

NVIDIA TensorRT 10.10.10 API Reference

for DRIVE OS 7.0.3

SWE-SWDOCTRT-006-APIS

TensorRT API Reference

10.10.10

May 2025

Table 1 Revision history

Date	Summary of change
2025-04-26	Initial draft
2025-04-28	Review
2025-05-12	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 **25**
7.1 File List 25

8 Namespace Documentation **27**
8.1 nvinfer1 Namespace Reference 27
8.1.1 Detailed Description 38
8.1.2 Typedef Documentation 38
8.1.2.1 AllocatorFlags 38
8.1.2.2 AsciiChar 38
8.1.2.3 BuilderFlags 38
8.1.2.4 char_t 39
8.1.2.5 Dims 39
8.1.2.6 IAlgorithmSelector 39
8.1.2.7 IDebugListener 39
8.1.2.8 IErrorRecorder 39
8.1.2.9 IGpuAllocator 39
8.1.2.10 IGpuAsyncAllocator 39
8.1.2.11 IInt8EntropyCalibrator 40
8.1.2.12 IInt8EntropyCalibrator2 40
8.1.2.13 IInt8LegacyCalibrator 40
8.1.2.14 IInt8MinMaxCalibrator 40
8.1.2.15 InterfaceKind 40
8.1.2.16 IOutputAllocator 40
8.1.2.17 IPluginCapability 40
8.1.2.18 IPluginCreator 41
8.1.2.19 IPluginCreatorInterface 41
8.1.2.20 IPluginCreatorV3One 41
8.1.2.21 IPluginResource 41

8.1.2.22 IPluginV3 . . . 41
8.1.2.23 IPluginV3OneBuild . . . 41
8.1.2.24 IPluginV3OneBuildV2 . . . 41
8.1.2.25 IPluginV3OneCore . . . 42
8.1.2.26 IPluginV3OneRuntime . . . 42
8.1.2.27 IProfiler . . . 42
8.1.2.28 IProgressMonitor . . . 42
8.1.2.29 IStreamReader . . . 42
8.1.2.30 IStreamReaderV2 . . . 42
8.1.2.31 NetworkDefinitionCreationFlags . . . 42
8.1.2.32 PluginFormat . . . 43
8.1.2.33 QuantizationFlags . . . 43
8.1.2.34 SerializationFlags . . . 43
8.1.2.35 TacticSources . . . 43
8.1.2.36 TempfileControlFlags . . . 44
8.1.2.37 TensorFormats . . . 44
8.1.3 Enumeration Type Documentation . . . 44
8.1.3.1 ActivationType . . . 44
8.1.3.2 AllocatorFlag . . . 45
8.1.3.3 APILanguage . . . 45
8.1.3.4 BoundingBoxFormat . . . 45
8.1.3.5 BuilderFlag . . . 46
8.1.3.6 CalibrationAlgoType . . . 49
8.1.3.7 CumulativeOperation . . . 49
8.1.3.8 DataType . . . 49
8.1.3.9 DeviceType . . . 50
8.1.3.10 DimensionOperation . . . 50
8.1.3.11 ElementWiseOperation . . . 51
8.1.3.12 EngineCapability . . . 52
8.1.3.13 ErrorCode . . . 52
8.1.3.14 ExecutionContextAllocationStrategy . . . 54
8.1.3.15 FillOperation . . . 54
8.1.3.16 GatherMode . . . 55
8.1.3.17 HardwareCompatibilityLevel . . . 55
8.1.3.18 InterpolationMode . . . 56
8.1.3.19 LayerInformationFormat . . . 56
8.1.3.20 LayerType . . . 57
8.1.3.21 LoopOutput . . . 58
8.1.3.22 MatrixOperation . . . 58
8.1.3.23 MemoryPoolType . . . 59
8.1.3.24 NetworkDefinitionCreationFlag . . . 60
8.1.3.25 OptProfileSelector . . . 60
8.1.3.26 PaddingMode . . . 61
8.1.3.27 PluginCapabilityType . . . 63
8.1.3.28 PluginCreatorVersion . . . 63
8.1.3.29 PluginFieldType . . . 64
8.1.3.30 PluginVersion . . . 64
8.1.3.31 PoolingType . . . 65
8.1.3.32 PreviewFeature . . . 65
8.1.3.33 ProfilingVerbosity . . . 65
8.1.3.34 QuantizationFlag . . . 66
8.1.3.35 ReduceOperation . . . 66
8.1.3.36 ResizeCoordinateTransformation . . . 67
8.1.3.37 ResizeRoundMode . . . 67

8.1.3.38 ResizeSelector . . . 68
8.1.3.39 RuntimePlatform . . . 68
8.1.3.40 SampleMode . . . 69
8.1.3.41 ScaleMode . . . 69
8.1.3.42 ScatterMode . . . 70
8.1.3.43 SeekPosition . . . 70
8.1.3.44 SerializationFlag . . . 70
8.1.3.45 TacticSource . . . 71
8.1.3.46 TempfileControlFlag . . . 72
8.1.3.47 TensorFormat . . . 72
8.1.3.48 TensorIOMode . . . 74
8.1.3.49 TensorLocation . . . 74
8.1.3.50 TensorRTPhase . . . 74
8.1.3.51 TilingOptimizationLevel . . . 75
8.1.3.52 TopKOperation . . . 75
8.1.3.53 TripLimit . . . 75
8.1.3.54 UnaryOperation . . . 76
8.1.3.55 WeightsRole . . . 77
8.1.4 Function Documentation . . . 77
8.1.4.1 EnumMax() . . . 77
8.1.4.2 EnumMax< BoundingBoxFormat >() . . . 77
8.1.4.3 EnumMax< BuilderFlag >() . . . 78
8.1.4.4 EnumMax< CalibrationAlgoType >() . . . 78
8.1.4.5 EnumMax< DeviceType >() . . . 78
8.1.4.6 EnumMax< DimensionOperation >() . . . 78
8.1.4.7 EnumMax< ExecutionContextAllocationStrategy >() . . . 79
8.1.4.8 EnumMax< FillOperation >() . . . 79
8.1.4.9 EnumMax< GatherMode >() . . . 79
8.1.4.10 EnumMax< LayerInformationFormat >() . . . 79
8.1.4.11 EnumMax< LayerType >() . . . 80
8.1.4.12 EnumMax< LoopOutput >() . . . 80
8.1.4.13 EnumMax< MatrixOperation >() . . . 80
8.1.4.14 EnumMax< MemoryPoolType >() . . . 80
8.1.4.15 EnumMax< NetworkDefinitionCreationFlag >() . . . 81
8.1.4.16 EnumMax< OptProfileSelector >() . . . 81
8.1.4.17 EnumMax< ProfilingVerbosity >() . . . 81
8.1.4.18 EnumMax< QuantizationFlag >() . . . 81
8.1.4.19 EnumMax< ReduceOperation >() . . . 82
8.1.4.20 EnumMax< SampleMode >() . . . 82
8.1.4.21 EnumMax< ScaleMode >() . . . 82
8.1.4.22 EnumMax< ScatterMode >() . . . 82
8.1.4.23 EnumMax< SerializationFlag >() . . . 83
8.1.4.24 EnumMax< TacticSource >() . . . 83
8.1.4.25 EnumMax< TempfileControlFlag >() . . . 83
8.1.4.26 EnumMax< TopKOperation >() . . . 83
8.1.4.27 EnumMax< TripLimit >() . . . 84
8.1.4.28 EnumMax< UnaryOperation >() . . . 84
8.1.4.29 EnumMax< WeightsRole >() . . . 84
8.1.4.30 getBuilderPluginRegistry() . . . 84
8.1.4.31 getBuilderSafePluginRegistry() . . . 85
8.2 nvinfer1::anonymous_namespace{NvInfer.h} Namespace Reference . . . 85
8.2.1 Function Documentation . . . 85
8.2.1.1 createInferBuilder() . . . 85
8.3 nvinfer1::anonymous_namespace{NvInferRuntime.h} Namespace Reference . . . 85

8.3.1 Function Documentation . . . 86
8.3.1.1 createInferRefitter() . . . 86
8.3.1.2 createInferRuntime() . . . 86
8.4 nvinfer1::impl Namespace Reference . . . 86
8.5 nvinfer1::plugin Namespace Reference . . . 87
8.5.1 Enumeration Type Documentation . . . 88
8.5.1.1 CodeTypeSSD . . . 88
8.6 nvinfer1::safe Namespace Reference . . . 88
8.7 nvinfer1::serialize Namespace Reference . . . 88
8.7.1 Function Documentation . . . 88
8.7.1.1 serializeNetwork() . . . 88
8.8 nvinfer1::v_1_0 Namespace Reference . . . 89
8.8.1 Detailed Description . . . 90
8.9 nvinfer1::v_2_0 Namespace Reference . . . 90
8.10 nvonnxparser Namespace Reference . . . 90
8.10.1 Detailed Description . . . 91
8.10.2 Typedef Documentation . . . 91
8.10.2.1 OnnxParserFlags . . . 91
8.10.3 Enumeration Type Documentation . . . 91
8.10.3.1 ErrorCode . . . 91
8.10.3.2 OnnxParserFlag . . . 92
8.10.4 Function Documentation . . . 92
8.10.4.1 createONNXConfig() . . . 92
8.10.4.2 EnumMax() . . . 93
8.10.4.3 EnumMax< ErrorCode >() . . . 93
8.10.4.4 EnumMax< OnnxParserFlag >() . . . 93
8.11 nvonnxparser::anonymous_namespace{NvOnnxParser.h} Namespace Reference . . . 93
8.11.1 Function Documentation . . . 93
8.11.1.1 createParser() . . . 93
8.11.1.2 createParserRefitter() . . . 94

9 Class Documentation **95**
9.1 nvinfer1::plugin::DetectionOutputParameters Struct Reference . . . 95
9.1.1 Detailed Description . . . 96
9.1.2 Member Data Documentation . . . 96
9.1.2.1 backgroundLabelId . . . 96
9.1.2.2 codeType . . . 96
9.1.2.3 confidenceThreshold . . . 96
9.1.2.4 confSigmoid . . . 96
9.1.2.5 inputOrder . . . 97
9.1.2.6 isBatchAgnostic . . . 97
9.1.2.7 isNormalized . . . 97
9.1.2.8 keepTopK . . . 97
9.1.2.9 nmsThreshold . . . 97
9.1.2.10 numClasses . . . 97
9.1.2.11 shareLocation . . . 98
9.1.2.12 topK . . . 98
9.1.2.13 varianceEncodedInTarget . . . 98
9.2 Dims Class Reference . . . 98
9.2.1 Detailed Description . . . 98
9.3 nvinfer1::Dims2 Class Reference . . . 99
9.3.1 Detailed Description . . . 99
9.3.2 Constructor & Destructor Documentation . . . 99
9.3.2.1 Dims2() [1/2] . . . 99

9.3.2.2 Dims2() [2/2] . . . 99
9.4 nvinfer1::Dims3 Class Reference . . . 100
9.4.1 Detailed Description . . . 100
9.4.2 Constructor & Destructor Documentation . . . 100
9.4.2.1 Dims3() [1/2] . . . 101
9.4.2.2 Dims3() [2/2] . . . 101
9.5 nvinfer1::Dims4 Class Reference . . . 101
9.5.1 Detailed Description . . . 102
9.5.2 Constructor & Destructor Documentation . . . 102
9.5.2.1 Dims4() [1/2] . . . 102
9.5.2.2 Dims4() [2/2] . . . 102
9.6 nvinfer1::Dims64 Class Reference . . . 103
9.6.1 Member Data Documentation . . . 103
9.6.1.1 d . . . 103
9.6.1.2 MAX_DIMS . . . 104
9.6.1.3 nbDims . . . 104
9.7 nvinfer1::DimsExprs Class Reference . . . 104
9.7.1 Detailed Description . . . 104
9.7.2 Member Data Documentation . . . 104
9.7.2.1 d . . . 105
9.7.2.2 nbDims . . . 105
9.8 nvinfer1::DimsHW Class Reference . . . 105
9.8.1 Detailed Description . . . 106
9.8.2 Constructor & Destructor Documentation . . . 106
9.8.2.1 DimsHW() [1/2] . . . 106
9.8.2.2 DimsHW() [2/2] . . . 106
9.8.3 Member Function Documentation . . . 106
9.8.3.1 h() [1/2] . . . 106
9.8.3.2 h() [2/2] . . . 107
9.8.3.3 w() [1/2] . . . 107
9.8.3.4 w() [2/2] . . . 107
9.9 nvinfer1::DynamicPluginTensorDesc Struct Reference . . . 107
9.9.1 Detailed Description . . . 108
9.9.2 Member Data Documentation . . . 108
9.9.2.1 desc . . . 108
9.9.2.2 max . . . 108
9.9.2.3 min . . . 108
9.9.2.4 opt . . . 109
9.10 nvinfer1::impl::EnumMaxImpl< T > Struct Template Reference . . . 109
9.10.1 Detailed Description . . . 109
9.11 nvinfer1::impl::EnumMaxImpl< ActivationType > Struct Reference . . . 109
9.11.1 Detailed Description . . . 109
9.11.2 Member Data Documentation . . . 110
9.11.2.1 kVALUE . . . 110
9.12 nvinfer1::impl::EnumMaxImpl< AllocatorFlag > Struct Reference . . . 110
9.12.1 Detailed Description . . . 110
9.12.2 Member Data Documentation . . . 110
9.12.2.1 kVALUE . . . 110
9.13 nvinfer1::impl::EnumMaxImpl< APILanguage > Struct Reference . . . 111
9.13.1 Detailed Description . . . 111
9.13.2 Member Data Documentation . . . 111
9.13.2.1 kVALUE . . . 111
9.14 nvinfer1::impl::EnumMaxImpl< CumulativeOperation > Struct Reference . . . 111
9.14.1 Detailed Description . . . 112

9.14.2 Member Data Documentation . . . 112
9.14.2.1 kVALUE . . . 112
9.15 nvinfer1::impl::EnumMaxImpl< DataType > Struct Reference . . . 112
9.15.1 Detailed Description . . . 112
9.15.2 Member Data Documentation . . . 113
9.15.2.1 kVALUE . . . 113
9.16 nvinfer1::impl::EnumMaxImpl< ElementWiseOperation > Struct Reference . . . 113
9.16.1 Detailed Description . . . 113
9.16.2 Member Data Documentation . . . 113
9.16.2.1 kVALUE . . . 113
9.17 nvinfer1::impl::EnumMaxImpl< EngineCapability > Struct Reference . . . 114
9.17.1 Detailed Description . . . 114
9.17.2 Member Data Documentation . . . 114
9.17.2.1 kVALUE . . . 114
9.18 nvinfer1::impl::EnumMaxImpl< ErrorCode > Struct Reference . . . 114
9.18.1 Detailed Description . . . 115
9.18.2 Member Data Documentation . . . 115
9.18.2.1 kVALUE . . . 115
9.19 nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel > Struct Reference . . . 115
9.19.1 Detailed Description . . . 115
9.19.2 Member Data Documentation . . . 115
9.19.2.1 kVALUE . . . 116
9.20 nvinfer1::impl::EnumMaxImpl< ILogger::Severity > Struct Reference . . . 116
9.20.1 Detailed Description . . . 116
9.20.2 Member Data Documentation . . . 116
9.20.2.1 kVALUE . . . 116
9.21 nvinfer1::impl::EnumMaxImpl< InterpolationMode > Struct Reference . . . 117
9.21.1 Detailed Description . . . 117
9.21.2 Member Data Documentation . . . 117
9.21.2.1 kVALUE . . . 117
9.22 nvinfer1::impl::EnumMaxImpl< PaddingMode > Struct Reference . . . 117
9.22.1 Detailed Description . . . 118
9.22.2 Member Data Documentation . . . 118
9.22.2.1 kVALUE . . . 118
9.23 nvinfer1::impl::EnumMaxImpl< PoolingType > Struct Reference . . . 118
9.23.1 Detailed Description . . . 118
9.23.2 Member Data Documentation . . . 118
9.23.2.1 kVALUE . . . 119
9.24 nvinfer1::impl::EnumMaxImpl< PreviewFeature > Struct Reference . . . 119
9.24.1 Detailed Description . . . 119
9.24.2 Member Data Documentation . . . 119
9.24.2.1 kVALUE . . . 119
9.25 nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation > Struct Reference . . . 119
9.25.1 Detailed Description . . . 120
9.25.2 Member Data Documentation . . . 120
9.25.2.1 kVALUE . . . 120
9.26 nvinfer1::impl::EnumMaxImpl< ResizeRoundMode > Struct Reference . . . 120
9.26.1 Detailed Description . . . 120
9.26.2 Member Data Documentation . . . 121
9.26.2.1 kVALUE . . . 121
9.27 nvinfer1::impl::EnumMaxImpl< ResizeSelector > Struct Reference . . . 121
9.27.1 Detailed Description . . . 121
9.27.2 Member Data Documentation . . . 121
9.27.2.1 kVALUE . . . 121

9.28 nvinfer1::impl::EnumMaxImpl< RuntimePlatform > Struct Reference . . . 122
9.28.1 Detailed Description . . . 122
9.28.2 Member Data Documentation . . . 122
9.28.2.1 kVALUE . . . 122
9.29 nvinfer1::impl::EnumMaxImpl< TensorFormat > Struct Reference . . . 122
9.29.1 Detailed Description . . . 123
9.29.2 Member Data Documentation . . . 123
9.29.2.1 kVALUE . . . 123
9.30 nvinfer1::impl::EnumMaxImpl< TensorIOMode > Struct Reference . . . 123
9.30.1 Detailed Description . . . 123
9.30.2 Member Data Documentation . . . 123
9.30.2.1 kVALUE . . . 124
9.31 nvinfer1::impl::EnumMaxImpl< TensorLocation > Struct Reference . . . 124
9.31.1 Detailed Description . . . 124
9.31.2 Member Data Documentation . . . 124
9.31.2.1 kVALUE . . . 124
9.32 nvinfer1::impl::EnumMaxImpl< TilingOptimizationLevel > Struct Reference . . . 124
9.32.1 Detailed Description . . . 125
9.32.2 Member Data Documentation . . . 125
9.32.2.1 kVALUE . . . 125
9.33 nvinfer1::plugin::GridAnchorParameters Struct Reference . . . 125
9.33.1 Detailed Description . . . 126
9.33.2 Member Data Documentation . . . 126
9.33.2.1 aspectRatios . . . 126
9.33.2.2 H . . . 126
9.33.2.3 maxSize . . . 126
9.33.2.4 minSize . . . 126
9.33.2.5 numAspectRatios . . . 127
9.33.2.6 variance . . . 127
9.33.2.7 W . . . 127
9.34 nvinfer1::IActivationLayer Class Reference . . . 127
9.34.1 Detailed Description . . . 128
9.34.2 Constructor & Destructor Documentation . . . 128
9.34.2.1 ~IActivationLayer() . . . 128
9.34.3 Member Function Documentation . . . 129
9.34.3.1 getActivationType() . . . 129
9.34.3.2 getAlpha() . . . 129
9.34.3.3 getBeta() . . . 129
9.34.3.4 setActivationType() . . . 130
9.34.3.5 setAlpha() . . . 130
9.34.3.6 setBeta() . . . 130
9.34.4 Member Data Documentation . . . 131
9.34.4.1 mImpl . . . 131
9.35 nvinfer1::IAlgorithm Class Reference . . . 131
9.35.1 Detailed Description . . . 132
9.35.2 Constructor & Destructor Documentation . . . 132
9.35.2.1 ~IAlgorithm() . . . 132
9.35.3 Member Function Documentation . . . 132
9.35.3.1 getAlgorithmIOInfoByIndex() . . . 132
9.35.3.2 getAlgorithmVariant() . . . 133
9.35.3.3 getTimingMSec() . . . 133
9.35.3.4 getWorkspaceSize() . . . 133
9.35.4 Member Data Documentation . . . 133
9.35.4.1 mImpl . . . 133

9.36 nvinfer1::IAlgorithmContext Class Reference . . . 134
9.36.1 Detailed Description . . . 134
9.36.2 Constructor & Destructor Documentation . . . 135
9.36.2.1 ~IAlgorithmContext() . . . 135
9.36.3 Member Function Documentation . . . 135
9.36.3.1 getDimensions() . . . 135
9.36.3.2 getName() . . . 135
9.36.3.3 getNbInputs() . . . 136
9.36.3.4 getNbOutputs() . . . 136
9.36.4 Member Data Documentation . . . 136
9.36.4.1 mImpl . . . 136
9.37 nvinfer1::IAlgorithmIOInfo Class Reference . . . 136
9.37.1 Detailed Description . . . 137
9.37.2 Constructor & Destructor Documentation . . . 137
9.37.2.1 ~IAlgorithmIOInfo() . . . 137
9.37.3 Member Function Documentation . . . 138
9.37.3.1 getComponentsPerElement() . . . 138
9.37.3.2 getDataType() . . . 138
9.37.3.3 getStrides() . . . 138
9.37.3.4 getVectorizedDim() . . . 138
9.37.4 Member Data Documentation . . . 139
9.37.4.1 mImpl . . . 139
9.38 IAlgorithmSelector Class Reference . . . 139
9.38.1 Detailed Description . . . 139
9.39 nvinfer1::v_1_0::IAlgorithmSelector Class Reference . . . 139
9.39.1 Constructor & Destructor Documentation . . . 140
9.39.1.1 ~IAlgorithmSelector() . . . 140
9.39.2 Member Function Documentation . . . 140
9.39.2.1 getInterfaceInfo() . . . 140
9.39.2.2 reportAlgorithms() . . . 140
9.39.2.3 selectAlgorithms() . . . 141
9.40 nvinfer1::IAlgorithmVariant Class Reference . . . 141
9.40.1 Detailed Description . . . 142
9.40.2 Constructor & Destructor Documentation . . . 143
9.40.2.1 ~IAlgorithmVariant() . . . 143
9.40.3 Member Function Documentation . . . 143
9.40.3.1 getImplementation() . . . 143
9.40.3.2 getTactic() . . . 143
9.40.4 Member Data Documentation . . . 143
9.40.4.1 mImpl . . . 143
9.41 nvinfer1::IAssertionLayer Class Reference . . . 144
9.41.1 Detailed Description . . . 144
9.41.2 Constructor & Destructor Documentation . . . 145
9.41.2.1 ~IAssertionLayer() . . . 145
9.41.3 Member Function Documentation . . . 145
9.41.3.1 getMessage() . . . 145
9.41.3.2 setMessage() . . . 145
9.41.4 Member Data Documentation . . . 145
9.41.4.1 mImpl . . . 146
9.42 nvinfer1::IBuilder Class Reference . . . 146
9.42.1 Detailed Description . . . 147
9.42.2 Constructor & Destructor Documentation . . . 147
9.42.2.1 ~IBuilder() . . . 147
9.42.3 Member Function Documentation . . . 148

9.42.3.1 buildEngineWithConfig() . . . 148
9.42.3.2 buildSerializedNetwork() . . . 148
9.42.3.3 createBuilderConfig() . . . 149
9.42.3.4 createNetworkV2() . . . 149
9.42.3.5 createOptimizationProfile() . . . 150
9.42.3.6 getErrorRecorder() . . . 150
9.42.3.7 getLogger() . . . 151
9.42.3.8 getMaxDLABatchSize() . . . 151
9.42.3.9 getMaxThreads() . . . 151
9.42.3.10 getNbDLACores() . . . 152
9.42.3.11 getPluginRegistry() . . . 152
9.42.3.12 isNetworkSupported() . . . 152
9.42.3.13 platformHasFastFp16() . . . 153
9.42.3.14 platformHasFastInt8() . . . 153
9.42.3.15 platformHasTf32() . . . 153
9.42.3.16 reset() . . . 153
9.42.3.17 setErrorRecorder() . . . 153
9.42.3.18 setGpuAllocator() . . . 154
9.42.3.19 setMaxThreads() . . . 154
9.42.4 Member Data Documentation . . . 155
9.42.4.1 mImpl . . . 155
9.43 nvinfer1::IBuilderConfig Class Reference . . . 155
9.43.1 Detailed Description . . . 158
9.43.2 Constructor & Destructor Documentation . . . 158
9.43.2.1 ~IBuilderConfig() . . . 159
9.43.3 Member Function Documentation . . . 159
9.43.3.1 addOptimizationProfile() . . . 159
9.43.3.2 canRunOnDLA() . . . 159
9.43.3.3 clearFlag() . . . 160
9.43.3.4 clearQuantizationFlag() . . . 160
9.43.3.5 createTimingCache() . . . 160
9.43.3.6 getAlgorithmSelector() . . . 161
9.43.3.7 getAvgTimingIterations() . . . 161
9.43.3.8 getBuilderOptimizationLevel() . . . 162
9.43.3.9 getCalibrationProfile() . . . 162
9.43.3.10 getDefaultDeviceType() . . . 162
9.43.3.11 getDeviceType() . . . 162
9.43.3.12 getDLACore() . . . 163
9.43.3.13 getEngineCapability() . . . 163
9.43.3.14 getFlag() . . . 163
9.43.3.15 getFlags() . . . 164
9.43.3.16 getHardwareCompatibilityLevel() . . . 164
9.43.3.17 getInt8Calibrator() . . . 164
9.43.3.18 getL2LimitForTiling() . . . 165
9.43.3.19 getMaxAuxStreams() . . . 165
9.43.3.20 getMaxNbTactics() . . . 165
9.43.3.21 getMemoryPoolLimit() . . . 165
9.43.3.22 getNbOptimizationProfiles() . . . 166
9.43.3.23 getNbPluginsToSerialize() . . . 166
9.43.3.24 getPluginToSerialize() . . . 166
9.43.3.25 getPreviewFeature() . . . 167
9.43.3.26 getProfileStream() . . . 167
9.43.3.27 getProfilingVerbosity() . . . 168
9.43.3.28 getProgressMonitor() . . . 168

9.43.3.29 getQuantizationFlag() . . . 168
9.43.3.30 getQuantizationFlags() . . . 169
9.43.3.31 getRuntimePlatform() . . . 169
9.43.3.32 getTacticSources() . . . 169
9.43.3.33 getTilingOptimizationLevel() . . . 170
9.43.3.34 getTimingCache() . . . 170
9.43.3.35 isDeviceTypeSet() . . . 170
9.43.3.36 reset() . . . 171
9.43.3.37 resetDeviceType() . . . 171
9.43.3.38 setAlgorithmSelector() . . . 171
9.43.3.39 setAvgTimingIterations() . . . 171
9.43.3.40 setBuilderOptimizationLevel() . . . 172
9.43.3.41 setCalibrationProfile() . . . 172
9.43.3.42 setDefaultDeviceType() . . . 173
9.43.3.43 setDeviceType() . . . 173
9.43.3.44 setDLACore() . . . 174
9.43.3.45 setEngineCapability() . . . 174
9.43.3.46 setFlag() . . . 175
9.43.3.47 setFlags() . . . 175
9.43.3.48 setHardwareCompatibilityLevel() . . . 175
9.43.3.49 setInt8Calibrator() . . . 176
9.43.3.50 setL2LimitForTiling() . . . 176
9.43.3.51 setMaxAuxStreams() . . . 177
9.43.3.52 setMaxNbTactics() . . . 177
9.43.3.53 setMemoryPoolLimit() . . . 178
9.43.3.54 setPluginsToSerialize() . . . 178
9.43.3.55 setPreviewFeature() . . . 179
9.43.3.56 setProfileStream() . . . 179
9.43.3.57 setProfilingVerbosity() . . . 179
9.43.3.58 setProgressMonitor() . . . 180
9.43.3.59 setQuantizationFlag() . . . 180
9.43.3.60 setQuantizationFlags() . . . 181
9.43.3.61 setRuntimePlatform() . . . 181
9.43.3.62 setTacticSources() . . . 182
9.43.3.63 setTilingOptimizationLevel() . . . 182
9.43.3.64 setTimingCache() . . . 183
9.43.4 Member Data Documentation . . . 183
9.43.4.1 mImpl . . . 183
9.44 nvinfer1::ICastLayer Class Reference . . . 184
9.44.1 Detailed Description . . . 184
9.44.2 Constructor & Destructor Documentation . . . 184
9.44.2.1 ~ICastLayer() . . . 185
9.44.3 Member Function Documentation . . . 185
9.44.3.1 getToType() . . . 185
9.44.3.2 setToType() . . . 185
9.44.4 Member Data Documentation . . . 185
9.44.4.1 mImpl . . . 185
9.45 nvinfer1::IConcatenationLayer Class Reference . . . 186
9.45.1 Detailed Description . . . 186
9.45.2 Constructor & Destructor Documentation . . . 186
9.45.2.1 ~IConcatenationLayer() . . . 187
9.45.3 Member Function Documentation . . . 187
9.45.3.1 getAxis() . . . 187
9.45.3.2 setAxis() . . . 187

9.45.4 Member Data Documentation . . . 187
9.45.4.1 mImpl . . . 188
9.46 nvinfer1::IConditionLayer Class Reference . . . 188
9.46.1 Detailed Description . . . 188
9.46.2 Constructor & Destructor Documentation . . . 188
9.46.2.1 ~IConditionLayer() . . . 189
9.46.3 Member Data Documentation . . . 189
9.46.3.1 mImpl . . . 189
9.47 nvinfer1::IConstantLayer Class Reference . . . 189
9.47.1 Detailed Description . . . 190
9.47.2 Constructor & Destructor Documentation . . . 190
9.47.2.1 ~IConstantLayer() . . . 190
9.47.3 Member Function Documentation . . . 190
9.47.3.1 getDimensions() . . . 190
9.47.3.2 getWeights() . . . 191
9.47.3.3 setDimensions() . . . 191
9.47.3.4 setWeights() . . . 191
9.47.4 Member Data Documentation . . . 191
9.47.4.1 mImpl . . . 192
9.48 nvinfer1::IConvolutionLayer Class Reference . . . 192
9.48.1 Detailed Description . . . 193
9.48.2 Constructor & Destructor Documentation . . . 194
9.48.2.1 ~IConvolutionLayer() . . . 194
9.48.3 Member Function Documentation . . . 194
9.48.3.1 getBiasWeights() . . . 194
9.48.3.2 getDilationNd() . . . 194
9.48.3.3 getKernelSizeNd() . . . 195
9.48.3.4 getKernelWeights() . . . 195
9.48.3.5 getNbGroups() . . . 195
9.48.3.6 getNbOutputMaps() . . . 195
9.48.3.7 getPaddingMode() . . . 196
9.48.3.8 getPaddingNd() . . . 196
9.48.3.9 getPostPadding() . . . 196
9.48.3.10 getPrePadding() . . . 196
9.48.3.11 getStrideNd() . . . 197
9.48.3.12 setBiasWeights() . . . 197
9.48.3.13 setDilationNd() . . . 197
9.48.3.14 setInput() . . . 197
9.48.3.15 setKernelSizeNd() . . . 198
9.48.3.16 setKernelWeights() . . . 198
9.48.3.17 setNbGroups() . . . 199
9.48.3.18 setNbOutputMaps() . . . 199
9.48.3.19 setPaddingMode() . . . 199
9.48.3.20 setPaddingNd() . . . 200
9.48.3.21 setPostPadding() . . . 200
9.48.3.22 setPrePadding() . . . 200
9.48.3.23 setStrideNd() . . . 201
9.48.4 Member Data Documentation . . . 201
9.48.4.1 mImpl . . . 201
9.49 nvinfer1::ICudaEngine Class Reference . . . 201
9.49.1 Detailed Description . . . 204
9.49.2 Constructor & Destructor Documentation . . . 204
9.49.2.1 ~ICudaEngine() . . . 204
9.49.3 Member Function Documentation . . . 205

9.49.3.1 createEngineInspector() . . . 205
9.49.3.2 createExecutionContext() . . . 205
9.49.3.3 createExecutionContextWithoutDeviceMemory() . . . 205
9.49.3.4 createSerializationConfig() . . . 206
9.49.3.5 getDeviceMemorySize() . . . 206
9.49.3.6 getDeviceMemorySizeForProfile() . . . 206
9.49.3.7 getDeviceMemorySizeForProfileV2() . . . 207
9.49.3.8 getDeviceMemorySizeV2() . . . 207
9.49.3.9 getEngineCapability() . . . 208
9.49.3.10 getErrorRecorder() . . . 208
9.49.3.11 getHardwareCompatibilityLevel() . . . 208
9.49.3.12 getIOTensorName() . . . 208
9.49.3.13 getMinimumWeightStreamingBudget() . . . 209
9.49.3.14 getName() . . . 209
9.49.3.15 getNbAuxStreams() . . . 210
9.49.3.16 getNbIOTensors() . . . 210
9.49.3.17 getNbLayers() . . . 210
9.49.3.18 getNbOptimizationProfiles() . . . 211
9.49.3.19 getProfileShape() . . . 211
9.49.3.20 getProfileTensorValues() . . . 211
9.49.3.21 getProfilingVerbosity() . . . 212
9.49.3.22 getStreamableWeightsSize() . . . 212
9.49.3.23 getTacticSources() . . . 213
9.49.3.24 getTensorBytesPerComponent() [1/2] . . . 213
9.49.3.25 getTensorBytesPerComponent() [2/2] . . . 213
9.49.3.26 getTensorComponentsPerElement() [1/2] . . . 214
9.49.3.27 getTensorComponentsPerElement() [2/2] . . . 215
9.49.3.28 getTensorDataType() . . . 215
9.49.3.29 getTensorFormat() [1/2] . . . 216
9.49.3.30 getTensorFormat() [2/2] . . . 216
9.49.3.31 getTensorFormatDesc() [1/2] . . . 216
9.49.3.32 getTensorFormatDesc() [2/2] . . . 217
9.49.3.33 getTensorIOMode() . . . 217
9.49.3.34 getTensorLocation() . . . 218
9.49.3.35 getTensorShape() . . . 218
9.49.3.36 getTensorVectorizedDim() [1/2] . . . 219
9.49.3.37 getTensorVectorizedDim() [2/2] . . . 219
9.49.3.38 getWeightStreamingAutomaticBudget() . . . 220
9.49.3.39 getWeightStreamingBudget() . . . 220
9.49.3.40 getWeightStreamingBudgetV2() . . . 221
9.49.3.41 getWeightStreamingScratchMemorySize() . . . 222
9.49.3.42 hasImplicitBatchDimension() . . . 222
9.49.3.43 isDebugTensor() . . . 223
9.49.3.44 isRefittable() . . . 223
9.49.3.45 isShapeInferenceIO() . . . 223
9.49.3.46 serialize() . . . 224
9.49.3.47 serializeWithConfig() . . . 224
9.49.3.48 setErrorRecorder() . . . 224
9.49.3.49 setWeightStreamingBudget() . . . 225
9.49.3.50 setWeightStreamingBudgetV2() . . . 226
9.49.4 Member Data Documentation . . . 227
9.49.4.1 mImpl . . . 227
9.50 nvinfer1::ICumulativeLayer Class Reference . . . 227
9.50.1 Detailed Description . . . 228

9.50.2 Constructor & Destructor Documentation . . . 229
9.50.2.1 ~ICumulativeLayer() . . . 229
9.50.3 Member Function Documentation . . . 229
9.50.3.1 getExclusive() . . . 229
9.50.3.2 getOperation() . . . 229
9.50.3.3 getReverse() . . . 230
9.50.3.4 setExclusive() . . . 230
9.50.3.5 setOperation() . . . 230
9.50.3.6 setReverse() . . . 231
9.50.4 Member Data Documentation . . . 231
9.50.4.1 mImpl . . . 231
9.51 IDebugListener Class Reference . . . 231
9.51.1 Detailed Description . . . 232
9.52 nvinfer1::v_1_0::IDebugListener Class Reference . . . 232
9.52.1 Constructor & Destructor Documentation . . . 232
9.52.1.1 ~IDebugListener() . . . 232
9.52.2 Member Function Documentation . . . 232
9.52.2.1 getInterfaceInfo() . . . 233
9.52.2.2 processDebugTensor() . . . 233
9.53 nvinfer1::IDeconvolutionLayer Class Reference . . . 233
9.53.1 Detailed Description . . . 235
9.53.2 Constructor & Destructor Documentation . . . 235
9.53.2.1 ~IDeconvolutionLayer() . . . 235
9.53.3 Member Function Documentation . . . 235
9.53.3.1 getBiasWeights() . . . 236
9.53.3.2 getDilationNd() . . . 236
9.53.3.3 getKernelSizeNd() . . . 236
9.53.3.4 getKernelWeights() . . . 236
9.53.3.5 getNbGroups() . . . 237
9.53.3.6 getNbOutputMaps() . . . 237
9.53.3.7 getPaddingMode() . . . 237
9.53.3.8 getPaddingNd() . . . 237
9.53.3.9 getPostPadding() . . . 238
9.53.3.10 getPrePadding() . . . 238
9.53.3.11 getStrideNd() . . . 238
9.53.3.12 setBiasWeights() . . . 238
9.53.3.13 setDilationNd() . . . 239
9.53.3.14 setInput() . . . 239
9.53.3.15 setKernelSizeNd() . . . 239
9.53.3.16 setKernelWeights() . . . 240
9.53.3.17 setNbGroups() . . . 240
9.53.3.18 setNbOutputMaps() . . . 240
9.53.3.19 setPaddingMode() . . . 241
9.53.3.20 setPaddingNd() . . . 241
9.53.3.21 setPostPadding() . . . 241
9.53.3.22 setPrePadding() . . . 242
9.53.3.23 setStrideNd() . . . 242
9.53.4 Member Data Documentation . . . 242
9.53.4.1 mImpl . . . 242
9.54 nvinfer1::IDequantizeLayer Class Reference . . . 243
9.54.1 Detailed Description . . . 244
9.54.2 Constructor & Destructor Documentation . . . 245
9.54.2.1 ~IDequantizeLayer() . . . 245
9.54.3 Member Function Documentation . . . 245

9.54.3.1 getAxis() . . . 245
9.54.3.2 getToType() . . . 245
9.54.3.3 setAxis() . . . 246
9.54.3.4 setToType() . . . 246
9.54.4 Member Data Documentation . . . 246
9.54.4.1 mImpl . . . 246
9.55 nvinfer1::IDimensionExpr Class Reference . . . 247
9.55.1 Detailed Description . . . 247
9.55.2 Constructor & Destructor Documentation . . . 248
9.55.2.1 ~IDimensionExpr() . . . 248
9.55.3 Member Function Documentation . . . 248
9.55.3.1 getConstantValue() . . . 248
9.55.3.2 isConstant() . . . 248
9.55.3.3 isSizeTensor() . . . 248
9.55.4 Member Data Documentation . . . 248
9.55.4.1 mImpl . . . 249
9.56 nvinfer1::IDynamicQuantizeLayer Class Reference . . . 249
9.56.1 Detailed Description . . . 250
9.56.2 Constructor & Destructor Documentation . . . 250
9.56.2.1 ~IDynamicQuantizeLayer() . . . 250
9.56.3 Member Function Documentation . . . 250
9.56.3.1 getAxis() . . . 251
9.56.3.2 getBlockSize() . . . 251
9.56.3.3 getScaleType() . . . 251
9.56.3.4 getToType() . . . 251
9.56.3.5 setAxis() . . . 252
9.56.3.6 setBlockSize() . . . 252
9.56.3.7 setInput() . . . 252
9.56.3.8 setScaleType() . . . 253
9.56.3.9 setToType() . . . 253
9.56.4 Member Data Documentation . . . 253
9.56.4.1 mImpl . . . 253
9.57 nvinfer1::IEinsumLayer Class Reference . . . 254
9.57.1 Detailed Description . . . 254
9.57.2 Constructor & Destructor Documentation . . . 255
9.57.2.1 ~IEinsumLayer() . . . 255
9.57.3 Member Function Documentation . . . 255
9.57.3.1 getEquation() . . . 255
9.57.3.2 setEquation() . . . 255
9.57.4 Member Data Documentation . . . 256
9.57.4.1 mImpl . . . 256
9.58 nvinfer1::IElementWiseLayer Class Reference . . . 256
9.58.1 Detailed Description . . . 257
9.58.2 Constructor & Destructor Documentation . . . 257
9.58.2.1 ~IElementWiseLayer() . . . 257
9.58.3 Member Function Documentation . . . 257
9.58.3.1 getOperation() . . . 257
9.58.3.2 setOperation() . . . 258
9.58.4 Member Data Documentation . . . 258
9.58.4.1 mImpl . . . 258
9.59 nvinfer1::IEngineInspector Class Reference . . . 258
9.59.1 Detailed Description . . . 259
9.59.2 Constructor & Destructor Documentation . . . 259
9.59.2.1 ~IEngineInspector() . . . 260

9.59.3 Member Function Documentation . . . 260
9.59.3.1 getEngineInformation() . . . 260
9.59.3.2 getErrorRecorder() . . . 260
9.59.3.3 getExecutionContext() . . . 261
9.59.3.4 getLayerInformation() . . . 261
9.59.3.5 setErrorRecorder() . . . 262
9.59.3.6 setExecutionContext() . . . 262
9.59.4 Member Data Documentation . . . 263
9.59.4.1 mImpl . . . 263
9.60 IErrorRecorder Class Reference . . . 263
9.60.1 Detailed Description . . . 263
9.61 nvinfer1::v_1_0::IErrorRecorder Class Reference . . . 264
9.61.1 Member Typedef Documentation . . . 265
9.61.1.1 ErrorDesc . . . 265
9.61.1.2 RefCount . . . 265
9.61.2 Constructor & Destructor Documentation . . . 265
9.61.2.1 IErrorRecorder() . . . 265
9.61.2.2 ~IErrorRecorder() . . . 265
9.61.3 Member Function Documentation . . . 265
9.61.3.1 clear() . . . 266
9.61.3.2 decRefCount() . . . 266
9.61.3.3 getErrorCode() . . . 266
9.61.3.4 getErrorDesc() . . . 267
9.61.3.5 getInterfaceInfo() . . . 268
9.61.3.6 getNbErrors() . . . 268
9.61.3.7 hasOverflowed() . . . 269
9.61.3.8 incRefCount() . . . 269
9.61.3.9 reportError() . . . 269
9.61.4 Member Data Documentation . . . 270
9.61.4.1 kMAX_DESC_LENGTH . . . 270
9.62 nvinfer1::IExecutionContext Class Reference . . . 271
9.62.1 Detailed Description . . . 273
9.62.2 Constructor & Destructor Documentation . . . 273
9.62.2.1 ~IExecutionContext() . . . 273
9.62.3 Member Function Documentation . . . 274
9.62.3.1 allInputDimensionsSpecified() . . . 274
9.62.3.2 allInputShapesSpecified() . . . 274
9.62.3.3 enqueueV3() . . . 274
9.62.3.4 executeV2() . . . 275
9.62.3.5 getDebugListener() . . . 276
9.62.3.6 getDebugState() . . . 276
9.62.3.7 getDebugSync() . . . 276
9.62.3.8 getEngine() . . . 276
9.62.3.9 getEnqueueEmitsProfile() . . . 277
9.62.3.10 getErrorRecorder() . . . 277
9.62.3.11 getInputConsumedEvent() . . . 277
9.62.3.12 getMaxOutputSize() . . . 277
9.62.3.13 getName() . . . 278
9.62.3.14 getNvtxVerbosity() . . . 278
9.62.3.15 getOptimizationProfile() . . . 279
9.62.3.16 getOutputAllocator() . . . 279
9.62.3.17 getOutputTensorAddress() . . . 279
9.62.3.18 getPersistentCacheLimit() . . . 280
9.62.3.19 getProfiler() . . . 280

