*self*: [tensorrt.tensorrt.OnnxParser](#), *model*: *buffer*, *path*: *str = None*) → Tuple[bool, [tensorrt.tensorrt.SubGraphCollection](#)]

Check whether TensorRT supports a particular ONNX model.

Parameters

- **model** – The serialized ONNX model.
- **path** – The path to the model file. Only required if the model has externally stored weights.

Returns Tuple[bool, List[Tuple[NodeIndices, bool]]] The first element of the tuple indicates whether the model is supported. The second indicates subgraphs (by node index) in the model and whether they are supported.

supports_operator(*self*: [tensorrt.tensorrt.OnnxParser](#), *op_name*: *str*) → bool

Returns whether the specified operator may be supported by the parser. Note that a result of true does not guarantee that the operator will be supported in all cases (i.e., this function may return false-positives).

Parameters **op_name** – The name of the ONNX operator to check for support

tensorrt.ErrorCode

The type of parser error

Members:

SUCCESS

INTERNAL_ERROR

MEM_ALLOC_FAILED

MODEL_DESERIALIZE_FAILED

INVALID_VALUE

INVALID_GRAPH

INVALID_NODE

UNSUPPORTED_GRAPH

UNSUPPORTED_NODE

class [tensorrt.ParserError](#)

code(*self*: [tensorrt.ParserError](#)) → [tensorrt.ErrorCode](#)

Returns The error code

desc(*self*: [tensorrt.ParserError](#)) → str

Returns Description of the error

file(*self*: [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), 20
 __del__() (*tensorrt.CaffeParser* method), 103
 __del__() (*tensorrt.IBuilderConfig* method), 15
 __del__() (*tensorrt.ICudaEngine* method), 22
 __del__() (*tensorrt.IExecutionContext* method), 27
 __del__() (*tensorrt.IHostMemory* method), 8
 __del__() (*tensorrt.INetworkDefinition* method), 39
 __del__() (*tensorrt.OnnxParser* method), 105
 __del__() (*tensorrt.Refitter* method), 31
 __del__() (*tensorrt.Runtime* method), 31
 __del__() (*tensorrt.UffParser* method), 99
 __exit__() (*tensorrt.Builder* method), 20
 __exit__() (*tensorrt.CaffeParser* method), 103
 __exit__() (*tensorrt.IBuilderConfig* method), 16
 __exit__() (*tensorrt.ICudaEngine* method), 22
 __exit__() (*tensorrt.IExecutionContext* method), 27
 __exit__() (*tensorrt.IHostMemory* method), 8
 __exit__() (*tensorrt.INetworkDefinition* method), 39
 __exit__() (*tensorrt.OnnxParser* method), 105
 __exit__() (*tensorrt.Refitter* method), 31
 __exit__() (*tensorrt.Runtime* method), 31
 __exit__() (*tensorrt.UffParser* method), 99
 __getitem__() (*tensorrt.ICudaEngine* method), 22
 __getitem__() (*tensorrt.INetworkDefinition* method), 39
 __init__() (*tensorrt.Builder* method), 20
 __init__() (*tensorrt.CaffeParser* method), 103
 __init__() (*tensorrt.EngineInspector* method), 38
 __init__() (*tensorrt.IAlgorithm* method), 96
 __init__() (*tensorrt.IAlgorithmContext* method), 95
 __init__() (*tensorrt.IAlgorithmIOInfo* method), 95
 __init__() (*tensorrt.IAlgorithmSelector* method), 96
 __init__() (*tensorrt.IAlgorithmVariant* method), 95
 __init__() (*tensorrt.IBuilderConfig* method), 16
 __init__() (*tensorrt.ICudaEngine* method), 22
 __init__() (*tensorrt.IExecutionContext* method), 27
 __init__() (*tensorrt.IGpuAllocator* method), 36
 __init__() (*tensorrt.IHostMemory* method), 8
 __init__() (*tensorrt.INetworkDefinition* method), 39
 __init__() (*tensorrt.OnnxParser* method), 105
 __init__() (*tensorrt.Refitter* method), 31