9.62.3.20 getTemporaryStorageAllocator() . . . 280
9.62.3.21 getTensorAddress() . . . 280
9.62.3.22 getTensorShape() . . . 281
9.62.3.23 getTensorStrides() . . . 282
9.62.3.24 inferShapes() . . . 282
9.62.3.25 reportToProfiler() . . . 284
9.62.3.26 setAllTensorsDebugState() . . . 285
9.62.3.27 setAuxStreams() . . . 285
9.62.3.28 setDebugListener() . . . 286
9.62.3.29 setDebugSync() . . . 286
9.62.3.30 setDeviceMemory() . . . 287
9.62.3.31 setDeviceMemoryV2() . . . 287
9.62.3.32 setEnqueueEmitsProfile() . . . 288
9.62.3.33 setErrorRecorder() . . . 288
9.62.3.34 setInputConsumedEvent() . . . 288
9.62.3.35 setInputShape() . . . 289
9.62.3.36 setInputTensorAddress() . . . 289
9.62.3.37 setName() . . . 290
9.62.3.38 setNvtxVerbosity() . . . 290
9.62.3.39 setOptimizationProfileAsync() . . . 291
9.62.3.40 setOutputAllocator() . . . 292
9.62.3.41 setOutputTensorAddress() . . . 293
9.62.3.42 setPersistentCacheLimit() . . . 293
9.62.3.43 setProfiler() . . . 294
9.62.3.44 setTemporaryStorageAllocator() . . . 294
9.62.3.45 setTensorAddress() . . . 295
9.62.3.46 setTensorDebugState() . . . 296
9.62.3.47 updateDeviceMemorySizeForShapes() . . . 296
9.62.4 Member Data Documentation . . . 297
9.62.4.1 mImpl . . . 297
9.63 nvinfer1::IExprBuilder Class Reference . . . 297
9.63.1 Detailed Description . . . 298
9.63.2 Constructor & Destructor Documentation . . . 298
9.63.2.1 ~IExprBuilder() . . . 298
9.63.3 Member Function Documentation . . . 298
9.63.3.1 constant() . . . 298
9.63.3.2 declareSizeTensor() . . . 299
9.63.3.3 operation() . . . 299
9.63.4 Member Data Documentation . . . 300
9.63.4.1 mImpl . . . 300
9.64 nvinfer1::IFillLayer Class Reference . . . 300
9.64.1 Detailed Description . . . 301
9.64.2 Constructor & Destructor Documentation . . . 302
9.64.2.1 ~IFillLayer() . . . 302
9.64.3 Member Function Documentation . . . 302
9.64.3.1 getAlpha() . . . 302
9.64.3.2 getAlphaInt64() . . . 303
9.64.3.3 getBeta() . . . 303
9.64.3.4 getBetaInt64() . . . 303
9.64.3.5 getDimensions() . . . 304
9.64.3.6 getOperation() . . . 304
9.64.3.7 getToType() . . . 304
9.64.3.8 isAlphaBetaInt64() . . . 304
9.64.3.9 setAlpha() . . . 304

9.64.3.10 setAlphaInt64() . . . 305
9.64.3.11 setBeta() . . . 305
9.64.3.12 setBetaInt64() . . . 306
9.64.3.13 setDimensions() . . . 306
9.64.3.14 setInput() . . . 307
9.64.3.15 setOperation() . . . 308
9.64.3.16 setToType() . . . 308
9.64.4 Member Data Documentation . . . 309
9.64.4.1 mImpl . . . 309
9.65 nvinfer1::IGatherLayer Class Reference . . . 309
9.65.1 Detailed Description . . . 310
9.65.2 Constructor & Destructor Documentation . . . 311
9.65.2.1 ~IGatherLayer() . . . 311
9.65.3 Member Function Documentation . . . 311
9.65.3.1 getGatherAxis() . . . 312
9.65.3.2 getMode() . . . 312
9.65.3.3 getNbElementWiseDims() . . . 312
9.65.3.4 setGatherAxis() . . . 312
9.65.3.5 setMode() . . . 313
9.65.3.6 setNbElementWiseDims() . . . 313
9.65.4 Member Data Documentation . . . 314
9.65.4.1 mImpl . . . 314
9.66 IGpuAllocator Class Reference . . . 314
9.66.1 Detailed Description . . . 314
9.67 nvinfer1::v_1_0::IGpuAllocator Class Reference . . . 315
9.67.1 Constructor & Destructor Documentation . . . 315
9.67.1.1 ~IGpuAllocator() . . . 315
9.67.1.2 IGpuAllocator() . . . 316
9.67.2 Member Function Documentation . . . 316
9.67.2.1 allocate() . . . 316
9.67.2.2 allocateAsync() . . . 317
9.67.2.3 deallocate() . . . 317
9.67.2.4 deallocateAsync() . . . 318
9.67.2.5 getInterfaceInfo() . . . 319
9.67.2.6 reallocate() . . . 319
9.68 IGpuAsyncAllocator Class Reference . . . 320
9.68.1 Detailed Description . . . 320
9.69 nvinfer1::v_1_0::IGpuAsyncAllocator Class Reference . . . 321
9.69.1 Constructor & Destructor Documentation . . . 321
9.69.1.1 IGpuAsyncAllocator() . . . 321
9.69.1.2 ~IGpuAsyncAllocator() . . . 322
9.69.2 Member Function Documentation . . . 322
9.69.2.1 allocate() . . . 322
9.69.2.2 allocateAsync() . . . 323
9.69.2.3 deallocate() . . . 323
9.69.2.4 deallocateAsync() . . . 324
9.69.2.5 getInterfaceInfo() . . . 325
9.70 nvinfer1::IGridSampleLayer Class Reference . . . 325
9.70.1 Detailed Description . . . 326
9.70.2 Constructor & Destructor Documentation . . . 326
9.70.2.1 ~IGridSampleLayer() . . . 326
9.70.3 Member Function Documentation . . . 327
9.70.3.1 getAlignCorners() . . . 327
9.70.3.2 getInterpolationMode() . . . 327

9.70.3.3 getSampleMode() . . . 327
9.70.3.4 setAlignCorners() . . . 328
9.70.3.5 setInterpolationMode() . . . 328
9.70.3.6 setSampleMode() . . . 328
9.70.4 Member Data Documentation . . . 328
9.70.4.1 mImpl . . . 329
9.71 nvinfer1::IHostMemory Class Reference . . . 329
9.71.1 Detailed Description . . . 329
9.71.2 Constructor & Destructor Documentation . . . 330
9.71.2.1 ~IHostMemory() . . . 330
9.71.3 Member Function Documentation . . . 330
9.71.3.1 data() . . . 330
9.71.3.2 size() . . . 330
9.71.3.3 type() . . . 330
9.71.4 Member Data Documentation . . . 330
9.71.4.1 mImpl . . . 331
9.72 nvinfer1::IIdentityLayer Class Reference . . . 331
9.72.1 Detailed Description . . . 331
9.72.2 Constructor & Destructor Documentation . . . 332
9.72.2.1 ~IIdentityLayer() . . . 332
9.72.3 Member Data Documentation . . . 332
9.72.3.1 mImpl . . . 332
9.73 nvinfer1::IIfConditional Class Reference . . . 332
9.73.1 Detailed Description . . . 333
9.73.2 Constructor & Destructor Documentation . . . 333
9.73.2.1 ~IIfConditional() . . . 334
9.73.3 Member Function Documentation . . . 334
9.73.3.1 addInput() . . . 334
9.73.3.2 addOutput() . . . 334
9.73.3.3 getName() . . . 335
9.73.3.4 setCondition() . . . 335
9.73.3.5 setName() . . . 335
9.73.4 Member Data Documentation . . . 336
9.73.4.1 mImpl . . . 336
9.74 nvinfer1::IIfConditionalBoundaryLayer Class Reference . . . 336
9.74.1 Detailed Description . . . 337
9.74.2 Constructor & Destructor Documentation . . . 337
9.74.2.1 ~IIfConditionalBoundaryLayer() . . . 337
9.74.3 Member Function Documentation . . . 337
9.74.3.1 getConditional() . . . 337
9.74.4 Member Data Documentation . . . 337
9.74.4.1 mBoundary . . . 338
9.75 nvinfer1::IIfConditionalInputLayer Class Reference . . . 338
9.75.1 Detailed Description . . . 338
9.75.2 Constructor & Destructor Documentation . . . 338
9.75.2.1 ~IIfConditionalInputLayer() . . . 339
9.75.3 Member Data Documentation . . . 339
9.75.3.1 mImpl . . . 339
9.76 nvinfer1::IIfConditionalOutputLayer Class Reference . . . 339
9.76.1 Detailed Description . . . 340
9.76.2 Constructor & Destructor Documentation . . . 340
9.76.2.1 ~IIfConditionalOutputLayer() . . . 340
9.76.3 Member Data Documentation . . . 340
9.76.3.1 mImpl . . . 340

9.77 nvinfer1::IInt8Calibrator Class Reference . . . 340
9.77.1 Detailed Description . . . 341
9.77.2 Constructor & Destructor Documentation . . . 341
9.77.2.1 ~IInt8Calibrator() . . . 341
9.77.3 Member Function Documentation . . . 341
9.77.3.1 getAlgorithm() . . . 342
9.77.3.2 getBatch() . . . 342
9.77.3.3 getBatchSize() . . . 342
9.77.3.4 readCalibrationCache() . . . 343
9.77.3.5 writeCalibrationCache() . . . 343
9.78 IInt8EntropyCalibrator Class Reference . . . 344
9.78.1 Detailed Description . . . 344
9.79 nvinfer1::v_1_0::IInt8EntropyCalibrator Class Reference . . . 344
9.79.1 Constructor & Destructor Documentation . . . 345
9.79.1.1 ~IInt8EntropyCalibrator() . . . 345
9.79.2 Member Function Documentation . . . 345
9.79.2.1 getAlgorithm() . . . 345
9.79.2.2 getInterfaceInfo() . . . 345
9.80 IInt8EntropyCalibrator2 Class Reference . . . 346
9.80.1 Detailed Description . . . 346
9.81 nvinfer1::v_1_0::IInt8EntropyCalibrator2 Class Reference . . . 346
9.81.1 Constructor & Destructor Documentation . . . 347
9.81.1.1 ~IInt8EntropyCalibrator2() . . . 347
9.81.2 Member Function Documentation . . . 347
9.81.2.1 getAlgorithm() . . . 347
9.81.2.2 getInterfaceInfo() . . . 347
9.82 IInt8LegacyCalibrator Class Reference . . . 347
9.82.1 Detailed Description . . . 348
9.83 nvinfer1::v_1_0::IInt8LegacyCalibrator Class Reference . . . 348
9.83.1 Constructor & Destructor Documentation . . . 349
9.83.1.1 ~IInt8LegacyCalibrator() . . . 349
9.83.2 Member Function Documentation . . . 349
9.83.2.1 getAlgorithm() . . . 349
9.83.2.2 getInterfaceInfo() . . . 349
9.83.2.3 getQuantile() . . . 349
9.83.2.4 getRegressionCutoff() . . . 350
9.83.2.5 readHistogramCache() . . . 350
9.83.2.6 writeHistogramCache() . . . 350
9.84 IInt8MinMaxCalibrator Class Reference . . . 351
9.84.1 Detailed Description . . . 351
9.85 nvinfer1::v_1_0::IInt8MinMaxCalibrator Class Reference . . . 351
9.85.1 Constructor & Destructor Documentation . . . 352
9.85.1.1 ~IInt8MinMaxCalibrator() . . . 352
9.85.2 Member Function Documentation . . . 352
9.85.2.1 getAlgorithm() . . . 352
9.85.2.2 getInterfaceInfo() . . . 352
9.86 nvinfer1::IIteratorLayer Class Reference . . . 353
9.86.1 Detailed Description . . . 353
9.86.2 Constructor & Destructor Documentation . . . 354
9.86.2.1 ~IIteratorLayer() . . . 354
9.86.3 Member Function Documentation . . . 354
9.86.3.1 getAxis() . . . 354
9.86.3.2 getReverse() . . . 354
9.86.3.3 setAxis() . . . 354

9.86.3.4 setReverse() . . . 355
9.86.4 Member Data Documentation . . . 355
9.86.4.1 mImpl . . . 355
9.87 nvinfer1::ILayer Class Reference . . . 355
9.87.1 Detailed Description . . . 357
9.87.2 Constructor & Destructor Documentation . . . 357
9.87.2.1 ~ILayer() . . . 357
9.87.3 Member Function Documentation . . . 358
9.87.3.1 getInput() . . . 358
9.87.3.2 getMetadata() . . . 358
9.87.3.3 getName() . . . 358
9.87.3.4 getNbInputs() . . . 359
9.87.3.5 getNbOutputs() . . . 359
9.87.3.6 getOutput() . . . 359
9.87.3.7 getOutputType() . . . 359
9.87.3.8 getPrecision() . . . 360
9.87.3.9 getType() . . . 360
9.87.3.10 outputTypeIsSet() . . . 360
9.87.3.11 precisionIsSet() . . . 361
9.87.3.12 resetOutputType() . . . 361
9.87.3.13 resetPrecision() . . . 362
9.87.3.14 setInput() . . . 362
9.87.3.15 setMetadata() . . . 362
9.87.3.16 setName() . . . 363
9.87.3.17 setOutputType() . . . 363
9.87.3.18 setPrecision() . . . 364
9.87.4 Member Data Documentation . . . 365
9.87.4.1 mLayer . . . 365
9.88 nvinfer1::ILogger Class Reference . . . 365
9.88.1 Detailed Description . . . 366
9.88.2 Member Enumeration Documentation . . . 366
9.88.2.1 Severity . . . 366
9.88.3 Constructor & Destructor Documentation . . . 366
9.88.3.1 ILogger() . . . 367
9.88.3.2 ~ILogger() . . . 367
9.88.4 Member Function Documentation . . . 367
9.88.4.1 log() . . . 367
9.89 nvinfer1::ILoggerFinder Class Reference . . . 368
9.89.1 Detailed Description . . . 368
9.89.2 Constructor & Destructor Documentation . . . 368
9.89.2.1 ~ILoggerFinder() . . . 368
9.89.3 Member Function Documentation . . . 368
9.89.3.1 findLogger() . . . 368
9.90 nvinfer1::ILoop Class Reference . . . 369
9.90.1 Detailed Description . . . 370
9.90.2 Constructor & Destructor Documentation . . . 370
9.90.2.1 ~ILoop() . . . 370
9.90.3 Member Function Documentation . . . 370
9.90.3.1 addIterator() . . . 370
9.90.3.2 addLoopOutput() . . . 370
9.90.3.3 addRecurrence() . . . 371
9.90.3.4 addTripLimit() . . . 371
9.90.3.5 getName() . . . 371
9.90.3.6 setName() . . . 371

9.90.4 Member Data Documentation . . . 372
9.90.4.1 mImpl . . . 372
9.91 nvinfer1::ILoopBoundaryLayer Class Reference . . . 372
9.91.1 Detailed Description . . . 373
9.91.2 Constructor & Destructor Documentation . . . 373
9.91.2.1 ~ILoopBoundaryLayer() . . . 373
9.91.3 Member Function Documentation . . . 373
9.91.3.1 getLoop() . . . 373
9.91.4 Member Data Documentation . . . 373
9.91.4.1 mBoundary . . . 374
9.92 nvinfer1::ILoopOutputLayer Class Reference . . . 374
9.92.1 Detailed Description . . . 375
9.92.2 Constructor & Destructor Documentation . . . 375
9.92.2.1 ~ILoopOutputLayer() . . . 375
9.92.3 Member Function Documentation . . . 375
9.92.3.1 getAxis() . . . 375
9.92.3.2 getLoopOutput() . . . 375
9.92.3.3 setAxis() . . . 376
9.92.3.4 setInput() . . . 376
9.92.4 Member Data Documentation . . . 376
9.92.4.1 mImpl . . . 376
9.93 nvinfer1::ILRNLayer Class Reference . . . 377
9.93.1 Detailed Description . . . 377
9.93.2 Constructor & Destructor Documentation . . . 378
9.93.2.1 ~ILRNLayer() . . . 378
9.93.3 Member Function Documentation . . . 378
9.93.3.1 getAlpha() . . . 378
9.93.3.2 getBeta() . . . 378
9.93.3.3 getK() . . . 379
9.93.3.4 getWindowSize() . . . 379
9.93.3.5 setAlpha() . . . 379
9.93.3.6 setBeta() . . . 379
9.93.3.7 setK() . . . 380
9.93.3.8 setWindowSize() . . . 380
9.93.4 Member Data Documentation . . . 380
9.93.4.1 mImpl . . . 380
9.94 nvinfer1::IMatrixMultiplyLayer Class Reference . . . 381
9.94.1 Detailed Description . . . 381
9.94.2 Constructor & Destructor Documentation . . . 382
9.94.2.1 ~IMatrixMultiplyLayer() . . . 382
9.94.3 Member Function Documentation . . . 382
9.94.3.1 getOperation() . . . 382
9.94.3.2 setOperation() . . . 382
9.94.4 Member Data Documentation . . . 383
9.94.4.1 mImpl . . . 383
9.95 nvinfer1::INetworkDefinition Class Reference . . . 383
9.95.1 Detailed Description . . . 387
9.95.2 Constructor & Destructor Documentation . . . 388
9.95.2.1 ~INetworkDefinition() . . . 388
9.95.3 Member Function Documentation . . . 388
9.95.3.1 addActivation() . . . 388
9.95.3.2 addAssertion() . . . 389
9.95.3.3 addCast() . . . 389
9.95.3.4 addConcatenation() . . . 390

9.95.3.5 addConstant() . . . 390
9.95.3.6 addConvolutionNd() . . . 391
9.95.3.7 addCumulative() . . . 392
9.95.3.8 addDeconvolutionNd() . . . 392
9.95.3.9 addDequantize() [1/2] . . . 393
9.95.3.10 addDequantize() [2/2] . . . 394
9.95.3.11 addDynamicQuantize() . . . 394
9.95.3.12 addEinsum() . . . 395
9.95.3.13 addElementWise() . . . 396
9.95.3.14 addFill() [1/2] . . . 396
9.95.3.15 addFill() [2/2] . . . 397
9.95.3.16 addGather() . . . 398
9.95.3.17 addGatherV2() . . . 398
9.95.3.18 addGridSample() . . . 399
9.95.3.19 addIdentity() . . . 399
9.95.3.20 addIfConditional() . . . 400
9.95.3.21 addInput() . . . 400
9.95.3.22 addLoop() . . . 401
9.95.3.23 addLRN() . . . 401
9.95.3.24 addMatrixMultiply() . . . 402
9.95.3.25 addNMS() . . . 403
9.95.3.26 addNonZero() . . . 403
9.95.3.27 addNormalization() . . . 404
9.95.3.28 addOneHot() . . . 405
9.95.3.29 addPaddingNd() . . . 405
9.95.3.30 addParametricReLU() . . . 406
9.95.3.31 addPluginV2() . . . 406
9.95.3.32 addPluginV3() . . . 407
9.95.3.33 addPoolingNd() . . . 408
9.95.3.34 addQuantize() [1/2] . . . 408
9.95.3.35 addQuantize() [2/2] . . . 409
9.95.3.36 addRaggedSoftMax() . . . 410
9.95.3.37 addReduce() . . . 410
9.95.3.38 addResize() . . . 411
9.95.3.39 addReverseSequence() . . . 411
9.95.3.40 addScale() . . . 412
9.95.3.41 addScaleNd() . . . 413
9.95.3.42 addScatter() . . . 414
9.95.3.43 addSelect() . . . 414
9.95.3.44 addShape() . . . 415
9.95.3.45 addShuffle() . . . 416
9.95.3.46 addSlice() . . . 416
9.95.3.47 addSoftMax() . . . 417
9.95.3.48 addSqueeze() . . . 417
9.95.3.49 addTopK() . . . 418
9.95.3.50 addUnary() . . . 419
9.95.3.51 addUnsqueeze() . . . 419
9.95.3.52 areWeightsMarkedRefittable() . . . 420
9.95.3.53 getBuilder() . . . 420
9.95.3.54 getErrorRecorder() . . . 421
9.95.3.55 getFlag() . . . 421
9.95.3.56 getFlags() . . . 422
9.95.3.57 getInput() . . . 422
9.95.3.58 getLayer() . . . 422

9.95.3.59 getName() . . . 423
9.95.3.60 getNbInputs() . . . 423
9.95.3.61 getNbLayers() . . . 424
9.95.3.62 getNbOutputs() . . . 424
9.95.3.63 getOutput() . . . 424
9.95.3.64 hasImplicitBatchDimension() . . . 425
9.95.3.65 isDebugTensor() . . . 425
9.95.3.66 markDebug() . . . 425
9.95.3.67 markOutput() . . . 426
9.95.3.68 markOutputForShapes() . . . 426
9.95.3.69 markWeightsRefittable() . . . 427
9.95.3.70 removeTensor() . . . 427
9.95.3.71 setErrorRecorder() . . . 427
9.95.3.72 setName() . . . 428
9.95.3.73 setWeightsName() . . . 429
9.95.3.74 unmarkDebug() . . . 429
9.95.3.75 unmarkOutput() . . . 430
9.95.3.76 unmarkOutputForShapes() . . . 430
9.95.3.77 unmarkWeightsRefittable() . . . 430
9.95.4 Member Data Documentation . . . 431
9.95.4.1 mImpl . . . 431
9.96 nvinfer1::INMSLayer Class Reference . . . 431
9.96.1 Detailed Description . . . 432
9.96.2 Constructor & Destructor Documentation . . . 433
9.96.2.1 ~INMSLayer() . . . 433
9.96.3 Member Function Documentation . . . 433
9.96.3.1 getBoundingBoxFormat() . . . 433
9.96.3.2 getTopKBoxLimit() . . . 433
9.96.3.3 setBoundingBoxFormat() . . . 434
9.96.3.4 setInput() . . . 434
9.96.3.5 setTopKBoxLimit() . . . 435
9.96.4 Member Data Documentation . . . 435
9.96.4.1 mImpl . . . 435
9.97 nvinfer1::INoCopy Class Reference . . . 435
9.97.1 Detailed Description . . . 436
9.97.2 Constructor & Destructor Documentation . . . 436
9.97.2.1 INoCopy() [1/3] . . . 437
9.97.2.2 ~INoCopy() . . . 437
9.97.2.3 INoCopy() [2/3] . . . 437
9.97.2.4 INoCopy() [3/3] . . . 437
9.97.3 Member Function Documentation . . . 437
9.97.3.1 operator=() [1/2] . . . 437
9.97.3.2 operator=() [2/2] . . . 437
9.98 INonZero Class Reference . . . 438
9.98.1 Detailed Description . . . 438
9.99 nvinfer1::INonZeroLayer Class Reference . . . 438
9.99.1 Constructor & Destructor Documentation . . . 439
9.99.1.1 ~INonZeroLayer() . . . 439
9.99.2 Member Data Documentation . . . 439
9.99.2.1 mImpl . . . 439
9.100 nvinfer1::INormalizationLayer Class Reference . . . 439
9.100.1 Detailed Description . . . 440
9.100.2 Constructor & Destructor Documentation . . . 440
9.100.2.1 ~INormalizationLayer() . . . 441

9.100.3 Member Function Documentation . . . 441
9.100.3.1 getAxes() . . . 441
9.100.3.2 getComputePrecision() . . . 441
9.100.3.3 getEpsilon() . . . 441
9.100.3.4 getNbGroups() . . . 442
9.100.3.5 setAxes() . . . 442
9.100.3.6 setComputePrecision() . . . 442
9.100.3.7 setEpsilon() . . . 443
9.100.3.8 setNbGroups() . . . 443
9.100.4 Member Data Documentation . . . 443
9.100.4.1 mImpl . . . 444
9.101 nvinfer1::InterfaceInfo Class Reference . . . 444
9.101.1 Detailed Description . . . 444
9.101.2 Member Data Documentation . . . 444
9.101.2.1 kind . . . 444
9.101.2.2 major . . . 444
9.101.2.3 minor . . . 445
9.102 nvinfer1::IOneHotLayer Class Reference . . . 445
9.102.1 Detailed Description . . . 446
9.102.2 Constructor & Destructor Documentation . . . 446
9.102.2.1 ~IOneHotLayer() . . . 446
9.102.3 Member Function Documentation . . . 446
9.102.3.1 getAxis() . . . 446
9.102.3.2 setAxis() . . . 447
9.102.4 Member Data Documentation . . . 447
9.102.4.1 mImpl . . . 447
9.103 nvonnxparser::IOnnxConfig Class Reference . . . 447
9.103.1 Detailed Description . . . 448
9.103.2 Member Typedef Documentation . . . 448
9.103.2.1 Verbosity . . . 448
9.103.3 Constructor & Destructor Documentation . . . 449
9.103.3.1 ~IOnnxConfig() . . . 449
9.103.4 Member Function Documentation . . . 449
9.103.4.1 addVerbosity() . . . 449
9.103.4.2 getFullTextFileName() . . . 449
9.103.4.3 getModelDtype() . . . 450
9.103.4.4 getModelFileName() . . . 450
9.103.4.5 getPrintLayerInfo() . . . 450
9.103.4.6 getTextFileName() . . . 451
9.103.4.7 getVerbosityLevel() . . . 451
9.103.4.8 reduceVerbosity() . . . 451
9.103.4.9 setFullTextFileName() . . . 451
9.103.4.10 setModelDtype() . . . 452
9.103.4.11 setModelFileName() . . . 452
9.103.4.12 setPrintLayerInfo() . . . 453
9.103.4.13 setTextFileName() . . . 453
9.103.4.14 setVerbosityLevel() . . . 453
9.104 nvinfer1::IOptimizationProfile Class Reference . . . 454
9.104.1 Detailed Description . . . 455
9.104.2 Constructor & Destructor Documentation . . . 455
9.104.2.1 ~IOptimizationProfile() . . . 455
9.104.3 Member Function Documentation . . . 455
9.104.3.1 getDimensions() . . . 455
9.104.3.2 getExtraMemoryTarget() . . . 456

9.104.3.3 getNbShapeValues() . . . 456
9.104.3.4 getShapeValues() . . . 456
9.104.3.5 isValid() . . . 457
9.104.3.6 setDimensions() . . . 457
9.104.3.7 setExtraMemoryTarget() . . . 458
9.104.3.8 setShapeValues() . . . 458
9.104.4 Member Data Documentation . . . 459
9.104.4.1 mImpl . . . 459
9.105 IOutputAllocator Class Reference . . . 460
9.105.1 Detailed Description . . . 460
9.106 nvinfer1::v_1_0::IOutputAllocator Class Reference . . . 460
9.106.1 Member Function Documentation . . . 461
9.106.1.1 getInterfaceInfo() . . . 461
9.106.1.2 notifyShape() . . . 461
9.106.1.3 reallocateOutput() . . . 461
9.106.1.4 reallocateOutputAsync() . . . 462
9.107 nvinfer1::IPaddingLayer Class Reference . . . 463
9.107.1 Detailed Description . . . 464
9.107.2 Constructor & Destructor Documentation . . . 464
9.107.2.1 ~IPaddingLayer() . . . 464
9.107.3 Member Function Documentation . . . 464
9.107.3.1 getPostPaddingNd() . . . 464
9.107.3.2 getPrePaddingNd() . . . 464
9.107.3.3 setPostPaddingNd() . . . 465
9.107.3.4 setPrePaddingNd() . . . 465
9.107.4 Member Data Documentation . . . 466
9.107.4.1 mImpl . . . 466
9.108 nvinfer1::IParametricReLULayer Class Reference . . . 466
9.108.1 Detailed Description . . . 466
9.108.2 Constructor & Destructor Documentation . . . 467
9.108.2.1 ~IParametricReLULayer() . . . 467
9.108.3 Member Data Documentation . . . 467
9.108.3.1 mImpl . . . 467
9.109 nvonnxparser::IParser Class Reference . . . 467
9.109.1 Detailed Description . . . 469
9.109.2 Constructor & Destructor Documentation . . . 469
9.109.2.1 ~IParser() . . . 469
9.109.3 Member Function Documentation . . . 469
9.109.3.1 clearErrors() . . . 469
9.109.3.2 clearFlag() . . . 469
9.109.3.3 getError() . . . 470
9.109.3.4 getFlag() . . . 470
9.109.3.5 getFlags() . . . 470
9.109.3.6 getLayerOutputTensor() . . . 470
9.109.3.7 getNbErrors() . . . 471
9.109.3.8 getNbSubgraphs() . . . 471
9.109.3.9 getSubgraphNodes() . . . 471
9.109.3.10 getUsedVCPluginLibraries() . . . 472
9.109.3.11 isSubgraphSupported() . . . 472
9.109.3.12 parse() . . . 473
9.109.3.13 parseFromFile() . . . 473
9.109.3.14 parseWithWeightDescriptors() . . . 474
9.109.3.15 setFlag() . . . 474
9.109.3.16 setFlags() . . . 474

9.109.3.17 supportsModel() . . . 475
9.109.3.18 supportsModelV2() . . . 475
9.109.3.19 supportsOperator() . . . 476
9.110 nvonnxparser::IParserError Class Reference . . . 476
9.110.1 Detailed Description . . . 477
9.110.2 Constructor & Destructor Documentation . . . 477
9.110.2.1 ~IParserError() . . . 477
9.110.3 Member Function Documentation . . . 477
9.110.3.1 code() . . . 478
9.110.3.2 desc() . . . 478
9.110.3.3 file() . . . 478
9.110.3.4 func() . . . 478
9.110.3.5 line() . . . 478
9.110.3.6 localFunctionStack() . . . 478
9.110.3.7 localFunctionStackSize() . . . 479
9.110.3.8 node() . . . 479
9.110.3.9 nodeName() . . . 479
9.110.3.10 nodeOperator() . . . 479
9.111 nvonnxparser::IParserRefitter Class Reference . . . 479
9.111.1 Detailed Description . . . 480
9.111.2 Constructor & Destructor Documentation . . . 480
9.111.2.1 ~IParserRefitter() . . . 480
9.111.3 Member Function Documentation . . . 480
9.111.3.1 clearErrors() . . . 480
9.111.3.2 getError() . . . 481
9.111.3.3 getNbErrors() . . . 481
9.111.3.4 refitFromBytes() . . . 481
9.111.3.5 refitFromFile() . . . 482
9.112 IPluginCapability Class Reference . . . 482
9.112.1 Detailed Description . . . 482
9.113 nvinfer1::v_1_0::IPluginCapability Class Reference . . . 483
9.114 IPluginCreator Class Reference . . . 483
9.114.1 Detailed Description . . . 483
9.115 nvinfer1::v_1_0::IPluginCreator Class Reference . . . 484
9.115.1 Constructor & Destructor Documentation . . . 484
9.115.1.1 IPluginCreator() . . . 485
9.115.1.2 ~IPluginCreator() . . . 485
9.115.2 Member Function Documentation . . . 485
9.115.2.1 createPlugin() . . . 485
9.115.2.2 deserializePlugin() . . . 485
9.115.2.3 getFieldNames() . . . 486
9.115.2.4 getInterfaceInfo() . . . 487
9.115.2.5 getPluginName() . . . 487
9.115.2.6 getPluginNamespace() . . . 487
9.115.2.7 getPluginVersion() . . . 488
9.115.2.8 setPluginNamespace() . . . 488
9.116 IPluginCreatorInterface Class Reference . . . 489
9.116.1 Detailed Description . . . 489
9.117 nvinfer1::v_1_0::IPluginCreatorInterface Class Reference . . . 489
9.117.1 Constructor & Destructor Documentation . . . 490
9.117.1.1 ~IPluginCreatorInterface() . . . 490
9.117.1.2 IPluginCreatorInterface() [1/3] . . . 490
9.117.1.3 IPluginCreatorInterface() [2/3] . . . 490
9.117.1.4 IPluginCreatorInterface() [3/3] . . . 490

9.117.2 Member Function Documentation . . . 490
9.117.2.1 operator=() [1/2] . . . 490
9.117.2.2 operator=() [2/2] . . . 491
9.118 IPluginCreatorV3One Class Reference . . . 491
9.118.1 Detailed Description . . . 491
9.119 nvinfer1::v_1_0::IPluginCreatorV3One Class Reference . . . 491
9.119.1 Constructor & Destructor Documentation . . . 492
9.119.1.1 IPluginCreatorV3One() [1/3] . . . 492
9.119.1.2 ~IPluginCreatorV3One() . . . 492
9.119.1.3 IPluginCreatorV3One() [2/3] . . . 493
9.119.1.4 IPluginCreatorV3One() [3/3] . . . 493
9.119.2 Member Function Documentation . . . 493
9.119.2.1 createPlugin() . . . 493
9.119.2.2 getFieldNames() . . . 494
9.119.2.3 getInterfaceInfo() . . . 494
9.119.2.4 getPluginName() . . . 494
9.119.2.5 getPluginNamespace() . . . 494
9.119.2.6 getPluginVersion() . . . 495
9.119.2.7 operator=() [1/2] . . . 495
9.119.2.8 operator=() [2/2] . . . 495
9.120 nvinfer1::IPluginRegistry Class Reference . . . 495
9.120.1 Detailed Description . . . 497
9.120.2 Member Typedef Documentation . . . 497
9.120.2.1 PluginLibraryHandle . . . 497
9.120.3 Constructor & Destructor Documentation . . . 497
9.120.3.1 ~IPluginRegistry() . . . 497
9.120.4 Member Function Documentation . . . 497
9.120.4.1 acquirePluginResource() . . . 497
9.120.4.2 deregisterCreator() [1/2] . . . 498
9.120.4.3 deregisterCreator() [2/2] . . . 499
9.120.4.4 deregisterLibrary() . . . 499
9.120.4.5 getAllCreators() . . . 499
9.120.4.6 getAllCreatorsRecursive() . . . 500
9.120.4.7 getCreator() . . . 501
9.120.4.8 getErrorRecorder() . . . 501
9.120.4.9 getPluginCreator() . . . 502
9.120.4.10 getPluginCreatorList() . . . 502
9.120.4.11 isParentSearchEnabled() . . . 503
9.120.4.12 loadLibrary() . . . 503
9.120.4.13 registerCreator() [1/2] . . . 503
9.120.4.14 registerCreator() [2/2] . . . 504
9.120.4.15 releasePluginResource() . . . 504
9.120.4.16 setErrorRecorder() . . . 505
9.120.4.17 setParentSearchEnabled() . . . 505
9.121 IPluginResource Class Reference . . . 506
9.121.1 Detailed Description . . . 506
9.122 nvinfer1::v_1_0::IPluginResource Class Reference . . . 506
9.122.1 Constructor & Destructor Documentation . . . 507
9.122.1.1 ~IPluginResource() . . . 507
9.122.1.2 IPluginResource() [1/3] . . . 507
9.122.1.3 IPluginResource() [2/3] . . . 507
9.122.1.4 IPluginResource() [3/3] . . . 507
9.122.2 Member Function Documentation . . . 508
9.122.2.1 clone() . . . 508

9.122.2.2 getInterfaceInfo() . . . 508
9.122.2.3 operator=() [1/2] . . . 508
9.122.2.4 operator=() [2/2] . . . 509
9.122.2.5 release() . . . 509
9.123 nvinfer1::IPluginResourceContext Class Reference . . . 509
9.123.1 Detailed Description . . . 510
9.123.2 Constructor & Destructor Documentation . . . 510
9.123.2.1 ~IPluginResourceContext() . . . 510
9.123.2.2 IPluginResourceContext() [1/3] . . . 510
9.123.2.3 IPluginResourceContext() [2/3] . . . 510
9.123.2.4 IPluginResourceContext() [3/3] . . . 511
9.123.3 Member Function Documentation . . . 511
9.123.3.1 getErrorRecorder() . . . 511
9.123.3.2 getGpuAllocator() . . . 511
9.123.3.3 operator=() [1/2] . . . 511
9.123.3.4 operator=() [2/2] . . . 512
9.124 nvinfer1::IPluginV2 Class Reference . . . 512
9.124.1 Detailed Description . . . 513
9.124.2 Member Function Documentation . . . 513
9.124.2.1 clone() . . . 514
9.124.2.2 configureWithFormat() . . . 514
9.124.2.3 destroy() . . . 515
9.124.2.4 enqueue() . . . 515
9.124.2.5 getNbOutputs() . . . 516
9.124.2.6 getOutputDimensions() . . . 517
9.124.2.7 getPluginNamespace() . . . 518
9.124.2.8 getPluginType() . . . 519
9.124.2.9 getPluginVersion() . . . 519
9.124.2.10 getSerializationSize() . . . 520
9.124.2.11 getTensorRTVersion() . . . 520
9.124.2.12 getWorkspaceSize() . . . 520
9.124.2.13 initialize() . . . 521
9.124.2.14 serialize() . . . 521
9.124.2.15 setPluginNamespace() . . . 522
9.124.2.16 supportsFormat() . . . 523
9.124.2.17 terminate() . . . 523
9.125 nvinfer1::IPluginV2DynamicExt Class Reference . . . 524
9.125.1 Detailed Description . . . 525
9.125.2 Constructor & Destructor Documentation . . . 526
9.125.2.1 ~IPluginV2DynamicExt() . . . 526
9.125.3 Member Function Documentation . . . 526
9.125.3.1 clone() . . . 526
9.125.3.2 configurePlugin() . . . 527
9.125.3.3 enqueue() . . . 528
9.125.3.4 getOutputDimensions() . . . 528
9.125.3.5 getTensorRTVersion() . . . 529
9.125.3.6 getWorkspaceSize() . . . 529
9.125.3.7 supportsFormatCombination() . . . 530
9.125.4 Member Data Documentation . . . 530
9.125.4.1 kFORMAT_COMBINATION_LIMIT . . . 530
9.126 nvinfer1::IPluginV2Ext Class Reference . . . 531
9.126.1 Detailed Description . . . 532
9.126.2 Constructor & Destructor Documentation . . . 532
9.126.2.1 IPluginV2Ext() . . . 532

9.126.2.2 ~IPluginV2Ext() . . . 532
9.126.3 Member Function Documentation . . . 532
9.126.3.1 attachToContext() . . . 532
9.126.3.2 canBroadcastInputAcrossBatch() . . . 533
9.126.3.3 clone() . . . 534
9.126.3.4 configurePlugin() . . . 535
9.126.3.5 configureWithFormat() . . . 536
9.126.3.6 detachFromContext() . . . 536
9.126.3.7 getOutputDataType() . . . 536
9.126.3.8 getTensorRTVersion() . . . 537
9.126.3.9 isOutputBroadcastAcrossBatch() . . . 538
9.127 nvinfer1::IPluginV2IOExt Class Reference . . . 538
9.127.1 Detailed Description . . . 539
9.127.2 Member Function Documentation . . . 539
9.127.2.1 configurePlugin() . . . 539
9.127.2.2 getTensorRTVersion() . . . 540
9.127.2.3 supportsFormatCombination() . . . 541
9.128 nvinfer1::IPluginV2Layer Class Reference . . . 542
9.128.1 Detailed Description . . . 542
9.128.2 Constructor & Destructor Documentation . . . 543
9.128.2.1 ~IPluginV2Layer() . . . 543
9.128.3 Member Function Documentation . . . 543
9.128.3.1 getPlugin() . . . 543
9.128.4 Member Data Documentation . . . 543
9.128.4.1 mImpl . . . 543
9.129 IPluginV3 Class Reference . . . 543
9.129.1 Detailed Description . . . 544
9.130 nvinfer1::v_1_0::IPluginV3 Class Reference . . . 544
9.130.1 Member Function Documentation . . . 544
9.130.1.1 clone() . . . 545
9.130.1.2 getCapabilityInterface() . . . 545
9.130.1.3 getInterfaceInfo() . . . 545
9.131 nvinfer1::IPluginV3Layer Class Reference . . . 546
9.131.1 Detailed Description . . . 546
9.131.2 Constructor & Destructor Documentation . . . 546
9.131.2.1 ~IPluginV3Layer() . . . 547
9.131.3 Member Function Documentation . . . 547
9.131.3.1 getPlugin() . . . 547
9.131.4 Member Data Documentation . . . 547
9.131.4.1 mImpl . . . 547
9.132 IPluginV3OneBuild Class Reference . . . 547
9.132.1 Detailed Description . . . 548
9.133 nvinfer1::v_1_0::IPluginV3OneBuild Class Reference . . . 548
9.133.1 Member Function Documentation . . . 549
9.133.1.1 configurePlugin() . . . 549
9.133.1.2 getFormatCombinationLimit() . . . 550
9.133.1.3 getInterfaceInfo() . . . 550
9.133.1.4 getMetadataString() . . . 550
9.133.1.5 getNbOutputs() . . . 551
9.133.1.6 getNbTactics() . . . 551
9.133.1.7 getOutputDataTypes() . . . 551
9.133.1.8 getOutputShapes() . . . 552
9.133.1.9 getTimingCacheID() . . . 553
9.133.1.10 getValidTactics() . . . 553

9.133.1.1 getWorkspaceSize() . . . 554
9.133.1.2 supportsFormatCombination() . . . 554
9.133.2 Member Data Documentation . . . 555
9.133.2.1 kDEFAULT_FORMAT_COMBINATION_LIMIT . . . 555
9.134 nvinfer1::v_2_0::IPluginV3OneBuild Class Reference . . . 555
9.134.1 Member Function Documentation . . . 556
9.134.1.1 getAliasedInput() . . . 556
9.134.1.2 getInterfaceInfo() . . . 557
9.135 IPluginV3OneBuildV2 Class Reference . . . 557
9.135.1 Detailed Description . . . 557
9.136 IPluginV3OneCore Class Reference . . . 557
9.136.1 Detailed Description . . . 558
9.137 nvinfer1::v_1_0::IPluginV3OneCore Class Reference . . . 558
9.137.1 Member Function Documentation . . . 558
9.137.1.1 getInterfaceInfo() . . . 559
9.137.1.2 getPluginName() . . . 559
9.137.1.3 getPluginNamespace() . . . 559
9.137.1.4 getPluginVersion() . . . 559
9.138 IPluginV3OneRuntime Class Reference . . . 560
9.138.1 Detailed Description . . . 560
9.139 nvinfer1::v_1_0::IPluginV3OneRuntime Class Reference . . . 560
9.139.1 Member Function Documentation . . . 561
9.139.1.1 attachToContext() . . . 561
9.139.1.2 enqueue() . . . 562
9.139.1.3 getFieldsToSerialize() . . . 562
9.139.1.4 getInterfaceInfo() . . . 563
9.139.1.5 onShapeChange() . . . 563
9.139.1.6 setTactic() . . . 564
9.140 nvinfer1::IPoolingLayer Class Reference . . . 564
9.140.1 Detailed Description . . . 565
9.140.2 Constructor & Destructor Documentation . . . 566
9.140.2.1 ~IPoolingLayer() . . . 566
9.140.3 Member Function Documentation . . . 566
9.140.3.1 getAverageCountExcludesPadding() . . . 566
9.140.3.2 getBlendFactor() . . . 566
9.140.3.3 getPaddingMode() . . . 567
9.140.3.4 getPaddingNd() . . . 567
9.140.3.5 getPoolingType() . . . 567
9.140.3.6 getPostPadding() . . . 567
9.140.3.7 getPrePadding() . . . 568
9.140.3.8 getStrideNd() . . . 568
9.140.3.9 getWindowSizeNd() . . . 568
9.140.3.10 setAverageCountExcludesPadding() . . . 568
9.140.3.11 setBlendFactor() . . . 569
9.140.3.12 setPaddingMode() . . . 569
9.140.3.13 setPaddingNd() . . . 569
9.140.3.14 setPoolingType() . . . 570
9.140.3.15 setPostPadding() . . . 570
9.140.3.16 setPrePadding() . . . 570
9.140.3.17 setStrideNd() . . . 571
9.140.3.18 setWindowSizeNd() . . . 571
9.140.4 Member Data Documentation . . . 571
9.140.4.1 mImpl . . . 571
9.141 IProfiler Class Reference . . . 572

9.141.1 Detailed Description . . . 572
9.142 nvinfer1::v_1_0::IProfiler Class Reference . . . 572
9.142.1 Constructor & Destructor Documentation . . . 572
9.142.1.1 ~IProfiler() . . . 572
9.142.2 Member Function Documentation . . . 572
9.142.2.1 reportLayerTime() . . . 572
9.143 IProgressMonitor Class Reference . . . 573
9.143.1 Detailed Description . . . 573
9.144 nvinfer1::v_1_0::IProgressMonitor Class Reference . . . 574
9.144.1 Constructor & Destructor Documentation . . . 574
9.144.1.1 IProgressMonitor() . . . 574
9.144.1.2 ~IProgressMonitor() . . . 574
9.144.2 Member Function Documentation . . . 575
9.144.2.1 getInterfaceInfo() . . . 575
9.144.2.2 phaseFinish() . . . 575
9.144.2.3 phaseStart() . . . 575
9.144.2.4 stepComplete() . . . 576
9.145 nvinfer1::IQuantizeLayer Class Reference . . . 577
9.145.1 Detailed Description . . . 577
9.145.2 Constructor & Destructor Documentation . . . 579
9.145.2.1 ~IQuantizeLayer() . . . 579
9.145.3 Member Function Documentation . . . 579
9.145.3.1 getAxis() . . . 579
9.145.3.2 getToType() . . . 579
9.145.3.3 setAxis() . . . 580
9.145.3.4 setToType() . . . 580
9.145.4 Member Data Documentation . . . 580
9.145.4.1 mImpl . . . 580
9.146 nvinfer1::IRaggedSoftMaxLayer Class Reference . . . 581
9.146.1 Detailed Description . . . 581
9.146.2 Constructor & Destructor Documentation . . . 581
9.146.2.1 ~IRaggedSoftMaxLayer() . . . 582
9.146.3 Member Data Documentation . . . 582
9.146.3.1 mImpl . . . 582
9.147 nvinfer1::IRecurrenceLayer Class Reference . . . 582
9.147.1 Detailed Description . . . 583
9.147.2 Constructor & Destructor Documentation . . . 583
9.147.2.1 ~IRecurrenceLayer() . . . 583
9.147.3 Member Function Documentation . . . 583
9.147.3.1 setInput() . . . 583
9.147.4 Member Data Documentation . . . 584
9.147.4.1 mImpl . . . 584
9.148 nvinfer1::IReduceLayer Class Reference . . . 584
9.148.1 Detailed Description . . . 585
9.148.2 Constructor & Destructor Documentation . . . 585
9.148.2.1 ~IReduceLayer() . . . 585
9.148.3 Member Function Documentation . . . 585
9.148.3.1 getKeepDimensions() . . . 585
9.148.3.2 getOperation() . . . 586
9.148.3.3 getReduceAxes() . . . 586
9.148.3.4 setKeepDimensions() . . . 586
9.148.3.5 setOperation() . . . 586
9.148.3.6 setReduceAxes() . . . 587
9.148.4 Member Data Documentation . . . 587

9.148.4.1 mImpl . . . 587
9.149 nvinfer1::IRefitter Class Reference . . . 587
9.149.1 Detailed Description . . . 589
9.149.2 Constructor & Destructor Documentation . . . 589
9.149.2.1 ~IRefitter() . . . 589
9.149.3 Member Function Documentation . . . 589
9.149.3.1 getAll() . . . 589
9.149.3.2 getAllWeights() . . . 590
9.149.3.3 getDynamicRangeMax() . . . 590
9.149.3.4 getDynamicRangeMin() . . . 591
9.149.3.5 getErrorRecorder() . . . 591
9.149.3.6 getLogger() . . . 592
9.149.3.7 getMaxThreads() . . . 592
9.149.3.8 getMissing() . . . 592
9.149.3.9 getMissingWeights() . . . 593
9.149.3.10 getNamedWeights() . . . 593
9.149.3.11 getTensorsWithDynamicRange() . . . 594
9.149.3.12 getWeightsLocation() . . . 594
9.149.3.13 getWeightsPrototype() . . . 595
9.149.3.14 getWeightsValidation() . . . 595
9.149.3.15 refitCudaEngine() . . . 596
9.149.3.16 refitCudaEngineAsync() . . . 596
9.149.3.17 setDynamicRange() . . . 596
9.149.3.18 setErrorRecorder() . . . 597
9.149.3.19 setMaxThreads() . . . 598
9.149.3.20 setNamedWeights() [1/2] . . . 598
9.149.3.21 setNamedWeights() [2/2] . . . 599
9.149.3.22 setWeights() . . . 599
9.149.3.23 setWeightsValidation() . . . 600
9.149.3.24 unsetNamedWeights() . . . 600
9.149.4 Member Data Documentation . . . 601
9.149.4.1 mImpl . . . 601
9.150 nvinfer1::IResizeLayer Class Reference . . . 601
9.150.1 Detailed Description . . . 603
9.150.2 Constructor & Destructor Documentation . . . 603
9.150.2.1 ~IResizeLayer() . . . 604
9.150.3 Member Function Documentation . . . 604
9.150.3.1 getCoordinateTransformation() . . . 604
9.150.3.2 getCubicCoeff() . . . 604
9.150.3.3 getExcludeOutside() . . . 604
9.150.3.4 getNearestRounding() . . . 605
9.150.3.5 getOutputDimensions() . . . 605
9.150.3.6 getResizeMode() . . . 605
9.150.3.7 getScales() . . . 605
9.150.3.8 getSelectorForSinglePixel() . . . 606
9.150.3.9 setCoordinateTransformation() . . . 606
9.150.3.10 setCubicCoeff() . . . 607
9.150.3.11 setExcludeOutside() . . . 607
9.150.3.12 setInput() . . . 607
9.150.3.13 setNearestRounding() . . . 608
9.150.3.14 setOutputDimensions() . . . 608
9.150.3.15 setResizeMode() . . . 609
9.150.3.16 setScales() . . . 609
9.150.3.17 setSelectorForSinglePixel() . . . 610

9.150.4 Member Data Documentation . . . 610
9.150.4.1 mImpl . . . 610
9.151 nvinfer1::IReverseSequenceLayer Class Reference . . . 610
9.151.1 Detailed Description . . . 611
9.151.2 Constructor & Destructor Documentation . . . 611
9.151.2.1 ~IReverseSequenceLayer() . . . 611
9.151.3 Member Function Documentation . . . 611
9.151.3.1 getBatchAxis() . . . 612
9.151.3.2 getSequenceAxis() . . . 612
9.151.3.3 setBatchAxis() . . . 612
9.151.3.4 setSequenceAxis() . . . 612
9.151.4 Member Data Documentation . . . 613
9.151.4.1 mImpl . . . 613
9.152 nvinfer1::IRuntime Class Reference . . . 613
9.152.1 Detailed Description . . . 614
9.152.2 Constructor & Destructor Documentation . . . 614
9.152.2.1 ~IRuntime() . . . 615
9.152.3 Member Function Documentation . . . 615
9.152.3.1 deserializeCudaEngine() [1/3] . . . 615
9.152.3.2 deserializeCudaEngine() [2/3] . . . 615
9.152.3.3 deserializeCudaEngine() [3/3] . . . 616
9.152.3.4 getDLACore() . . . 616
9.152.3.5 getEngineHostCodeAllowed() . . . 617
9.152.3.6 getErrorRecorder() . . . 617
9.152.3.7 getLogger() . . . 617
9.152.3.8 getMaxThreads() . . . 618
9.152.3.9 getNbDLACores() . . . 618
9.152.3.10 getPluginRegistry() . . . 618
9.152.3.11 getTempfileControlFlags() . . . 618
9.152.3.12 getTemporaryDirectory() . . . 619
9.152.3.13 loadRuntime() . . . 619
9.152.3.14 setDLACore() . . . 619
9.152.3.15 setEngineHostCodeAllowed() . . . 620
9.152.3.16 setErrorRecorder() . . . 620
9.152.3.17 setGpuAllocator() . . . 621
9.152.3.18 setMaxThreads() . . . 621
9.152.3.19 setTempfileControlFlags() . . . 621
9.152.3.20 setTemporaryDirectory() . . . 622
9.152.4 Member Data Documentation . . . 623
9.152.4.1 mImpl . . . 623
9.153 nvinfer1::IScaleLayer Class Reference . . . 623
9.153.1 Detailed Description . . . 624
9.153.2 Constructor & Destructor Documentation . . . 625
9.153.2.1 ~IScaleLayer() . . . 625
9.153.3 Member Function Documentation . . . 625
9.153.3.1 getChannelAxis() . . . 625
9.153.3.2 getMode() . . . 626
9.153.3.3 getPower() . . . 626
9.153.3.4 getScale() . . . 626
9.153.3.5 getShift() . . . 626
9.153.3.6 setChannelAxis() . . . 627
9.153.3.7 setMode() . . . 627
9.153.3.8 setPower() . . . 627
9.153.3.9 setScale() . . . 628

9.153.3.10 setShift() . . . 628
9.153.4 Member Data Documentation . . . 628
9.153.4.1 mImpl . . . 628
9.154 nvinfer1::IScatterLayer Class Reference . . . 628
9.154.1 Detailed Description . . . 629
9.154.2 Constructor & Destructor Documentation . . . 630
9.154.2.1 ~IScatterLayer() . . . 630
9.154.3 Member Function Documentation . . . 630
9.154.3.1 getAxis() . . . 630
9.154.3.2 getMode() . . . 631
9.154.3.3 setAxis() . . . 631
9.154.3.4 setMode() . . . 631
9.154.4 Member Data Documentation . . . 631
9.154.4.1 mImpl . . . 631
9.155 nvinfer1::ISelectLayer Class Reference . . . 632
9.155.1 Detailed Description . . . 632
9.155.2 Constructor & Destructor Documentation . . . 632
9.155.2.1 ~ISelectLayer() . . . 632
9.155.3 Member Data Documentation . . . 633
9.155.3.1 mImpl . . . 633
9.156 nvinfer1::ISerializationConfig Class Reference . . . 633
9.156.1 Detailed Description . . . 634
9.156.2 Constructor & Destructor Documentation . . . 634
9.156.2.1 ~ISerializationConfig() . . . 634
9.156.3 Member Function Documentation . . . 634
9.156.3.1 clearFlag() . . . 634
9.156.3.2 getFlag() . . . 635
9.156.3.3 getFlags() . . . 635
9.156.3.4 setFlag() . . . 635
9.156.3.5 setFlags() . . . 635
9.156.4 Member Data Documentation . . . 636
9.156.4.1 mImpl . . . 636
9.157 nvinfer1::IShapeLayer Class Reference . . . 636
9.157.1 Detailed Description . . . 637
9.157.2 Constructor & Destructor Documentation . . . 637
9.157.2.1 ~IShapeLayer() . . . 637
9.157.3 Member Data Documentation . . . 637
9.157.3.1 mImpl . . . 637
9.158 nvinfer1::IShuffleLayer Class Reference . . . 638
9.158.1 Detailed Description . . . 639
9.158.2 Constructor & Destructor Documentation . . . 639
9.158.2.1 ~IShuffleLayer() . . . 639
9.158.3 Member Function Documentation . . . 639
9.158.3.1 getFirstTranspose() . . . 639
9.158.3.2 getReshapeDimensions() . . . 640
9.158.3.3 getSecondTranspose() . . . 640
9.158.3.4 getZeroIsPlaceholder() . . . 640
9.158.3.5 setFirstTranspose() . . . 640
9.158.3.6 setInput() . . . 641
9.158.3.7 setReshapeDimensions() . . . 642
9.158.3.8 setSecondTranspose() . . . 642
9.158.3.9 setZeroIsPlaceholder() . . . 643
9.158.4 Member Data Documentation . . . 643
9.158.4.1 mImpl . . . 643

9.159 nvinfer1::ISliceLayer Class Reference . . . 643
9.159.1 Detailed Description . . . 644
9.159.2 Constructor & Destructor Documentation . . . 645
9.159.2.1 ~ISliceLayer() . . . 646
9.159.3 Member Function Documentation . . . 646
9.159.3.1 getAxes() . . . 646
9.159.3.2 getMode() . . . 646
9.159.3.3 getSize() . . . 647
9.159.3.4 getStart() . . . 647
9.159.3.5 getStride() . . . 647
9.159.3.6 setAxes() . . . 647
9.159.3.7 setInput() . . . 648
9.159.3.8 setMode() . . . 649
9.159.3.9 setSize() . . . 649
9.159.3.10 setStart() . . . 649
9.159.3.11 setStride() . . . 650
9.159.4 Member Data Documentation . . . 650
9.159.4.1 mImpl . . . 650
9.160 nvinfer1::ISoftMaxLayer Class Reference . . . 650
9.160.1 Detailed Description . . . 651
9.160.2 Constructor & Destructor Documentation . . . 651
9.160.2.1 ~ISoftMaxLayer() . . . 651
9.160.3 Member Function Documentation . . . 652
9.160.3.1 getAxes() . . . 652
9.160.3.2 setAxes() . . . 652
9.160.4 Member Data Documentation . . . 652
9.160.4.1 mImpl . . . 652
9.161 nvinfer1::ISqueezeLayer Class Reference . . . 653
9.161.1 Detailed Description . . . 653
9.161.2 Constructor & Destructor Documentation . . . 653
9.161.2.1 ~ISqueezeLayer() . . . 654
9.161.3 Member Function Documentation . . . 654
9.161.3.1 setInput() . . . 654
9.161.4 Member Data Documentation . . . 654
9.161.4.1 mImpl . . . 654
9.162 IStreamReader Class Reference . . . 654
9.162.1 Detailed Description . . . 655
9.163 nvinfer1::v_1_0::IStreamReader Class Reference . . . 655
9.163.1 Constructor & Destructor Documentation . . . 655
9.163.1.1 ~IStreamReader() . . . 656
9.163.1.2 IStreamReader() [1/3] . . . 656
9.163.1.3 IStreamReader() [2/3] . . . 656
9.163.1.4 IStreamReader() [3/3] . . . 656
9.163.2 Member Function Documentation . . . 656
9.163.2.1 getInterfaceInfo() . . . 656
9.163.2.2 operator=() [1/2] . . . 656
9.163.2.3 operator=() [2/2] . . . 657
9.163.2.4 read() . . . 657
9.164 IStreamReaderV2 Class Reference . . . 657
9.164.1 Detailed Description . . . 657
9.165 nvinfer1::v_1_0::IStreamReaderV2 Class Reference . . . 658
9.165.1 Constructor & Destructor Documentation . . . 658
9.165.1.1 ~IStreamReaderV2() . . . 658
9.165.1.2 IStreamReaderV2() [1/3] . . . 658

9.165.1.3 IStreamReaderV2() [2/3] . . . 659
9.165.1.4 IStreamReaderV2() [3/3] . . . 659
9.165.2 Member Function Documentation . . . 659
9.165.2.1 getInterfaceInfo() . . . 659
9.165.2.2 operator=() [1/2] . . . 659
9.165.2.3 operator=() [2/2] . . . 659
9.165.2.4 read() . . . 659
9.165.2.5 seek() . . . 660
9.166 nvinfer1::ITensor Class Reference . . . 660
9.166.1 Detailed Description . . . 662
9.166.2 Constructor & Destructor Documentation . . . 662
9.166.2.1 ~ITensor() . . . 662
9.166.3 Member Function Documentation . . . 663
9.166.3.1 dynamicRangeIsSet() . . . 663
9.166.3.2 getAllowedFormats() . . . 663
9.166.3.3 getBroadcastAcrossBatch() . . . 663
9.166.3.4 getDimensionName() . . . 663
9.166.3.5 getDimensions() . . . 664
9.166.3.6 getDynamicRangeMax() . . . 664
9.166.3.7 getDynamicRangeMin() . . . 665
9.166.3.8 getLocation() . . . 665
9.166.3.9 getName() . . . 665
9.166.3.10 getType() . . . 666
9.166.3.11 isExecutionTensor() . . . 666
9.166.3.12 isNetworkInput() . . . 666
9.166.3.13 isNetworkOutput() . . . 666
9.166.3.14 isShapeTensor() . . . 667
9.166.3.15 resetDynamicRange() . . . 667
9.166.3.16 setAllowedFormats() . . . 667
9.166.3.17 setBroadcastAcrossBatch() . . . 668
9.166.3.18 setDimensionName() . . . 668
9.166.3.19 setDimensions() . . . 670
9.166.3.20 setDynamicRange() . . . 670
9.166.3.21 setLocation() . . . 671
9.166.3.22 setName() . . . 671
9.166.3.23 setType() . . . 672
9.166.4 Member Data Documentation . . . 672
9.166.4.1 mImpl . . . 672
9.167 nvinfer1::ITimingCache Class Reference . . . 673
9.167.1 Detailed Description . . . 673
9.167.2 Constructor & Destructor Documentation . . . 674
9.167.2.1 ~ITimingCache() . . . 674
9.167.3 Member Function Documentation . . . 674
9.167.3.1 combine() . . . 674
9.167.3.2 query() . . . 675
9.167.3.3 queryKeys() . . . 675
9.167.3.4 reset() . . . 676
9.167.3.5 serialize() . . . 676
9.167.3.6 update() . . . 676
9.167.4 Member Data Documentation . . . 677
9.167.4.1 mImpl . . . 677
9.168 nvinfer1::ITopKLayer Class Reference . . . 677
9.168.1 Detailed Description . . . 678
9.168.2 Constructor & Destructor Documentation . . . 678

9.168.2.1 ~ITopKLayer() . . . 678
9.168.3 Member Function Documentation . . . 679
9.168.3.1 getK() . . . 679
9.168.3.2 getOperation() . . . 679
9.168.3.3 getReduceAxes() . . . 679
9.168.3.4 setInput() . . . 679
9.168.3.5 setK() . . . 680
9.168.3.6 setOperation() . . . 680
9.168.3.7 setReduceAxes() . . . 681
9.168.4 Member Data Documentation . . . 681
9.168.4.1 mImpl . . . 681
9.169 nvinfer1::ITripLimitLayer Class Reference . . . 681
9.169.1 Detailed Description . . . 682
9.169.2 Constructor & Destructor Documentation . . . 682
9.169.2.1 ~ITripLimitLayer() . . . 682
9.169.3 Member Function Documentation . . . 682
9.169.3.1 getTripLimit() . . . 682
9.169.4 Member Data Documentation . . . 682
9.169.4.1 mImpl . . . 683
9.170 nvinfer1::IUnaryLayer Class Reference . . . 683
9.170.1 Detailed Description . . . 683
9.170.2 Constructor & Destructor Documentation . . . 684
9.170.2.1 ~IUnaryLayer() . . . 684
9.170.3 Member Function Documentation . . . 684
9.170.3.1 getOperation() . . . 684
9.170.3.2 setOperation() . . . 684
9.170.4 Member Data Documentation . . . 684
9.170.4.1 mImpl . . . 685
9.171 nvinfer1::IUnsqueezeLayer Class Reference . . . 685
9.171.1 Detailed Description . . . 685
9.171.2 Constructor & Destructor Documentation . . . 686
9.171.2.1 ~IUnsqueezeLayer() . . . 686
9.171.3 Member Function Documentation . . . 686
9.171.3.1 setInput() . . . 686
9.171.4 Member Data Documentation . . . 686
9.171.4.1 mImpl . . . 687
9.172 nvinfer1::IVersionedInterface Class Reference . . . 687
9.172.1 Detailed Description . . . 688
9.172.2 Constructor & Destructor Documentation . . . 688
9.172.2.1 ~IVersionedInterface() . . . 688
9.172.2.2 IVersionedInterface() [1/3] . . . 688
9.172.2.3 IVersionedInterface() [2/3] . . . 688
9.172.2.4 IVersionedInterface() [3/3] . . . 688
9.172.3 Member Function Documentation . . . 688
9.172.3.1 getAPILanguage() . . . 689
9.172.3.2 getInterfaceInfo() . . . 689
9.172.3.3 operator=() [1/2] . . . 689
9.172.3.4 operator=() [2/2] . . . 689
9.173 nvinfer1::plugin::NMSParameters Struct Reference . . . 689
9.173.1 Detailed Description . . . 690
9.173.2 Member Data Documentation . . . 690
9.173.2.1 backgroundLabelId . . . 690
9.173.2.2 iouThreshold . . . 690
9.173.2.3 isNormalized . . . 690

9.173.2.4 keepTopK . . . 691
9.173.2.5 numClasses . . . 691
9.173.2.6 scoreThreshold . . . 691
9.173.2.7 shareLocation . . . 691
9.173.2.8 topK . . . 691
9.174 nvinfer1::Permutation Struct Reference . . . 691
9.174.1 Detailed Description . . . 692
9.174.2 Member Data Documentation . . . 692
9.174.2.1 order . . . 692
9.175 nvinfer1::PluginField Class Reference . . . 692
9.175.1 Detailed Description . . . 693
9.175.2 Constructor & Destructor Documentation . . . 693
9.175.2.1 PluginField() . . . 693
9.175.3 Member Data Documentation . . . 693
9.175.3.1 data . . . 693
9.175.3.2 length . . . 693
9.175.3.3 name . . . 693
9.175.3.4 type . . . 694
9.176 nvinfer1::PluginFieldCollection Struct Reference . . . 694
9.176.1 Detailed Description . . . 694
9.176.2 Member Data Documentation . . . 694
9.176.2.1 fields . . . 694
9.176.2.2 nbFields . . . 695
9.177 nvinfer1::PluginRegistrar< T > Class Template Reference . . . 695
9.177.1 Detailed Description . . . 695
9.177.2 Constructor & Destructor Documentation . . . 695
9.177.2.1 PluginRegistrar() . . . 695
9.178 nvinfer1::PluginTensorDesc Struct Reference . . . 696
9.178.1 Detailed Description . . . 696
9.178.2 Member Data Documentation . . . 696
9.178.2.1 dims . . . 696
9.178.2.2 format . . . 696
9.178.2.3 scale . . . 697
9.178.2.4 type . . . 697
9.179 PluginVersion Struct Reference . . . 697
9.179.1 Detailed Description . . . 697
9.180 nvinfer1::plugin::PriorBoxParameters Struct Reference . . . 697
9.180.1 Detailed Description . . . 698
9.180.2 Member Data Documentation . . . 698
9.180.2.1 aspectRatios . . . 698
9.180.2.2 clip . . . 699
9.180.2.3 flip . . . 699
9.180.2.4 imgH . . . 699
9.180.2.5 imgW . . . 699
9.180.2.6 maxSize . . . 699
9.180.2.7 minSize . . . 699
9.180.2.8 numAspectRatios . . . 700
9.180.2.9 numMaxSize . . . 700
9.180.2.10 numMinSize . . . 700
9.180.2.11 offset . . . 700
9.180.2.12 stepH . . . 700
9.180.2.13 stepW . . . 700
9.180.2.14 variance . . . 701
9.181 nvinfer1::plugin::RegionParameters Struct Reference . . . 701

9.181.1 Detailed Description . . . 701
9.181.2 Member Data Documentation . . . 701
9.181.2.1 classes . . . 702
9.181.2.2 coords . . . 702
9.181.2.3 num . . . 702
9.181.2.4 smTree . . . 702
9.182 nvinfer1::plugin::RPROIParams Struct Reference . . . 702
9.182.1 Detailed Description . . . 703
9.182.2 Member Data Documentation . . . 703
9.182.2.1 anchorsRatioCount . . . 703
9.182.2.2 anchorsScaleCount . . . 703
9.182.2.3 featureStride . . . 704
9.182.2.4 iouThreshold . . . 704
9.182.2.5 minBoxSize . . . 704
9.182.2.6 nmsMaxOut . . . 704
9.182.2.7 poolingH . . . 704
9.182.2.8 poolingW . . . 704
9.182.2.9 preNmsTop . . . 705
9.182.2.10 spatialScale . . . 705
9.183 nvinfer1::plugin::softmaxTree Struct Reference . . . 705
9.183.1 Detailed Description . . . 705
9.183.2 Member Data Documentation . . . 705
9.183.2.1 child . . . 706
9.183.2.2 group . . . 706
9.183.2.3 groupOffset . . . 706
9.183.2.4 groups . . . 706
9.183.2.5 groupSize . . . 706
9.183.2.6 leaf . . . 706
9.183.2.7 n . . . 706
9.183.2.8 name . . . 707
9.183.2.9 parent . . . 707
9.184 nvinfer1::v_1_0::TimingCacheKey Struct Reference . . . 707
9.184.1 Detailed Description . . . 707
9.184.2 Member Data Documentation . . . 707
9.184.2.1 data . . . 708
9.185 nvinfer1::v_1_0::TimingCacheValue Struct Reference . . . 708
9.185.1 Member Data Documentation . . . 708
9.185.1.1 kINVALID_TACTIC_HASH . . . 708
9.185.1.2 tacticHash . . . 708
9.185.1.3 timingMSec . . . 709
9.186 Value Struct Reference . . . 709
9.186.1 Detailed Description . . . 709
9.187 nvinfer1::Weights Class Reference . . . 709
9.187.1 Detailed Description . . . 710
9.187.2 Member Data Documentation . . . 710
9.187.2.1 count . . . 710
9.187.2.2 type . . . 710
9.187.2.3 values . . . 710

10 File Documentation **711**
10.1 NvInfer.h File Reference . . . 711
10.1.1 Detailed Description . . . 719
10.2 NvInfer.h . . . 719
10.3 NvInferImpl.h File Reference . . . 766

10.4 NvInferImpl.h . . . 766
10.5 NvInferLegacyDims.h File Reference . . . 782
10.5.1 Detailed Description . . . 783
10.5.2 Macro Definition Documentation . . . 783
10.5.2.1 NV_INFER_INTERNAL_INCLUDE . . . 783
10.6 NvInferLegacyDims.h . . . 783
10.7 NvInferPlugin.h File Reference . . . 785
10.7.1 Detailed Description . . . 785
10.7.2 Function Documentation . . . 785
10.7.2.1 initLibNvInferPlugins() . . . 785
10.8 NvInferPlugin.h . . . 785
10.9 NvInferPluginBase.h File Reference . . . 786
10.9.1 Macro Definition Documentation . . . 787
10.9.1.1 NV_INFER_INTERNAL_INCLUDE . . . 787
10.10 NvInferPluginBase.h . . . 787
10.11 NvInferPluginUtils.h File Reference . . . 789
10.11.1 Detailed Description . . . 790
10.12 NvInferPluginUtils.h . . . 790
10.13 NvInferRuntime.h File Reference . . . 792
10.13.1 Detailed Description . . . 796
10.13.2 Macro Definition Documentation . . . 797
10.13.2.1 NV_INFER_INTERNAL_INCLUDE . . . 797
10.13.2.2 REGISTER_TENSORRT_PLUGIN . . . 797
10.13.3 Function Documentation . . . 797
10.13.3.1 getInferLibBuildVersion() . . . 797
10.13.3.2 getInferLibMajorVersion() . . . 797
10.13.3.3 getInferLibMinorVersion() . . . 797
10.13.3.4 getInferLibPatchVersion() . . . 798
10.13.3.5 getLogger() . . . 798
10.13.3.6 getPluginRegistry() . . . 798
10.14 NvInferRuntime.h . . . 798
10.15 NvInferRuntimeBase.h File Reference . . . 821
10.15.1 Detailed Description . . . 822
10.15.2 Macro Definition Documentation . . . 823
10.15.2.1 NV_TENSORRT_VERSION . . . 823
10.15.2.2 NV_TENSORRT_VERSION_INT . . . 823
10.15.2.3 TENSORRTAPI . . . 823
10.15.2.4 TRT_DEPRECATED . . . 823
10.15.2.5 TRT_DEPRECATED_API . . . 823
10.15.2.6 TRT_DEPRECATED_ENUM . . . 823
10.15.2.7 TRTNOEXCEPT . . . 824
10.15.3 Function Documentation . . . 824
10.15.3.1 getInferLibVersion() . . . 824
10.16 NvInferRuntimeBase.h . . . 824
10.17 NvInferRuntimeCommon.h File Reference . . . 828
10.17.1 Detailed Description . . . 828
10.17.2 Macro Definition Documentation . . . 828
10.17.2.1 NV_INFER_INTERNAL_INCLUDE . . . 829
10.18 NvInferRuntimeCommon.h . . . 829
10.19 NvInferRuntimePlugin.h File Reference . . . 830
10.19.1 Detailed Description . . . 831
10.19.2 Macro Definition Documentation . . . 831
10.19.2.1 NV_INFER_INTERNAL_INCLUDE . . . 831
10.20 NvInferRuntimePlugin.h . . . 831

10.21 NvInferSerialize.h File Reference . . . 835
10.22 NvInferSerialize.h . . . 835
10.23 NvInferVersion.h File Reference . . . 836
10.23.1 Detailed Description . . . 836
10.23.2 Macro Definition Documentation . . . 836
10.23.2.1 NV_TENSORRT_BUILD . . . 836
10.23.2.2 NV_TENSORRT_LWS_MAJOR . . . 837
10.23.2.3 NV_TENSORRT_LWS_MINOR . . . 837
10.23.2.4 NV_TENSORRT_LWS_PATCH . . . 837
10.23.2.5 NV_TENSORRT_MAJOR . . . 837
10.23.2.6 NV_TENSORRT_MINOR . . . 837
10.23.2.7 NV_TENSORRT_PATCH . . . 837
10.23.2.8 NV_TENSORRT_RELEASE_TYPE . . . 838
10.23.2.9 NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS . . . 838
10.23.2.10 NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY . . . 838
10.23.2.11 NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE . . . 838
10.24 NvInferVersion.h . . . 838
10.25 NvOnnxConfig.h File Reference . . . 839
10.25.1 Detailed Description . . . 839
10.26 NvOnnxConfig.h . . . 840
10.27 NvOnnxParser.h File Reference . . . 841
10.27.1 Detailed Description . . . 842
10.27.2 Macro Definition Documentation . . . 842
10.27.2.1 NV_ONNX_PARSER_MAJOR . . . 842
10.27.2.2 NV_ONNX_PARSER_MINOR . . . 843
10.27.2.3 NV_ONNX_PARSER_PATCH . . . 843
10.27.3 Typedef Documentation . . . 843
10.27.3.1 SubGraph_t . . . 843
10.27.3.2 SubGraphCollection_t . . . 843
10.27.4 Function Documentation . . . 843
10.27.4.1 createNvOnnxParser_INTERNAL() . . . 843
10.27.4.2 createNvOnnxParserRefitter_INTERNAL() . . . 844
10.27.4.3 getNvOnnxParserVersion() . . . 844
10.28 NvOnnxParser.h . . . 844

Index **849**

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 828
NvInferSafeRuntime.h . . . . . **??**
NvInferVersion.h 836

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.

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

Chapter 3

Deprecated List

Class IAlgorithmSelector
Deprecated in TensorRT 10.8. Please use editable mode in ITimingCache instead.

Class IInt8EntropyCalibrator
Deprecated in TensorRT 10.1. Superseded by explicit quantization.

Class IInt8EntropyCalibrator2
Deprecated in TensorRT 10.1. Superseded by explicit quantization.

Class IInt8LegacyCalibrator
Deprecated in TensorRT 10.1. Superseded by explicit quantization.

Class IInt8MinMaxCalibrator
Deprecated in TensorRT 10.1. Superseded by explicit quantization.

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

Member nvinfer1::CalibrationAlgoType
Deprecated in TensorRT 10.1. Superseded by explicit quantization.

Class nvinfer1::IAlgorithm
Deprecated in TensorRT 10.8. Please use editable mode in ITimingCache instead.

Class nvinfer1::IAlgorithmContext
Deprecated in TensorRT 10.8. Please use editable mode in ITimingCache instead.

Class nvinfer1::IAlgorithmIOInfo
Deprecated in TensorRT 10.8. Please use editable mode in ITimingCache instead.

Class nvinfer1::IAlgorithmVariant
Deprecated in TensorRT 10.8. Please use editable mode in ITimingCache instead.

Member nvinfer1::IBuilder::platformHasFastFp16 () const noexcept
Deprecated in TensorRT 10.5. Please query data type support from CUDA directly.

Member nvinfer1::IBuilder::platformHasFastInt8 () const noexcept
Deprecated in TensorRT 10.5. Please query data type support from CUDA directly.

Member nvinfer1::IBuilder::platformHasTf32 () const noexcept
Deprecated in TensorRT 10.5. Please query data type support from CUDA directly.

Member nvinfer1::IBuilderConfig::clearQuantizationFlag (QuantizationFlag flag) noexcept

Deprecated in TensorRT 10.10. Superseded by explicit quantization.

Member nvinfer1::IBuilderConfig::getAlgorithmSelector () const noexcept

Deprecated in TensorRT 10.8. Please use editable mode in ITimingCache instead.

Member nvinfer1::IBuilderConfig::getCalibrationProfile () noexcept

Deprecated in TensorRT 10.1. Superseded by explicit quantization.

Member nvinfer1::IBuilderConfig::getInt8Calibrator () const noexcept

Deprecated in TensorRT 10.1. Superseded by explicit quantization.

Member nvinfer1::IBuilderConfig::getQuantizationFlag (QuantizationFlag flag) const noexcept

Deprecated in TensorRT 10.10. Superseded by explicit quantization.

Member nvinfer1::IBuilderConfig::getQuantizationFlags () const noexcept

Deprecated in TensorRT 10.10. Superseded by explicit quantization.

Member nvinfer1::IBuilderConfig::setAlgorithmSelector (IAlgorithmSelector *selector) noexcept

Deprecated in TensorRT 10.8. Please use editable mode in ITimingCache instead.

Member nvinfer1::IBuilderConfig::setCalibrationProfile (IOptimizationProfile const *profile) noexcept

Deprecated in TensorRT 10.1. Superseded by explicit quantization.

Member nvinfer1::IBuilderConfig::setInt8Calibrator (IInt8Calibrator *calibrator) noexcept

Deprecated in TensorRT 10.1. Superseded by explicit quantization.

Member nvinfer1::IBuilderConfig::setQuantizationFlag (QuantizationFlag flag) noexcept

Deprecated in TensorRT 10.10. Superseded by explicit quantization.

Member nvinfer1::IBuilderConfig::setQuantizationFlags (QuantizationFlags flags) noexcept

Deprecated in TensorRT 10.10. Superseded by explicit quantization.

Member nvinfer1::ICudaEngine::createExecutionContextWithoutDeviceMemory () noexcept

Deprecated in TensorRT 10.0. Superseded by createExecutionContext() with parameter.

Member nvinfer1::ICudaEngine::getDeviceMemorySize () const noexcept

Deprecated in TensorRT 10.1. Superseded by getDeviceMemorySizeV2().

Member nvinfer1::ICudaEngine::getDeviceMemorySizeForProfile (int32_t profileIndex) const noexcept

Deprecated in TensorRT 10.1. Superseded by getDeviceMemorySizeForProfileV2(int32_t).

Member nvinfer1::ICudaEngine::getMinimumWeightStreamingBudget () const noexcept

Deprecated in TensorRT 10.1. The minimum budget is 0 in the V2 APIs.

Member nvinfer1::ICudaEngine::getWeightStreamingBudget () const noexcept

Deprecated in TensorRT 10.1. Superceded by getWeightStreamingBudgetV2().

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

Deprecated in TensorRT 10.0. Implicit batch is no supported since TensorRT 10.0.

Member nvinfer1::ICudaEngine::setWeightStreamingBudget (int64_t gpuMemoryBudget) noexcept

Deprecated in TensorRT 10.1. Superceded by setWeightStreamingBudgetV2().

Member nvinfer1::IExecutionContext::allInputShapesSpecified () const noexcept

Deprecated in TensorRT 10.0. setInputShapeBinding() is removed since TensorRT 10.0.

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

Deprecated in TensorRT 10.1. Superceded by setDeviceMemoryV2().

Class nvinfer1::IInt8Calibrator

Deprecated in TensorRT 10.1. Superseded by explicit quantization.

Member nvinfer1::IInt8Calibrator::getBatchSize () const noexcept=0

Deprecated in TensorRT 10.0. Implicit batch support is removed in TensorRT 10.0.

Member nvinfer1::INetworkDefinition::addDequantize (ITensor &input, ITensor &scale) noexcept

Deprecated in TensorRT 9.0. Superseded by three-argument addDequantize.

Member nvinfer1::INetworkDefinition::addFill (Dims const &dimensions, FillOperation op) noexcept

Deprecated in TensorRT 9.0. Superseded by three-argument addFill.

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

Deprecated in TensorRT 10.8. Superseded by addPluginV3.

Member nvinfer1::INetworkDefinition::addQuantize (ITensor &input, ITensor &scale) noexcept

Deprecated in TensorRT 9.0. Superseded by three-argument addQuantize.

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

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

Member nvinfer1::IPluginRegistry::deregisterCreator (IPluginCreator const &creator) noexcept=0

Deprecated in TensorRT 10.0. Superseded by IPluginRegistry::deregisterCreator(IPluginCreatorInterface const&).

Member nvinfer1::IPluginRegistry::getPluginCreator (AsciiChar const ∗const pluginName, AsciiChar const ∗const pluginVersion, AsciiChar const ∗const pluginNamespace="") noexcept=0

Deprecated in TensorRT 10.0. Superseded by IPluginRegistry::getCreator(AsciiChar const∗ const, AsciiChar const∗ const, AsciiChar const∗ const).

Member nvinfer1::IPluginRegistry::getPluginCreatorList (int32_t ∗const numCreators) const noexcept=0

Deprecated in TensorRT 10.0. Superseded by IPluginRegistry::getAllCreators(int32_t∗ const).

Member nvinfer1::IPluginRegistry::registerCreator (IPluginCreator &creator, AsciiChar const ∗const pluginNamespace) noexcept=0

Deprecated in TensorRT 10.0. Superseded by IPluginRegistry::registerCreator(IPluginCreatorInterface&, AsciiChar const∗ const).

Class nvinfer1::IPluginV2

Deprecated in TensorRT 8.5. Implement IPluginV3 instead.

Class nvinfer1::IPluginV2DynamicExt

Deprecated in TensorRT 10.0. Please implement IPluginV3 instead.

Class nvinfer1::IPluginV2Ext

Deprecated in TensorRT 8.5. Implement IPluginV3 instead.

Member nvinfer1::IPluginV2Ext::canBroadcastInputAcrossBatch (int32_t inputIndex) const noexcept=0

Deprecated in TensorRT 10.0. Implicit batch support is removed in TensorRT 10.0.

Member nvinfer1::IPluginV2Ext::isOutputBroadcastAcrossBatch (int32_t outputIndex, bool const ∗inputIsBroadcasted, int32_t nbInputs) const noexcept=0

Deprecated in TensorRT 10.0. Implicit batch support is removed in TensorRT 10.0.

Class nvinfer1::IPluginV2IOExt

Deprecated in TensorRT 10.0. Implement IPluginV3 instead.

Class nvinfer1::IPluginV2Layer

Deprecated in TensorRT 10.8. Superseded by IPluginV3Layer.

Member nvinfer1::IRefitter::getDynamicRangeMax (char const ∗tensorName) const noexcept

Deprecated in TensorRT 10.1. Superseded by explicit quantization.

Member nvinfer1::IRefitter::getDynamicRangeMin (char const ∗tensorName) const noexcept

Deprecated in TensorRT 10.1. Superseded by explicit quantization.