__init__() (*tensorrt.Runtime* method), 31
 __init__() (*tensorrt.UffParser* method), 99
 __len__() (*tensorrt.ICudaEngine* method), 22
 __len__() (*tensorrt.INetworkDefinition* method), 39

A

ActivationType (*in module tensorrt*), 58
 add_activation() (*tensorrt.INetworkDefinition* method), 39
 add_assertion() (*tensorrt.INetworkDefinition* method), 40
 add_concatenation() (*tensorrt.INetworkDefinition* method), 40
 add_constant() (*tensorrt.INetworkDefinition* method), 40
 add_convolution() (*tensorrt.INetworkDefinition* method), 40
 add_convolution_nd() (*tensorrt.INetworkDefinition* method), 40
 add_deconvolution() (*tensorrt.INetworkDefinition* method), 41
 add_deconvolution_nd() (*tensorrt.INetworkDefinition* method), 41
 add_dequantize() (*tensorrt.INetworkDefinition* method), 41
 add_einsum() (*tensorrt.INetworkDefinition* method), 42
 add_elementwise() (*tensorrt.INetworkDefinition* method), 42
 add_fill() (*tensorrt.INetworkDefinition* method), 42
 add_fully_connected() (*tensorrt.INetworkDefinition* method), 42
 add_gather() (*tensorrt.INetworkDefinition* method), 42
 add_gather_v2() (*tensorrt.INetworkDefinition* method), 43
 add_identity() (*tensorrt.INetworkDefinition* method), 43
 add_if_conditional() (*tensorrt.INetworkDefinition* method), 43
 add_input() (*tensorrt.IfConditional* method), 80
 add_input() (*tensorrt.INetworkDefinition* method), 43
 add_iterator() (*tensorrt.ILoop* method), 76
 add_loop() (*tensorrt.INetworkDefinition* method), 43

[add_loop_output\(\)](#) (*tensorrt.ILoop method*), 76
[add_lrn\(\)](#) (*tensorrt.INetworkDefinition method*), 43
[add_matrix_multiply\(\)](#) (*tensorrt.INetworkDefinition method*), 44
[add_optimization_profile\(\)](#) (*tensorrt.IBuilderConfig method*), 16
[add_output\(\)](#) (*tensorrt.IfConditional method*), 81
[add_padding\(\)](#) (*tensorrt.INetworkDefinition method*), 44
[add_padding_nd\(\)](#) (*tensorrt.INetworkDefinition method*), 44
[add_parametric_relu\(\)](#) (*tensorrt.INetworkDefinition method*), 44
[add_plugin_v2\(\)](#) (*tensorrt.INetworkDefinition method*), 45
[add_pooling\(\)](#) (*tensorrt.INetworkDefinition method*), 45
[add_pooling_nd\(\)](#) (*tensorrt.INetworkDefinition method*), 45
[add_quantize\(\)](#) (*tensorrt.INetworkDefinition method*), 45
[add_ragged_softmax\(\)](#) (*tensorrt.INetworkDefinition method*), 45
[add_recurrence\(\)](#) (*tensorrt.ILoop method*), 76
[add_reduce\(\)](#) (*tensorrt.INetworkDefinition method*), 46
[add_resize\(\)](#) (*tensorrt.INetworkDefinition method*), 46
[add_rnn_v2\(\)](#) (*tensorrt.INetworkDefinition method*), 46
[add_scale\(\)](#) (*tensorrt.INetworkDefinition method*), 47
[add_scale_nd\(\)](#) (*tensorrt.INetworkDefinition method*), 48
[add_scatter\(\)](#) (*tensorrt.INetworkDefinition method*), 48
[add_select\(\)](#) (*tensorrt.INetworkDefinition method*), 48
[add_shape\(\)](#) (*tensorrt.INetworkDefinition method*), 48
[add_shuffle\(\)](#) (*tensorrt.INetworkDefinition method*), 48
[add_slice\(\)](#) (*tensorrt.INetworkDefinition method*), 49
[add_softmax\(\)](#) (*tensorrt.INetworkDefinition method*), 49
[add_topk\(\)](#) (*tensorrt.INetworkDefinition method*), 49
[add_trip_limit\(\)](#) (*tensorrt.ILoop method*), 76
[add_unary\(\)](#) (*tensorrt.INetworkDefinition method*), 49
[allocate\(\)](#) (*tensorrt.IGpuAllocator method*), 36
[AllocatorFlag](#) (*in module tensorrt*), 36
[append\(\)](#) (*graphsurgeon.DynamicGraph method*), 128
[append\(\)](#) (*tensorrt.PluginFieldCollection method*), 84
[as_graph_def\(\)](#) (*graphsurgeon.StaticGraph method*), 126

B

[binding_is_input\(\)](#) (*tensorrt.ICudaEngine method*), 22
[build_engine\(\)](#) (*tensorrt.Builder method*), 20

[build_serialized_network\(\)](#) (*tensorrt.Builder method*), 20
[Builder](#) (*class in tensorrt*), 19
[BuilderFlag](#) (*in module tensorrt*), 14

C

[CaffeParser](#) (*class in tensorrt*), 103
[CalibrationAlgoType](#) (*in module tensorrt*), 87
[can_run_on_DLA\(\)](#) (*tensorrt.IBuilderConfig method*), 16
[clear\(\)](#) (*tensorrt.IErrorRecorder method*), 34
[clear\(\)](#) (*tensorrt.PluginFieldCollection method*), 84
[clear_errors\(\)](#) (*tensorrt.OnnxParser method*), 105
[clear_flag\(\)](#) (*tensorrt.IBuilderConfig method*), 16
[clear_quantization_flag\(\)](#) (*tensorrt.IBuilderConfig method*), 16
[code\(\)](#) (*tensorrt.ParserError method*), 106
[collapse_namespaces\(\)](#) (*graphsurgeon.DynamicGraph method*), 128
[combine\(\)](#) (*tensorrt.ITimingCache method*), 36
[create_builder_config\(\)](#) (*tensorrt.Builder method*), 20
[create_engine_inspector\(\)](#) (*tensorrt.ICudaEngine method*), 22
[create_execution_context\(\)](#) (*tensorrt.ICudaEngine method*), 23
[create_execution_context_without_device_memory\(\)](#) (*tensorrt.ICudaEngine method*), 23
[create_network\(\)](#) (*tensorrt.Builder method*), 20
[create_node\(\)](#) (*in module graphsurgeon*), 125
[create_optimization_profile\(\)](#) (*tensorrt.Builder method*), 20
[create_plugin\(\)](#) (*tensorrt.ICaffePluginFactoryV2 method*), 104
[create_plugin\(\)](#) (*tensorrt.IPluginCreator method*), 84
[create_plugin_node\(\)](#) (*in module graphsurgeon*), 125
[create_timing_cache\(\)](#) (*tensorrt.IBuilderConfig method*), 16

D

[DataType](#) (*in module tensorrt*), 5
[deallocate\(\)](#) (*tensorrt.IGpuAllocator method*), 37
[deregister_creator\(\)](#) (*tensorrt.IPluginRegistry method*), 85
[desc\(\)](#) (*tensorrt.ParserError method*), 106
[deserialize_cuda_engine\(\)](#) (*tensorrt.Runtime method*), 31
[deserialize_plugin\(\)](#) (*tensorrt.IPluginCreator method*), 84
[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), 128