Member nvinfer1::IRefitter::getTensorsWithDynamicRange (int32_t size, char const **tensorNames) const noexcept

Deprecated in TensorRT 10.1. Superseded by explicit quantization.

Member nvinfer1::IRefitter::setDynamicRange (char const *tensorName, float min, float max) noexcept

Deprecated in TensorRT 10.1. Superseded by explicit quantization.

Member nvinfer1::IRuntime::deserializeCudaEngine (IStreamReader &streamReader)

Deprecated in TensorRT 10.7. Superseded by deserializeCudaEngine that takes an IStreamReaderV2 instead of IStreamReader.

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

Deprecated in TensorRT 10.1. Superseded by explicit quantization.

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

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

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

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

Member nvinfer1::ITensor::setDynamicRange (float min, float max) noexcept

Deprecated in TensorRT 10.1. Superseded by explicit quantization.

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

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

Member nvinfer1::kCUBLAS

Deprecated in TensorRT 10.0.

Member nvinfer1::kCUBLAS_LT

Deprecated in TensorRT 9.0.

Member nvinfer1::kCUDNN

Deprecated in TensorRT 10.0.

Member nvinfer1::kDIRECT_IO

Deprecated in TensorRT 10.7. Unneeded API.

Member nvinfer1::kPROFILE_SHARING_0806

Deprecated in TensorRT 10.0. The default value for this flag is on and can not be changed.

Member nvinfer1::kREJECT_EMPTY_ALGORITHMS

Deprecated in TensorRT 10.10. Unneeded API due to IAlgorithmSelector deprecation.

Member nvinfer1::kWEIGHTLESS

Deprecated in TensorRT 10.0. Superseded by kSTRIP_PLAN.

Member nvinfer1::plugin::CodeTypeSSD

Deprecated in TensorRT 10.0. DetectionOutput plugin is deprecated.

Class nvinfer1::plugin::DetectionOutputParameters

Deprecated in TensorRT 10.0. DetectionOutput plugin is deprecated.

Class nvinfer1::plugin::NMSParameters

Deprecated in TensorRT 10.0. BatchedNMSPlugin plugin is deprecated.

Member nvinfer1::PluginCreatorVersion

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

Member nvinfer1::QuantizationFlag

Deprecated in TensorRT 10.1. Superseded by explicit quantization.

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

Deprecated in TensorRT 10.0. Superseded by allocateAsync

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

Deprecated in TensorRT 10.0. Superseded by deallocateAsync

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

Deprecated in TensorRT 10.0. Superseded by allocateAsync

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

Deprecated in TensorRT 10.0. Superseded by deallocateAsync

Member nvinfer1::v_1_0::IOutputAllocator::reallocateOutput (char const *tensorName, void *currentMemory, uint64_t size, uint64_t alignment) noexcept

Deprecated in TensorRT 10.0. Superseded by reallocateOutputAsync with cudaStream_t argument

Class PluginVersion

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

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

nvinfer1
    The TensorRT API version 1 namespace 27
nvinfer1::anonymous_namespace{NvInfer.h} 85
nvinfer1::anonymous_namespace{NvInferRuntime.h} 85
nvinfer1::impl 86
nvinfer1::plugin 87
nvinfer1::safe 88
nvinfer1::serialize 88
nvinfer1::v_1_0
    Forward declare IErrorRecorder for use in other interfaces 89
nvinfer1::v_2_0 90
nvonnxparser
    The TensorRT ONNX parser API namespace 90
nvonnxparser::anonymous_namespace{NvOnnxParser.h} 93

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

nvinfer1::plugin::DetectionOutputParameters . . . 95
Dims . . . 98
nvinfer1::Dims64 . . . 103
nvinfer1::Dims2 . . . 99
nvinfer1::Dims3 . . . 100
nvinfer1::Dims4 . . . 101
nvinfer1::DimsHW . . . 105
nvinfer1::DimsExprs . . . 104
nvinfer1::DynamicPluginTensorDesc . . . 107
nvinfer1::impl::EnumMaxImpl< T > . . . 109
nvinfer1::impl::EnumMaxImpl< ActivationType > . . . 109
nvinfer1::impl::EnumMaxImpl< AllocatorFlag > . . . 110
nvinfer1::impl::EnumMaxImpl< APILanguage > . . . 111
nvinfer1::impl::EnumMaxImpl< CumulativeOperation > . . . 111
nvinfer1::impl::EnumMaxImpl< DataType > . . . 112
nvinfer1::impl::EnumMaxImpl< ElementWiseOperation > . . . 113
nvinfer1::impl::EnumMaxImpl< EngineCapability > . . . 114
nvinfer1::impl::EnumMaxImpl< ErrorCode > . . . 114
nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel > . . . 115
nvinfer1::impl::EnumMaxImpl< ILogger::Severity > . . . 116
nvinfer1::impl::EnumMaxImpl< InterpolationMode > . . . 117
nvinfer1::impl::EnumMaxImpl< PaddingMode > . . . 117
nvinfer1::impl::EnumMaxImpl< PoolingType > . . . 118
nvinfer1::impl::EnumMaxImpl< PreviewFeature > . . . 119
nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation > . . . 119
nvinfer1::impl::EnumMaxImpl< ResizeRoundMode > . . . 120
nvinfer1::impl::EnumMaxImpl< ResizeSelector > . . . 121
nvinfer1::impl::EnumMaxImpl< RuntimePlatform > . . . 122
nvinfer1::impl::EnumMaxImpl< TensorFormat > . . . 122
nvinfer1::impl::EnumMaxImpl< TensorIOMode > . . . 123
nvinfer1::impl::EnumMaxImpl< TensorLocation > . . . 124

nvinfer1::impl::EnumMaxImpl< TilingOptimizationLevel > . . . 124
nvinfer1::plugin::GridAnchorParameters . . . 125
IAlgorithmSelector . . . 139
IDebugListener . . . 231
IErrorRecorder . . . 263
IGpuAllocator . . . 314
IGpuAsyncAllocator . . . 320
IInt8EntropyCalibrator . . . 344
IInt8EntropyCalibrator2 . . . 346
IInt8LegacyCalibrator . . . 347
IInt8MinMaxCalibrator . . . 351
nvinfer1::ILogger . . . 365
nvinfer1::ILoggerFinder . . . 368
nvinfer1::INoCopy . . . 435
  nvinfer1::IAlgorithm . . . 131
  nvinfer1::IAlgorithmContext . . . 134
  nvinfer1::IAlgorithmIOInfo . . . 136
  nvinfer1::IAlgorithmVariant . . . 141
  nvinfer1::IBuilder . . . 146
  nvinfer1::IBuilderConfig . . . 155
  nvinfer1::ICudaEngine . . . 201
  nvinfer1::IDimensionExpr . . . 247
  nvinfer1::IEngineInspector . . . 258
  nvinfer1::IExecutionContext . . . 271
  nvinfer1::IExprBuilder . . . 297
  nvinfer1::IHostMemory . . . 329
  nvinfer1::IIfConditional . . . 332
  nvinfer1::ILayer . . . 355
    nvinfer1::IActivationLayer . . . 127
    nvinfer1::IAssertionLayer . . . 144
    nvinfer1::ICastLayer . . . 184
    nvinfer1::IConcatenationLayer . . . 186
    nvinfer1::IConstantLayer . . . 189
    nvinfer1::IConvolutionLayer . . . 192
    nvinfer1::ICumulativeLayer . . . 227
    nvinfer1::IDeconvolutionLayer . . . 233
    nvinfer1::IDequantizeLayer . . . 243
    nvinfer1::IDynamicQuantizeLayer . . . 249
    nvinfer1::IEinsumLayer . . . 254
    nvinfer1::IElementWiseLayer . . . 256
    nvinfer1::IFillLayer . . . 300
    nvinfer1::IGatherLayer . . . 309
    nvinfer1::IGridSampleLayer . . . 325
    nvinfer1::IIdentityLayer . . . 331
    nvinfer1::IIfConditionalBoundaryLayer . . . 336
      nvinfer1::IConditionLayer . . . 188
      nvinfer1::IIfConditionalInputLayer . . . 338
      nvinfer1::IIfConditionalOutputLayer . . . 339
    nvinfer1::ILRNLayer . . . 377
    nvinfer1::ILoopBoundaryLayer . . . 372
      nvinfer1::IIteratorLayer . . . 353
      nvinfer1::ILoopOutputLayer . . . 374
      nvinfer1::IRecurrenceLayer . . . 582
      nvinfer1::ITripLimitLayer . . . 681

nvinfer1::IMatrixMultiplyLayer . . . 381
nvinfer1::INMSLayer . . . 431
nvinfer1::INonZeroLayer . . . 438
nvinfer1::INormalizationLayer . . . 439
nvinfer1::IOneHotLayer . . . 445
nvinfer1::IPaddingLayer . . . 463
nvinfer1::IParametricReLULayer . . . 466
nvinfer1::IPluginV2Layer . . . 542
nvinfer1::IPluginV3Layer . . . 546
nvinfer1::IPoolingLayer . . . 564
nvinfer1::IQuantizeLayer . . . 577
nvinfer1::IRaggedSoftMaxLayer . . . 581
nvinfer1::IReduceLayer . . . 584
nvinfer1::IResizeLayer . . . 601
nvinfer1::IReverseSequenceLayer . . . 610
nvinfer1::IScaleLayer . . . 623
nvinfer1::IScatterLayer . . . 628
nvinfer1::ISelectLayer . . . 632
nvinfer1::IShapeLayer . . . 636
nvinfer1::IShuffleLayer . . . 638
nvinfer1::ISliceLayer . . . 643
nvinfer1::ISoftMaxLayer . . . 650
nvinfer1::ISqueezeLayer . . . 653
nvinfer1::ITopKLayer . . . 677
nvinfer1::IUnaryLayer . . . 683
nvinfer1::IUnsqueezeLayer . . . 685
nvinfer1::ILoop . . . 369
nvinfer1::INetworkDefinition . . . 383
nvinfer1::IOptimizationProfile . . . 454
nvinfer1::IRefitter . . . 587
nvinfer1::IRuntime . . . 613
nvinfer1::ISerializationConfig . . . 633
nvinfer1::ITensor . . . 660
nvinfer1::ITimingCache . . . 673
INonZero . . . 438
nvinfer1::InterfaceInfo . . . 444
nvonnxparser::IOnnxConfig . . . 447
IOutputAllocator . . . 460
nvonnxparser::IParser . . . 467
nvonnxparser::IParserError . . . 476
nvonnxparser::IParserRefitter . . . 479
IPluginCapability . . . 482
IPluginCreator . . . 483
IPluginCreatorInterface . . . 489
IPluginCreatorV3One . . . 491
nvinfer1::IPluginRegistry . . . 495
IPluginResource . . . 506
nvinfer1::IPluginResourceContext . . . 509
nvinfer1::IPluginV2 . . . 512
nvinfer1::IPluginV2Ext . . . 531
nvinfer1::IPluginV2DynamicExt . . . 524
nvinfer1::IPluginV2IOExt . . . 538
IPluginV3 . . . 543
IPluginV3OneBuild . . . 547

IPluginV3OneBuildV2 . . . 557
IPluginV3OneCore . . . 557
IPluginV3OneRuntime . . . 560
IProfiler . . . 572
nvinfer1::v_1_0::IProfiler . . . 572
IProgressMonitor . . . 573
IStreamReader . . . 654
IStreamReaderV2 . . . 657
nvinfer1::IVersionedInterface . . . 687
  nvinfer1::IInt8Calibrator . . . 340
    nvinfer1::v_1_0::IInt8EntropyCalibrator . . . 344
    nvinfer1::v_1_0::IInt8EntropyCalibrator2 . . . 346
    nvinfer1::v_1_0::IInt8LegacyCalibrator . . . 348
    nvinfer1::v_1_0::IInt8MinMaxCalibrator . . . 351
  nvinfer1::v_1_0::IAlgorithmSelector . . . 139
  nvinfer1::v_1_0::IDebugListener . . . 232
  nvinfer1::v_1_0::IErrorRecorder . . . 264
  nvinfer1::v_1_0::IGpuAllocator . . . 315
    nvinfer1::v_1_0::IGpuAsyncAllocator . . . 321
  nvinfer1::v_1_0::IOutputAllocator . . . 460
  nvinfer1::v_1_0::IPluginCapability . . . 483
    nvinfer1::v_1_0::IPluginV3OneBuild . . . 548
      nvinfer1::v_2_0::IPluginV3OneBuild . . . 555
    nvinfer1::v_1_0::IPluginV3OneCore . . . 558
    nvinfer1::v_1_0::IPluginV3OneRuntime . . . 560
  nvinfer1::v_1_0::IPluginCreatorInterface . . . 489
    nvinfer1::v_1_0::IPluginCreator . . . 484
    nvinfer1::v_1_0::IPluginCreatorV3One . . . 491
  nvinfer1::v_1_0::IPluginResource . . . 506
  nvinfer1::v_1_0::IPluginV3 . . . 544
  nvinfer1::v_1_0::IProgressMonitor . . . 574
  nvinfer1::v_1_0::IStreamReader . . . 655
  nvinfer1::v_1_0::IStreamReaderV2 . . . 658
nvinfer1::plugin::NMSParameters . . . 689
nvinfer1::Permutation . . . 691
nvinfer1::PluginField . . . 692
nvinfer1::PluginFieldCollection . . . 694
nvinfer1::PluginRegistrar< T > . . . 695
nvinfer1::PluginTensorDesc . . . 696
PluginVersion . . . 697
nvinfer1::plugin::PriorBoxParameters . . . 697
nvinfer1::plugin::RegionParameters . . . 701
nvinfer1::plugin::RPROIParams . . . 702
nvinfer1::plugin::softmaxTree . . . 705
nvinfer1::v_1_0::TimingCacheKey . . . 707
nvinfer1::v_1_0::TimingCacheValue . . . 708
Value . . . 709
nvinfer1::Weights . . . 709

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 . . . 95

Dims
: Structure to define the dimensions of a tensor . . . 98

nvinfer1::Dims2
: Descriptor for two-dimensional data . . . 99

nvinfer1::Dims3
: Descriptor for three-dimensional data . . . 100

nvinfer1::Dims4
: Descriptor for four-dimensional data . . . 101

nvinfer1::Dims64 . . . 103

nvinfer1::DimsExprs
: Analog of class Dims with expressions instead of constants for the dimensions . . . 104

nvinfer1::DimsHW
: Descriptor for two-dimensional spatial data . . . 105

nvinfer1::DynamicPluginTensorDesc
: Summarizes tensors that a plugin might see for an input or output . . . 107

nvinfer1::impl::EnumMaxImpl< T >
: Declaration of EnumMaxImpl struct to store maximum number of elements in an enumeration type 109

nvinfer1::impl::EnumMaxImpl< ActivationType > . . . 109

nvinfer1::impl::EnumMaxImpl< AllocatorFlag >
: Maximum number of elements in AllocatorFlag enum . . . 110

nvinfer1::impl::EnumMaxImpl< APILanguage >
: Maximum number of elements in APILanguage enum . . . 111

nvinfer1::impl::EnumMaxImpl< CumulativeOperation >
: Maximum number of elements in CumulativeOperation enum . . . 111

nvinfer1::impl::EnumMaxImpl< DataType >
: Maximum number of elements in DataType enum . . . 112

nvinfer1::impl::EnumMaxImpl< ElementWiseOperation > . . . 113

nvinfer1::impl::EnumMaxImpl< EngineCapability >
: Maximum number of elements in EngineCapability enum . . . 114

nvinfer1::impl::EnumMaxImpl< ErrorCode >
Maximum number of elements in ErrorCode enum . . . 114
nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel > . . . 115
nvinfer1::impl::EnumMaxImpl< ILogger::Severity >
Maximum number of elements in ILogger::Severity enum . . . 116
nvinfer1::impl::EnumMaxImpl< InterpolationMode > . . . 117
nvinfer1::impl::EnumMaxImpl< PaddingMode > . . . 117
nvinfer1::impl::EnumMaxImpl< PoolingType > . . . 118
nvinfer1::impl::EnumMaxImpl< PreviewFeature > . . . 119
nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation > . . . 119
nvinfer1::impl::EnumMaxImpl< ResizeRoundMode > . . . 120
nvinfer1::impl::EnumMaxImpl< ResizeSelector > . . . 121
nvinfer1::impl::EnumMaxImpl< RuntimePlatform > . . . 122
nvinfer1::impl::EnumMaxImpl< TensorFormat >
Maximum number of elements in TensorFormat enum . . . 122
nvinfer1::impl::EnumMaxImpl< TensorIOMode >
Maximum number of elements in TensorIOMode enum . . . 123
nvinfer1::impl::EnumMaxImpl< TensorLocation >
Maximum number of elements in TensorLocation enum . . . 124
nvinfer1::impl::EnumMaxImpl< TilingOptimizationLevel > . . . 124
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 . . . 125
nvinfer1::IActivationLayer
An Activation layer in a network definition . . . 127
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() . . . 131
nvinfer1::IAlgorithmContext
Describes the context and requirements, that could be fulfilled by one or more instances of IAlgorithm . . . 134
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() . . . 136
IAlgorithmSelector
Interface implemented by application for selecting and reporting algorithms of a layer provided by the builder . . . 139
nvinfer1::v_1_0::IAlgorithmSelector . . . 139
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() . . . 141
nvinfer1::IAssertionLayer
An assertion layer in a network . . . 144
nvinfer1::IBuilder
Builds an engine from a network definition . . . 146
nvinfer1::IBuilderConfig
Holds properties for configuring a builder to produce an engine . . . 155
nvinfer1::ICastLayer
A cast layer in a network . . . 184

nvinfer1::IConcatenationLayer
A concatenation layer in a network definition . . . 186
nvinfer1::IConditionLayer
This layer represents a condition input to an IIfConditional . . . 188
nvinfer1::IConstantLayer
Layer that represents a constant value . . . 189
nvinfer1::IConvolutionLayer
A convolution layer in a network definition . . . 192
nvinfer1::ICudaEngine
An engine for executing inference on a built network, with functionally unsafe features . . . 201
nvinfer1::ICumulativeLayer
Layer that represents a cumulative operation across a tensor . . . 227
IDebugListener
User-implemented callback for notification when value of a debug tensor is updated . . . 231
nvinfer1::v_1_0::IDebugListener . . . 232
nvinfer1::IDeconvolutionLayer
A deconvolution layer in a network definition . . . 233
nvinfer1::IDequantizeLayer
A Dequantize layer in a network definition . . . 243
nvinfer1::IDimensionExpr
An IDimensionExpr represents an integer expression constructed from constants, input dimensions, and binary operations. These expressions are can be used in overrides of IPluginV2DynamicExt::getOutputDimensions or IPluginV3OneBuild::getOutputShapes() to define output dimensions in terms of input dimensions . . . 247
nvinfer1::IDynamicQuantizeLayer
A network layer to perform dynamic quantization . . . 249
nvinfer1::IEinsumLayer
An Einsum layer in a network . . . 254
nvinfer1::IElementWiseLayer
A elementwise layer in a network definition . . . 256
nvinfer1::IEngineInspector
An engine inspector which prints out the layer information of an engine or an execution context . 258
IErrorRecorder
Reference counted application-implemented error reporting interface for TensorRT objects . . . 263
nvinfer1::v_1_0::IErrorRecorder . . . 264
nvinfer1::IExecutionContext
Context for executing inference using an engine, with functionally unsafe features . . . 271
nvinfer1::IExprBuilder
Object for constructing IDimensionExpr . . . 297
nvinfer1::IFillLayer
Generate a tensor according to a specified mode . . . 300
nvinfer1::IGatherLayer
A Gather layer in a network definition. Supports several kinds of gathering . . . 309
IGpuAllocator
Application-implemented class for controlling allocation on the GPU . . . 314
nvinfer1::v_1_0::IGpuAllocator . . . 315
IGpuAsyncAllocator
Application-implemented class for controlling asynchronous (stream ordered) memory allocation on the GPU . . . 320
nvinfer1::v_1_0::IGpuAsyncAllocator . . . 321
nvinfer1::IGridSampleLayer
A GridSample layer in a network definition . . . 325
nvinfer1::IHostMemory
Class to handle library allocated memory that is accessible to the user . . . 329

nvinfer1::IIdentityLayer
A layer that represents the identity function . . . 331
nvinfer1::IIfConditional
Helper for constructing conditionally-executed subgraphs . . . 332
nvinfer1::IIfConditionalBoundaryLayer
This is a base class for Conditional boundary layers . . . 336
nvinfer1::IIfConditionalInputLayer
This layer represents an input to an IIfConditional . . . 338
nvinfer1::IIfConditionalOutputLayer
This layer represents an output of an IIfConditional . . . 339
nvinfer1::IInt8Calibrator
Application-implemented interface for calibration . . . 340
IInt8EntropyCalibrator
Entropy calibrator . . . 344
nvinfer1::v_1_0::IInt8EntropyCalibrator . . . 344
IInt8EntropyCalibrator2
Entropy calibrator 2 . . . 346
nvinfer1::v_1_0::IInt8EntropyCalibrator2 . . . 346
IInt8LegacyCalibrator
Legacy calibrator . . . 347
nvinfer1::v_1_0::IInt8LegacyCalibrator . . . 348
IInt8MinMaxCalibrator
MinMax Calibrator . . . 351
nvinfer1::v_1_0::IInt8MinMaxCalibrator . . . 351
nvinfer1::IIteratorLayer
A layer to do iterations . . . 353
nvinfer1::ILayer
Base class for all layer classes in a network definition . . . 355
nvinfer1::ILogger
Application-implemented logging interface for the builder, refitter and runtime . . . 365
nvinfer1::ILoggerFinder
A virtual base class to find a logger. Allows a plugin to find an instance of a logger if it needs to emit a log message. A pointer to an instance of this class is passed to a plugin shared library on initialization when that plugin is serialized as part of a version-compatible plan. See the plugin chapter in the developer guide for details . . . 368
nvinfer1::ILoop
Helper for creating a recurrent subgraph . . . 369
nvinfer1::ILoopBoundaryLayer
This is a base class for Loop boundary layers . . . 372
nvinfer1::ILoopOutputLayer
An ILoopOutputLayer is the sole way to get output from a loop . . . 374
nvinfer1::ILRNLayer
A LRN layer in a network definition . . . 377
nvinfer1::IMatrixMultiplyLayer
Layer that represents a Matrix Multiplication . . . 381
nvinfer1::INetworkDefinition
A network definition for input to the builder . . . 383
nvinfer1::INMSLayer
A non-maximum suppression layer in a network definition . . . 431
nvinfer1::INoCopy
Forward declaration of IEngineInspector for use by other interfaces . . . 435
INonZero
A NonZero layer in a network . . . 438
nvinfer1::INonZeroLayer . . . 438

nvinfer1::INormalizationLayer
A normalization layer in a network definition . . . 439
nvinfer1::InterfaceInfo
Version information associated with a TRT interface . . . 444
nvinfer1::IOneHotLayer
A OneHot layer in a network definition . . . 445
nvonnxparser::IOnnxConfig
Configuration Manager Class . . . 447
nvinfer1::IOptimizationProfile
Optimization profile for dynamic input dimensions and shape tensors . . . 454
IOutputAllocator
Callback from ExecutionContext::enqueueV3() . . . 460
nvinfer1::v_1_0::IOutputAllocator . . . 460
nvinfer1::IPaddingLayer
Layer that represents a padding operation . . . 463
nvinfer1::IParametricReLULayer
Layer that represents a parametric ReLU operation . . . 466
nvonnxparser::IParser
Object for parsing ONNX models into a TensorRT network definition . . . 467
nvonnxparser::IParserError
Object containing information about an error . . . 476
nvonnxparser::IParserRefitter
An interface designed to refit weights from an ONNX model . . . 479
IPluginCapability
Base class for plugin capability interfaces . . . 482
nvinfer1::v_1_0::IPluginCapability . . . 483
IPluginCreator
Plugin creator class for user implemented layers . . . 483
nvinfer1::v_1_0::IPluginCreator . . . 484
IPluginCreatorInterface
Base class for all plugin creator versions . . . 489
nvinfer1::v_1_0::IPluginCreatorInterface . . . 489
IPluginCreatorV3One
A plugin creator class capable of producing IPluginV3 objects . . . 491
nvinfer1::v_1_0::IPluginCreatorV3One . . . 491
nvinfer1::IPluginRegistry
Single registration point for all plugins in an application. It is used to find plugin implementations during engine deserialization. Internally, the plugin registry is considered to be a singleton so all plugins in an application are part of the same global registry. Note that the plugin registry is only supported for plugins of type IPluginV2 and should also have a corresponding IPluginCreator implementation . . . 495
IPluginResource
Interface for plugins to define custom resources that could be shared through the plugin registry . 506
nvinfer1::v_1_0::IPluginResource . . . 506
nvinfer1::IPluginResourceContext
Interface for plugins to access per context resources provided by TensorRT . . . 509
nvinfer1::IPluginV2
Plugin class for user-implemented layers . . . 512
nvinfer1::IPluginV2DynamicExt
Similar to IPluginV2Ext, but with support for dynamic shapes . . . 524
nvinfer1::IPluginV2Ext
Plugin class for user-implemented layers . . . 531
nvinfer1::IPluginV2IOExt
Plugin class for user-implemented layers . . . 538

nvinfer1::IPluginV2Layer
Layer type for pluginV2 . . . 542
IPluginV3
Plugin class for the V3 generation of user-implemented layers . . . 543
nvinfer1::v_1_0::IPluginV3 . . . 544
nvinfer1::IPluginV3Layer
Layer type for V3 plugins . . . 546
IPluginV3OneBuild
A plugin capability interface that enables the build capability (PluginCapabilityType::kBUILD). Exposes methods that allow the expression of the build time properties and behavior of a plugin . . . 547
nvinfer1::v_1_0::IPluginV3OneBuild . . . 548
nvinfer1::v_2_0::IPluginV3OneBuild . . . 555
IPluginV3OneBuildV2
A plugin capability interface that extends IPluginV3OneBuild by providing I/O aliasing functionality . . . 557
IPluginV3OneCore
A plugin capability interface that enables the core capability (PluginCapabilityType::kCORE) . . . 557
nvinfer1::v_1_0::IPluginV3OneCore . . . 558
IPluginV3OneRuntime
A plugin capability interface that enables the runtime capability (PluginCapabilityType::k↩ RUNTIME). Exposes methods that allow the expression of the runtime properties and behavior of a plugin . . . 560
nvinfer1::v_1_0::IPluginV3OneRuntime . . . 560
nvinfer1::IPoolingLayer
A Pooling layer in a network definition . . . 564
IProfiler
Application-implemented interface for profiling . . . 572
nvinfer1::v_1_0::IProfiler . . . 572
IProgressMonitor
Application-implemented progress reporting interface for TensorRT . . . 573
nvinfer1::v_1_0::IProgressMonitor . . . 574
nvinfer1::IQuantizeLayer
A Quantize layer in a network definition . . . 577
nvinfer1::IRaggedSoftMaxLayer
A RaggedSoftmax layer in a network definition . . . 581
nvinfer1::IRecurrenceLayer
A recurrence layer in a network definition . . . 582
nvinfer1::IReduceLayer
Layer that represents a reduction across a non-bool tensor . . . 584
nvinfer1::IRefitter
Updates weights in an engine . . . 587
nvinfer1::IResizeLayer
A resize layer in a network definition . . . 601
nvinfer1::IReverseSequenceLayer
A ReverseSequence layer in a network definition . . . 610
nvinfer1::IRuntime
Allows a serialized functionally unsafe engine to be deserialized . . . 613
nvinfer1::IScaleLayer
A Scale layer in a network definition . . . 623
nvinfer1::IScatterLayer
A scatter layer in a network definition. Supports several kinds of scattering . . . 628
nvinfer1::ISelectLayer
Select elements from two data tensors based on a condition tensor . . . 632

nvinfer1::ISerializationConfig
Holds properties for configuring an engine to serialize the binary . . . 633
nvinfer1::IShapeLayer
Layer type for getting shape of a tensor . . . 636
nvinfer1::IShuffleLayer
Layer type for shuffling data . . . 638
nvinfer1::ISliceLayer
Slices an input tensor into an output tensor based on the offset and strides . . . 643
nvinfer1::ISoftMaxLayer
A Softmax layer in a network definition . . . 650
nvinfer1::ISqueezeLayer
Layer that represents a squeeze operation, removing unit dimensions of the input tensor on a set of axes . . . 653
IStreamReader
Application-implemented class for reading data in a stream-based manner . . . 654
nvinfer1::v_1_0::IStreamReader . . . 655
IStreamReaderV2
Application-implemented class for reading data in a stream-based manner asynchronously. Intended for use with the GDS API for optimizing load times . . . 657
nvinfer1::v_1_0::IStreamReaderV2 . . . 658
nvinfer1::ITensor
A tensor in a network definition . . . 660
nvinfer1::ITimingCache
Class to handle tactic timing info collected from builder . . . 673
nvinfer1::ITopKLayer
Layer that represents a TopK reduction . . . 677
nvinfer1::ITripLimitLayer
A layer that represents a trip-count limiter . . . 681
nvinfer1::IUnaryLayer
Layer that represents an unary operation . . . 683
nvinfer1::IUnsqueezeLayer
Layer that represents an unsqueeze operation, which reshapes the input tensor by inserting unit-length dimensions at specified axes of the output . . . 685
nvinfer1::IVersionedInterface
An Interface class for version control . . . 687
nvinfer1::plugin::NMSParameters
The NMSParameters are used by the BatchedNMSPlugin for performing the non_max_suppression operation over boxes for object detection networks . . . 689
nvinfer1::Permutation
Represents a permutation of dimensions . . . 691
nvinfer1::PluginField
Structure containing plugin attribute field names and associated data This information can be parsed to decode necessary plugin metadata . . . 692
nvinfer1::PluginFieldCollection
Plugin field collection struct . . . 694
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 . . . 695
nvinfer1::PluginTensorDesc
Fields that a plugin might see for an input or output . . . 696
PluginVersion
Definition of plugin versions . . . 697

nvinfer1::plugin::PriorBoxParameters
The PriorBox plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions (H x W) . . . 697
nvinfer1::plugin::RegionParameters
The Region plugin layer performs region proposal calculation . . . 701
nvinfer1::plugin::RPROIParams
RPROIParams is used to create the RPROIPlugin instance . . . 702
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 . . . 705
nvinfer1::v_1_0::TimingCacheKey
The key to retrieve timing cache entries . . . 707
nvinfer1::v_1_0::TimingCacheValue . . . 708
Value
The values in the cache entry . . . 709
nvinfer1::Weights
An array of weights used as a layer parameter . . . 709

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

NvInfer.h . . . 711
NvInferImpl.h . . . 766
NvInferLegacyDims.h . . . 782
NvInferPlugin.h . . . 785
NvInferPluginBase.h . . . 786
NvInferPluginUtils.h . . . 789
NvInferRuntime.h . . . 792
NvInferRuntimeBase.h . . . 821
NvInferRuntimeCommon.h . . . 828
NvInferRuntimePlugin.h . . . 830
NvInferSerialize.h . . . 835
NvInferVersion.h . . . 836
NvOnnxConfig.h . . . 839
NvOnnxParser.h . . . 841

Chapter 8

Namespace Documentation

8.1 nvinfer1 Namespace Reference

The TensorRT API version 1 namespace.

Namespaces

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

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

Classes

- class Dims2

  Descriptor for two-dimensional data.
- class Dims3

  Descriptor for three-dimensional data.
- class Dims4

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

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

  Descriptor for two-dimensional spatial data.
- struct DynamicPluginTensorDesc

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

- class IActivationLayer

  An Activation layer in a network definition.

- class 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 IAlgorithmVariant

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

- class IAssertionLayer

  An assertion layer in a network.

- class IBuilder

  Builds an engine from a network definition.

- class IBuilderConfig

  Holds properties for configuring a builder to produce an engine.

- class ICastLayer

  A cast layer in a network.

- class IConcatenationLayer

  A concatenation layer in a network definition.

- class IConditionLayer

  This layer represents a condition input to an IIfConditional.

- class IConstantLayer

  Layer that represents a constant value.

- class IConvolutionLayer

  A convolution layer in a network definition.

- class ICudaEngine

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

- class ICumulativeLayer

  Layer that represents a cumulative operation across a tensor.

- class IDeconvolutionLayer

  A deconvolution layer in a network definition.

- class IDequantizeLayer

  A Dequantize layer in a network definition.

- class IDimensionExpr

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

- class IDynamicQuantizeLayer

  A network layer to perform dynamic quantization.

- class IEinsumLayer

  An Einsum layer in a network.

- class IElementWiseLayer

  A elementwise layer in a network definition.
- class IEngineInspector

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

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

  Object for constructing IDimensionExpr.
- class IFillLayer

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

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

  A GridSample layer in a network definition.
- class IHostMemory

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

  A layer that represents the identity function.
- class IIfConditional

  Helper for constructing conditionally-executed subgraphs.
- class IIfConditionalBoundaryLayer

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

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

  This layer represents an output of an IIfConditional.
- class IInt8Calibrator

  Application-implemented interface for calibration.
- class IIteratorLayer

  A layer to do iterations.
- class ILayer

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

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

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

  Helper for creating a recurrent subgraph.
- class ILoopBoundaryLayer

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

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

  A LRN layer in a network definition.
- class IMatrixMultiplyLayer

Layer that represents a Matrix Multiplication.

- class INetworkDefinition

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

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

  Forward declaration of IEngineInspector for use by other interfaces.
- class INonZeroLayer
- class INormalizationLayer

  A normalization layer in a network definition.
- class InterfaceInfo

  Version information associated with a TRT interface.
- class IOneHotLayer

  A OneHot layer in a network definition.
- class IOptimizationProfile

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

  Layer that represents a padding operation.
- class IParametricReLULayer

  Layer that represents a parametric ReLU operation.
- class IPluginRegistry

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

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

  Plugin class for user-implemented layers.
- class IPluginV2DynamicExt

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

  Plugin class for user-implemented layers.
- class IPluginV2IOExt

  Plugin class for user-implemented layers.
- class IPluginV2Layer

  Layer type for pluginV2.
- class IPluginV3Layer

  Layer type for V3 plugins.
- class IPoolingLayer

  A Pooling layer in a network definition.
- class IQuantizeLayer

  A Quantize layer in a network definition.
- class IRaggedSoftMaxLayer

  A RaggedSoftmax layer in a network definition.
- class IRecurrenceLayer

  A recurrence layer in a network definition.
- class IReduceLayer

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

- class IRefitter

  Updates weights in an engine.
- class IResizeLayer

  A resize layer in a network definition.
- class IReverseSequenceLayer

  A ReverseSequence layer in a network definition.
- class 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

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

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

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

  Layer type for shuffling data.
- class ISliceLayer

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

  A Softmax layer in a network definition.
- class ISqueezeLayer

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

  A tensor in a network definition.
- class ITimingCache

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

  Layer that represents a TopK reduction.
- class ITripLimitLayer

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

  Layer that represents an unary operation.
- class IUnsqueezeLayer

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

  An Interface class for version control.
- struct Permutation

  Represents a permutation of dimensions.
- class PluginField

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

- struct PluginFieldCollection

  Plugin field collection struct.
- class PluginRegistrar

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

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

  An array of weights used as a layer parameter.

Typedefs

- using TensorFormats = uint32_t

  It is capable of representing one or more TensorFormat by binary OR operations, e.g., 1U << TensorFormat::kCHW4 | 1U << TensorFormat::kCHW32.
- using IInt8EntropyCalibrator = v_1_0::IInt8EntropyCalibrator
- using IInt8EntropyCalibrator2 = v_1_0::IInt8EntropyCalibrator2
- using IInt8MinMaxCalibrator = v_1_0::IInt8MinMaxCalibrator
- using IInt8LegacyCalibrator = v_1_0::IInt8LegacyCalibrator
- using IAlgorithmSelector = v_1_0::IAlgorithmSelector
- using QuantizationFlags = uint32_t

  Represents one or more QuantizationFlag values using binary OR operations.
- using BuilderFlags = uint32_t

  Represents one or more BuilderFlag values using binary OR operations, e.g., 1U << BuilderFlag::kFP16 | 1U << BuilderFlag::kDEBUG.
- using IProgressMonitor = v_1_0::IProgressMonitor
- using NetworkDefinitionCreationFlags = uint32_t

  Represents one or more NetworkDefinitionCreationFlag flags using binary OR operations. e.g., 1U << NetworkDefinitionCreationFlag::kSTRONGLY_TYPED.
- using IPluginResource = v_1_0::IPluginResource
- using IPluginCreatorInterface = v_1_0::IPluginCreatorInterface
- using IPluginV3 = v_1_0::IPluginV3
- using IPluginCapability = v_1_0::IPluginCapability
- using IStreamReader = v_1_0::IStreamReader
- using IStreamReaderV2 = v_1_0::IStreamReaderV2
- using IPluginV3OneCore = v_1_0::IPluginV3OneCore
- using IPluginV3OneBuild = v_1_0::IPluginV3OneBuild
- using IPluginV3OneRuntime = v_1_0::IPluginV3OneRuntime
- using IPluginV3OneBuildV2 = v_2_0::IPluginV3OneBuild
- using IProfiler = v_1_0::IProfiler
- using TempfileControlFlags = uint32_t

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

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

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

- using IDebugListener = v_1_0::IDebugListener
- using IGpuAsyncAllocator = v_1_0::IGpuAsyncAllocator
- using IPluginCreatorV3One = v_1_0::IPluginCreatorV3One
- using char_t = char

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

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

Enumerations

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

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

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

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

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

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

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

- enum class GatherMode : int32_t { kDEFAULT = 0 , kELEMENT = 1 , kND = 2 }

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

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

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

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

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

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

  Enumerates various modes of interpolation.
- enum class ResizeCoordinateTransformation : int32_t { kALIGN_CORNERS = 0 , kASYMMETRIC = 1 , kHALF_PIXEL = 2 }

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

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

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

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

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

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

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

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

  Enumerates the cumulative operations that may be performed by a Cumulative layer.
- enum class 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 RuntimePlatform : int32_t { kSAME_AS_BUILD = 0 , kWINDOWS_AMD64 = 1 }

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

- enum class BuilderFlag : int32_t {
  kFP16 = 0 , kINT8 = 1 , kDEBUG = 2 , kGPU_FALLBACK = 3 ,
  kREFIT = 4 , kDISABLE_TIMING_CACHE = 5 , kTF32 = 6 , kSPARSE_WEIGHTS = 7 ,
  kSAFETY_SCOPE = 8 , kOBEY_PRECISION_CONSTRAINTS = 9 , kPREFER_PRECISION_CONSTRAINTS = 10 , kDIRECT_IO = 11 ,
  kREJECT_EMPTY_ALGORITHMS = 12 , kVERSION_COMPATIBLE = 13 , kEXCLUDE_LEAN_RUNTIME = 14 , kFP8 = 15 ,
  kERROR_ON_TIMING_CACHE_MISS = 16 , kBF16 = 17 , kDISABLE_COMPILATION_CACHE = 18 , kSTRIP_PLAN = 19 ,
  kWEIGHTLESS = kSTRIP_PLAN , kREFIT_IDENTICAL = 20 , kWEIGHT_STREAMING = 21 , kINT4 = 22 ,
  kREFIT_INDIVIDUAL = 23 , kSTRICT_NANS = 24 , kMONITOR_MEMORY = 25 , kFP4 = 26 ,
  kEDITABLE_TIMING_CACHE = 27 }

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

- enum class MemoryPoolType : int32_t {
  kWORKSPACE = 0 , kDLA_MANAGED_SRAM = 1 , kDLA_LOCAL_DRAM = 2 , kDLA_GLOBAL_DRAM = 3 ,
  kTACTIC_DRAM = 4 , kTACTIC_SHARED_MEMORY = 5 }

  The type for memory pools used by TensorRT.

- enum class PreviewFeature : int32_t { kPROFILE_SHARING_0806 = 0 , kALIASED_PLUGIN_IO_10_03 = 1 , kRUNTIME_ACTIVATION_RESIZE_10_10 = 2 }

  Define preview features.

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

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

- enum class TilingOptimizationLevel : int32_t { kNONE = 0 , kFAST = 1 , kMODERATE = 2 , kFULL = 3 }

  Define the optimization levels for Tiling.

- enum class NetworkDefinitionCreationFlag : int32_t { kEXPLICIT_BATCH = 0 , kSTRONGLY_TYPED = 1 }

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

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

  The possible field types for custom layer.

- enum class TensorRTPhase : int32_t { kBUILD = 0 , kRUNTIME = 1 }

  Indicates a phase of operation of TensorRT.

- enum class PluginCapabilityType : int32_t { kCORE = 0 , kBUILD = 1 , kRUNTIME = 2 }

  Enumerates the different capability types a IPluginV3 object may have.

- enum class EngineCapability : int32_t { kSTANDARD = 0 , kSAFETY = 1 , kDLA_STANDALONE = 2 }

  List of supported engine capability flows.

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

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

- enum class TensorLocation : int32_t { kDEVICE = 0 , kHOST = 1 }

  The location for tensor data storage, device or host.

- enum class SeekPosition : int32_t { kSET = 0 , kCUR = 1 , kEND = 2 }

  Controls the seek mode of IStreamReaderV2.

- enum class WeightsRole : int32_t {
  kKERNEL = 0 , kBIAS = 1 , kSHIFT = 2 , kSCALE = 3 ,
  kCONSTANT = 4 , kANY = 5 }

  How a layer uses particular Weights.

- enum class DeviceType : int32_t { kGPU = 0 , kDLA = 1 }

  The device that this layer/network will execute on.

- enum class TempfileControlFlag : int32_t { kALLOW_IN_MEMORY_FILES = 0 , kALLOW_TEMPORARY_FILES = 1 }

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

- enum class TensorFormat : int32_t {
  kLINEAR = 0 , kCHW2 = 1 , kHWC8 = 2 , kCHW4 = 3 ,
  kCHW16 = 4 , kCHW32 = 5 , kDHWC8 = 6 , kCDHW32 = 7 ,
  kHWC = 8 , kDLA_LINEAR = 9 , kDLA_HWC4 = 10 , kHWC16 = 11 ,
  kDHWC = 12 }

  Format of the input/output tensors.

- enum class AllocatorFlag : int32_t { kRESIZABLE = 0 }

  Allowed type of memory allocation.