E

ElementWiseOperation (in module tensorrt), 63

EngineCapability (in module tensorrt), 13

EngineInspector (class in tensorrt), 38

ErrorCode (in module tensorrt), 106

ErrorCodeTRT (in module tensorrt), 33

execute() (tensorrt.IExecutionContext method), 27

execute_async() (tensorrt.IExecutionContext method), 27

execute_async_v2() (tensorrt.IExecutionContext method), 28

execute_v2() (tensorrt.IExecutionContext method), 28

extend() (graphsurgeon.DynamicGraph method), 128

extend() (tensorrt.PluginFieldCollection method), 84

F

FieldCollection (class in tensorrt), 101

FieldMap (class in tensorrt), 100

FieldType (in module tensorrt), 100

file() (tensorrt.ParserError method), 106

FillOperation (in module tensorrt), 78

find() (tensorrt.IBlobNameToTensor method), 103

find_node_chains_by_op() (graphsurgeon.StaticGraph method), 126

find_node_inputs() (graphsurgeon.StaticGraph method), 126

find_node_inputs_by_name() (graphsurgeon.StaticGraph method), 126

find_node_inputs_by_op() (graphsurgeon.StaticGraph method), 127

find_nodes_by_name() (graphsurgeon.StaticGraph method), 127

find_nodes_by_op() (graphsurgeon.StaticGraph method), 127

find_nodes_by_path() (graphsurgeon.StaticGraph method), 127

forward_inputs() (graphsurgeon.DynamicGraph method), 128

free() (tensorrt.IGpuAllocator method), 37

from_tensorflow() (in module uff), 109

from_tensorflow_frozen_model() (in module uff), 110

func() (tensorrt.ParserError method), 106

G

get_algorithm() (tensorrt.IInt8Calibrator method), 87

get_algorithm() (tensorrt.IInt8EntropyCalibrator method), 90

get_algorithm() (tensorrt.IInt8EntropyCalibrator2 method), 92

get_algorithm() (tensorrt.IInt8LegacyCalibrator method), 89

get_algorithm() (tensorrt.IInt8MinMaxCalibrator method), 93

get_algorithm_io_info() (tensorrt.IAlgorithm method), 96

get_all() (tensorrt.Refitter method), 32

get_all_weights() (tensorrt.Refitter method), 32

get_batch() (tensorrt.IInt8Calibrator method), 87

get_batch() (tensorrt.IInt8EntropyCalibrator method), 90

get_batch() (tensorrt.IInt8EntropyCalibrator2 method), 92

get_batch() (tensorrt.IInt8LegacyCalibrator method), 89

get_batch() (tensorrt.IInt8MinMaxCalibrator method), 93

get_batch_size() (tensorrt.IInt8Calibrator method), 88

get_batch_size() (tensorrt.IInt8EntropyCalibrator method), 91

get_batch_size() (tensorrt.IInt8EntropyCalibrator2 method), 92

get_batch_size() (tensorrt.IInt8LegacyCalibrator method), 89

get_batch_size() (tensorrt.IInt8MinMaxCalibrator method), 94

get_bias_for_gate() (tensorrt.IRNNv2Layer method), 67

get_binding_bytes_per_component() (tensorrt.ICudaEngine method), 23

get_binding_components_per_element() (tensorrt.ICudaEngine method), 23

get_binding_dtype() (tensorrt.ICudaEngine method), 23

get_binding_format() (tensorrt.ICudaEngine method), 23

get_binding_format_desc() (tensorrt.ICudaEngine method), 23

get_binding_index() (tensorrt.ICudaEngine method), 24

get_binding_name() (tensorrt.ICudaEngine method), 24

get_binding_shape() (tensorrt.ICudaEngine method), 24

get_binding_shape() (tensorrt.IExecutionContext method), 28

get_binding_vectorized_dim() (tensorrt.ICudaEngine method), 24

get_builder_plugin_registry() (in module tensorrt), 86

get_calibration_profile() (tensorrt.IBuilderConfig method), 16

get_device_type() (tensorrt.IBuilderConfig method),

16
 get_dynamic_range() (*tensorrt.Refitter method*), 32
 get_engine_information() (*tensorrt.EngineInspector method*), 38
 get_error() (*tensorrt.OnnxParser method*), 105
 get_error_code() (*tensorrt.IErrorRecorder method*), 34
 get_error_desc() (*tensorrt.IErrorRecorder method*), 35
 get_flag() (*tensorrt.IBuilderConfig method*), 16
 get_input() (*tensorrt.ILayer method*), 55
 get_input() (*tensorrt.INetworkDefinition method*), 50
 get_layer() (*tensorrt.INetworkDefinition method*), 50
 get_layer_information() (*tensorrt.EngineInspector method*), 38
 get_location() (*tensorrt.ICudaEngine method*), 24
 get_memory_pool_limit() (*tensorrt.IBuilderConfig method*), 17
 get_missing() (*tensorrt.Refitter method*), 32
 get_missing_weights() (*tensorrt.Refitter method*), 32
 get_output() (*tensorrt.ILayer method*), 55
 get_output() (*tensorrt.INetworkDefinition method*), 50
 get_output_type() (*tensorrt.ILayer method*), 55
 get_plugin_creator() (*tensorrt.IPluginRegistry method*), 85
 get_plugin_registry() (*in module tensorrt*), 85
 get_profile_shape() (*tensorrt.ICudaEngine method*), 25
 get_profile_shape_input() (*tensorrt.ICudaEngine method*), 25
 get_quantization_flag() (*tensorrt.IBuilderConfig method*), 17
 get_shape() (*tensorrt.IAlgorithmContext method*), 95
 get_shape() (*tensorrt.IExecutionContext method*), 29
 get_shape() (*tensorrt.IOptimizationProfile method*), 11
 get_shape_input() (*tensorrt.IOptimizationProfile method*), 11
 get_strides() (*tensorrt.IExecutionContext method*), 29
 get_tactic_sources() (*tensorrt.IBuilderConfig method*), 17
 get_tensors_with_dynamic_range() (*tensorrt.Refitter method*), 32
 get_timing_cache() (*tensorrt.IBuilderConfig method*), 17
 get_weights_for_gate() (*tensorrt.IRNNv2Layer method*), 67
 graph_inputs (*graphsurgeon.StaticGraph attribute*), 126
 graph_outputs (*graphsurgeon.StaticGraph attribute*), 126

H

has_overflowed() (*tensorrt.IErrorRecorder method*),

35

I

IActivationLayer (*class in tensorrt*), 59
 IAlgorithm (*class in tensorrt*), 96
 IAlgorithmContext (*class in tensorrt*), 95
 IAlgorithmIOInfo (*class in tensorrt*), 95
 IAlgorithmSelector (*class in tensorrt*), 96
 IAlgorithmVariant (*class in tensorrt*), 95
 IAssertionLayer (*class in tensorrt*), 82
 IBlobNameToTensor (*class in tensorrt*), 103
 IBuilderConfig (*class in tensorrt*), 15
 ICaffePluginFactoryV2 (*class in tensorrt*), 104
 IConcatenationLayer (*class in tensorrt*), 62
 IConditionLayer (*class in tensorrt*), 81
 IConstantLayer (*class in tensorrt*), 74
 IConvolutionLayer (*class in tensorrt*), 57
 ICudaEngine (*class in tensorrt*), 21
 IDEconvolutionLayer (*class in tensorrt*), 62
 IDEquantizeLayer (*class in tensorrt*), 80
 IEinsumLayer (*class in tensorrt*), 82
 IElementWiseLayer (*class in tensorrt*), 63
 IErrorRecorder (*class in tensorrt*), 34
 IExecutionContext (*class in tensorrt*), 26
 IFillLayer (*class in tensorrt*), 78
 IFullyConnectedLayer (*class in tensorrt*), 58
 IGatherLayer (*class in tensorrt*), 63
 IGPUAllocator (*class in tensorrt*), 36
 IHostMemory (*class in tensorrt*), 8
 IIdentityLayer (*class in tensorrt*), 74
 IIfConditional (*class in tensorrt*), 80
 IIfConditionalInputLayer (*class in tensorrt*), 81
 IIfConditionalOutputLayer (*class in tensorrt*), 81
 IInt8Calibrator (*class in tensorrt*), 87
 IInt8EntropyCalibrator (*class in tensorrt*), 90
 IInt8EntropyCalibrator2 (*class in tensorrt*), 92
 IInt8LegacyCalibrator (*class in tensorrt*), 89
 IInt8MinMaxCalibrator (*class in tensorrt*), 93
 IIteratorLayer (*class in tensorrt*), 77
 ILayer (*class in tensorrt*), 55
 ILogger (*class in tensorrt*), 9
 ILogger.Severity (*class in tensorrt*), 9
 ILoop (*class in tensorrt*), 76
 ILoopBoundaryLayer (*class in tensorrt*), 77
 ILoopOutputLayer (*class in tensorrt*), 78
 ILRNLayer (*class in tensorrt*), 60
 IMatrixMultiplyLayer (*class in tensorrt*), 73
 INetworkDefinition (*class in tensorrt*), 39
 init_libnvinfer_plugins() (*in module tensorrt*), 86
 insert() (*tensorrt.PluginFieldCollection method*), 84
 IOptimizationProfile (*class in tensorrt*), 11
 IPaddingLayer (*class in tensorrt*), 69
 IParametricReLULayer (*class in tensorrt*), 70
 IPluginCreator (*class in tensorrt*), 84