- enum class OptProfileSelector : int32_t { kMIN = 0 , kOPT = 1 , kMAX = 2 }

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

- enum class TacticSource : int32_t {
  kCUBLAS = 0 , kCUBLAS_LT = 1 , kCUDNN = 2 , kEDGE_MASK_CONVOLUTIONS = 3 ,
  kJIT_CONVOLUTIONS = 4 }

  List of tactic sources for TensorRT.

- enum class ProfilingVerbosity : int32_t { kLAYER_NAMES_ONLY = 0 , kNONE = 1 , kDETAILED = 2 }

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

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

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

- enum class ExecutionContextAllocationStrategy : int32_t { kSTATIC = 0 , kON_PROFILE_CHANGE = 1 , kUSER_MANAGED = 2 }

  Different memory allocation behaviors for IExecutionContext.

- enum class LayerInformationFormat : int32_t { kONELINE = 0 , kJSON = 1 }

  The format in which the IEngineInspector prints the layer information.

- enum class DataType : int32_t {
  kFLOAT = 0 , kHALF = 1 , kINT8 = 2 , kINT32 = 3 ,
  kBOOL = 4 , kUINT8 = 5 , kFP8 = 6 , kBF16 = 7 ,
  kINT64 = 8 , kINT4 = 9 , kFP4 = 10 }

  The type of weights and tensors.

- enum class APILanguage : int32_t { kCPP = 0 , kPYTHON = 1 }

  Programming language used in the implementation of a TRT interface.

- enum class ErrorCode : int32_t {
  kSUCCESS = 0 , kUNSPECIFIED_ERROR = 1 , kINTERNAL_ERROR = 2 , kINVALID_ARGUMENT = 3 ,
  kINVALID_CONFIG = 4 , kFAILED_ALLOCATION = 5 , kFAILED_INITIALIZATION = 6 , kFAILED_EXECUTION = 7 ,
  kFAILED_COMPUTATION = 8 , kINVALID_STATE = 9 , kUNSUPPORTED_STATE = 10 }

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

  Definition of tensor IO Mode.
- enum class PluginVersion : uint8_t {
  kV2 = 0 , kV2_EXT = 1 , kV2_IOEXT = 2 , kV2_DYNAMICEXT = 3 ,
  kV2_DYNAMICEXT_PYTHON = kPLUGIN_VERSION_PYTHON_BIT | 3 }
- enum class PluginCreatorVersion : int32_t { kV1 = 0 , kV1_PYTHON = kPLUGIN_VERSION_PYTHON_BIT }

  Enum to identify version of the plugin creator.

Functions

- template<> 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< UnaryOperation > () noexcept
- template<> constexpr int32_t EnumMax< ReduceOperation > () noexcept
- template<> constexpr int32_t EnumMax< SampleMode > () noexcept
- template<> constexpr int32_t EnumMax< TopKOperation > () noexcept
- template<> constexpr int32_t EnumMax< MatrixOperation > () noexcept
- template<> constexpr int32_t EnumMax< LoopOutput > () noexcept
- template<> constexpr int32_t EnumMax< TripLimit > () noexcept
- template<> constexpr int32_t EnumMax< FillOperation > () noexcept
- template<> constexpr int32_t EnumMax< ScatterMode > () noexcept
- template<> constexpr int32_t EnumMax< BoundingBoxFormat > () noexcept
- template<> constexpr int32_t EnumMax< CalibrationAlgoType > () noexcept
- template<> constexpr int32_t EnumMax< QuantizationFlag > () noexcept
- template<> constexpr int32_t EnumMax< BuilderFlag > () noexcept
- template<> constexpr int32_t EnumMax< MemoryPoolType > () noexcept
- template<> constexpr int32_t EnumMax< NetworkDefinitionCreationFlag > () noexcept
- nvinfer1::IPluginRegistry * getBuilderPluginRegistry (nvinfer1::EngineCapability capability) noexcept

  Return the plugin registry for building a Standard engine, or nullptr if no registry exists.
- nvinfer1::safe::IPluginRegistry * getBuilderSafePluginRegistry (nvinfer1::EngineCapability capability) noexcept

  Return the plugin registry for building a Safety engine, or nullptr if no registry exists.
- template<> constexpr int32_t EnumMax< DimensionOperation > () noexcept

  Maximum number of elements in DimensionOperation enum.
- template<> constexpr int32_t EnumMax< WeightsRole > () noexcept

  Maximum number of elements in WeightsRole enum.
- template<> constexpr int32_t EnumMax< DeviceType > () noexcept

  Maximum number of elements in DeviceType enum.
- template<> constexpr int32_t EnumMax< TempfileControlFlag > () noexcept

  Maximum number of elements in TempfileControlFlag enum.
- template<> constexpr int32_t EnumMax< OptProfileSelector > () noexcept

Number of different values of OptProfileSelector enum.

- template<> constexpr int32_t EnumMax< TacticSource > () noexcept

  Maximum number of tactic sources in TacticSource enum.
- template<> constexpr int32_t EnumMax< ProfilingVerbosity > () noexcept

  Maximum number of profile verbosity levels in ProfilingVerbosity enum.
- template<> constexpr int32_t EnumMax< SerializationFlag > () noexcept

  Maximum number of serialization flags in SerializationFlag enum.
- template<> constexpr int32_t EnumMax< ExecutionContextAllocationStrategy > () noexcept

  Maximum number of memory allocation strategies in ExecutionContextAllocationStrategy enum.
- template<> constexpr int32_t EnumMax< LayerInformationFormat > () noexcept
- template<typename T >
  constexpr int32_t EnumMax () noexcept

  Maximum number of elements in an enumeration type.

8.1.1 Detailed Description

The TensorRT API version 1 namespace.

8.1.2 Typedef Documentation

8.1.2.1 AllocatorFlags

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

8.1.2.2 AsciiChar

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

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

8.1.2.3 BuilderFlags

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

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

See also

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

8.1.2.4 char_t

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

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

8.1.2.5 Dims

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

Alias for Dims64.

8.1.2.6 IAlgorithmSelector

```
using nvinfer1::IAlgorithmSelector = typedef v_1_0::IAlgorithmSelector
```

8.1.2.7 IDebugListener

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

8.1.2.8 IErrorRecorder

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

8.1.2.9 IGpuAllocator

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

8.1.2.10 IGpuAsyncAllocator

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

8.1.2.11 IInt8EntropyCalibrator

```
using nvinfer1::IInt8EntropyCalibrator = typedef v_1_0::IInt8EntropyCalibrator
```

8.1.2.12 IInt8EntropyCalibrator2

```
using nvinfer1::IInt8EntropyCalibrator2 = typedef v_1_0::IInt8EntropyCalibrator2
```

8.1.2.13 IInt8LegacyCalibrator

```
using nvinfer1::IInt8LegacyCalibrator = typedef v_1_0::IInt8LegacyCalibrator
```

8.1.2.14 IInt8MinMaxCalibrator

```
using nvinfer1::IInt8MinMaxCalibrator = typedef v_1_0::IInt8MinMaxCalibrator
```

8.1.2.15 InterfaceKind

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

8.1.2.16 IOutputAllocator

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

8.1.2.17 IPluginCapability

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

8.1.2.18 IPluginCreator

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

8.1.2.19 IPluginCreatorInterface

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

8.1.2.20 IPluginCreatorV3One

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

8.1.2.21 IPluginResource

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

8.1.2.22 IPluginV3

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

8.1.2.23 IPluginV3OneBuild

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

8.1.2.24 IPluginV3OneBuildV2

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

8.1.2.25 IPluginV3OneCore

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

8.1.2.26 IPluginV3OneRuntime

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

8.1.2.27 IProfiler

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

8.1.2.28 IProgressMonitor

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

8.1.2.29 IStreamReader

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

8.1.2.30 IStreamReaderV2

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

8.1.2.31 NetworkDefinitionCreationFlags

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

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

See also

IBuilder::createNetworkV2

8.1.2.32 PluginFormat

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

PluginFormat is reserved for backward compatibility.

See also

IPluginV2::supportsFormat()

8.1.2.33 QuantizationFlags

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

Represents one or more QuantizationFlag values using binary OR operations.

See also

IBuilderConfig::getQuantizationFlags(), IBuilderConfig::setQuantizationFlags()

8.1.2.34 SerializationFlags

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

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

See also

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

8.1.2.35 TacticSources

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

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

See also

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

8.1.2.36 TempfileControlFlags

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

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

See also

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

8.1.2.37 TensorFormats

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

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

See also

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

8.1.3 Enumeration Type Documentation

8.1.3.1 ActivationType

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

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

Enumerator

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

8.1.3.2 AllocatorFlag

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

Allowed type of memory allocation.

Enumerator

kRESIZABLE	TensorRT may call realloc() on this allocation.

8.1.3.3 APILanguage

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

Programming language used in the implementation of a TRT interface.

Enumerator

kCPP	
kPYTHON	

8.1.3.4 BoundingBoxFormat

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

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

See also

INMSLayer

Enumerator

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

8.1.3.5 BuilderFlag

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

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

See also

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

Enumerator

kFP16	Enable FP16 layer selection, with FP32 fallback.
kINT8	Enable Int8 layer selection, with FP32 fallback with FP16 fallback if kFP16 also specified.
kDEBUG	Enable debugging of layers via synchronizing after every layer.
kGPU_FALLBACK	Enable layers marked to execute on GPU if layer cannot execute on DLA.
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. **Deprecated** Deprecated in TensorRT 10.7. Unneeded API.
kREJECT_EMPTY_ALGORITHMS	Fail if IAlgorithmSelector::selectAlgorithms returns an empty set of algorithms. **Deprecated** Deprecated in TensorRT 10.10. Unneeded API due to IAlgorithmSelector deprecation.
kVERSION_COMPATIBLE	Restrict to lean runtime operators to provide version forward compatibility for the plan. This flag is only supported by NVIDIA Volta and later GPUs. This flag is not supported in NVIDIA Drive(R) products.

Enumerator

kEXCLUDE_LEAN_RUNTIME	Exclude lean runtime from the plan when version forward compatability is enabled. By default, this flag is unset, so the lean runtime will be included in the plan. If BuilderFlag::kVERSION_COMPATIBLE is not set then the value of this flag will be ignored.

kFP8	Enable plugins with FP8 input/output.
	`This flag is not supported when HardwareCompatibilityLevel::kAMPERE_PLUS i`
	`\see HardwareCompatibilityLevel`

kERROR_ON_TIMING_CACHE_MISS	Emit error when a tactic being timed is not present in the timing cache. This flag has an effect only when IBuilderConfig has an associated ITimingCache.
kBF16	Enable DataType::kBF16 layer selection, with FP32 fallback. This flag is only supported by NVIDIA Ampere and later GPUs.
kDISABLE_COMPILATION_CACHE	Disable caching of JIT-compilation results during engine build. By default, JIT-compiled code will be serialized as part of the timing cache, which may significantly increase the cache size. Setting this flag prevents the code from being serialized. This flag has an effect only when BuilderFlag::DISABLE_TIMING_CACHE is not set.
kSTRIP_PLAN	Strip the refittable weights from the engine plan file.
kWEIGHTLESS	**Deprecated** Deprecated in TensorRT 10.0. Superseded by kSTRIP_PLAN.
kREFIT_IDENTICAL	Create a refittable engine under the assumption that the refit weights will be identical to those provided at build time. The resulting engine will have the same performance as a non-refittable one. All refittable weights can be refitted through the refit API, but if the refit weights are not identical to the build-time weights, behavior is undefined. When used alongside 'kSTRIP_PLAN', this flag will result in a small plan file for which weights are later supplied via refitting. This enables use of a single set of weights with different inference backends, or with TensorRT plans for multiple GPU architectures.

Enumerator

kWEIGHT_STREAMING	Enable weight streaming for the current engine. Weight streaming from the host enables execution of models that do not fit in GPU memory by allowing TensorRT to intelligently stream network weights from the CPU DRAM. Please see ICudaEngine::getMinimumWeightStreamingBudget for the default memory budget when this flag is enabled. Enabling this feature changes the behavior of IRuntime::deserializeCudaEngine to allocate the entire network's weights on the CPU DRAM instead of GPU memory. Then, ICudaEngine::createExecutionContext will determine the optimal split of weights between the CPU and GPU and place weights accordingly. Future TensorRT versions may enable this flag by default. Warning Enabling this flag may marginally increase build time. Enabling this feature will significantly increase the latency of ICudaEngine::createExecutionContext. See also IRuntime::deserializeCudaEngine, ICudaEngine::getMinimumWeightStreamingBudget, ICudaEngine::setWeightStreamingBudget
kINT4	Enable plugins with INT4 input/output.
kREFIT_INDIVIDUAL	Enable building a refittable engine and provide fine-grained control. This allows control over which weights are refittable or not using INetworkDefinition::markWeightsRefittable and INetworkDefinition::unmarkWeightsRefittable. By default, all weights are non-refittable when this flag is enabled. This flag cannot be used together with kREFIT or kREFIT_IDENTICAL.
kSTRICT_NANS	Disable floating-point optimizations: 0∗x => 0, x-x => 0, or x/x => 1. These identities are not true when x is a NaN or Inf, and thus might hide propagation or generation of NaNs. This flag is typically used in combination with kSPARSE_WEIGHTS. There are three valid sparsity configurations. 1. Disable all sparsity. Both kSPARSE_WEIGHTS and kSTRICT_NANS are unset 2. Enable sparsity only where it does not affect propagation/generation of NaNs. Both kSPARSE_WEIGHTS and kSTRICT_NANS are set 3. Enable all sparsity. kSPARSE_WEIGHTS is set and kSTRICT_NANS is unset
kMONITOR_MEMORY	Enable memory monitor during build time.
kFP4	Enable plugins with FP4 input/output.
kEDITABLE_TIMING_CACHE	Enable editable timing cache.

8.1.3.6 CalibrationAlgoType

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

Version of calibration algorithm to use.

Deprecated Deprecated in TensorRT 10.1. Superseded by explicit quantization.

Enumerator

kLEGACY_CALIBRATION	Legacy calibration.
kENTROPY_CALIBRATION	Legacy entropy calibration.
kENTROPY_↩ CALIBRATION_2	Entropy calibration.
kMINMAX_CALIBRATION	Minmax calibration.

8.1.3.7 CumulativeOperation

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

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

The table shows the initial value of each Cumulative operation.

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

Enumerator

kSUM	Calculate cumulative sum.

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

8.1.3.9 DeviceType

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

The device that this layer/network will execute on.

Enumerator

kGPU	GPU Device.
kDLA	DLA Core.

8.1.3.10 DimensionOperation

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

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

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

See also

IDimensionExpr, IExprBuilder

Enumerator

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

8.1.3.11 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 floating-point type or DataType::kINT8.

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

See also

IElementWiseLayer

Enumerator

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

8.1.3.12 EngineCapability

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

List of supported engine capability flows.

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

Enumerator

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

8.1.3.13 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.

Enumerator

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

8.1.3.14 ExecutionContextAllocationStrategy

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

Different memory allocation behaviors for IExecutionContext.

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

See also

ICudaEngine::createExecutionContext()

IExecutionContext::setDeviceMemory()

Enumerator

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

8.1.3.15 FillOperation

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

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

See also

IFillLayer

Enumerator

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

8.1.3.16 GatherMode

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

Control form of IGatherLayer.

See also

IGatherLayer

Enumerator

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

8.1.3.17 HardwareCompatibilityLevel

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

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

Warning

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

Enumerator

kNONE	Do not require hardware compatibility with GPU architectures other than that of the GPU on which the engine was built.
kAMPERE_PLUS	Require that the engine is compatible with Ampere and newer GPUs. This will limit the combined usage of driver reserved and backend kernel max shared memory to 48KiB, may reduce the number of available tactics for each layer, and may prevent some fusions from occurring. Thus this can decrease the performance, especially for tf32 models. This option will disable cuDNN, cuBLAS, and cuBLASLt as tactic sources. This option is only supported for engines built on NVIDIA Ampere and later GPUs. The driver reserved shared memory can be queried from cuDeviceGetAttribute(&reservedShmem, CU_DEVICE_ATTRIBUTE_↩ RESERVED_SHARED_MEMORY_PER_BLOCK).

Enumerator

kSAME_COMPUTE_CAPABILITY	Require that the engine is compatible with GPUs that have the same Compute Capability (https://developer.nvidia.com/cuda-gpus) as the one it was built on. This may decrease the performance compared to an engine with no compatibility. This option will disable cuDNN, cuBLAS, and cuBLASLt as tactic sources. This option is only supported for engines built on NVIDIA Turing and later GPUs.

8.1.3.18 InterpolationMode

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

Enumerates various modes of interpolation.

Enumerator

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

8.1.3.19 LayerInformationFormat

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

The format in which the IEngineInspector prints the layer information.

See also

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

Enumerator

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

8.1.3.20 LayerType

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

The type values of layer classes.

See also

ILayer::getType()

Enumerator

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

Enumerator

kDEQUANTIZE	Dequantize layer.
kCONDITION	Condition layer.
kCONDITIONAL_INPUT	Conditional Input layer.
kCONDITIONAL_OUTPUT	Conditional Output layer.
kSCATTER	Scatter layer.
kEINSUM	Einsum layer.
kASSERTION	Assertion layer.
kONE_HOT	OneHot layer.
kNON_ZERO	NonZero layer.
kGRID_SAMPLE	Grid sample layer.
kNMS	NMS layer.
kREVERSE_SEQUENCE	Reverse sequence layer.
kNORMALIZATION	Normalization layer.
kPLUGIN_V3	PluginV3 layer.
kSQUEEZE	Squeeze Layer.
kUNSQUEEZE	Unsqueeze Layer.
kCUMULATIVE	Cumulative layer.
kDYNAMIC_QUANTIZE	Dynamic Quantize layer.

8.1.3.21 LoopOutput

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

Enum that describes kinds of loop outputs.

Enumerator

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

8.1.3.22 MatrixOperation

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

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

Enumerator

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

8.1.3.23 MemoryPoolType

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

The type for memory pools used by TensorRT.

See also

IBuilderConfig::setMemoryPoolLimit, IBuilderConfig::getMemoryPoolLimit

Enumerator

kWORKSPACE	kWORKSPACE is used by TensorRT to store intermediate buffers within an operation. This defaults to max device memory. Set to a smaller value to restrict tactics 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.
kDLA_LOCAL_DRAM	kDLA_LOCAL_DRAM is host RAM used by DLA to share intermediate tensor data across operations. The size of this pool must be at least 4 KiB and must be a power of 2. This defaults to 1 GiB.
kDLA_GLOBAL_DRAM	kDLA_GLOBAL_DRAM is host RAM used by DLA to store weights and metadata for execution. The size of this pool must be at least 4 KiB and must be a power of 2. This defaults to 512 MiB.
kTACTIC_DRAM	kTACTIC_DRAM is the device DRAM used by the optimizer to run tactics. On embedded devices, where host and device memory are unified, this includes all host memory required by TensorRT to build the network up to the point of each memory allocation. This defaults to 75% of totalGlobalMem as reported by cudaGetDeviceProperties when cudaGetDeviceProperties.embedded is true, and 100% otherwise.

Enumerator

kTACTIC_SHARED_MEMORY	kTACTIC_SHARED_MEMORY defines the maximum sum of shared memory reserved by the driver and used for executing CUDA kernels. Adjust this value to restrict tactics that exceed the specified threshold en masse. The default value is device max capability. This value must be less than 1GiB. The driver reserved shared memory can be queried from cuDeviceGetAttribute(&reservedShmem, CU_DEVICE_ATTRIBUTE_↩ RESERVED_SHARED_MEMORY_PER_BLOCK). Updating this flag will override the shared memory limit set by HardwareCompatibilityLevel, which defaults to 48KiB - reservedShmem.

8.1.3.24 NetworkDefinitionCreationFlag

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

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

See also

IBuilder::createNetworkV2

Enumerator

kEXPLICIT_BATCH	Ignored because networks are always "explicit batch" in TensorRT 10.0. `\deprecated Deprecated in TensorRT 10.0.`
kSTRONGLY_TYPED	Mark the network to be strongly typed. Every tensor in the network has a data type defined in the network following only type inference rules and the inputs/operator annotations. Setting layer precision and layer output types is not allowed, and the network output types will be inferred based on the input types and the type inference rules.

8.1.3.25 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.

8.1.3.26 PaddingMode

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

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

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

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

Formulas for Convolution:

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

Formulas for Deconvolution:

- EXPLICIT_ROUND_DOWN:

- EXPLICIT_ROUND_UP:
```
O = (I - 1) * S + DK - (B + A)
```

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

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

Formulas for Pooling:

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

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

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

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

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

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

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

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

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

- SAME_LOWER:

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

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

Pooling Example 2:

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

Enumerator

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

8.1.3.27 PluginCapabilityType

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

Enumerates the different capability types a IPluginV3 object may have.

Enumerator

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

8.1.3.28 PluginCreatorVersion

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

Enum to identify version of the plugin creator.

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

Enumerator

kV1	IPluginCreator.
kV1_PYTHON	IPluginCreator-based Python plugin creators.

8.1.3.29 PluginFieldType

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

The possible field types for custom layer.

Enumerator

kFLOAT16	FP16 field type.
kFLOAT32	FP32 field type.
kFLOAT64	FP64 field type.
kINT8	INT8 field type.
kINT16	INT16 field type.
kINT32	INT32 field type.
kCHAR	char field type.
kDIMS	nvinfer1::Dims field type.
kUNKNOWN	Unknown field type.
kBF16	BF16 field type.
kINT64	INT64 field type.
kFP8	FP8 field type.
kINT4	INT4 field type.
kFP4	FP4 field type.

8.1.3.30 PluginVersion

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

Enumerator

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

8.1.3.31 PoolingType

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

The type of pooling to perform in a pooling layer.

Enumerator

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

8.1.3.32 PreviewFeature

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

Define preview features.

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

Enumerator

kPROFILE_SHARING_0806	Allows optimization profiles to be shared across execution contexts. **Deprecated** Deprecated in TensorRT 10.0. The default value for this flag is on and can not be changed.
kALIASED_PLUGIN_IO_10_03	Allows plugin I/O to be aliased when using IPluginV3OneBuildV2
kRUNTIME_ACTIVATION_RESIZE_10_10	Allows IExecutionContext::updateDeviceMemorySizeForShapes to resize runner internal activation memory. Using this feature can reduce runtime memory requirement when the actual input tensor shapes are smaller than the maximum input tensor dimensions.

8.1.3.33 ProfilingVerbosity

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

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

See also

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

Enumerator

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

8.1.3.34 QuantizationFlag

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

List of valid flags for quantizing the network to int8.

See also

IBuilderConfig::setQuantizationFlag(), IBuilderConfig::getQuantizationFlag()

Deprecated Deprecated in TensorRT 10.1. Superseded by explicit quantization.

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.

8.1.3.35 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	

8.1.3.36 ResizeCoordinateTransformation

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

The resize coordinate transformation function.

See also

IResizeLayer::setCoordinateTransformation()

Enumerator

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

kASYMMETRIC	|<----------length----------\>| | 0 | 1 | 2 | 3 | x_origin = x_resized ∗ (length_origin / length_resize)
kHALF_PIXEL	|<----------length----------\>| | 0 | 1 | 2 | 3 | x_origin = (x_resized + 0.5) ∗ (length_origin / length_resize) - 0.5

8.1.3.37 ResizeRoundMode

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

The rounding mode for nearest neighbor resize.

See also

IResizeLayer::setNearestRounding()

Enumerator

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

8.1.3.38 ResizeSelector

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

The coordinate selector when resize to single pixel output.

See also

IResizeLayer::setSelectorForSinglePixel()

Enumerator

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

8.1.3.39 RuntimePlatform

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

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

Note

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

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

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

See also

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

Enumerator

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

8.1.3.40 SampleMode

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

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

See also

ISliceLayer and IGridSample

Enumerator

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

8.1.3.41 ScaleMode

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

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

See also

IScaleLayer

Enumerator

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

8.1.3.42 ScatterMode

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

Control form of IScatterLayer.

See also

IScatterLayer

Enumerator

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

8.1.3.43 SeekPosition

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

Controls the seek mode of IStreamReaderV2.

Enumerator

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

8.1.3.44 SerializationFlag

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

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

See also

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

Enumerator

kEXCLUDE_WEIGHTS	Exclude the weights that can be refitted.
kEXCLUDE_LEAN_RUNTIME	Exclude the lean runtime.

8.1.3.45 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. Disabled by default. Note Disabling kCUBLAS will cause the cuBLAS handle passed to plugins in attachToContext to be null. **Deprecated** Deprecated in TensorRT 10.0.
kCUBLAS_LT	cuBLAS LT tactics. Disabled by default. **Deprecated** Deprecated in TensorRT 9.0.
kCUDNN	cuDNN tactics. Disabled by default. Note Disabling kCUDNN will cause the cuDNN handle passed to plugins in attachToContext to be null. **Deprecated** Deprecated in TensorRT 10.0.
kEDGE_MASK_CONVOLUTIONS	Enables convolution tactics implemented with edge mask tables. These tactics tradeoff memory for performance by consuming additional memory space proportional to the input size. Enabled by default.
kJIT_CONVOLUTIONS	Enables convolution tactics implemented with source-code JIT fusion. The engine building time may increase when this is enabled. Enabled by default.

8.1.3.46 TempfileControlFlag

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

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

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

Enumerator

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

8.1.3.47 TensorFormat

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

Format of the input/output tensors.

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

See also

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

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

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

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

For more information about data formats, see the topic ”Data Format Description” located in the TensorRT Developer Guide. https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.↩html#data-format-desc

Enumerator

kLINEAR	Memory layout is similar to an array in C or C++. The stride of each dimension is the product of the dimensions after it. The last dimension has unit stride. This format supports all TensorRT types. For DLA usage, the tensor sizes are limited to C,H,W in the range [1,8192].
kCHW2	Vector-major format with two scalars per vector. Vector dimension is third to last. This format requires FP16 or BF16 and at least three dimensions.
kHWC8	Vector-minor format with eight scalars per vector. Vector dimension is third to last. This format requires FP16 or BF16 and at least three dimensions.
kCHW4	Vector-major format with four scalars per vector. Vector dimension is third to last. This format requires INT8, FP16, or BF16 and at least three dimensions. For INT8, the length of the vector dimension must be a build-time constant. Deprecated usage: If running on the DLA, this format can be used for acceleration with the caveat that C must be less than or equal to 4. If used as DLA input and the build option kGPU_FALLBACK is not specified, it needs to meet line stride requirement of DLA format. Column stride in bytes must be a multiple of 64 on Orin.
kCHW16	Vector-major format with 16 scalars per vector. Vector dimension is third to last. This format requires FP16 and at least three dimensions. For DLA usage, this format maps to the native feature format for FP16, and the tensor sizes are limited to C,H,W in the range [1,8192].
kCHW32	Vector-major format with 32 scalars per vector. Vector dimension is third to last. This format requires INT8, FP32, or FP16 and at least three dimensions. For DLA usage, this format maps to the native feature format for INT8, and the tensor sizes are limited to C,H,W in the range [1,8192].
kDHWC8	Vector-minor format with eight scalars per vector. Vector dimension is fourth to last. This format requires FP16 or BF16 and at least four dimensions.
kCDHW32	Vector-major format with 32 scalars per vector. Vector dimension is fourth to last. This format requires FP16 or INT8 and at least four dimensions.
kHWC	Vector-minor format where channel dimension is third to last and unpadded. `This format requires either FP32 or UINT8 and at least three dimensions.`
kDLA_LINEAR	DLA planar format. For a tensor with dimension {N, C, H, W}, the W axis always has unit stride. The stride for stepping along the H axis is rounded up to 64 bytes. The memory layout is equivalent to a C array with dimensions [N][C][H][roundUp(W, 64/elementSize)] where elementSize is 2 for FP16 and 1 for Int8, with the tensor coordinates (n, c, h, w) mapping to array subscript [n][c][h][w].
kDLA_HWC4	DLA image format. For a tensor with dimension {N, C, H, W} the C axis always has unit stride. The stride for stepping along the H axis is rounded up to 64 bytes on Orin. C can only be 1, 3 or 4. If C == 1, it will map to grayscale format. If C == 3 or C == 4, it will map to color image format. And if C == 3, the stride for stepping along the W axis needs to be padded to 4 in elements. When C is {1, 3, 4}, then C' is {1, 4, 4} respectively, the memory layout is equivalent to a C array with dimensions [N][H][roundUp(W, 64/C'/elementSize)][C'] on Orin where elementSize is 2 for FP16 and 1 for Int8. The tensor coordinates (n, c, h, w) mapping to array subscript [n][h][w][c].
kHWC16	Vector-minor format with 16 scalars per vector. Vector dimension is third to last. This requires FP16, INT8 or FP8 and at least three dimensions.
kDHWC	Vector-minor format with one scalar per vector. Vector dimension is fourth to last. This format requires FP32 and at least four dimensions.

8.1.3.48 TensorIOMode

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

Definition of tensor IO Mode.

Enumerator

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

8.1.3.49 TensorLocation

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

The location for tensor data storage, device or host.

Enumerator

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

8.1.3.50 TensorRTPhase

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

Indicates a phase of operation of TensorRT.

Enumerator

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

8.1.3.51 TilingOptimizationLevel

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

Define the optimization levels for Tiling.

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

Enumerator

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

8.1.3.52 TopKOperation

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

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

Enumerator

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

8.1.3.53 TripLimit

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

Enum that describes kinds of trip limits.

Enumerator

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

8.1.3.54 UnaryOperation

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

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

Operations kNOT must have inputs of DataType::kBOOL.

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

Operation kISINF must have inputs of floating-point type.

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

See also

IUnaryLayer

Enumerator

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

8.1.3.55 WeightsRole

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

How a layer uses particular Weights.

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

Enumerator

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

8.1.4 Function Documentation

8.1.4.1 EnumMax()

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

Maximum number of elements in an enumeration type.

8.1.4.2 EnumMax< BoundingBoxFormat >()

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

Maximum number of elements in BoundingBoxFormat enum.

See also

BoundingBoxFormat

8.1.4.3 EnumMax< BuilderFlag >()

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

Maximum number of builder flags in BuilderFlag enum.

See also

BuilderFlag

8.1.4.4 EnumMax< CalibrationAlgoType >()

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

Maximum number of elements in CalibrationAlgoType enum.

See also

DataType

8.1.4.5 EnumMax< DeviceType >()

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

Maximum number of elements in DeviceType enum.

See also

DeviceType

8.1.4.6 EnumMax< DimensionOperation >()

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

Maximum number of elements in DimensionOperation enum.

See also

DimensionOperation

8.1.4.7 EnumMax< ExecutionContextAllocationStrategy >()

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

Maximum number of memory allocation strategies in ExecutionContextAllocationStrategy enum.

See also

ExecutionContextAllocationStrategy

8.1.4.8 EnumMax< FillOperation >()

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

Maximum number of elements in FillOperation enum.

See also

FillOperation

8.1.4.9 EnumMax< GatherMode >()

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

Maximum number of elements in GatherMode enum.

See also

GatherMode

8.1.4.10 EnumMax< LayerInformationFormat >()

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

Maximum number of layer information formats in LayerInformationFormat enum.

See also

LayerInformationFormat

8.1.4.11 EnumMax< LayerType >()

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

Maximum number of elements in LayerType enum.

See also

LayerType

8.1.4.12 EnumMax< LoopOutput >()

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

Maximum number of elements in LoopOutput enum.

See also

DataType

8.1.4.13 EnumMax< MatrixOperation >()

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

Maximum number of elements in MatrixOperation enum.

See also

DataType

8.1.4.14 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

8.1.4.15 EnumMax< NetworkDefinitionCreationFlag >()

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

Maximum number of elements in NetworkDefinitionCreationFlag enum.

See also

NetworkDefinitionCreationFlag

8.1.4.16 EnumMax< OptProfileSelector >()

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

Number of different values of OptProfileSelector enum.

See also

OptProfileSelector

8.1.4.17 EnumMax< ProfilingVerbosity >()

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

Maximum number of profile verbosity levels in ProfilingVerbosity enum.

See also

ProfilingVerbosity

8.1.4.18 EnumMax< QuantizationFlag >()

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

Maximum number of quantization flags in QuantizationFlag enum.

See also

QuantizationFlag

8.1.4.19 EnumMax< ReduceOperation >()

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

Maximum number of elements in ReduceOperation enum.

See also

ReduceOperation

8.1.4.20 EnumMax< SampleMode >()

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

Maximum number of elements in SampleMode enum.

See also

SampleMode

8.1.4.21 EnumMax< ScaleMode >()

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

Maximum number of elements in ScaleMode enum.

See also

ScaleMode

8.1.4.22 EnumMax< ScatterMode >()

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

Maximum number of elements in ScatterMode enum.

See also

ScatterMode

8.1.4.23 EnumMax< SerializationFlag >()

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

Maximum number of serialization flags in SerializationFlag enum.

See also

SerializationFlag

8.1.4.24 EnumMax< TacticSource >()

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

Maximum number of tactic sources in TacticSource enum.

See also

TacticSource

8.1.4.25 EnumMax< TempfileControlFlag >()

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

Maximum number of elements in TempfileControlFlag enum.

See also

TempfileControlFlag

8.1.4.26 EnumMax< TopKOperation >()

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

Maximum number of elements in TopKOperation enum.

See also

TopKOperation

8.1.4.27 EnumMax< TripLimit >()

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

Maximum number of elements in TripLimit enum.

See also

DataType

8.1.4.28 EnumMax< UnaryOperation >()

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

Maximum number of elements in UnaryOperation enum.

See also

UnaryOperation

8.1.4.29 EnumMax< WeightsRole >()

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

Maximum number of elements in WeightsRole enum.

See also

WeightsRole

8.1.4.30 getBuilderPluginRegistry()

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

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

Also return nullptr if the input argument is not EngineCapability::kSTANDARD. Engine capabilities EngineCapability::kSTANDARD and EngineCapability::kSAFETY have distinct plugin registries. When building a Safety engine, use nvinfer1::getBuilderSafePluginRegi
Use IPluginRegistry::registerCreator from the registry to register plugins. Plugins registered in a registry associated with a specific engine capability are only available when building engines with that engine capability.

There is no plugin registry for EngineCapability::kDLA_STANDALONE.

8.1.4.31 getBuilderSafePluginRegistry()

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

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

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

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

Functions

- IBuilder * createInferBuilder (ILogger &logger) noexcept

  Create an instance of an IBuilder class.

8.2.1 Function Documentation

8.2.1.1 createInferBuilder()

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

Create an instance of an IBuilder class.

Parameters

logger	The logging class for the builder.

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

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

Functions

- IRuntime * createInferRuntime (ILogger &logger) noexcept

  Create an instance of an IRuntime class.
- IRefitter * createInferRefitter (ICudaEngine &engine, ILogger &logger) noexcept

  Create an instance of an IRefitter class.

8.3.1 Function Documentation

8.3.1.1 createInferRefitter()

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

Create an instance of an IRefitter class.

Parameters

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

8.3.1.2 createInferRuntime()

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

Create an instance of an IRuntime class.

Parameters

logger	The logging class for the runtime.

8.4 nvinfer1::impl Namespace Reference

Classes

- struct EnumMaxImpl

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

  Maximum number of elements in AllocatorFlag enum.
- struct EnumMaxImpl< APILanguage >

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

  Maximum number of elements in CumulativeOperation enum.

- struct EnumMaxImpl< DataType >

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

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

  Maximum number of elements in ErrorCode enum.
- struct EnumMaxImpl< HardwareCompatibilityLevel >
- struct EnumMaxImpl< ILogger::Severity >

  Maximum number of elements in ILogger::Severity enum.
- struct EnumMaxImpl< InterpolationMode >
- struct EnumMaxImpl< PaddingMode >
- struct EnumMaxImpl< PoolingType >
- struct EnumMaxImpl< PreviewFeature >
- struct EnumMaxImpl< ResizeCoordinateTransformation >
- struct EnumMaxImpl< ResizeRoundMode >
- struct EnumMaxImpl< ResizeSelector >
- struct EnumMaxImpl< RuntimePlatform >
- struct EnumMaxImpl< TensorFormat >

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

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

  Maximum number of elements in TensorLocation enum.
- struct EnumMaxImpl< TilingOptimizationLevel >

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.
- struct GridAnchorParameters

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

  The NMSParameters are used by the BatchedNMSPlugin 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).
- struct RegionParameters

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

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

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

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.

Deprecated Deprecated in TensorRT 10.0. DetectionOutput plugin is deprecated.

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

8.7 nvinfer1::serialize Namespace Reference

Functions

- nvinfer1::IHostMemory * serializeNetwork (nvinfer1::INetworkDefinition const &network) noexcept

  Serialize TensorRT INetworkDefinition.

8.7.1 Function Documentation

8.7.1.1 serializeNetwork()

```
nvinfer1::IHostMemory * nvinfer1::serialize::serializeNetwork (
            nvinfer1::INetworkDefinition const & network )  [inline], [noexcept]
```

Serialize TensorRT INetworkDefinition.

Parameters

network	Network to serialize.

Returns

A nvinfer1::IHostMemory object that contains serialized network.

The network might be deserialized with nvserialize::deserializeINetwork.

8.8 nvinfer1::v_1_0 Namespace Reference

Forward declare IErrorRecorder for use in other interfaces.

Classes

- class IAlgorithmSelector
- class IDebugListener
- class IErrorRecorder
- class IGpuAllocator
- class IGpuAsyncAllocator
- class IInt8EntropyCalibrator
- class IInt8EntropyCalibrator2
- class IInt8LegacyCalibrator
- class IInt8MinMaxCalibrator
- class IOutputAllocator
- class IPluginCapability
- class IPluginCreator
- class IPluginCreatorInterface
- class IPluginCreatorV3One
- class IPluginResource
- class IPluginV3
- class IPluginV3OneBuild
- class IPluginV3OneCore
- class IPluginV3OneRuntime
- class IProfiler
- class IProgressMonitor
- class IStreamReader
- class IStreamReaderV2
- struct TimingCacheKey

  The key to retrieve timing cache entries.
- struct TimingCacheValue

8.8.1 Detailed Description

Forward declare IErrorRecorder for use in other interfaces.

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

8.9 nvinfer1::v_2_0 Namespace Reference

Classes

- class IPluginV3OneBuild

8.10 nvonnxparser Namespace Reference

The TensorRT ONNX parser API namespace.

Namespaces

- namespace anonymous_namespace{NvOnnxParser.h}

Classes

- class IOnnxConfig

  Configuration Manager Class.
- class IParser

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

  an object containing information about an error
- class IParserRefitter

  An interface designed to refit weights from an ONNX model.

Typedefs

- using OnnxParserFlags = uint32_t

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

Enumerations

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

  The type of error that the parser or refitter may return.
- enum class OnnxParserFlag : int32_t { kNATIVE_INSTANCENORM = 0 }

Functions

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

8.10.1 Detailed Description

The TensorRT ONNX parser API namespace.

8.10.2 Typedef Documentation

8.10.2.1 OnnxParserFlags

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

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

See also

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

8.10.3 Enumeration Type Documentation

8.10.3.1 ErrorCode

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

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

Enumerator

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

8.10.3.2 OnnxParserFlag

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

Enumerator

kNATIVE_INSTANCENORM	Parse the ONNX model into the INetworkDefinition with the intention of using TensorRT's native layer implementation over the plugin implementation for InstanceNormalization nodes. This flag is required when building version-compatible or hardware-compatible engines. This flag is set to be ON by default.

8.10.4 Function Documentation

8.10.4.1 createONNXConfig()

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

8.10.4.2 EnumMax()

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

8.10.4.3 EnumMax< ErrorCode >()

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

Maximum number of flags in the ErrorCode enum.

See also

ErrorCode

8.10.4.4 EnumMax< OnnxParserFlag >()

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

Maximum number of flags in the OnnxParserFlag enum.

See also

OnnxParserFlag

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

Functions

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

8.11.1 Function Documentation

8.11.1.1 createParser()

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

Create a new parser object.

Parameters

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

Returns

a new parser object or NULL if an error occurred

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

See also

IParser

8.11.1.2 createParserRefitter()

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

Create a new ONNX refitter object.

Parameters

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

Returns

a new ParserRefitter object or NULL if an error occurred

See also

IParserRefitter

Chapter 9

Class Documentation

9.1 nvinfer1::plugin::DetectionOutputParameters Struct Reference

The DetectionOutput plugin layer generates the detection output based on location and confidence predictions by doing non maximum suppression.

`#include <NvInferPluginUtils.h>`

Public Attributes

- bool shareLocation

  If true, bounding box are shared among different classes.
- bool varianceEncodedInTarget
- int32_t backgroundLabelId

  Background label ID. If there is no background class, set it as -1.
- int32_t numClasses

  Number of classes to be predicted.
- int32_t topK
- int32_t keepTopK

  Number of total bounding boxes to be kept per image after NMS step.
- float confidenceThreshold

  Only consider detections whose confidences are larger than a threshold.
- float nmsThreshold

  Threshold to be used in NMS.
- CodeTypeSSD codeType

  Type of coding method for bbox.
- int32_t inputOrder [3]

  Specifies the order of inputs {loc_data, conf_data, priorbox_data}.
- bool confSigmoid

  Set to true to calculate sigmoid of confidence scores.
- bool isNormalized

  Set to true if bounding box data is normalized by the network.
- bool isBatchAgnostic {true}

  Defaults to true. Set to false if prior boxes are unique per batch.

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.

Deprecated Deprecated in TensorRT 10.0. DetectionOutput plugin is deprecated.

9.1.2 Member Data Documentation

9.1.2.1 backgroundLabelId

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

Background label ID. If there is no background class, set it as -1.

9.1.2.2 codeType

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

Type of coding method for bbox.

9.1.2.3 confidenceThreshold

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

Only consider detections whose confidences are larger than a threshold.

9.1.2.4 confSigmoid

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

Set to true to calculate sigmoid of confidence scores.

9.1.2.5 inputOrder

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

Specifies the order of inputs {loc_data, conf_data, priorbox_data}.

9.1.2.6 isBatchAgnostic

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

Defaults to true. Set to false if prior boxes are unique per batch.

9.1.2.7 isNormalized

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

Set to true if bounding box data is normalized by the network.

9.1.2.8 keepTopK

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

Number of total bounding boxes to be kept per image after NMS step.