IPluginRegistry (class in *tensorrt*), 85
 IPluginV2Layer (class in *tensorrt*), 68
 IPoolingLayer (class in *tensorrt*), 59
 IProfiler (class in *tensorrt*), 10
 IQuantizeLayer (class in *tensorrt*), 79
 IRaggedSoftMaxLayer (class in *tensorrt*), 74
 IRecurrenceLayer (class in *tensorrt*), 77
 IReduceLayer (class in *tensorrt*), 69
 IResizeLayer (class in *tensorrt*), 74
 IRNNv2Layer (class in *tensorrt*), 66
 is_device_type_set() (*tensorrt.IBuilderConfig* method), 17
 is_execution_binding() (*tensorrt.ICudaEngine* method), 26
 is_network_supported() (*tensorrt.Builder* method), 21
 is_plugin_v2() (*tensorrt.ICaffePluginFactoryV2* method), 104
 is_shape_binding() (*tensorrt.ICudaEngine* method), 26
 IScaleLayer (class in *tensorrt*), 60
 IScatterLayer (class in *tensorrt*), 80
 ISelectLayer (class in *tensorrt*), 70
 IShapeLayer (class in *tensorrt*), 72
 IShuffleLayer (class in *tensorrt*), 70
 ISliceLayer (class in *tensorrt*), 71
 ISoftMaxLayer (class in *tensorrt*), 61
 ITensor (class in *tensorrt*), 52
 ITimingCache (class in *tensorrt*), 36
 ITopKLayer (class in *tensorrt*), 73
 ITripLimitLayer (class in *tensorrt*), 77
 IUnaryLayer (class in *tensorrt*), 69

L

LayerType (in module *tensorrt*), 54
 line() (*tensorrt.ParserError* method), 107
 log() (*tensorrt.ILogger* method), 9
 log() (*tensorrt.Logger* method), 10
 Logger (class in *tensorrt*), 10
 LoopOutput (in module *tensorrt*), 78

M

mark_output() (*tensorrt.INetworkDefinition* method), 50
 mark_output_for_shapes() (*tensorrt.INetworkDefinition* method), 50
 MatrixOperation (in module *tensorrt*), 73
 MAX_DIMS (*tensorrt.Dims* property), 7
 MemoryPoolType (in module *tensorrt*), 14

N

name (*tensorrt.ILogger.Severity* property), 9
 NetworkDefinitionCreationFlag (in module *tensorrt*), 19

node() (*tensorrt.ParserError* method), 107
 node_map (*graphsurgeon.StaticGraph* attribute), 126
 node_outputs (*graphsurgeon.StaticGraph* attribute), 126
 nptype() (in module *tensorrt*), 5
 num_errors() (*tensorrt.IErrorRecorder* method), 35
 numpy() (*tensorrt.Weights* method), 6

O

OnnxParser (class in *tensorrt*), 105
 output_type_is_set() (*tensorrt.ILayer* method), 55

P

PaddingMode (in module *tensorrt*), 56
 parse() (*tensorrt.CaffeParser* method), 103
 parse() (*tensorrt.OnnxParser* method), 105
 parse() (*tensorrt.UffParser* method), 99
 parse_binary_proto() (*tensorrt.CaffeParser* method), 104
 parse_buffer() (*tensorrt.CaffeParser* method), 104
 parse_buffer() (*tensorrt.UffParser* method), 99
 parse_from_file() (*tensorrt.OnnxParser* method), 105
 parse_with_weight_descriptors() (*tensorrt.OnnxParser* method), 106
 ParserError (class in *tensorrt*), 106
 Permutation (class in *tensorrt*), 70
 PluginField (class in *tensorrt*), 83
 PluginFieldCollection (class in *tensorrt*), 83
 PluginFieldType (in module *tensorrt*), 83
 PoolingType (in module *tensorrt*), 59
 pop() (*tensorrt.PluginFieldCollection* method), 84
 Profiler (class in *tensorrt*), 10
 ProfilingVerbosity (in module *tensorrt*), 13

Q

QuantizationFlag (in module *tensorrt*), 13

R

read() (*graphsurgeon.StaticGraph* method), 127
 read_calibration_cache() (*tensorrt.IInt8Calibrator* method), 88
 read_calibration_cache() (*tensorrt.IInt8EntropyCalibrator* method), 91
 read_calibration_cache() (*tensorrt.IInt8EntropyCalibrator2* method), 92
 read_calibration_cache() (*tensorrt.IInt8LegacyCalibrator* method), 89
 read_calibration_cache() (*tensorrt.IInt8MinMaxCalibrator* method), 94
 reallocate() (*tensorrt.IGpuAllocator* method), 37
 ReduceOperation (in module *tensorrt*), 69

refit_cuda_engine() (*tensorrt.Refitter method*), 32
 Refitter (*class in tensorrt*), 31
 register_creator() (*tensorrt.IPluginRegistry method*), 85
 register_input() (*tensorrt.UffParser method*), 100
 register_output() (*tensorrt.UffParser method*), 100
 remove() (*graphsurgeon.DynamicGraph method*), 129
 remove_tensor() (*tensorrt.INetworkDefinition method*), 50
 report_algorithms() (*tensorrt.IAlgorithmSelector method*), 96
 report_error() (*tensorrt.IErrorRecorder method*), 35
 report_layer_time() (*tensorrt.IProfiler method*), 10
 report_layer_time() (*tensorrt.Profiler method*), 10
 report_to_profiler() (*tensorrt.IExecutionContext method*), 29
 reset() (*tensorrt.Builder method*), 21
 reset() (*tensorrt.IBuilderConfig method*), 17
 reset() (*tensorrt.ITimingCache method*), 36
 reset_device_type() (*tensorrt.IBuilderConfig method*), 17
 reset_dynamic_range() (*tensorrt.ITensor method*), 53
 reset_output_type() (*tensorrt.ILayer method*), 55
 reset_precision() (*tensorrt.ILayer method*), 55
 ResizeMode (*in module tensorrt*), 74
 RNNDirection (*in module tensorrt*), 65
 RNNGateType (*in module tensorrt*), 66
 RNNInputMode (*in module tensorrt*), 65
 RNNOperation (*in module tensorrt*), 64
 Runtime (*class in tensorrt*), 30