9.1.2.9 nmsThreshold

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

Threshold to be used in NMS.

9.1.2.10 numClasses

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

Number of classes to be predicted.

9.1.2.11 shareLocation

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

If true, bounding box are shared among different classes.

9.1.2.12 topK

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

Number of boxes per image with top confidence scores that are fed into the NMS algorithm.

9.1.2.13 varianceEncodedInTarget

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

If true, variance is encoded in target. Otherwise we need to adjust the predicted offset accordingly.

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

- NvInferPluginUtils.h

9.2 Dims Class Reference

Structure to define the dimensions of a tensor.

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

9.2.1 Detailed Description

Structure to define the dimensions of a tensor.

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

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

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

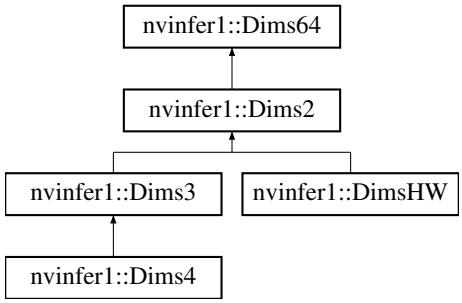
- NvInferRuntimeBase.h

9.3 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 (int64_t d0, int64_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 (
            int64_t d0,
            int64_t d1 )  [inline]
```

Construct a Dims2 from 2 elements.

Parameters

d0	The first element.
d1	The second element.

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

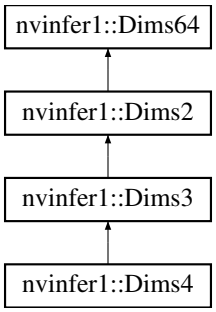
- NvInferLegacyDims.h

9.4 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 (int64_t d0, int64_t d1, int64_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 (
            int64_t d0,
            int64_t d1,
            int64_t d2 )  [inline]
```

Construct a Dims3 from 3 elements.

Parameters

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

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

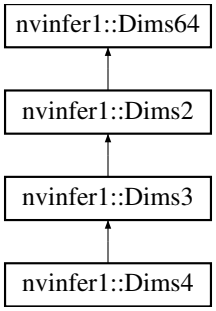
- NvInferLegacyDims.h

9.5 nvinfer1::Dims4 Class Reference

Descriptor for four-dimensional data.

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

Inheritance diagram for nvinfer1::Dims4:



Public Member Functions

- Dims4 ()

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

  Construct a Dims4 from 4 elements.

Additional Inherited Members

9.5.1 Detailed Description

Descriptor for four-dimensional data.

9.5.2 Constructor & Destructor Documentation

9.5.2.1 Dims4() [1/2]

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

Construct an empty Dims4 object.

9.5.2.2 Dims4() [2/2]

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

Construct a Dims4 from 4 elements.

Parameters

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

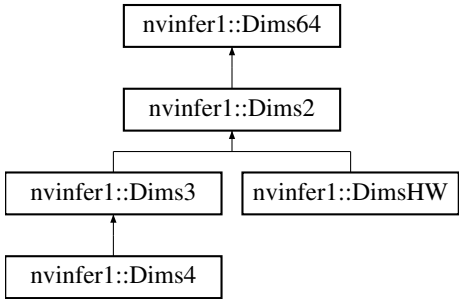
The documentation for this class was generated from the following file:

- NvInferLegacyDims.h

9.6 nvinfer1::Dims64 Class Reference

`#include <NvInferRuntimeBase.h>`

Inheritance diagram for nvinfer1::Dims64:



Public Attributes

- int32_t nbDims

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

  The extent of each dimension.

Static Public Attributes

- static constexpr int32_t MAX_DIMS {8}

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

9.6.1 Member Data Documentation

9.6.1.1 d

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

The extent of each dimension.

9.6.1.2 MAX_DIMS

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

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

9.6.1.3 nbDims

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

The rank (number of dimensions).

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

- NvInferRuntimeBase.h

9.7 nvinfer1::DimsExprs Class Reference

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

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

Public Attributes

- int32_t nbDims

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

  The extent of each dimension.

9.7.1 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:

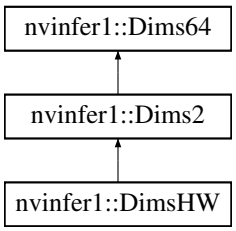
- NvInferRuntime.h

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

Additional Inherited Members

9.8.1 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 (
            int64_t height,
            int64_t width )  [inline]
```

Construct a DimsHW given height and width.

Parameters

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

9.8.3 Member Function Documentation

9.8.3.1 h() [1/2]

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

Get the height.

Returns

The height.

9.8.3.2 h() [2/2]

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

Get the height.

Returns

The height.

9.8.3.3 w() [1/2]

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

Get the width.

Returns

The width.

9.8.3.4 w() [2/2]

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

Get the width.

Returns

The width.

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

- NvInferLegacyDims.h

9.9 nvinfer1::DynamicPluginTensorDesc Struct Reference

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

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

Public Attributes

- PluginTensorDesc desc

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

  Lower bounds on tensor's dimensions.
- Dims max

  Upper bounds on tensor's dimensions.
- Dims opt

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

9.9.1 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.

9.9.2.4 opt

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

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

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

- NvInferRuntime.h

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

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

9.10.1 Detailed Description

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

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

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

- NvInferRuntimeBase.h

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

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

Static Public Attributes

- static constexpr int32_t kVALUE = 14

9.11.1 Detailed Description

Maximum number of elements in ActivationType enum.

See also

ActivationType

9.11.2 Member Data Documentation

9.11.2.1 kVALUE

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

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

- NvInfer.h

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

Maximum number of elements in AllocatorFlag enum.

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

Static Public Attributes

- static constexpr int32_t kVALUE = 1

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

9.12.1 Detailed Description

Maximum number of elements in AllocatorFlag enum.

See also

AllocatorFlag

9.12.2 Member Data Documentation

9.12.2.1 kVALUE

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

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

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

- NvInferRuntime.h

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

Maximum number of elements in APILanguage enum.

`#include <NvInferRuntimeBase.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 2

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

9.13.1 Detailed Description

Maximum number of elements in APILanguage enum.

See also

APILanguage

9.13.2 Member Data Documentation

9.13.2.1 kVALUE

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

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

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

- NvInferRuntimeBase.h

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

Maximum number of elements in CumulativeOperation enum.

`#include <NvInfer.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 1

9.14.1 Detailed Description

Maximum number of elements in CumulativeOperation enum.

See also

CumulativeOperation

9.14.2 Member Data Documentation

9.14.2.1 kVALUE

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

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

- NvInfer.h

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

Maximum number of elements in DataType enum.

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

Static Public Attributes

- static constexpr int32_t kVALUE = 11

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

9.15.1 Detailed Description

Maximum number of elements in DataType enum.

See also

DataType

9.15.2 Member Data Documentation

9.15.2.1 kVALUE

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

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

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

- NvInferRuntimeBase.h

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

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

Static Public Attributes

- static constexpr int32_t kVALUE = 14

9.16.1 Detailed Description

Maximum number of elements in ElementWiseOperation enum.

See also

ElementWiseOperation

9.16.2 Member Data Documentation

9.16.2.1 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

9.17 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

9.17.1 Detailed Description

Maximum number of elements in EngineCapability enum.

See also

EngineCapability

9.17.2 Member Data Documentation

9.17.2.1 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

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

Maximum number of elements in ErrorCode enum.

`#include <NvInferRuntimeBase.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 11

  Declaration of kVALUE.

9.18.1 Detailed Description

Maximum number of elements in ErrorCode enum.

See also

ErrorCode

9.18.2 Member Data Documentation

9.18.2.1 kVALUE

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

Declaration of kVALUE.

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

- NvInferRuntimeBase.h

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

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

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.19.1 Detailed Description

Maximum number of elements in HardwareCompatibilityLevel enum.

See also

HardwareCompatibilityLevel

9.19.2 Member Data Documentation

9.19.2.1 kVALUE

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

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

- NvInfer.h

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

Maximum number of elements in ILogger::Severity enum.

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

Static Public Attributes

- static constexpr int32_t kVALUE = 5

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

9.20.1 Detailed Description

Maximum number of elements in ILogger::Severity enum.

See also

ILogger::Severity

9.20.2 Member Data Documentation

9.20.2.1 kVALUE

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

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

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

- NvInferRuntime.h

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

`#include <NvInfer.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.21.1 Detailed Description

Maximum number of elements in InterpolationMode enum.

See also

InterpolationMode

9.21.2 Member Data Documentation

9.21.2.1 kVALUE

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

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

- NvInfer.h

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

`#include <NvInfer.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 4

9.22.1 Detailed Description

Maximum number of elements in PaddingMode enum.

See also

PaddingMode

9.22.2 Member Data Documentation

9.22.2.1 kVALUE

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

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

- NvInfer.h

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

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

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.23.1 Detailed Description

Maximum number of elements in PoolingType enum.

See also

PoolingType

9.23.2 Member Data Documentation

9.23.2.1 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

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

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

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.24.1 Detailed Description

Maximum number of elements in PreviewFeature enum.

See also

PreviewFeature

9.24.2 Member Data Documentation

9.24.2.1 kVALUE

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

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

- NvInfer.h

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

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

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.25.1 Detailed Description

Maximum number of elements in ResizeCoordinateTransformation enum.

See also

ResizeCoordinateTransformation

9.25.2 Member Data Documentation

9.25.2.1 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

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

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

Static Public Attributes

- static constexpr int32_t kVALUE = 4

9.26.1 Detailed Description

Maximum number of elements in ResizeRoundMode enum.

See also

ResizeRoundMode

9.26.2 Member Data Documentation

9.26.2.1 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

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

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

Static Public Attributes

- static constexpr int32_t kVALUE = 2

9.27.1 Detailed Description

Maximum number of elements in ResizeSelector enum.

See also

ResizeSelector

9.27.2 Member Data Documentation

9.27.2.1 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

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

`#include <NvInfer.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 2

9.28.1 Detailed Description

Maximum number of elements in RuntimePlatform enum.

See also

RuntimePlatform

9.28.2 Member Data Documentation

9.28.2.1 kVALUE

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

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

- NvInfer.h

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

Maximum number of elements in TensorFormat enum.

`#include <NvInferRuntime.h>`

Static Public Attributes

- static constexpr int32_t kVALUE = 13

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

9.29.1 Detailed Description

Maximum number of elements in TensorFormat enum.

See also

TensorFormat

9.29.2 Member Data Documentation

9.29.2.1 kVALUE

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

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

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

- NvInferRuntime.h

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

Maximum number of elements in TensorIOMode enum.

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

Static Public Attributes

- static constexpr int32_t kVALUE = 3

9.30.1 Detailed Description

Maximum number of elements in TensorIOMode enum.

See also

TensorIOMode

9.30.2 Member Data Documentation

9.30.2.1 kVALUE

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

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

- NvInferRuntimeBase.h

9.31 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

9.31.1 Detailed Description

Maximum number of elements in TensorLocation enum.

See also

TensorLocation

9.31.2 Member Data Documentation

9.31.2.1 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

9.32 nvinfer1::impl::EnumMaxImpl< TilingOptimizationLevel > Struct Reference

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

Static Public Attributes

- static constexpr int32_t kVALUE = 4

9.32.1 Detailed Description

Maximum number of elements in TilingOptimizationLevel enum.

See also

TilingOptimizationLevel

9.32.2 Member Data Documentation

9.32.2.1 kVALUE

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

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

- NvInfer.h

9.33 nvinfer1::plugin::GridAnchorParameters Struct Reference

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

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

Public Attributes

- float minSize

  Scale of anchors corresponding to finest resolution.
- float maxSize

  Scale of anchors corresponding to coarsest resolution.
- float * aspectRatios

  List of aspect ratios to place on each grid point.
- int32_t numAspectRatios

  Number of elements in aspectRatios.
- int32_t H

  Height of feature map to generate anchors for.
- int32_t W

  Width of feature map to generate anchors for.
- float variance [4]

  Variance for adjusting the prior boxes.

9.33.1 Detailed Description

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

9.33.2 Member Data Documentation

9.33.2.1 aspectRatios

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

List of aspect ratios to place on each grid point.

9.33.2.2 H

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

Height of feature map to generate anchors for.

9.33.2.3 maxSize

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

Scale of anchors corresponding to coarsest resolution.

9.33.2.4 minSize

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

Scale of anchors corresponding to finest resolution.

9.33.2.5 numAspectRatios

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

Number of elements in aspectRatios.

9.33.2.6 variance

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

Variance for adjusting the prior boxes.

9.33.2.7 W

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

Width of feature map to generate anchors for.

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

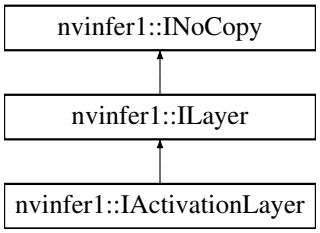
- NvInferPluginUtils.h

9.34 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

9.34.1 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.

9.34.2 Constructor & Destructor Documentation

9.34.2.1 ~IActivationLayer()

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

9.34.3 Member Function Documentation

9.34.3.1 getActivationType()

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

Get the type of activation to be performed.

See also

setActivationType(), ActivationType

9.34.3.2 getAlpha()

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

Get the alpha parameter.

See also

getBeta(), setAlpha()

9.34.3.3 getBeta()

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

Get the beta parameter.

See also

getAlpha(), setBeta()

9.34.3.4 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

9.34.3.5 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()

9.34.3.6 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()

9.34.4 Member Data Documentation

9.34.4.1 mImpl

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

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

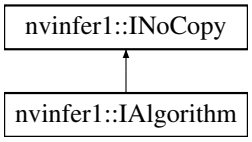
- NvInfer.h

9.35 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 Algorithm↩ Selector::selectAlgorithms().

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

Inheritance diagram for nvinfer1::IAlgorithm:



Public Member Functions

- 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

9.35.1 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.

Deprecated Deprecated in TensorRT 10.8. Please use editable mode in ITimingCache instead.

9.35.2 Constructor & Destructor Documentation

9.35.2.1 ~IAlgorithm()

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

9.35.3 Member Function Documentation

9.35.3.1 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

index	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.

9.35.3.2 getAlgorithmVariant()

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

Returns the algorithm variant.

9.35.3.3 getTimingMSec()

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

The time in milliseconds to execute the algorithm.

9.35.3.4 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.

9.35.4 Member Data Documentation

9.35.4.1 mImpl

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

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

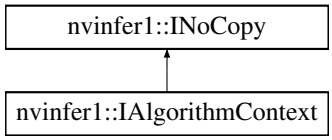
- NvInfer.h

9.36 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.
- 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

9.36.1 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.

Deprecated Deprecated in TensorRT 10.8. Please use editable mode in ITimingCache instead.

9.36.2 Constructor & Destructor Documentation

9.36.2.1 ~IAlgorithmContext()

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

9.36.3 Member Function Documentation

9.36.3.1 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

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

9.36.3.2 getName()

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

Return name of the algorithm node.

This is a unique identifier for the IAlgorithmContext.

9.36.3.3 getNbInputs()

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

Return number of inputs of the algorithm.

9.36.3.4 getNbOutputs()

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

Return number of outputs of the algorithm.

9.36.4 Member Data Documentation

9.36.4.1 mImpl

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

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

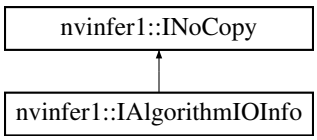
- NvInfer.h

9.37 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

- DataType getDataType () const noexcept

  Return DataType of the input/output of algorithm.
- Dims getStrides () const noexcept

  Return strides of the input/output tensor of algorithm. For vectorized formats, strides are given in units of vectors.
- int64_t getVectorizedDim () const noexcept

  Return the index of the vectorized dimension or -1 for non-vectorized formats.
- int64_t getComponentsPerElement () const noexcept

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

Protected Member Functions

- virtual ~IAlgorithmIOInfo () noexcept=default

Protected Attributes

- apiv::VAlgorithmIOInfo * mImpl

9.37.1 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.

Deprecated Deprecated in TensorRT 10.8. Please use editable mode in ITimingCache instead.

9.37.2 Constructor & Destructor Documentation

9.37.2.1 ~IAlgorithmIOInfo()

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

9.37.3 Member Function Documentation

9.37.3.1 getComponentsPerElement()

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

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

Returns

the number of components per element.

9.37.3.2 getDataType()

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

Return DataType of the input/output of algorithm.

Returns

the data type.

9.37.3.3 getStrides()

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

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

Returns

the strides of the tensor.

9.37.3.4 getVectorizedDim()

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

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

Returns

the index of the vectorized dimension.

9.37.4 Member Data Documentation

9.37.4.1 mImpl

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

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

- NvInfer.h

9.38 IAlgorithmSelector Class Reference

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

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

9.38.1 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.

To ensure compatibility of source code with future versions of TensorRT, use IAlgorithmSelector, not v_1_0::↩ IAlgorithmSelector

Deprecated Deprecated in TensorRT 10.8. Please use editable mode in ITimingCache instead.

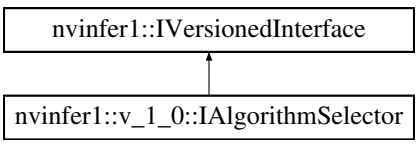
The documentation for this class was generated from the following file:

- NvInfer.h

9.39 nvinfer1::v_1_0::IAlgorithmSelector Class Reference

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

Inheritance diagram for nvinfer1::v_1_0::IAlgorithmSelector:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- 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

Additional Inherited Members

9.39.1 Constructor & Destructor Documentation

9.39.1.1 ~IAlgorithmSelector()

```
virtual nvinfer1::v_1_0::IAlgorithmSelector::~IAlgorithmSelector ( )  [virtual], [default], [noexcept]
```

9.39.2 Member Function Documentation

9.39.2.1 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IAlgorithmSelector::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.39.2.2 reportAlgorithms()

```
virtual void nvinfer1::v_1_0::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

algoContexts	The list of all algorithm contexts.
algoChoices	The list of algorithm choices made by TensorRT
nbAlgorithms	The size of algoContexts as well as algoChoices.

9.39.2.3 selectAlgorithms()

```
virtual int32_t nvinfer1::v_1_0::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

context	The context for which the algorithm choices are valid.
choices	The list of algorithm choices to select for implementation of this layer.
nbChoices	Number of algorithm choices.
selection	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 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:

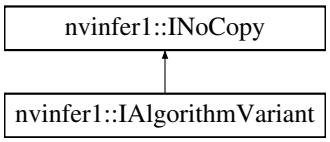
- NvInfer.h

9.40 nvinfer1::IAlgorithmVariant Class Reference

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

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IAlgorithmVariant:



Public Member Functions

- int64_t getImplementation () const noexcept

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

  Return tactic of the algorithm.

Protected Member Functions

- virtual ~IAlgorithmVariant () noexcept=default

Protected Attributes

- apiv::VAlgorithmVariant * mImpl

9.40.1 Detailed Description

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

See also

IAlgorithmIOInfo, IAlgorithm, IAlgorithmSelector::selectAlgorithms()

Note

A single implementation can have multiple tactics.

Warning

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

Deprecated Deprecated in TensorRT 10.8. Please use editable mode in ITimingCache instead.

9.40.2 Constructor & Destructor Documentation

9.40.2.1 ~IAlgorithmVariant()

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

9.40.3 Member Function Documentation

9.40.3.1 getImplementation()

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

Return implementation of the algorithm.

9.40.3.2 getTactic()

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

Return tactic of the algorithm.

9.40.4 Member Data Documentation

9.40.4.1 mImpl

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

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

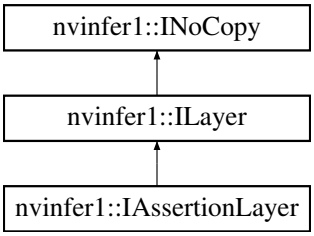
- NvInfer.h

9.41 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

9.41.1 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.

9.41.2 Constructor & Destructor Documentation

9.41.2.1 ~IAssertionLayer()

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

9.41.3 Member Function Documentation

9.41.3.1 getMessage()

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

Return the assertion message.

See also

setMessage()

9.41.3.2 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()

9.41.4 Member Data Documentation

9.41.4.1 mImpl

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

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

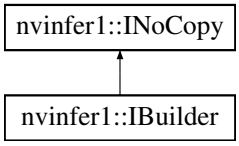
- NvInfer.h

9.42 nvinfer1::IBuilder Class Reference

Builds an engine from a network definition.

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

Inheritance diagram for nvinfer1::IBuilder:



Public Member Functions

- virtual ~IBuilder () noexcept=default
- TRT_DEPRECATED bool platformHasFastFp16 () const noexcept

  Determine whether the platform has fast native fp16.
- TRT_DEPRECATED bool platformHasFastInt8 () const noexcept

  Determine whether the platform has fast native int8.
- int32_t getMaxDLABatchSize () const noexcept

  Get the maximum batch size DLA can support. For any tensor the total volume of index dimensions combined(dimensions other than CHW) with the requested batch size should not exceed the value returned by this function.
- int32_t getNbDLACores () const noexcept

  Return the number of DLA engines available to this builder.
- void setGpuAllocator (IGpuAllocator *allocator) noexcept

  Set the GPU allocator.
- nvinfer1::IBuilderConfig * createBuilderConfig () noexcept

  Create a builder configuration object.
- nvinfer1::INetworkDefinition * createNetworkV2 (NetworkDefinitionCreationFlags flags) noexcept

  Create a network definition object.
- nvinfer1::IOptimizationProfile * createOptimizationProfile () noexcept

  Create a new optimization profile.
- void setErrorRecorder (IErrorRecorder *recorder) noexcept

  Set the ErrorRecorder for this interface.
- IErrorRecorder * getErrorRecorder () const noexcept

get the ErrorRecorder assigned to this interface.

- void reset () noexcept

  Resets the builder state to default values.

- TRT_DEPRECATED 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.

- nvinfer1::ICudaEngine * buildEngineWithConfig (INetworkDefinition &network, IBuilderConfig &config) noexcept

  Builds a network for the given INetworkDefinition and IBuilderConfig.

- bool isNetworkSupported (INetworkDefinition const &network, IBuilderConfig const &config) const noexcept

  Checks that a network is within the scope of the IBuilderConfig settings.

- ILogger * getLogger () const noexcept

  get the logger with which the builder was created

- bool setMaxThreads (int32_t maxThreads) noexcept

  Set the maximum number of threads.

- int32_t getMaxThreads () const noexcept

  get the maximum number of threads that can be used by the builder.

- IPluginRegistry & getPluginRegistry () noexcept

  get the local plugin registry that can be used by the builder.

Protected Attributes

- apiv::VBuilder * mImpl

Additional Inherited Members

9.42.1 Detailed Description

Builds an engine from a network definition.

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

9.42.2 Constructor & Destructor Documentation

9.42.2.1 ~IBuilder()

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

9.42.3 Member Function Documentation

9.42.3.1 buildEngineWithConfig()

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

Builds a network for the given INetworkDefinition and IBuilderConfig.

Parameters

network	Network definition.
config	Builder configuration.

Returns

A pointer to a ICudaEngine object that contains an engine.

Note

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

This function does not support `BuilderFlag::kVERSION_COMPATIBLE`. Please use `buildSerializedNetwork` to get a version compatible engine.

See also

INetworkDefinition, IBuilderConfig, ICudaEngine

9.42.3.2 buildSerializedNetwork()

```
nvinfer1::IHostMemory * nvinfer1::IBuilder::buildSerializedNetwork (
            INetworkDefinition & network,
            IBuilderConfig & config )  [inline], [noexcept]
```

Builds and serializes a network for the given INetworkDefinition and IBuilderConfig.

This function allows building and serialization of a network without creating an engine.

Parameters

network	Network definition.
config	Builder configuration.

Returns

A pointer to a IHostMemory object that contains a serialized network.

Note

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

See also

INetworkDefinition, IBuilderConfig, IHostMemory

9.42.3.3 createBuilderConfig()

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

Create a builder configuration object.

The caller owns the new IBuilderConfig, which must be destroyed with operator delete before this IBuilder is destroyed. Destroying this IBuilder before destroying the IBuilderConfig causes undefined behavior.

See also

IBuilderConfig

9.42.3.4 createNetworkV2()

```
nvinfer1::INetworkDefinition * nvinfer1::IBuilder::createNetworkV2 (
            NetworkDefinitionCreationFlags flags )  [inline], [noexcept]
```

Create a network definition object.

Creates a network definition object with immutable properties specified using the flags parameter.

createNetworkV2 supports creating network with properties from NetworkDefinitionCreationFlags.

CreateNetworkV2 supports dynamic shapes and explicit batch dimensions by default.

createNetworkV2 with NetworkDefinitionCreationFlag::kSTRONGLY_TYPED flag supports creating a strongly typed plan where tensor data types are inferred from network input types and operator type specification.

The caller owns the new INetworkDefinition, which must be destroyed with operator delete before this IBuilder is destroyed. Destroying this IBuilder before destroying the INetworkDefinition causes undefined behavior.

Parameters

flags	Bitset of NetworkDefinitionCreationFlags specifying network properties combined with bitwise OR, e.g., 1U << NetworkDefinitionCreationFlag::kSTRONGLY_TYPED.

See also

INetworkDefinition, NetworkDefinitionCreationFlags

9.42.3.5 createOptimizationProfile()

```
nvinfer1::IOptimizationProfile * nvinfer1::IBuilder::createOptimizationProfile ( )  [inline], [noexcept]
```

Create a new optimization profile.

If the network has any dynamic input tensors, the appropriate calls to setDimensions() must be made. Likewise, if there are any shape input tensors, the appropriate calls to setShapeValues() are required. The builder retains ownership of the created optimization profile and returns a raw pointer, i.e. the users must not attempt to delete the returned pointer.

See also

IOptimizationProfile

9.42.3.6 getErrorRecorder()

```
IErrorRecorder * nvinfer1::IBuilder::getErrorRecorder ( ) const  [inline], [noexcept]
```

get the ErrorRecorder assigned to this interface.

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

Returns

A pointer to the IErrorRecorder object that has been registered.

See also

setErrorRecorder()

9.42.3.7 getLogger()

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

get the logger with which the builder was created

Returns

the logger

9.42.3.8 getMaxDLABatchSize()

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

Get the maximum batch size DLA can support. For any tensor the total volume of index dimensions combined(dimensions other than CHW) with the requested batch size should not exceed the value returned by this function.

Warning

getMaxDLABatchSize does not work with dynamic shapes.

9.42.3.9 getMaxThreads()

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

get the maximum number of threads that can be used by the builder.

Retrieves the maximum number of threads that can be used by the builder.

Returns

The maximum number of threads that can be used by the builder.

See also

setMaxThreads()

9.42.3.10 getNbDLACores()

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

Return the number of DLA engines available to this builder.

9.42.3.11 getPluginRegistry()

```
IPluginRegistry & nvinfer1::IBuilder::getPluginRegistry ( )  [inline], [noexcept]
```

get the local plugin registry that can be used by the builder.

Returns

The local plugin registry that can be used by the builder.

9.42.3.12 isNetworkSupported()

```
bool nvinfer1::IBuilder::isNetworkSupported (
            INetworkDefinition const & network,
            IBuilderConfig const & config ) const  [inline], [noexcept]
```

Checks that a network is within the scope of the IBuilderConfig settings.

Parameters

network	The network definition to check for configuration compliance.
config	The configuration of the builder to use when checking `network`.

Given an INetworkDefinition, `network`, and an IBuilderConfig, `config`, check if the network falls within the constraints of the builder configuration based on the EngineCapability, BuilderFlag, and DeviceType. If the network is within the constraints, then the function returns true, and false if a violation occurs. This function reports the conditions that are violated to the registered ErrorRecorder.

Returns

True if network is within the scope of the restrictions specified by the builder config, false otherwise.

Note

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

9.42.3.13 platformHasFastFp16()

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

Determine whether the platform has fast native fp16.

Deprecated Deprecated in TensorRT 10.5. Please query data type support from CUDA directly.

9.42.3.14 platformHasFastInt8()

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

Determine whether the platform has fast native int8.

Deprecated Deprecated in TensorRT 10.5. Please query data type support from CUDA directly.

9.42.3.15 platformHasTf32()

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

Determine whether the platform has TF32 support.

Deprecated Deprecated in TensorRT 10.5. Please query data type support from CUDA directly.

9.42.3.16 reset()

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

Resets the builder state to default values.

9.42.3.17 setErrorRecorder()

```
void nvinfer1::IBuilder::setErrorRecorder (
            IErrorRecorder * recorder )  [inline], [noexcept]
```

Set the ErrorRecorder for this interface.

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

If an error recorder is not set, messages will be sent to the global log stream.

Parameters

recorder	The error recorder to register with this interface.

See also

getErrorRecorder()

9.42.3.18 setGpuAllocator()

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

Set the GPU allocator.

Parameters

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

Default: allocateAsync uses cudaMallocAsync if cudaDevAttrMemoryPoolsSupported returns true, otherwise falls back to cudaMalloc. allocate always uses cudaMalloc.

Note

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

9.42.3.19 setMaxThreads()

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

Set the maximum number of threads.

Parameters

maxThreads	The maximum number of threads that can be used by the builder.

Returns

True if successful, false otherwise.

The default value is 1 and includes the current thread. A value greater than 1 permits TensorRT to use multi-threaded algorithms. A value less than 1 triggers a kINVALID_ARGUMENT error.

9.42.4 Member Data Documentation

9.42.4.1 mImpl

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

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

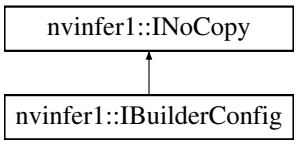
- NvInfer.h

9.43 nvinfer1::IBuilderConfig Class Reference

Holds properties for configuring a builder to produce an engine.

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

Inheritance diagram for nvinfer1::IBuilderConfig:



Public Member Functions

- virtual ~IBuilderConfig () noexcept=default
- virtual 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.

- TRT_DEPRECATED void setInt8Calibrator (IInt8Calibrator *calibrator) noexcept

  Set Int8 Calibration interface.
- TRT_DEPRECATED IInt8Calibrator * getInt8Calibrator () const noexcept

  Get Int8 Calibration interface.
- void setFlags (BuilderFlags builderFlags) noexcept

  Set the build mode flags to turn on builder options for this network.
- BuilderFlags getFlags () const noexcept

  Get the build mode flags for this builder config. Defaults to 0.
- void clearFlag (BuilderFlag builderFlag) noexcept

  clear a single build mode flag.
- void setFlag (BuilderFlag builderFlag) noexcept

  Set a single build mode flag.
- bool getFlag (BuilderFlag builderFlag) const noexcept

  Returns true if the build mode flag is set.
- void setDeviceType (ILayer const *layer, DeviceType deviceType) noexcept

  Set the device that this layer must execute on.
- DeviceType getDeviceType (ILayer const *layer) const noexcept

  Get the device that this layer executes on.
- bool isDeviceTypeSet (ILayer const *layer) const noexcept

  whether the DeviceType has been explicitly set for this layer
- void resetDeviceType (ILayer const *layer) noexcept

  reset the DeviceType for this layer
- bool canRunOnDLA (ILayer const *layer) const noexcept

  Checks if a layer can run on DLA.
- void setDLACore (int32_t dlaCore) noexcept

  Sets the DLA core used by the network. Defaults to -1.
- int32_t getDLACore () const noexcept

  Get the DLA core that the engine executes on.
- void setDefaultDeviceType (DeviceType deviceType) noexcept

  Sets the default DeviceType to be used by the builder. It ensures that all the layers that can run on this device will run on it, unless setDeviceType is used to override the default DeviceType for a layer.
- DeviceType getDefaultDeviceType () const noexcept

  Get the default DeviceType which was set by setDefaultDeviceType.
- void reset () noexcept

  Resets the builder configuration to defaults.
- void setProfileStream (const cudaStream_t stream) noexcept

  Set the CUDA stream that is used to profile this network.
- cudaStream_t getProfileStream () const noexcept

  Get the CUDA stream that is used to profile this network.
- int32_t addOptimizationProfile (IOptimizationProfile const *profile) noexcept

  Add an optimization profile.
- int32_t getNbOptimizationProfiles () const noexcept

  Get number of optimization profiles.
- void setProfilingVerbosity (ProfilingVerbosity verbosity) noexcept

  Set verbosity level of layer information exposed in NVTX annotations and IEngineInspector.
- ProfilingVerbosity getProfilingVerbosity () const noexcept

  Get verbosity level of layer information exposed in NVTX annotations and IEngineInspector.

- TRT_DEPRECATED void setAlgorithmSelector (IAlgorithmSelector *selector) noexcept

  Set Algorithm Selector.
- TRT_DEPRECATED IAlgorithmSelector * getAlgorithmSelector () const noexcept

  Get Algorithm Selector.
- TRT_DEPRECATED bool setCalibrationProfile (IOptimizationProfile const *profile) noexcept

  Add a calibration profile.
- TRT_DEPRECATED IOptimizationProfile const * getCalibrationProfile () noexcept

  Get the current calibration profile.
- TRT_DEPRECATED void setQuantizationFlags (QuantizationFlags flags) noexcept

  Set the quantization flags.
- TRT_DEPRECATED QuantizationFlags getQuantizationFlags () const noexcept

  Get the quantization flags.
- TRT_DEPRECATED void clearQuantizationFlag (QuantizationFlag flag) noexcept

  clear a quantization flag.
- TRT_DEPRECATED void setQuantizationFlag (QuantizationFlag flag) noexcept

  Set a single quantization flag.
- TRT_DEPRECATED bool getQuantizationFlag (QuantizationFlag flag) const noexcept

  Returns true if the quantization flag is set.
- bool setTacticSources (TacticSources tacticSources) noexcept

  Set tactic sources.
- TacticSources getTacticSources () const noexcept

  Get tactic sources.
- nvinfer1::ITimingCache * createTimingCache (void const *blob, std::size_t size) const noexcept

  Create timing cache.
- bool setTimingCache (ITimingCache const &cache, bool ignoreMismatch) noexcept

  Attach a timing cache to IBuilderConfig.
- nvinfer1::ITimingCache const * getTimingCache () const noexcept

  Get the pointer to the timing cache from current IBuilderConfig.
- void setMemoryPoolLimit (MemoryPoolType pool, std::size_t poolSize) noexcept

  Set the memory size for the memory pool.
- std::size_t getMemoryPoolLimit (MemoryPoolType pool) const noexcept

  Get the memory size limit of the memory pool.
- void setPreviewFeature (PreviewFeature feature, bool enable) noexcept

  Enable or disable a specific preview feature.
- bool getPreviewFeature (PreviewFeature feature) const noexcept

  Get status of preview feature.
- void setBuilderOptimizationLevel (int32_t level) noexcept

  Set builder optimization level.
- int32_t getBuilderOptimizationLevel () noexcept

  Get builder optimization level.
- void setHardwareCompatibilityLevel (HardwareCompatibilityLevel hardwareCompatibilityLevel) noexcept

  Set the hardware compatibility level.
- HardwareCompatibilityLevel getHardwareCompatibilityLevel () const noexcept

  Get the hardware compatibility level.
- void setPluginsToSerialize (char const *const *paths, int32_t nbPaths) noexcept

  Set the plugin libraries to be serialized with version-compatible engines.
- char const * getPluginToSerialize (int32_t index) const noexcept

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

- int32_t getNbPluginsToSerialize () const noexcept

  Get the number of plugin library paths to be serialized with version-compatible engines.
- void setMaxAuxStreams (int32_t nbStreams) noexcept

  Set the maximum number of auxiliary streams that TRT is allowed to use.
- int32_t getMaxAuxStreams () const noexcept

  Get the maximum number of auxiliary streams that TRT is allowed to use.
- void setProgressMonitor (IProgressMonitor *monitor) noexcept

  Sets the progress monitor for building a network.
- IProgressMonitor * getProgressMonitor () const noexcept
- void setRuntimePlatform (RuntimePlatform runtimePlatform) noexcept

  Set the target platform for runtime execution.
- RuntimePlatform getRuntimePlatform () const noexcept

  Get the target platform for runtime execution.
- void setMaxNbTactics (int32_t maxNbTactics) noexcept

  Set the maximum number of tactics to time when there is a choice of tactics.
- int32_t getMaxNbTactics () const noexcept

  Query the maximum number of tactics timed when there is a choice.
- bool setTilingOptimizationLevel (TilingOptimizationLevel level) noexcept

  Set the Tiling optimization level.
- TilingOptimizationLevel getTilingOptimizationLevel () const noexcept

  Get the Tiling optimization level.
- bool setL2LimitForTiling (int64_t size) noexcept

  Set the L2 cache usage limit for Tiling optimization.
- int64_t getL2LimitForTiling () const noexcept

  Get the L2 cache usage limit for tiling optimization.

Protected Attributes

- apiv::VBuilderConfig * mImpl

Additional Inherited Members

9.43.1 Detailed Description

Holds properties for configuring a builder to produce an engine.

See also

BuilderFlags

9.43.2 Constructor & Destructor Documentation

9.43.2.1 ~IBuilderConfig()

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

9.43.3 Member Function Documentation

9.43.3.1 addOptimizationProfile()

```
int32_t nvinfer1::IBuilderConfig::addOptimizationProfile (
            IOptimizationProfile const * profile )  [inline], [noexcept]
```

Add an optimization profile.

This function must be called at least once if the network has dynamic or shape input tensors. This function may be called at most once when building a refittable engine, as more than a single optimization profile are not supported for refittable engines.

Parameters

profile	The new optimization profile, which must satisfy profile->isValid() == true

Returns

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

9.43.3.2 canRunOnDLA()

```
bool nvinfer1::IBuilderConfig::canRunOnDLA (
            ILayer const * layer ) const  [inline], [noexcept]
```

Checks if a layer can run on DLA.

Returns

status true if the layer can on DLA else returns false.

9.43.3.3 clearFlag()

```
void nvinfer1::IBuilderConfig::clearFlag (
            BuilderFlag builderFlag )  [inline], [noexcept]
```

clear a single build mode flag.

clears the builder mode flag from the enabled flags.

See also

setFlags()

9.43.3.4 clearQuantizationFlag()

```
TRT_DEPRECATED void nvinfer1::IBuilderConfig::clearQuantizationFlag (
            QuantizationFlag flag )  [inline], [noexcept]
```

clear a quantization flag.

Clears the quantization flag from the enabled quantization flags.

See also

setQuantizationFlags()

Deprecated Deprecated in TensorRT 10.10. Superseded by explicit quantization.

9.43.3.5 createTimingCache()

```
nvinfer1::ITimingCache * nvinfer1::IBuilderConfig::createTimingCache (
            void const * blob,
            std::size_t size ) const  [inline], [noexcept]
```

Create timing cache.

Create ITimingCache instance from serialized raw data. The created timing cache doesn't belong to a specific IBuilderConfig. It can be shared by multiple builder instances. Call setTimingCache() before launching a builder to attach cache to builder instance. The lifetime of the ITimingCache must exceed the lifetime of all builders that use it.

Parameters

blob	A pointer to the raw data that contains serialized timing cache
size	The size in bytes of the serialized timing cache. Size 0 means create a new cache from scratch

See also

setTimingCache

Returns

the pointer to ITimingCache created

9.43.3.6 getAlgorithmSelector()

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

Get Algorithm Selector.

Deprecated Deprecated in TensorRT 10.8. Please use editable mode in ITimingCache instead.

9.43.3.7 getAvgTimingIterations()

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

Query the number of averaging iterations.

By default the number of averaging iterations is 1.

See also

setAvgTimingIterations()

9.43.3.8 getBuilderOptimizationLevel()

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

Get builder optimization level.

Returns

the current builder optimization level

See also

setBuilderOptimizationLevel

9.43.3.9 getCalibrationProfile()

```
TRT_DEPRECATED 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.

Deprecated Deprecated in TensorRT 10.1. Superseded by explicit quantization.

9.43.3.10 getDefaultDeviceType()

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

Get the default DeviceType which was set by setDefaultDeviceType.

By default it returns DeviceType::kGPU.

9.43.3.11 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.

9.43.3.12 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.

9.43.3.13 getEngineCapability()

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

Query EngineCapability flow configured for the builder.

By default it returns EngineCapability::kSTANDARD.

See also

setEngineCapability()

9.43.3.14 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.

9.43.3.15 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()

9.43.3.16 getHardwareCompatibilityLevel()

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

Get the hardware compatibility level.

Returns

hardwareCompatibilityLevel The level of hardware compatibility.

See also

setHardwareCompatiblityLevel()

9.43.3.17 getInt8Calibrator()

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

Get Int8 Calibration interface.

Deprecated Deprecated in TensorRT 10.1. Superseded by explicit quantization.

9.43.3.18 getL2LimitForTiling()

```
int64_t nvinfer1::IBuilderConfig::getL2LimitForTiling ( ) const  [inline], [noexcept]
```

Get the L2 cache usage limit for tiling optimization.

Returns

L2 cache usage limit for tiling optimization.

See also

setL2LimitForTiling()

9.43.3.19 getMaxAuxStreams()

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

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

See also

setMaxAuxStreams()

9.43.3.20 getMaxNbTactics()

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

Query the maximum number of tactics timed when there is a choice.

By default the value is -1, indicating TensorRT can determine the number of tactics based on its own heuristic.

See also

setMaxNbTactics()

9.43.3.21 getMemoryPoolLimit()

```
std::size_t nvinfer1::IBuilderConfig::getMemoryPoolLimit (
            MemoryPoolType pool ) const  [inline], [noexcept]
```

Get the memory size limit of the memory pool.

Retrieve the memory size limit of the corresponding pool in bytes. If setMemoryPoolLimit for the pool has not been called, this returns the default value used by TensorRT. This default value is not necessarily the maximum possible value for that configuration.

Parameters

pool	The memory pool to get the limit for.

Returns

The size of the memory limit, in bytes, for the corresponding pool.

See also

setMemoryPoolLimit

9.43.3.22 getNbOptimizationProfiles()

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

Get number of optimization profiles.

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

Returns

The number of the optimization profiles.

9.43.3.23 getNbPluginsToSerialize()

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

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

Returns

The number of paths.

9.43.3.24 getPluginToSerialize()

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

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

Parameters

index	Index of the plugin library path in the list. Should be in the range `[0, getNbPluginsToSerialize())`.

Returns

The path to the plugin library.