S

ScaleMode (*in module tensorrt*), 60
 select_algorithms() (*tensorrt.IAlgorithmSelector method*), 97
 serialize() (*tensorrt.ICudaEngine method*), 26
 serialize() (*tensorrt.ITimingCache method*), 36
 set_bias_for_gate() (*tensorrt.IRNNv2Layer method*), 67
 set_binding_shape() (*tensorrt.IExecutionContext method*), 29
 set_calibration_profile() (*tensorrt.IBuilderConfig method*), 17
 set_condition() (*tensorrt.IfConditional method*), 81
 set_device_type() (*tensorrt.IBuilderConfig method*), 17
 set_dynamic_range() (*tensorrt.ITensor method*), 53
 set_dynamic_range() (*tensorrt.Refitter method*), 32
 set_flag() (*tensorrt.IBuilderConfig method*), 18
 set_input() (*tensorrt.IFillLayer method*), 78
 set_input() (*tensorrt.ILayer method*), 56
 set_input() (*tensorrt.ILoopOutputLayer method*), 78
 set_input() (*tensorrt.IRecurrenceLayer method*), 77
 set_input() (*tensorrt.IResizeLayer method*), 75

set_input() (*tensorrt.IShuffleLayer method*), 71
 set_input() (*tensorrt.ISliceLayer method*), 72
 set_memory_pool_limit() (*tensorrt.IBuilderConfig method*), 18
 set_named_weights() (*tensorrt.Refitter method*), 32
 set_optimization_profile_async() (*tensorrt.IExecutionContext method*), 30
 set_output_type() (*tensorrt.ILayer method*), 56
 set_quantization_flag() (*tensorrt.IBuilderConfig method*), 18
 set_shape() (*tensorrt.IOptimizationProfile method*), 11
 set_shape_input() (*tensorrt.IExecutionContext method*), 30
 set_shape_input() (*tensorrt.IOptimizationProfile method*), 12
 set_tactic_sources() (*tensorrt.IBuilderConfig method*), 18
 set_timing_cache() (*tensorrt.IBuilderConfig method*), 18
 set_weights() (*tensorrt.Refitter method*), 33
 set_weights_for_gate() (*tensorrt.IRNNv2Layer method*), 67
 set_weights_name() (*tensorrt.INetworkDefinition method*), 50
 shutdown_protobuf_library() (*in module tensorrt*), 104
 StaticGraph (*class in graphsurgeon*), 126
 supports_model() (*tensorrt.OnnxParser method*), 106
 supports_operator() (*tensorrt.OnnxParser method*), 106

T

TacticSource (*in module tensorrt*), 13
 TensorFormat (*in module tensorrt*), 51
 TensorLocation (*in module tensorrt*), 51
 TopKOperation (*in module tensorrt*), 73
 TripLimit (*in module tensorrt*), 77

U

UffInputOrder (*in module tensorrt*), 99
 UffParser (*class in tensorrt*), 99
 UnaryOperation (*in module tensorrt*), 68
 unmark_output() (*tensorrt.INetworkDefinition method*), 50
 unmark_output_for_shapes() (*tensorrt.INetworkDefinition method*), 51

V

volume() (*in module tensorrt*), 6

W

Weights (*class in tensorrt*), 6
 WeightsRole (*in module tensorrt*), 6

[write\(\)](#) (*graphsurgeon.StaticGraph* method), 127
[write_calibration_cache\(\)](#) (*ten-
sorr.IInt8Calibrator* method), 88
[write_calibration_cache\(\)](#) (*ten-
sorr.IInt8EntropyCalibrator* method), 91
[write_calibration_cache\(\)](#) (*ten-
sorr.IInt8EntropyCalibrator2* method),
93
[write_calibration_cache\(\)](#) (*ten-
sorr.IInt8LegacyCalibrator* method), 90
[write_calibration_cache\(\)](#) (*ten-
sorr.IInt8MinMaxCalibrator* method), 94
[write_tensorboard\(\)](#) (*graphsurgeon.StaticGraph*
method), 127

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

© 2017-2022 NVIDIA Corporation & affiliates. All rights reserved.