9.43.3.25 getPreviewFeature()

```
bool nvinfer1::IBuilderConfig::getPreviewFeature (
            PreviewFeature feature ) const  [inline], [noexcept]
```

Get status of preview feature.

Parameters

feature	the feature to query

Returns

true if the `feature` is enabled, false otherwise

See also

PreviewFeature, setPreviewFeature

9.43.3.26 getProfileStream()

```
cudaStream_t nvinfer1::IBuilderConfig::getProfileStream ( ) const  [inline], [noexcept]
```

Get the CUDA stream that is used to profile this network.

Returns

The CUDA stream set by setProfileStream, nullptr if setProfileStream has not been called.

See also

setProfileStream()

9.43.3.27 getProfilingVerbosity()

```
ProfilingVerbosity nvinfer1::IBuilderConfig::getProfilingVerbosity ( ) const  [inline], [noexcept]
```

Get verbosity level of layer information exposed in NVTX annotations and IEngineInspector.

Get the current setting of verbosity level of layer information exposed in NVTX annotations and IEngineInspector. Default value is ProfilingVerbosity::kLAYER_NAMES_ONLY.

See also

ProfilingVerbosity, setProfilingVerbosity(), IEngineInspector

9.43.3.28 getProgressMonitor()

```
IProgressMonitor * nvinfer1::IBuilderConfig::getProgressMonitor ( ) const  [inline], [noexcept]
```

Returns

The progress monitor set by the application or nullptr.

See also

IBuilderConfig::setProgressMonitor

9.43.3.29 getQuantizationFlag()

```
TRT_DEPRECATED 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.

Deprecated Deprecated in TensorRT 10.10. Superseded by explicit quantization.

9.43.3.30 getQuantizationFlags()

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

Get the quantization flags.

Returns

The quantization flags as a bitmask.

See also

setQuantizationFlag()

Deprecated Deprecated in TensorRT 10.10. Superseded by explicit quantization.

9.43.3.31 getRuntimePlatform()

```
RuntimePlatform nvinfer1::IBuilderConfig::getRuntimePlatform ( ) const  [inline], [noexcept]
```

Get the target platform for runtime execution.

Returns

The target platform for runtime execution.

See also

IBuilderConfig::setRuntimePlatform()

9.43.3.32 getTacticSources()

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

Get tactic sources.

Get the tactic sources currently set in the engine build configuration.

See also

setTacticSources()

Returns

tactic sources

9.43.3.33 getTilingOptimizationLevel()

```
TilingOptimizationLevel nvinfer1::IBuilderConfig::getTilingOptimizationLevel ( ) const  [inline],
[noexcept]
```

Get the Tiling optimization level.

Returns

TilingOptimizationLevel The level of Tiling optimization.

See also

setTilingOptimizationLevel()

9.43.3.34 getTimingCache()

```
nvinfer1::ITimingCache const * nvinfer1::IBuilderConfig::getTimingCache ( ) const  [inline], [noexcept]
```

Get the pointer to the timing cache from current IBuilderConfig.

Returns

pointer to the timing cache used in current IBuilderConfig

9.43.3.35 isDeviceTypeSet()

```
bool nvinfer1::IBuilderConfig::isDeviceTypeSet (
            ILayer const * layer ) const  [inline], [noexcept]
```

whether the DeviceType has been explicitly set for this layer

Returns

true if device type is not default

See also

setDeviceType() getDeviceType() resetDeviceType()

9.43.3.36 reset()

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

Resets the builder configuration to defaults.

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

9.43.3.37 resetDeviceType()

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

reset the DeviceType for this layer

See also

setDeviceType() getDeviceType() isDeviceTypeSet()

9.43.3.38 setAlgorithmSelector()

```
TRT_DEPRECATED void nvinfer1::IBuilderConfig::setAlgorithmSelector (
            IAlgorithmSelector * selector )  [inline], [noexcept]
```

Set Algorithm Selector.

Parameters

selector	The algorithm selector to be set in the build config.

Deprecated Deprecated in TensorRT 10.8. Please use editable mode in ITimingCache instead.

9.43.3.39 setAvgTimingIterations()

```
virtual void nvinfer1::IBuilderConfig::setAvgTimingIterations (
            int32_t avgTiming )  [inline], [virtual], [noexcept]
```

Set the number of averaging iterations used when timing layers.

When timing layers, the builder minimizes over a set of average times for layer execution. This parameter controls the number of iterations used in averaging.

See also

getAvgTimingIterations()

9.43.3.40 setBuilderOptimizationLevel()

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

Set builder optimization level.

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

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

Below are the descriptions about each builder optimization level:

- Level 0: This enables the fastest compilation by disabling dynamic kernel generation and selecting the first tactic that succeeds in execution. This will also not respect a timing cache.
- Level 1: Available tactics are sorted by heuristics, but only the top are tested to select the best. If a dynamic kernel is generated its compile optimization is low.
- Level 2: Available tactics are sorted by heuristics, but only the fastest tactics are tested to select the best.
- Level 3: Apply heuristics to see if a static precompiled kernel is applicable or if a new one has to be compiled dynamically.
- Level 4: Always compiles a dynamic kernel.
- Level 5: Always compiles a dynamic kernel and compares it to static kernels.

Parameters

level	The optimization level to set to. Must be non-negative.

See also

getBuilderOptimizationLevel

9.43.3.41 setCalibrationProfile()

```
TRT_DEPRECATED bool nvinfer1::IBuilderConfig::setCalibrationProfile (
            IOptimizationProfile const * profile )  [inline], [noexcept]
```

Add a calibration profile.

Calibration optimization profile must be set if int8 calibration is used to set scales for a network with runtime dimensions.

Parameters

profile	The new calibration profile, which must satisfy profile->isValid() == true or be nullptr. MIN and MAX values will be overwritten by kOPT.

Returns

True if the calibration profile was set correctly.

Deprecated Deprecated in TensorRT 10.1. Superseded by explicit quantization.

9.43.3.42 setDefaultDeviceType()

```
void nvinfer1::IBuilderConfig::setDefaultDeviceType (
            DeviceType deviceType )  [inline], [noexcept]
```

Sets the default DeviceType to be used by the builder. It ensures that all the layers that can run on this device will run on it, unless setDeviceType is used to override the default DeviceType for a layer.

See also

getDefaultDeviceType()

9.43.3.43 setDeviceType()

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

Set the device that this layer must execute on.

Parameters

layer	which layer to execute.
deviceType	that this layer must execute on. If DeviceType is not set or is reset, TensorRT will use the default DeviceType set in the builder.

Note

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

See also

getDeviceType()

9.43.3.44 setDLACore()

```
void nvinfer1::IBuilderConfig::setDLACore (
            int32_t dlaCore )  [inline], [noexcept]
```

Sets the DLA core used by the network. Defaults to -1.

Parameters

dlaCore	The DLA core to execute the engine on, in the range [0,getNbDlaCores()).

This function is used to specify which DLA core to use via indexing, if multiple DLA cores are available.

Warning

if getNbDLACores() returns 0, then this function does nothing.

See also

IRuntime::setDLACore() getDLACore()

9.43.3.45 setEngineCapability()

```
void nvinfer1::IBuilderConfig::setEngineCapability (
            EngineCapability capability )  [inline], [noexcept]
```

Configure the builder to target specified EngineCapability flow.

The flow means a sequence of API calls that allow an application to set up a runtime, engine, and execution context in order to run inference.

The supported flows are specified in the EngineCapability enum.

9.43.3.46 setFlag()

```
void nvinfer1::IBuilderConfig::setFlag (
            BuilderFlag builderFlag )  [inline], [noexcept]
```

Set a single build mode flag.

Add the input builder mode flag to the already enabled flags.

See also

setFlags()

9.43.3.47 setFlags()

```
void nvinfer1::IBuilderConfig::setFlags (
            BuilderFlags builderFlags )  [inline], [noexcept]
```

Set the build mode flags to turn on builder options for this network.

The flags are listed in the BuilderFlags enum. The flags set configuration options to build the network.

Parameters

builderFlags	The build option for an engine.

Note

This function will override the previous set flags, rather than bitwise ORing the new flag.

See also

getFlags()

9.43.3.48 setHardwareCompatibilityLevel()

```
void nvinfer1::IBuilderConfig::setHardwareCompatibilityLevel (
            HardwareCompatibilityLevel hardwareCompatibilityLevel )  [inline], [noexcept]
```

Set the hardware compatibility level.

Hardware compatibility allows an engine to run on GPU architectures other than that of the GPU where the engine was built.

The default hardware compatibility level is HardwareCompatibilityLevel::kNONE.

Parameters

hardwareCompatibilityLevel	The level of hardware compatibility.

9.43.3.49 setInt8Calibrator()

```
TRT_DEPRECATED 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.

Deprecated Deprecated in TensorRT 10.1. Superseded by explicit quantization.

9.43.3.50 setL2LimitForTiling()

```
bool nvinfer1::IBuilderConfig::setL2LimitForTiling (
            int64_t size )  [inline], [noexcept]
```

Set the L2 cache usage limit for Tiling optimization.

Parameter for tiling optimization. This API only takes effect when TilingOptimizationLevel is not kNONE.

Note

If setL2LimitForTiling() has not been called, TensorRT would choose a default value between 0 and L2 capacity size.

Parameters

size	The size of the L2 cache usage limit for Tiling optimization.

Returns

True if successful, false otherwise

9.43.3.51 setMaxAuxStreams()

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

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

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

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

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

Note

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

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

Parameters

nbStreams	The maximum number of auxiliary streams that TRT is allowed to use.

See also

getMaxAuxStreams(), ICudaEngine::getNbAuxStreams(), IExecutionContext::setAuxStreams()

9.43.3.52 setMaxNbTactics()

```
void nvinfer1::IBuilderConfig::setMaxNbTactics (
            int32_t maxNbTactics )  [inline], [noexcept]
```

Set the maximum number of tactics to time when there is a choice of tactics.

This function controls the number of tactics timed when there are multiple tactics to choose from.

See also

getMaxNbTactics()

9.43.3.53 setMemoryPoolLimit()

```
void nvinfer1::IBuilderConfig::setMemoryPoolLimit (
            MemoryPoolType pool,
            std::size_t poolSize )  [inline], [noexcept]
```

Set the memory size for the memory pool.

TensorRT layers access different memory pools depending on the operation. This function sets in the IBuilderConfig the size limit, specified by `poolSize`, for the corresponding memory pool, specified by `pool`. TensorRT will build a plan file that is constrained by these limits or report which constraint caused the failure.

If the size of the pool, specified by `poolSize`, fails to meet the size requirements for the pool, this function does nothing and emits the recoverable error, ErrorCode::kINVALID_ARGUMENT, to the registered IErrorRecorder.

If the size of the pool is larger than the maximum possible value for the configuration, this function does nothing and emits ErrorCode::kUNSUPPORTED_STATE.

If the pool does not exist on the requested device type when building the network, a warning is emitted to the logger, and the memory pool value is ignored.

Refer to MemoryPoolType to see the size requirements for each pool.

Parameters

pool	The memory pool to limit the available memory for.
poolSize	The size of the pool in bytes.

See also

getMemoryPoolLimit, MemoryPoolType

9.43.3.54 setPluginsToSerialize()

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

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

Each entry in the list of libraries must be unique.

Parameters

paths	The paths of plugin libraries.
nbPaths	The number of paths.

9.43.3.55 setPreviewFeature()

```
void nvinfer1::IBuilderConfig::setPreviewFeature (
            PreviewFeature feature,
            bool enable )  [inline], [noexcept]
```

Enable or disable a specific preview feature.

Allows enabling or disabling experimental features, which are not enabled by default in the current release.

Refer to PreviewFeature for additional information, and a list of the available features.

Parameters

feature	the feature to enable / disable
enable	true for enable, false for disable

See also

PreviewFeature, getPreviewFeature

9.43.3.56 setProfileStream()

```
void nvinfer1::IBuilderConfig::setProfileStream (
            const cudaStream_t stream )  [inline], [noexcept]
```

Set the CUDA stream that is used to profile this network.

Parameters

stream	The CUDA stream used for profiling by the builder.

See also

getProfileStream()

9.43.3.57 setProfilingVerbosity()

```
void nvinfer1::IBuilderConfig::setProfilingVerbosity (
            ProfilingVerbosity verbosity )  [inline], [noexcept]
```

Set verbosity level of layer information exposed in NVTX annotations and IEngineInspector.

Control how much layer information will be exposed in NVTX annotations and IEngineInspector.

See also

ProfilingVerbosity, getProfilingVerbosity(), IEngineInspector

9.43.3.58 setProgressMonitor()

```
void nvinfer1::IBuilderConfig::setProgressMonitor (
            IProgressMonitor * monitor )  [inline], [noexcept]
```

Sets the progress monitor for building a network.

Parameters

monitor	The progress monitor to assign to the IBuilderConfig.

The progress monitor signals to the application when different phases of the compiler are being executed. Setting to nullptr unsets the monitor so that the application is not signaled.

See also

IBuilderConfig::getProgressMonitor

9.43.3.59 setQuantizationFlag()

```
TRT_DEPRECATED 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()

Deprecated Deprecated in TensorRT 10.10. Superseded by explicit quantization.

9.43.3.60 setQuantizationFlags()

```
TRT_DEPRECATED 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

flags	The quantization flags.

Note

This function will override the previous set flags, rather than bitwise ORing the new flag.

See also

getQuantizationFlags()

Deprecated Deprecated in TensorRT 10.10. Superseded by explicit quantization.

9.43.3.61 setRuntimePlatform()

```
void nvinfer1::IBuilderConfig::setRuntimePlatform (
            RuntimePlatform runtimePlatform )  [inline], [noexcept]
```

Set the target platform for runtime execution.

Cross-platform compatibility allows an engine to be built and executed on different platforms.

The default cross-platform target is RuntimePlatform::kSAME_AS_BUILD.

Parameters

runtimePlatform	The target platform for runtime execution.

See also

IBuilderConfig::getRuntimePlatform()

9.43.3.62 setTacticSources()

```
bool nvinfer1::IBuilderConfig::setTacticSources (
            TacticSources tacticSources )  [inline], [noexcept]
```

Set tactic sources.

This bitset controls which tactic sources TensorRT is allowed to use for tactic selection.

Multiple tactic sources may be combined with a bitwise OR operation. For example, to enable cublas and cublasLt as tactic sources, use a value of:

1U << static_cast<uint32_t>(TacticSource::kCUBLAS) | 1U << static_cast<uint32_t>(TacticSource::kCUBLAS↩_LT)

See also

getTacticSources

Returns

true if the tactic sources in the build configuration were updated. The tactic sources in the build configuration will not be updated if the provided value is invalid.

9.43.3.63 setTilingOptimizationLevel()

```
bool nvinfer1::IBuilderConfig::setTilingOptimizationLevel (
            TilingOptimizationLevel level )  [inline], [noexcept]
```

Set the Tiling optimization level.

Tiling allows TensorRT to try an on-chip caching strategy.

The default getTilingOptimizationLevel is TilingOptimizationLevel::kNONE.

Parameters

level	The level of Tiling optimization.

Returns

True if successful, false otherwise

9.43.3.64 setTimingCache()

```
bool nvinfer1::IBuilderConfig::setTimingCache (
            ITimingCache const & cache,
            bool ignoreMismatch )  [inline], [noexcept]
```

Attach a timing cache to IBuilderConfig.

The timing cache has verification header to make sure the provided cache can be used in current environment. A failure will be reported if the CUDA device property in the provided cache is different from current environment. ignoreMismatch = true skips strict verification and allows loading cache created from a different device.

The cache must not be destroyed until after the engine is built.

Parameters

cache	the timing cache to be used
ignoreMismatch	whether or not allow using a cache that contains different CUDA device property

Returns

true if set successfully, false otherwise

Warning

Using cache generated from devices with different CUDA device properties may lead to functional/performance bugs.

9.43.4 Member Data Documentation

9.43.4.1 mImpl

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

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

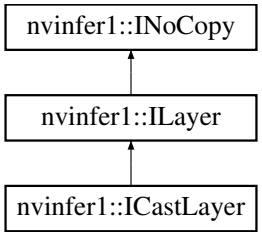
- NvInfer.h

9.44 nvinfer1::ICastLayer Class Reference

A cast layer in a network.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::ICastLayer:



Public Member Functions

- void setToType (DataType toType) noexcept

  Set cast layer output type.
- DataType getToType () const noexcept

  Return cast layer output type.

Protected Member Functions

- virtual ~ICastLayer () noexcept=default

Protected Attributes

- apiv::VCastLayer * mImpl

9.44.1 Detailed Description

A cast layer in a network.

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

9.44.2 Constructor & Destructor Documentation

9.44.2.1 ~ICastLayer()

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

9.44.3 Member Function Documentation

9.44.3.1 getToType()

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

Return cast layer output type.

Returns

toType parameter set during layer creation or by setToType(). The return value is the output type of the cast layer.

9.44.3.2 setToType()

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

Set cast layer output type.

Parameters

toType	The DataType of the output tensor.

Set the output type of the cast layer.

9.44.4 Member Data Documentation

9.44.4.1 mImpl

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

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

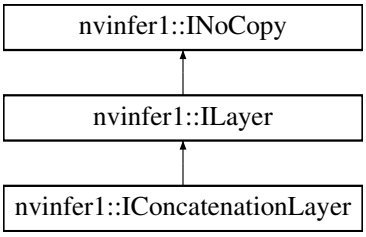
- NvInfer.h

9.45 nvinfer1::IConcatenationLayer Class Reference

A concatenation layer in a network definition.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IConcatenationLayer:



Public Member Functions

- void setAxis (int32_t axis) noexcept

  Set the axis along which concatenation occurs.
- int32_t getAxis () const noexcept

  Get the axis along which concatenation occurs.

Protected Member Functions

- virtual ~IConcatenationLayer () noexcept=default

Protected Attributes

- apiv::VConcatenationLayer * mImpl

9.45.1 Detailed Description

A concatenation layer in a network definition.

The output dimension along the concatenation axis is the sum of the corresponding input dimensions. Every other output dimension is the same as the corresponding dimension of the inputs.

Warning

All tensors must have the same dimensions except along the concatenation axis.

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

9.45.2 Constructor & Destructor Documentation

9.45.2.1 ~IConcatenationLayer()

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

9.45.3 Member Function Documentation

9.45.3.1 getAxis()

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

Get the axis along which concatenation occurs.

See also

setAxis()

9.45.3.2 setAxis()

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

Set the axis along which concatenation occurs.

The default axis is the number of tensor dimensions minus three, or zero if the tensor has fewer than three dimensions. For example, for a tensor with dimensions NCHW, it is C.

When running this layer on the DLA, the concatenation axis must be the third to last axis, e.g. C if tensor dimensions are NCHW.

Parameters

axis	The axis along which concatenation occurs.

9.45.4 Member Data Documentation

9.45.4.1 mImpl

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

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

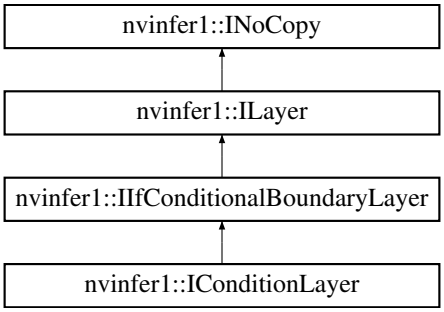
- NvInfer.h

9.46 nvinfer1::IConditionLayer Class Reference

This layer represents a condition input to an IIfConditional.

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

Inheritance diagram for nvinfer1::IConditionLayer:



Protected Member Functions

- virtual ~IConditionLayer () noexcept=default

Protected Attributes

- apiv::VConditionLayer * mImpl

Additional Inherited Members

9.46.1 Detailed Description

This layer represents a condition input to an IIfConditional.

9.46.2 Constructor & Destructor Documentation

9.46.2.1 ~IConditionLayer()

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

9.46.3 Member Data Documentation

9.46.3.1 mImpl

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

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

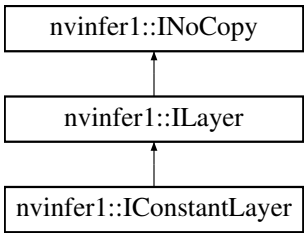
- NvInfer.h

9.47 nvinfer1::IConstantLayer Class Reference

Layer that represents a constant value.

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

Inheritance diagram for nvinfer1::IConstantLayer:



Public Member Functions

- void setWeights (Weights weights) noexcept

  Set the weights for the layer.
- Weights getWeights () const noexcept

  Get the weights for the layer.
- void setDimensions (Dims const &dimensions) noexcept

  Set the dimensions for the layer.
- Dims getDimensions () const noexcept

  Get the dimensions for the layer.

Protected Member Functions

- virtual ~IConstantLayer () noexcept=default

Protected Attributes

- apiv::VConstantLayer * mImpl

9.47.1 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.

9.47.2 Constructor & Destructor Documentation

9.47.2.1 ~IConstantLayer()

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

9.47.3 Member Function Documentation

9.47.3.1 getDimensions()

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

Get the dimensions for the layer.

Returns

the dimensions for the layer

See also

getDimensions

9.47.3.2 getWeights()

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

Get the weights for the layer.

See also

setWeights

9.47.3.3 setDimensions()

```
void nvinfer1::IConstantLayer::setDimensions (
            Dims const & dimensions )  [inline], [noexcept]
```

Set the dimensions for the layer.

Parameters

dimensions	The dimensions of the layer

See also

setDimensions

9.47.3.4 setWeights()

```
void nvinfer1::IConstantLayer::setWeights (
            Weights weights )  [inline], [noexcept]
```

Set the weights for the layer.

The output type is weights.type. If the network is weakly typed and the weights have a real type, the output type might be different per TensorRT's type conversion rules.

See also

getWeights()

9.47.4 Member Data Documentation

9.47.4.1 mImpl

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

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

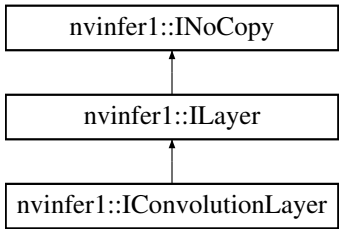
- NvInfer.h

9.48 nvinfer1::IConvolutionLayer Class Reference

A convolution layer in a network definition.

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

Inheritance diagram for nvinfer1::IConvolutionLayer:



Public Member Functions

- void setNbOutputMaps (int64_t nbOutputMaps) noexcept

  Set the number of output maps for the convolution.
- int64_t getNbOutputMaps () const noexcept

  Get the number of output maps for the convolution.
- void setNbGroups (int64_t nbGroups) noexcept

  Set the number of groups for a convolution.
- int64_t getNbGroups () const noexcept

  Get the number of groups of the convolution.
- void setKernelWeights (Weights weights) noexcept

  Set the kernel weights for the convolution.
- Weights getKernelWeights () const noexcept

  Get the kernel weights of the convolution.
- void setBiasWeights (Weights weights) noexcept

  Set the bias weights for the convolution.
- Weights getBiasWeights () const noexcept

  Get the bias weights for the convolution.
- void setPrePadding (Dims const &padding) noexcept

  Set the multi-dimension pre-padding of the convolution.
- Dims getPrePadding () const noexcept

Get the pre-padding.

- void setPostPadding (Dims const &padding) noexcept

  Set the multi-dimension post-padding of the convolution.

- Dims getPostPadding () const noexcept

  Get the post-padding.

- void setPaddingMode (PaddingMode paddingMode) noexcept

  Set the padding mode.

- PaddingMode getPaddingMode () const noexcept

  Get the padding mode.

- void setKernelSizeNd (Dims const &kernelSize) noexcept

  Set the multi-dimension kernel size of the convolution.

- Dims getKernelSizeNd () const noexcept

  Get the multi-dimension kernel size of the convolution.

- void setStrideNd (Dims const &stride) noexcept

  Set the multi-dimension stride of the convolution.

- Dims getStrideNd () const noexcept

  Get the multi-dimension stride of the convolution.

- void setPaddingNd (Dims const &padding) noexcept

  Set the multi-dimension padding of the convolution.

- Dims getPaddingNd () const noexcept

  Get the multi-dimension padding of the convolution.

- void setDilationNd (Dims const &dilation) noexcept

  Set the multi-dimension dilation of the convolution.

- Dims getDilationNd () const noexcept

  Get the multi-dimension dilation of the convolution.

- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~IConvolutionLayer () noexcept=default

Protected Attributes

- apiv::VConvolutionLayer * mImpl

9.48.1 Detailed Description

A convolution layer in a network definition.

This layer performs a correlation operation between 3 or 4 dimensional filter with a 4 or 5 dimensional tensor to produce another 4 or 5 dimensional tensor.

An optional bias argument is supported, which adds a per-channel constant to each value in the output.

Warning

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

9.48.2 Constructor & Destructor Documentation

9.48.2.1 ~IConvolutionLayer()

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

9.48.3 Member Function Documentation

9.48.3.1 getBiasWeights()

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

Get the bias weights for the convolution.

See also

setBiasWeights()

9.48.3.2 getDilationNd()

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

Get the multi-dimension dilation of the convolution.

See also

setDilationNd()

9.48.3.3 getKernelSizeNd()

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

Get the multi-dimension kernel size of the convolution.

See also

setKernelSizeNd()

9.48.3.4 getKernelWeights()

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

Get the kernel weights of the convolution.

See also

setKernelWeights()

9.48.3.5 getNbGroups()

```
int64_t nvinfer1::IConvolutionLayer::getNbGroups ( ) const  [inline], [noexcept]
```

Get the number of groups of the convolution.

See also

setNbGroups()

9.48.3.6 getNbOutputMaps()

```
int64_t nvinfer1::IConvolutionLayer::getNbOutputMaps ( ) const  [inline], [noexcept]
```

Get the number of output maps for the convolution.

See also

setNbOutputMaps()

9.48.3.7 getPaddingMode()

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

Get the padding mode.

Default: kEXPLICIT_ROUND_DOWN

See also

setPaddingMode()

9.48.3.8 getPaddingNd()

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

Get the multi-dimension padding of the convolution.

If the padding is asymmetric, the pre-padding is returned.

See also

setPaddingNd()

9.48.3.9 getPostPadding()

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

Get the post-padding.

See also

setPostPadding()

9.48.3.10 getPrePadding()

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

Get the pre-padding.

See also

setPrePadding()

9.48.3.11 getStrideNd()

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

Get the multi-dimension stride of the convolution.

See also

setStrideNd()

9.48.3.12 setBiasWeights()

```
void nvinfer1::IConvolutionLayer::setBiasWeights (
            Weights weights )  [inline], [noexcept]
```

Set the bias weights for the convolution.

Bias is optional. To omit bias, set the count value of the weights structure to zero.

The bias is applied per-channel, so the number of weights (if non-zero) must be equal to the number of output feature maps.

See also

getBiasWeights()

9.48.3.13 setDilationNd()

```
void nvinfer1::IConvolutionLayer::setDilationNd (
            Dims const & dilation )  [inline], [noexcept]
```

Set the multi-dimension dilation of the convolution.

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

If executing this layer on DLA, only support 2D padding, both height and width must be in the range [1,32].

See also

getDilationNd()

9.48.3.14 setInput()

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

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor

The indices are as follows:

Input 0 is the input activation tensor. Input 1 is the kernel tensor. If used, the kernel weights parameter must be set to empty weights. Input 2 is the bias tensor. If used, the bias parameter must be set to empty weights.

See also

getKernelWeights(), setKernelWeights(), getBiasWeights(), setBiasWeights()

9.48.3.15 setKernelSizeNd()

```
void nvinfer1::IConvolutionLayer::setKernelSizeNd (
            Dims const & kernelSize )  [inline], [noexcept]
```

Set the multi-dimension kernel size of the convolution.

If executing this layer on DLA, only support 2D kernel size, both height and width of kernel size must be in the range [1,32].

See also

getKernelSizeNd()

9.48.3.16 setKernelWeights()

```
void nvinfer1::IConvolutionLayer::setKernelWeights (
            Weights weights )  [inline], [noexcept]
```

Set the kernel weights for the convolution.

The weights are specified as a contiguous array in `GKCRS` order, where `G` is the number of groups, `K` the number of output feature maps, `C` the number of input channels, and `R` and `S` are the height and width of the filter.

See also

getKernelWeights()

9.48.3.17 setNbGroups()

```
void nvinfer1::IConvolutionLayer::setNbGroups (
            int64_t nbGroups ) [inline], [noexcept]
```

Set the number of groups for a convolution.

The input tensor channels are divided into `nbGroups` groups, and a convolution is executed for each group, using a filter per group. The results of the group convolutions are concatenated to form the output.

Note

When using groups in int8 mode, the size of the groups (i.e. the channel count divided by the group count) must be a multiple of 4 for both input and output.

Default: 1

If executing this layer on DLA, the max number of groups is 8192.

See also

getNbGroups()

9.48.3.18 setNbOutputMaps()

```
void nvinfer1::IConvolutionLayer::setNbOutputMaps (
            int64_t nbOutputMaps ) [inline], [noexcept]
```

Set the number of output maps for the convolution.

If executing this layer on DLA, the number of output maps must be in the range [1,8192].

See also

getNbOutputMaps()

9.48.3.19 setPaddingMode()

```
void nvinfer1::IConvolutionLayer::setPaddingMode (
            PaddingMode paddingMode ) [inline], [noexcept]
```

Set the padding mode.

Padding mode takes precedence if both setPaddingMode and setPre/PostPadding are used.

Default: kEXPLICIT_ROUND_DOWN

See also

getPaddingMode()

9.48.3.20 setPaddingNd()

```
void nvinfer1::IConvolutionLayer::setPaddingNd (
            Dims const & padding )  [inline], [noexcept]
```

Set the multi-dimension padding of the convolution.

The input will be zero-padded by this number of elements in each dimension. Padding is symmetric.

Default: (0, 0, ..., 0)

If executing this layer on DLA, only support 2D padding, both height and width of padding must be in the range [0,31], and the padding must be less than the kernel size.

See also

getPaddingNd() setPadding() getPadding()

9.48.3.21 setPostPadding()

```
void nvinfer1::IConvolutionLayer::setPostPadding (
            Dims const & padding )  [inline], [noexcept]
```

Set the multi-dimension post-padding of the convolution.

The end of the input will be zero-padded by this number of elements in each dimension.

Default: (0, 0, ..., 0)

If executing this layer on DLA, only support 2D padding, both height and width of padding must be in the range [0,31], and the padding must be less than the kernel size.

See also

getPostPadding()

9.48.3.22 setPrePadding()

```
void nvinfer1::IConvolutionLayer::setPrePadding (
            Dims const & padding )  [inline], [noexcept]
```

Set the multi-dimension pre-padding of the convolution.

The start of the input will be zero-padded by this number of elements in each dimension.

Default: (0, 0, ..., 0)

If executing this layer on DLA, only support 2D padding, both height and width of padding must be in the range [0,31], and the padding must be less than the kernel size.

See also

getPrePadding()

9.48.3.23 setStrideNd()

```
void nvinfer1::IConvolutionLayer::setStrideNd (
            Dims const & stride )  [inline], [noexcept]
```

Set the multi-dimension stride of the convolution.

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

If executing this layer on DLA, only support 2D stride, both height and width of stride must be in the range [1,8].

See also

getStrideNd()

9.48.4 Member Data Documentation

9.48.4.1 mImpl

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

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

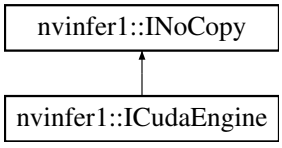
- NvInfer.h

9.49 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
- Dims getTensorShape (char const *tensorName) const noexcept

  Get shape of an input or output tensor.
- DataType getTensorDataType (char const *tensorName) const noexcept

  Determine the required data type for a buffer from its tensor name.
- int32_t getNbLayers () const noexcept

  Get the number of layers in the network.
- IHostMemory * serialize () const noexcept

  Serialize the network to a stream.
- IExecutionContext * createExecutionContext (ExecutionContextAllocationStrategy strategy=ExecutionContextAllocationStrategy noexcept

  Create an execution context and specify the strategy for allocating internal activation memory.
- TensorLocation getTensorLocation (char const *tensorName) const noexcept

  Get whether an input or output tensor must be on GPU or CPU.
- bool isShapeInferenceIO (char const *tensorName) const noexcept

  True if tensor is required as input for shape calculations or is output from shape calculations.
- TensorIOMode getTensorIOMode (char const *tensorName) const noexcept

  Determine whether a tensor is an input or output tensor.
- TRT_DEPRECATED IExecutionContext * createExecutionContextWithoutDeviceMemory () noexcept

  create an execution context without any device memory allocated
- TRT_DEPRECATED size_t getDeviceMemorySize () const noexcept

  Return the maximum device memory required by the context over all profiles.
- TRT_DEPRECATED size_t getDeviceMemorySizeForProfile (int32_t profileIndex) const noexcept

  Return the maximum device memory required by the context for a profile.
- int64_t getDeviceMemorySizeV2 () const noexcept

  Return the maximum device memory required by the context over all profiles.
- int64_t getDeviceMemorySizeForProfileV2 (int32_t profileIndex) const noexcept

  Return the maximum device memory required by the context for a profile.
- bool isRefittable () const noexcept

  Return true if an engine can be refit.
- int32_t getTensorBytesPerComponent (char const *tensorName) const noexcept

  Return the number of bytes per component of an element, or -1 if the tensor is not vectorized or provided name does not map to an input or output tensor.
- int32_t getTensorBytesPerComponent (char const *tensorName, int32_t profileIndex) const noexcept

  Return the number of bytes per component of an element given of given profile, or -1 if the tensor is not vectorized or provided name does not map to an input or output tensor.
- int32_t getTensorComponentsPerElement (char const *tensorName) const noexcept

  Return the number of components included in one element, or -1 if tensor is not vectorized or if the provided name does not map to an input or output tensor.
- int32_t getTensorComponentsPerElement (char const *tensorName, int32_t profileIndex) const noexcept

  Return the number of components included in one element of given profile, or -1 if tensor is not vectorized or the provided name does not map to an input or output tensor.
- TensorFormat getTensorFormat (char const *tensorName) const noexcept

  Return the tensor format, or TensorFormat::kLINEAR if the provided name does not map to an input or output tensor.
- TensorFormat getTensorFormat (char const *tensorName, int32_t profileIndex) const noexcept

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

- char const * getTensorFormatDesc (char const *tensorName) const noexcept

  Return the human readable description of the tensor format, or empty string if the provided name does not map to an input or output tensor.
- char const * getTensorFormatDesc (char const *tensorName, int32_t profileIndex) const noexcept

  Return the human readable description of the tensor format of given profile, or empty string if the provided name does not map to an input or output tensor.
- int32_t getTensorVectorizedDim (char const *tensorName) const noexcept

  Return the dimension index that the buffer is vectorized, or -1 if the provided name does not map to an input or output tensor.
- int32_t getTensorVectorizedDim (char const *tensorName, int32_t profileIndex) const noexcept

  Return the dimension index that the buffer is vectorized of given profile, or -1 if the provided name does not map to an input or output tensor.
- char const * getName () const noexcept

  Returns the name of the network associated with the engine.
- int32_t getNbOptimizationProfiles () const noexcept

  Get the number of optimization profiles defined for this engine.
- Dims getProfileShape (char const *tensorName, int32_t profileIndex, OptProfileSelector select) const noexcept

  Get the minimum / optimum / maximum dimensions for an input tensor given its name under an optimization profile.
- int32_t const * getProfileTensorValues (char const *tensorName, int32_t profileIndex, OptProfileSelector select) const noexcept

  Get the minimum / optimum / maximum values (not dimensions) for an input tensor given its name under an optimization profile. These correspond to the values set using IOptimizationProfile::setShapeValues when the engine was built.
- 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.
- TRT_DEPRECATED bool hasImplicitBatchDimension () const noexcept

  Query whether the engine was built with an implicit batch dimension.
- TacticSources getTacticSources () const noexcept

  return the tactic sources required by this engine.
- ProfilingVerbosity getProfilingVerbosity () const noexcept

  Return the ProfilingVerbosity the builder config was set to when the engine was built.
- IEngineInspector * createEngineInspector () const noexcept

  Create a new engine inspector which prints the layer information in an engine or an execution context.
- int32_t getNbIOTensors () const noexcept

  Return number of IO tensors.
- char const * getIOTensorName (int32_t index) const noexcept

  Return name of an IO tensor.
- HardwareCompatibilityLevel getHardwareCompatibilityLevel () const noexcept

  Return the hardware compatibility level of this engine.
- int32_t getNbAuxStreams () const noexcept

  Return the number of auxiliary streams used by this engine.
- ISerializationConfig * createSerializationConfig () noexcept

  Create a serialization configuration object.
- IHostMemory * serializeWithConfig (ISerializationConfig &config) const noexcept

  Serialize the network to a stream with the provided SerializationConfig.

- TRT_DEPRECATED bool setWeightStreamingBudget (int64_t gpuMemoryBudget) noexcept

  Limit the maximum amount of GPU memory usable for network weights in bytes.
- TRT_DEPRECATED int64_t getWeightStreamingBudget () const noexcept

  Returns the current weight streaming device memory budget in bytes.
- TRT_DEPRECATED int64_t getMinimumWeightStreamingBudget () const noexcept

  The minimum number of bytes of GPU memory required by network weights for successful weight streaming.
- int64_t getStreamableWeightsSize () const noexcept

  Get the total size in bytes of all streamable weights.
- bool setWeightStreamingBudgetV2 (int64_t gpuMemoryBudget) noexcept

  Limit the maximum amount of GPU memory usable for network weights in bytes.
- int64_t getWeightStreamingBudgetV2 () const noexcept

  Returns the current weight streaming device memory budget in bytes.
- int64_t getWeightStreamingAutomaticBudget () const noexcept

  TensorRT automatically determines a device memory budget for the model to run. The budget is close to the current free memory size, leaving some space for other memory needs in the user's application. If the budget exceeds the size obtained from getStreamableWeightsSize(), it is capped to that size, effectively disabling weight streaming. Since TensorRT lacks information about the user's allocations, the remaining memory size might be larger than required, leading to wasted memory, or smaller than required, causing an out-of-memory error. For optimal memory allocation, it is recommended to manually calculate and set the budget.
- int64_t getWeightStreamingScratchMemorySize () const noexcept

  Returns the size of the scratch memory required by the current weight streaming budget.
- bool isDebugTensor (char const *name) const noexcept

  Check if a tensor is marked as a debug tensor.

Protected Attributes

- apiv::VCudaEngine * mImpl

Additional Inherited Members

9.49.1 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.

9.49.2 Constructor & Destructor Documentation

9.49.2.1 ~ICudaEngine()

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

9.49.3 Member Function Documentation

9.49.3.1 createEngineInspector()

```
IEngineInspector * nvinfer1::ICudaEngine::createEngineInspector ( ) const  [inline], [noexcept]
```

Create a new engine inspector which prints the layer information in an engine or an execution context.

See also

IEngineInspector.

9.49.3.2 createExecutionContext()

```
IExecutionContext * nvinfer1::ICudaEngine::createExecutionContext (
            ExecutionContextAllocationStrategy strategy = ExecutionContextAllocationStrategy::kSTATIC
) [inline], [noexcept]
```

Create an execution context and specify the strategy for allocating internal activation memory.

The default value for the allocation strategy is ExecutionContextAllocationStrategy::kSTATIC, which means the context will pre-allocate a block of device memory that is sufficient for all profiles. The newly created execution context will be assigned optimization profile 0. If an error recorder has been set for the engine, it will also be passed to the execution context.

See also

IExecutionContext

IExecutionContext::setOptimizationProfileAsync()

ExecutionContextAllocationStrategy

9.49.3.3 createExecutionContextWithoutDeviceMemory()

```
TRT_DEPRECATED IExecutionContext * nvinfer1::ICudaEngine::createExecutionContextWithoutDevice↩
Memory ( ) [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.

Deprecated Deprecated in TensorRT 10.0. Superseded by createExecutionContext() with parameter.

9.49.3.4 createSerializationConfig()

```
ISerializationConfig * nvinfer1::ICudaEngine::createSerializationConfig ( )  [inline], [noexcept]
```

Create a serialization configuration object.

See also

ISerializationConfig

9.49.3.5 getDeviceMemorySize()

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

Return the maximum device memory required by the context over all profiles.

Deprecated Deprecated in TensorRT 10.1. Superseded by getDeviceMemorySizeV2().

See also

IExecutionContext::setDeviceMemory()

9.49.3.6 getDeviceMemorySizeForProfile()

```
TRT_DEPRECATED size_t nvinfer1::ICudaEngine::getDeviceMemorySizeForProfile (
            int32_t profileIndex ) const  [inline], [noexcept]
```

Return the maximum device memory required by the context for a profile.

Deprecated Deprecated in TensorRT 10.1. Superseded by getDeviceMemorySizeForProfileV2(int32_t).

See also

IExecutionContext::setDeviceMemoryV2()

9.49.3.7 getDeviceMemorySizeForProfileV2()

```
int64_t nvinfer1::ICudaEngine::getDeviceMemorySizeForProfileV2 (
            int32_t profileIndex ) const  [inline], [noexcept]
```

Return the maximum device memory required by the context for a profile.

This API is stateful, so its call returns different values based on the following calls:

- setWeightStreamingBudget()
- setWeightStreamingBudgetV2()

See also

IExecutionContext::setDeviceMemoryV2()
setWeightStreamingBudget()
setWeightStreamingBudgetV2()

9.49.3.8 getDeviceMemorySizeV2()

```
int64_t nvinfer1::ICudaEngine::getDeviceMemorySizeV2 ( ) const  [inline], [noexcept]
```

Return the maximum device memory required by the context over all profiles.

This API is stateful, so its call returns different values based on the following calls:

- setWeightStreamingBudget()
- setWeightStreamingBudgetV2()

See also

IExecutionContext::setDeviceMemoryV2()
setWeightStreamingBudget()
setWeightStreamingBudgetV2()

9.49.3.9 getEngineCapability()

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

Determine what execution capability this engine has.

If the engine has EngineCapability::kSTANDARD, then all engine functionality is valid. If the engine has EngineCapability::kSAFETY, then only the functionality in safe engine is valid. If the engine has EngineCapability::kDLA_STANDALONE, then only serialize, destroy, and const-accessor functions are valid.

Returns

The EngineCapability flag that the engine was built for.

9.49.3.10 getErrorRecorder()

```
IErrorRecorder * nvinfer1::ICudaEngine::getErrorRecorder ( ) const  [inline], [noexcept]
```

Get the ErrorRecorder assigned to this interface.

Retrieves the assigned error recorder object for the given class. A nullptr will be returned if an error handler has not been set.

Returns

A pointer to the IErrorRecorder object that has been registered.

See also

setErrorRecorder()

9.49.3.11 getHardwareCompatibilityLevel()

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

Return the hardware compatibility level of this engine.

Returns

hardwareCompatibilityLevel The level of hardware compatibility.

9.49.3.12 getIOTensorName()

```
char const * nvinfer1::ICudaEngine::getIOTensorName (
            int32_t index ) const  [inline], [noexcept]
```

Return name of an IO tensor.

Parameters

index	value between 0 and getNbIOTensors()-1

See also

getNbIOTensors()

9.49.3.13 getMinimumWeightStreamingBudget()

```
TRT_DEPRECATED int64_t nvinfer1::ICudaEngine::getMinimumWeightStreamingBudget ( ) const  [inline], [noexcept]
```

The minimum number of bytes of GPU memory required by network weights for successful weight streaming.

This is a positive integer for engines with streamable weights because a staging buffer on the GPU is required to temporarily hold the streamed weights. The size of the staging buffer is determined by TensorRT and must be at least as large as the size of the largest streamable weight in the network.

Warning

BuilderFlag::kWEIGHT_STREAMING must be set during engine building.

Returns

The minimum number of bytes of GPU memory required for streaming.

Deprecated Deprecated in TensorRT 10.1. The minimum budget is 0 in the V2 APIs.

See also

setWeightStreamingBudget()

9.49.3.14 getName()

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

Returns the name of the network associated with the engine.

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

See also

INetworkDefinition::setName(), INetworkDefinition::getName()

Returns

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

9.49.3.15 getNbAuxStreams()

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

Return the number of auxiliary streams used by this engine.

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

Returns

the number of auxiliary streams used by this engine.

See also

IBuilderConfig::setMaxAuxStreams(), IExecutionContext::setAuxStreams()

9.49.3.16 getNbIOTensors()

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

Return number of IO tensors.

It is the number of input and output tensors for the network from which the engine was built. The names of the IO tensors can be discovered by calling getIOTensorName(i) for i in 0 to getNbIOTensors()-1.

See also

getIOTensorName()

9.49.3.17 getNbLayers()

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

Get the number of layers in the network.

The number of layers in the network is not necessarily the number in the original network definition, as layers may be combined or eliminated as the engine is optimized. This value can be useful when building per-layer tables, such as when aggregating profiling data over a number of executions.

Returns

The number of layers in the network.

9.49.3.18 getNbOptimizationProfiles()

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

Get the number of optimization profiles defined for this engine.

Returns

Number of optimization profiles. It is always at least 1.

See also

IExecutionContext::setOptimizationProfileAsync()

9.49.3.19 getProfileShape()

```
Dims nvinfer1::ICudaEngine::getProfileShape (
            char const * tensorName,
            int32_t profileIndex,
            OptProfileSelector select ) const  [inline], [noexcept]
```

Get the minimum / optimum / maximum dimensions for an input tensor given its name under an optimization profile.

Parameters

tensorName	The name of an input tensor.
profileIndex	The profile index, which must be between 0 and getNbOptimizationProfiles()-1.
select	Whether to query the minimum, optimum, or maximum dimensions for this input tensor.

Returns

The minimum / optimum / maximum dimensions for an input tensor in this profile. If the profileIndex is invalid or provided name does not map to an input tensor, return Dims{-1, {}}

Warning

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

9.49.3.20 getProfileTensorValues()

```
int32_t const * nvinfer1::ICudaEngine::getProfileTensorValues (
            char const * tensorName,
```

```
    int32_t profileIndex,
    OptProfileSelector select ) const  [inline], [noexcept]
```

Get the minimum / optimum / maximum values (not dimensions) for an input tensor given its name under an optimization profile. These correspond to the values set using IOptimizationProfile::setShapeValues when the engine was built.

Parameters

tensorName	The name of an input tensor.
profileIndex	The profile index, which must be between 0 and getNbOptimizationProfiles()-1.
select	Whether to query the minimum, optimum, or maximum values for this input tensor.

Returns

The minimum / optimum / maximum values for an input tensor in this profile. If the profileIndex is invalid or the provided name does not map to an input tensor, or the tensor is not a shape binding, return nullptr.

Warning

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

9.49.3.21 getProfilingVerbosity()

```
ProfilingVerbosity nvinfer1::ICudaEngine::getProfilingVerbosity ( ) const  [inline], [noexcept]
```

Return the ProfilingVerbosity the builder config was set to when the engine was built.

Returns

the profiling verbosity the builder config was set to when the engine was built.

See also

IBuilderConfig::setProfilingVerbosity()

9.49.3.22 getStreamableWeightsSize()

```
int64_t nvinfer1::ICudaEngine::getStreamableWeightsSize ( ) const  [inline], [noexcept]
```

Get the total size in bytes of all streamable weights.

The set of streamable weights is a subset of all network weights. The total size may exceed free GPU memory.

Returns

The total size in bytes of all streamable weights. Returns 0 if BuilderFlag::kWEIGHT_STREAMING is unset during engine building.

See also

setWeightStreamingBudget()

9.49.3.23 getTacticSources()

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

return the tactic sources required by this engine.

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

See also

IBuilderConfig::setTacticSources()

9.49.3.24 getTensorBytesPerComponent() [1/2]

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

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

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

Parameters

tensorName	The name of an input or output tensor.

Warning

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

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

See also

getTensorVectorizedDim()

getTensorBytesPerComponent(tensorName, profileIndex)

9.49.3.25 getTensorBytesPerComponent() [2/2]

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

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

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

Parameters

tensorName	The name of an input or output tensor.
profileIndex	The profile index to query

Warning

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

See also

getTensorVectorizedDim(tensorName, profileIndex)

9.49.3.26 getTensorComponentsPerElement() [1/2]

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

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

The number of elements in the vectors is returned if getTensorVectorizedDim(tensorName) != -1.

Parameters

tensorName	The name of an input or output tensor.

Warning

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

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

See also

getTensorVectorizedDim()

getTensorComponentsPerElement(tensorName, profileIndex)

9.49.3.27 getTensorComponentsPerElement() [2/2]

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

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

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

Parameters

tensorName	The name of an input or output tensor.
profileIndex	The profile index to query

Warning

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

See also

getTensorVectorizedDim(tensorName, profileIndex)

9.49.3.28 getTensorDataType()

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

Determine the required data type for a buffer from its tensor name.

Parameters

tensorName	The name of an input or output tensor.

Returns

The type of the data in the buffer, or DataType::kFLOAT if the provided name does not map to an input or output tensor.

Warning

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

9.49.3.29 getTensorFormat() [1/2]

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

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

Warning

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

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

See also

getTensorFormat(tensorName, profileIndex)

9.49.3.30 getTensorFormat() [2/2]

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

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

Parameters

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

Warning

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

9.49.3.31 getTensorFormatDesc() [1/2]

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

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

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

Parameters

tensorName	The name of an input or output tensor.

Warning

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

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

9.49.3.32 getTensorFormatDesc() [2/2]

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

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

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

Parameters

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

Warning

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

9.49.3.33 getTensorIOMode()

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

Determine whether a tensor is an input or output tensor.

Parameters

tensorName	The name of an input or output tensor.

Returns

kINPUT if tensorName is an input, kOUTPUT if tensorName is an output, or kNONE if neither.

Warning

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

9.49.3.34 getTensorLocation()

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

Get whether an input or output tensor must be on GPU or CPU.

Parameters

tensorName	The name of an input or output tensor.

Returns

TensorLocation::kDEVICE if tensorName must be on GPU, or TensorLocation::kHOST if on CPU, or TensorLocation::kDEVICE if the provided name does not map to an input or output tensor.

The location is established at build time. E.g. shape tensors inputs are typically required to be on the CPU.

Warning

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

9.49.3.35 getTensorShape()

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

Get shape of an input or output tensor.

Parameters

tensorName	The name of an input or output tensor.

Returns

shape of the tensor, with -1 in place of each dynamic runtime dimension, or Dims{-1, {}} if the provided name does not map to an input or output tensor.

Warning

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

9.49.3.36 getTensorVectorizedDim() [1/2]

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

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

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

Parameters

tensorName	The name of an input or output tensor.

Warning

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

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

9.49.3.37 getTensorVectorizedDim() [2/2]

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

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

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

Parameters

tensorName	The name of an input.
profileIndex	The profile index to query the format for.

Warning

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

9.49.3.38 getWeightStreamingAutomaticBudget()

```
int64_t nvinfer1::ICudaEngine::getWeightStreamingAutomaticBudget ( ) const  [inline], [noexcept]
```

TensorRT automatically determines a device memory budget for the model to run. The budget is close to the current free memory size, leaving some space for other memory needs in the user's application. If the budget exceeds the size obtained from getStreamableWeightsSize(), it is capped to that size, effectively disabling weight streaming. Since TensorRT lacks information about the user's allocations, the remaining memory size might be larger than required, leading to wasted memory, or smaller than required, causing an out-of-memory error. For optimal memory allocation, it is recommended to manually calculate and set the budget.

Warning

BuilderFlag::kWEIGHT_STREAMING must be set during engine building.

The return value may change between TensorRT minor versions.

Setting the returned budget with V1 APIs (setWeightStreamingBudget()) will lead to undefined behavior. Please use V2 APIs.

Returns

The weight streaming budget in bytes. Please set with setWeightStreamingBudgetV2().

See also

BuilderFlag::kWEIGHT_STREAMING

setWeightStreamingBudgetV2()

9.49.3.39 getWeightStreamingBudget()

```
TRT_DEPRECATED int64_t nvinfer1::ICudaEngine::getWeightStreamingBudget ( ) const  [inline], [noexcept]
```

Returns the current weight streaming device memory budget in bytes.

Warning

BuilderFlag::kWEIGHT_STREAMING must be set during engine building.

Returns

The weight streaming budget in bytes. Please see setWeightStreamingBudget() for the possible values.

Deprecated Deprecated in TensorRT 10.1. Superceded by getWeightStreamingBudgetV2().

See also

BuilderFlag::kWEIGHT_STREAMING,

setWeightStreamingBudget()

getMinimumWeightStreamingBudget()

getStreamableWeightsSize()

9.49.3.40 getWeightStreamingBudgetV2()

```
int64_t nvinfer1::ICudaEngine::getWeightStreamingBudgetV2 ( ) const  [inline], [noexcept]
```

Returns the current weight streaming device memory budget in bytes.

Warning

BuilderFlag::kWEIGHT_STREAMING must be set during engine building.

Returns

The weight streaming budget in bytes. Please see setWeightStreamingBudgetV2() for the possible return values. Returns getStreamableWeightsSize() if weight streaming is disabled.

See also

BuilderFlag::kWEIGHT_STREAMING

setWeightStreamingBudget()

getMinimumWeightStreamingBudget()

getStreamableWeightsSize()

9.49.3.41 getWeightStreamingScratchMemorySize()

```
int64_t nvinfer1::ICudaEngine::getWeightStreamingScratchMemorySize ( ) const  [inline], [noexcept]
```

Returns the size of the scratch memory required by the current weight streaming budget.

Weight streaming requires small amounts of scratch memory on the GPU to stage CPU weights right before execution. This value is typically much smaller than the total streamable weights size. Each IExecutionContext will then allocate this additional memory or the user can provide the additional memory through getDeviceMemorySizeV2() and IExecutionContext::setDeviceMemoryV2().

The return value of this call depends on

1. setWeightStreamingBudget()
2. setWeightStreamingBudgetV2()

> Warning
>
> BuilderFlag::kWEIGHT_STREAMING must be set during engine building.

Returns

The weight streaming scratch memory in bytes. Returns 0 if weight streaming is disabled.

See also

BuilderFlag::kWEIGHT_STREAMING
setWeightStreamingBudgetV2()
getStreamableWeightsSize()
getDeviceMemorySizeV2()
getDeviceMemorySizeForProfileV2()
IExecutionContext::setDeviceMemoryV2()

9.49.3.42 hasImplicitBatchDimension()

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

Query whether the engine was built with an implicit batch dimension.

Returns

Always false since TensorRT 10.0 does not support an implicit batch dimension.

See also

createNetworkV2

Deprecated Deprecated in TensorRT 10.0. Implicit batch is no supported since TensorRT 10.0.

9.49.3.43 isDebugTensor()

```
bool nvinfer1::ICudaEngine::isDebugTensor (
            char const * name ) const  [inline], [noexcept]
```

Check if a tensor is marked as a debug tensor.

Determine whether the given name corresponds to a debug tensor.

Returns

True if tensor is a debug tensor, false otherwise.

See also

INetworkDefinition::markDebug

9.49.3.44 isRefittable()

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

Return true if an engine can be refit.

See also

nvinfer1::createInferRefitter()

9.49.3.45 isShapeInferenceIO()

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

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

Return true for either of the following conditions:

- The tensor is a network input, and its value is required for IExecutionContext::getTensorShape() to return the shape of a network output.
- The tensor is a network output, and inferShape() will compute its values.

For example, if a network uses an input tensor "foo" as an addend to an IElementWiseLayer that computes the "reshape dimensions" for IShuffleLayer, then isShapeInferenceIO("foo") == true. If the network copies said input tensor "foo" to an output "bar", then isShapeInferenceIO("bar") == true and IExecutionContext::inferShapes() will write to "bar".

9.49.3.46 serialize()

```
IHostMemory * nvinfer1::ICudaEngine::serialize ( ) const  [inline], [noexcept]
```

Serialize the network to a stream.

Returns

A IHostMemory object that contains the serialized engine.

The network may be deserialized with IRuntime::deserializeCudaEngine().

See also

IRuntime::deserializeCudaEngine()

9.49.3.47 serializeWithConfig()

```
IHostMemory * nvinfer1::ICudaEngine::serializeWithConfig (
            ISerializationConfig & config ) const  [inline], [noexcept]
```

Serialize the network to a stream with the provided SerializationConfig.

Returns

An IHostMemory object that contains the serialized engine.

The network may be deserialized with IRuntime::deserializeCudaEngine(). Serializing plan file with SerializationFlag::kEXCLUDE_WE requires building the engine with kREFIT, kREFIT_IDENTICAL or kREFIT_INDIVIDUAL.

See also

IRuntime::deserializeCudaEngine()

9.49.3.48 setErrorRecorder()

```
void nvinfer1::ICudaEngine::setErrorRecorder (
            IErrorRecorder * recorder )  [inline], [noexcept]
```

Set the ErrorRecorder for this interface.

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

If an error recorder is not set, messages will be sent to the global log stream.

Parameters

recorder	The error recorder to register with this interface.

See also

getErrorRecorder()

9.49.3.49 setWeightStreamingBudget()

```
TRT_DEPRECATED bool nvinfer1::ICudaEngine::setWeightStreamingBudget (
            int64_t gpuMemoryBudget ) [inline], [noexcept]
```

Limit the maximum amount of GPU memory usable for network weights in bytes.

Parameters

gpuMemoryBudget	This parameter may take on 3 types of values: -1: Allows TensorRT to choose the budget according to the streamable weights size. Free CUDA memory will be queried at createExecutionContext() and accordingly: • If streamable weights all fit: weight streaming is not required and disabled. • Otherwise: Budget is set to getMinimumWeightStreamingBudget 0: (default) Disables weight streaming. The execution may fail if the network is too large for GPU memory. >0: The maximum bytes of GPU memory that weights can occupy. It must be bounded by [getMinimumWeightStreamingBudget, free GPU memory)].

By setting a weight limit, users can expect a GPU memory usage reduction of (total bytes for network weights) - gpuMemoryBudget bytes. Maximum memory savings occur when gpuMemoryBudget is set to getMinimumWeightStreamingBudget(). Creating additional IExecutionContexts will increase memory usage by O(getMinimumStreamingBudget()).

Streaming larger amounts of memory will likely result in lower performance except in some boundary cases where streaming weights allows the user to run larger batch sizes. The higher throughput offsets the increased latency in these cases. Tuning the value of the memory limit is recommended for best performance.

Warning

GPU memory for the weights is allocated in this call and will be deallocated by enabling weight streaming or destroying the ICudaEngine.

BuilderFlag::kWEIGHT_STREAMING must be set during engine building.

The weights streaming budget cannot be modified while there are active IExecutionContexts.

Returns

true if the memory limit is valid and the call was successful, false otherwise.

Deprecated Deprecated in TensorRT 10.1. Superceded by setWeightStreamingBudgetV2().

See also

BuilderFlag::kWEIGHT_STREAMING

getWeightStreamingBudget()

getMinimumWeightStreamingBudget()

getStreamableWeightsSize()

9.49.3.50 setWeightStreamingBudgetV2()

```
bool nvinfer1::ICudaEngine::setWeightStreamingBudgetV2 (
            int64_t gpuMemoryBudget ) [inline], [noexcept]
```

Limit the maximum amount of GPU memory usable for network weights in bytes.

Parameters

gpuMemoryBudget	This parameter must be a non-negative value. 0: Only small amounts of scratch memory will required to run the model. >= getStreamableWeightsSize (default): Disables weight streaming. The execution may fail if the network is too large for GPU memory.

By setting a weight limit, users can expect a GPU memory usage reduction on the order of (total bytes for network weights) - gpuMemoryBudget bytes. Maximum memory savings occur when gpuMemoryBudget is set to 0. Each IExecutionContext will require getWeightStreamingScratchMemorySize() bytes of additional device memory if the engine is streaming its weights (budget < getStreamableWeightsSize()).

Streaming larger amounts of memory will likely result in lower performance except in some boundary cases where streaming weights allows the user to run larger batch sizes. The higher throughput offsets the increased latency in these cases. Tuning the value of the memory limit is recommended for best performance.

Warning

GPU memory for the weights is allocated in this call and will be deallocated by enabling weight streaming or destroying the ICudaEngine.

BuilderFlag::kWEIGHT_STREAMING must be set during engine building.

The weights streaming budget cannot be modified while there are active IExecutionContexts.

Using the V2 weight streaming APIs with V1 APIs (setWeightStreamingBudget(), getWeightStreamingBudget(), getWeightStreamingMinimumBudget()) leads to undefined behavior.

Returns

true if the memory limit is valid and the call was successful, false otherwise.

See also

BuilderFlag::kWEIGHT_STREAMING

getWeightStreamingBudgetV2()

getWeightStreamingScratchMemorySize()

getWeightStreamingAutomaticBudget()

getStreamableWeightsSize()

9.49.4 Member Data Documentation

9.49.4.1 mImpl

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

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

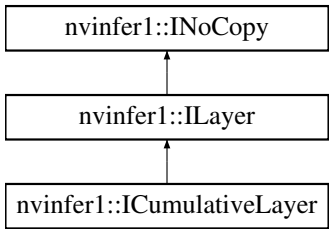
- NvInferRuntime.h

9.50 nvinfer1::ICumulativeLayer Class Reference

Layer that represents a cumulative operation across a tensor.

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

Inheritance diagram for nvinfer1::ICumulativeLayer:



Public Member Functions

- bool setOperation (CumulativeOperation op) noexcept

  Set the cumulative operation for the layer.
- CumulativeOperation getOperation () const noexcept

  Get the cumulative operation for the layer.
- void setExclusive (bool exclusive) noexcept

  Set whether it is an exclusive accumulation or inclusive accumulation.
- bool getExclusive () const noexcept

  Get whether it is exclusive accumulation or inclusive accumulation.
- void setReverse (bool reverse) noexcept

  Specify whether the cumulative operation should be applied backward.
- bool getReverse () const noexcept

  Get the boolean that specifies whether the cumulative operation should be applied backward.

Protected Member Functions

- virtual ~ICumulativeLayer () noexcept=default

Protected Attributes

- apiv::VCumulativeLayer * mImpl

9.50.1 Detailed Description

Layer that represents a cumulative operation across a tensor.

It computes successive reductions across an axis of a tensor. The output always has the same shape as the input.

If the reduction operation is summation, then this is also known as prefix-sum or cumulative sum.

The operation has forward vs. reverse variants, and inclusive vs. exclusive variants.

For example, let the input be a vector x of length n and the output be vector y. Then y[j] = sum(x[...]) where ... denotes a sequence of indices from this table:

```
      | forward   | reverse
```

---------—|----------—| -------— inclusive | 0..j | j..n-1 exclusive | 0..j-1 | j+1..n-1

For multidimensional tensors, the reductions apply across a specified axis. For example, given a 2D input, a forward inclusive cumulative operation across axis 0 generates cumulative sums within each column.

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

9.50.2 Constructor & Destructor Documentation

9.50.2.1 ~ICumulativeLayer()

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

9.50.3 Member Function Documentation

9.50.3.1 getExclusive()

```
bool nvinfer1::ICumulativeLayer::getExclusive ( ) const  [inline], [noexcept]
```

Get whether it is exclusive accumulation or inclusive accumulation.

Returns

Whether the operation will exclude the element at the current index

See also

setExclusive

9.50.3.2 getOperation()

```
CumulativeOperation nvinfer1::ICumulativeLayer::getOperation ( ) const  [inline], [noexcept]
```

Get the cumulative operation for the layer.

Returns

The reduction operation to be performed

See also

setOperation(), CumulativeOperation

9.50.3.3 getReverse()

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

Get the boolean that specifies whether the cumulative operation should be applied backward.

Returns

Whether the cumulative will run in the reverse direction from the last element

See also

setReverse

9.50.3.4 setExclusive()

```
void nvinfer1::ICumulativeLayer::setExclusive (
            bool exclusive )  [inline], [noexcept]
```

Set whether it is an exclusive accumulation or inclusive accumulation.

Parameters

exclusive	Whether the operation will exclude the element at the current index

See also

getExclusive

9.50.3.5 setOperation()

```
bool nvinfer1::ICumulativeLayer::setOperation (
            CumulativeOperation op )  [inline], [noexcept]
```

Set the cumulative operation for the layer.

Parameters

op	The reduction operation to be performed

Returns

Whether `op` is valid and the operation successfully set

See also

getOperation(), CumulativeOperation

9.50.3.6 setReverse()

```
void nvinfer1::ICumulativeLayer::setReverse (
            bool reverse )  [inline], [noexcept]
```

Specify whether the cumulative operation should be applied backward.

Parameters

reverse	Whether the cumulative will run in the reverse direction from the last element

See also

getReverse

9.50.4 Member Data Documentation

9.50.4.1 mImpl

```
apiv::VCumulativeLayer* nvinfer1::ICumulativeLayer::mImpl  [protected]
```

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

- NvInfer.h

9.51 IDebugListener Class Reference

User-implemented callback for notification when value of a debug tensor is updated.

`#include <NvInferRuntime.h>`

9.51.1 Detailed Description

User-implemented callback for notification when value of a debug tensor is updated.

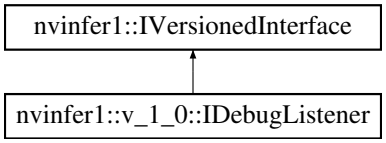
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.52 nvinfer1::v_1_0::IDebugListener Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IDebugListener:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual bool processDebugTensor (void const *addr, TensorLocation location, DataType type, Dims const &shape, char const *name, cudaStream_t stream)=0

  Callback function that is called when a debug tensor's value is updated and the debug state of the tensor is set to true. Content in the given address is only guaranteed to be valid for the duration of the callback.
- ~IDebugListener () override=default

Additional Inherited Members

9.52.1 Constructor & Destructor Documentation

9.52.1.1 ~IDebugListener()

`nvinfer1::v_1_0::IDebugListener::~IDebugListener ( ) [override], [default]`

9.52.2 Member Function Documentation

9.52.2.1 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IDebugListener::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.52.2.2 processDebugTensor()

```
virtual bool nvinfer1::v_1_0::IDebugListener::processDebugTensor (
            void const * addr,
            TensorLocation location,
            DataType type,
            Dims const & shape,
            char const * name,
            cudaStream_t stream )  [pure virtual]
```

Callback function that is called when a debug tensor's value is updated and the debug state of the tensor is set to true. Content in the given address is only guaranteed to be valid for the duration of the callback.

Parameters

location	TensorLocation of the tensor.
addr	pointer to buffer.
type	data Type of the tensor.
shape	shape of the tensor.
name	name of the tensor.
stream	CUDA stream object.

Returns

True on success, false otherwise.

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

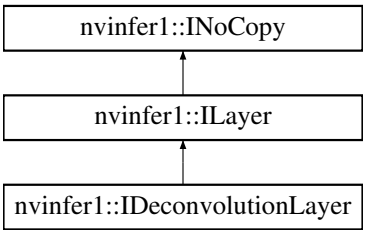
- NvInferRuntime.h

9.53 nvinfer1::IDeconvolutionLayer Class Reference

A deconvolution layer in a network definition.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IDeconvolutionLayer:



Public Member Functions

- void setNbOutputMaps (int64_t nbOutputMaps) noexcept

  Set the number of output feature maps for the deconvolution.
- int64_t getNbOutputMaps () const noexcept

  Get the number of output feature maps for the deconvolution.
- void setNbGroups (int64_t nbGroups) noexcept

  Set the number of groups for a deconvolution.
- int64_t getNbGroups () const noexcept

  Get the number of groups for a deconvolution.
- void setKernelWeights (Weights weights) noexcept

  Set the kernel weights for the deconvolution.
- Weights getKernelWeights () const noexcept

  Get the kernel weights for the deconvolution.
- void setBiasWeights (Weights weights) noexcept

  Set the bias weights for the deconvolution.
- Weights getBiasWeights () const noexcept

  Get the bias weights for the deconvolution.
- void setPrePadding (Dims const &padding) noexcept

  Set the multi-dimension pre-padding of the deconvolution.
- Dims getPrePadding () const noexcept

  Get the pre-padding.
- void setPostPadding (Dims const &padding) noexcept

  Set the multi-dimension post-padding of the deconvolution.
- Dims getPostPadding () const noexcept

  Get the padding.
- void setPaddingMode (PaddingMode paddingMode) noexcept

  Set the padding mode.
- PaddingMode getPaddingMode () const noexcept

  Get the padding mode.
- void setKernelSizeNd (Dims const &kernelSize) noexcept

  Set the multi-dimension kernel size of the deconvolution.
- Dims getKernelSizeNd () const noexcept

  Get the multi-dimension kernel size of the deconvolution.
- void setStrideNd (Dims const &stride) noexcept

Set the multi-dimension stride of the deconvolution.

- Dims getStrideNd () const noexcept

  Get the multi-dimension stride of the deconvolution.
- void setPaddingNd (Dims const &padding) noexcept

  Set the multi-dimension padding of the deconvolution.
- Dims getPaddingNd () const noexcept

  Get the multi-dimension padding of the deconvolution.
- void setDilationNd (Dims const &dilation) noexcept

  Set the multi-dimension dilation of the deconvolution.
- Dims getDilationNd () const noexcept

  Get the multi-dimension dilation of the deconvolution.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~IDeconvolutionLayer () noexcept=default

Protected Attributes

- apiv::VDeconvolutionLayer * mImpl

9.53.1 Detailed Description

A deconvolution layer in a network definition.

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

9.53.2 Constructor & Destructor Documentation

9.53.2.1 ~IDeconvolutionLayer()

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

9.53.3 Member Function Documentation

9.53.3.1 getBiasWeights()

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

Get the bias weights for the deconvolution.

See also

getBiasWeights()

9.53.3.2 getDilationNd()

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

Get the multi-dimension dilation of the deconvolution.

See also

setDilationNd()

9.53.3.3 getKernelSizeNd()

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

Get the multi-dimension kernel size of the deconvolution.

See also

setKernelSizeNd()

9.53.3.4 getKernelWeights()

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

Get the kernel weights for the deconvolution.

See also

setNbGroups()

9.53.3.5 getNbGroups()

```
int64_t nvinfer1::IDeconvolutionLayer::getNbGroups ( ) const  [inline], [noexcept]
```

Get the number of groups for a deconvolution.

See also

setNbGroups()

9.53.3.6 getNbOutputMaps()

```
int64_t nvinfer1::IDeconvolutionLayer::getNbOutputMaps ( ) const  [inline], [noexcept]
```

Get the number of output feature maps for the deconvolution.

See also

setNbOutputMaps()

9.53.3.7 getPaddingMode()

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

Get the padding mode.

Default: kEXPLICIT_ROUND_DOWN

See also

setPaddingMode()

9.53.3.8 getPaddingNd()

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

Get the multi-dimension padding of the deconvolution.

If the padding is asymmetric, the pre-padding is returned.

See also

setPaddingNd()

9.53.3.9 getPostPadding()

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

Get the padding.

See also

setPostPadding()

9.53.3.10 getPrePadding()

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

Get the pre-padding.

See also

setPrePadding()

9.53.3.11 getStrideNd()

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

Get the multi-dimension stride of the deconvolution.

See also

setStrideNd()

9.53.3.12 setBiasWeights()

```
void nvinfer1::IDeconvolutionLayer::setBiasWeights (
            Weights weights )  [inline], [noexcept]
```

Set the bias weights for the deconvolution.

Bias is optional. To omit bias, set the count value of the weights structure to zero.

The bias is applied per-feature-map, so the number of weights (if non-zero) must be equal to the number of output feature maps.

See also

getBiasWeights()

9.53.3.13 setDilationNd()

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

Set the multi-dimension dilation of the deconvolution.

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

See also

getDilationNd()

9.53.3.14 setInput()

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

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor

Input 0 is the input activation tensor. Input 1 is the kernel tensor. If used, the kernel weights parameter must be set to empty weights. Input 2 is the bias tensor. If used, the bias parameter must be set to empty weights.

See also

getKernelWeights(), setKernelWeights(), getBiasWeights(), setBiasWeights()

9.53.3.15 setKernelSizeNd()

```
void nvinfer1::IDeconvolutionLayer::setKernelSizeNd (
            Dims const & kernelSize )  [inline], [noexcept]
```

Set the multi-dimension kernel size of the deconvolution.

If executing this layer on DLA, there are ttwo restrictions: 1) Only 2D Kernel is supported. 2) Kernel height and width must be in the range [1,32] or the combinations of [64, 96, 128] in one dimension and 1 in the other dimensions, i.e. [1x64] or [64x1] are valid, but not [64x64].

See also

getKernelSizeNd()

9.53.3.16 setKernelWeights()

```
void nvinfer1::IDeconvolutionLayer::setKernelWeights (
            Weights weights )  [inline], [noexcept]
```

Set the kernel weights for the deconvolution.

The weights are specified as a contiguous array in `CKRS` order, where `C` the number of input channels, `K` the number of output feature maps, and `R` and `S` are the height and width of the filter.

See also

getWeights()

9.53.3.17 setNbGroups()

```
void nvinfer1::IDeconvolutionLayer::setNbGroups (
            int64_t nbGroups )  [inline], [noexcept]
```

Set the number of groups for a deconvolution.

The input tensor channels are divided into `nbGroups` groups, and a deconvolution is executed for each group, using a filter per group. The results of the group convolutions are concatenated to form the output.

If executing this layer on DLA, nbGroups must be one

Note

When using groups in int8 mode, the size of the groups (i.e. the channel count divided by the group count) must be a multiple of 4 for both input and output.

Default: 1

See also

getNbGroups()

9.53.3.18 setNbOutputMaps()

```
void nvinfer1::IDeconvolutionLayer::setNbOutputMaps (
            int64_t nbOutputMaps )  [inline], [noexcept]
```

Set the number of output feature maps for the deconvolution.

If executing this layer on DLA, the number of output maps must be in the range [1,8192].

See also

getNbOutputMaps()

9.53.3.19 setPaddingMode()

```
void nvinfer1::IDeconvolutionLayer::setPaddingMode (
            PaddingMode paddingMode )  [inline], [noexcept]
```

Set the padding mode.

Padding mode takes precedence if both setPaddingMode and setPre/PostPadding are used.

Default: kEXPLICIT_ROUND_DOWN

See also

getPaddingMode()

9.53.3.20 setPaddingNd()

```
void nvinfer1::IDeconvolutionLayer::setPaddingNd (
            Dims const & padding )  [inline], [noexcept]
```

Set the multi-dimension padding of the deconvolution.

The output will be trimmed by this number of elements on both sides of every dimension. In other words, it resembles the inverse of a convolution layer with this padding size. Padding is symmetric, and negative padding is not supported.

Default: (0, 0, ..., 0)

If executing this layer on DLA, padding must be 0.

See also

getPaddingNd() setPadding() getPadding()

9.53.3.21 setPostPadding()

```
void nvinfer1::IDeconvolutionLayer::setPostPadding (
            Dims const & padding )  [inline], [noexcept]
```

Set the multi-dimension post-padding of the deconvolution.

The output will be trimmed by this number of elements on the end of every dimension. In other words, it resembles the inverse of a convolution layer with this padding size. Negative padding is not supported.

Default: (0, 0, ..., 0)

See also

getPostPadding()

9.53.3.22 setPrePadding()

```
void nvinfer1::IDeconvolutionLayer::setPrePadding (
            Dims const & padding )  [inline], [noexcept]
```

Set the multi-dimension pre-padding of the deconvolution.

The output will be trimmed by this number of elements on the start of every dimension. In other words, it resembles the inverse of a convolution layer with this padding size. Negative padding is not supported.

Default: (0, 0, ..., 0)

See also

getPrePadding()

9.53.3.23 setStrideNd()

```
void nvinfer1::IDeconvolutionLayer::setStrideNd (
            Dims const & stride )  [inline], [noexcept]
```

Set the multi-dimension stride of the deconvolution.

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

If executing this layer on DLA, there are two restrictions: 1) Only 2D Stride is supported. 2) Stride height and width must be in the range [1,32] or the combinations of [64, 96, 128] in one dimension and 1 in the other dimensions, i.e. [1x64] or [64x1] are valid, but not [64x64].

See also

getStrideNd()

9.53.4 Member Data Documentation

9.53.4.1 mImpl

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

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

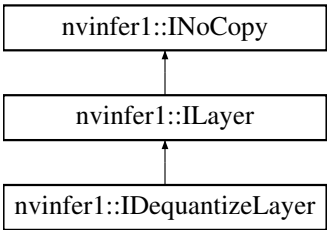
- NvInfer.h

9.54 nvinfer1::IDequantizeLayer Class Reference

A Dequantize layer in a network definition.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IDequantizeLayer:



Public Member Functions

- int32_t getAxis () const noexcept

  Get the quantization axis.
- void setAxis (int32_t axis) noexcept

  Set the quantization axis.
- void setToType (DataType toType) noexcept

  Set the Dequantize layer output type.
- DataType getToType () const noexcept

  Return the Dequantize layer output type.

Protected Member Functions

- virtual ~IDequantizeLayer () noexcept=default

Protected Attributes

- apiv::VDequantizeLayer * mImpl

9.54.1 Detailed Description

A Dequantize layer in a network definition.

This layer accepts a quantized type input tensor, and uses the configured scale and zeroPt inputs to dequantize the input according to: `output` = (`input` - `zeroPt`) * `scale`

The first input (index 0) is the tensor to be quantized. The second (index 1) and third (index 2) are the scale and zero point respectively. `scale` and `zeroPt` should have identical dimensions, and rank lower or equal to 2.

The `zeroPt` tensor is optional, and if not set, will be assumed to be zero. Its data type must be identical to the input's data type. `zeroPt` must only contain zero-valued coefficients, because only symmetric quantization is supported. The `scale` value must be a scalar for per-tensor quantization, a 1D tensor for per-channel quantization, or the same rank as the input tensor for block quantization (supported for DataType::kINT4 only). All `scale` coefficients must have positive values. The size of the 1D `scale` tensor must match the size of the quantization axis. For block quantization, the shape of `scale` tensor must match the shape of the input, except for one dimension (the last or second to last dimension) in which blocking occurs. The size of `zeroPt` must match the size of `scale`.

The subgraph which terminates with the `scale` tensor must be a build-time constant. The same restrictions apply to the `zeroPt`. The output type, if constrained, must be constrained to DataType::kFLOAT, DataType::kHALF, or DataType::kBF16. The input type, if constrained, must be constrained to DataType::kINT8, DataType::kFP8 or DataType::kINT4. The output size is the same as the input size. The quantization axis is in reference to the input tensor's dimensions.

IDequantizeLayer supports DataType::kINT8, DataType::kFP8 or DataType::kINT4 precision and will default to DataType::kINT8 precision during instantiation. For strongly typed networks, `input` data type must be same as `zeroPt` data type.

IDequantizeLayer supports DataType::kFLOAT, DataType::kHALF, or DataType::kBF16 output. For strongly typed networks, `output` data type is inferred from `scale` data type.

As an example of the operation of this layer, imagine a 4D NCHW activation input which can be quantized using a single scale coefficient (referred to as per-tensor quantization): For each n in N: For each c in C: For each h in H: For each w in W: output[n,c,h,w] = (`input`[n,c,h,w] - `zeroPt`) * `scale`

Per-channel dequantization is supported only for input that is rooted at an IConstantLayer (i.e. weights). Activations cannot be quantized per-channel. As an example of per-channel operation, imagine a 4D KCRS weights input and K (dimension 0) as the quantization axis. The scale is an array of coefficients, which is the same size as the quantization axis. For each k in K: For each c in C: For each r in R: For each s in S: output[k,c,r,s] = (`input`[k,c,r,s] - `zeroPt`[k]) * `scale`[k]

Block dequantization is supported only for input tensors with DataType::kINT4 that are rooted at an IConstantLayer (i.e. weights). As an example of blocked operation, imagine a 2D RS weights input with R (dimension 0) as the blocking axis and B as the block size. The scale is a 2D array of coefficients, with dimensions (R//B, S). For each r in R: For each s in S: output[r,s] = (`input`[r,s] - `zeroPt`[r//B, s]) * `scale`[r//B, s]

Note

Only symmetric quantization is supported.

Currently the only allowed build-time constant `scale` and `zeroPt` subgraphs are:

1. Constant -> Quantize
2. Constant -> Cast -> Quantize

The input tensor for this layer must not be a scalar.

Warning

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

9.54.2 Constructor & Destructor Documentation

9.54.2.1 ~IDequantizeLayer()

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

9.54.3 Member Function Documentation

9.54.3.1 getAxis()

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

Get the quantization axis.

Returns

axis parameter set by setAxis(). The return value is the index of the quantization axis in the input tensor's dimensions. A value of -1 indicates per-tensor quantization. The default value is -1.

9.54.3.2 getToType()

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

Return the Dequantize layer output type.

Returns

toType parameter set during layer creation or by setToType(). The return value is the output type of the quantize layer. The default value is DataType::kFLOAT.

9.54.3.3 setAxis()

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

Set the quantization axis.

Set the index of the quantization axis (with reference to the input tensor's dimensions). The axis must be a valid axis if the scale tensor has more than one coefficient. The axis value will be ignored if the scale tensor has exactly one coefficient (per-tensor quantization).

9.54.3.4 setToType()

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

Set the Dequantize layer output type.

Parameters

toType	The DataType of the output tensor.

Set the output type of the dequantize layer. Valid values are DataType::kFLOAT and DataType::kHALF. If the network is strongly typed, setToType must be used to set the output type, and use of setOutputType is an error. Otherwise, types passed to setOutputType and setToType must be the same.

See also

NetworkDefinitionCreationFlag::kSTRONGLY_TYPED

9.54.4 Member Data Documentation

9.54.4.1 mImpl

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

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

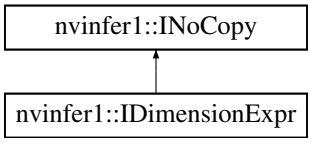
- NvInfer.h

9.55 nvinfer1::IDimensionExpr Class Reference

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

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::IDimensionExpr:



Public Member Functions

- bool isConstant () const noexcept

  Return true if expression is a build-time constant.
- int64_t getConstantValue () const noexcept

  Get the value of the constant.
- bool isSizeTensor () const noexcept

  Return true if this denotes the value of a size tensor.

Protected Member Functions

- virtual ~IDimensionExpr () noexcept=default

Protected Attributes

- apiv::VDimensionExpr * mImpl

9.55.1 Detailed Description

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

Warning

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

See also

DimensionOperation, IPluginV2DynamicExt::getOutputDimensions, IPluginV3OneBuild::getOutputShapes()

9.55.2 Constructor & Destructor Documentation

9.55.2.1 ~IDimensionExpr()

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

9.55.3 Member Function Documentation

9.55.3.1 getConstantValue()

```
int64_t nvinfer1::IDimensionExpr::getConstantValue ( ) const  [inline], [noexcept]
```

Get the value of the constant.

If isConstant(), returns value of the constant. If !isConstant(), return std::numeric_limits<int64_t>::min().

9.55.3.2 isConstant()

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

Return true if expression is a build-time constant.

9.55.3.3 isSizeTensor()

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

Return true if this denotes the value of a size tensor.

Returns

True if this was created with method IExprBuilder::declareSizeTensor, false otherwise

9.55.4 Member Data Documentation

9.55.4.1 mImpl

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

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

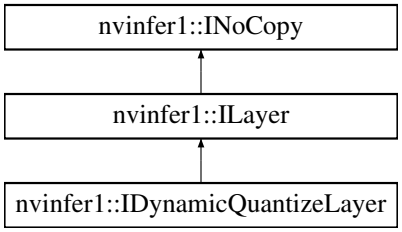
- NvInferRuntime.h

9.56 nvinfer1::IDynamicQuantizeLayer Class Reference

A network layer to perform dynamic quantization.

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

Inheritance diagram for nvinfer1::IDynamicQuantizeLayer:



Public Member Functions

- void setToType (DataType toType) noexcept

  Set DynamicQuantizeLayer's quantized output type.
- DataType getToType () const noexcept

  Return DynamicQuantizeLayer's quantized output type.
- void setScaleType (DataType scaleType) noexcept

  Set the data type of the scale factors used to quantize the data.
- DataType getScaleType () const noexcept

  Return the scale factors data type.
- void setAxis (int32_t axis) noexcept

  Set the axis along which block quantization occurs.
- int32_t getAxis () const noexcept

  Get the axis along which blocking occurs.
- void setBlockSize (int32_t size) noexcept

  Set the size of the quantization block.
- int32_t getBlockSize () const noexcept

  Get the size of the quantization block.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~IDynamicQuantizeLayer () noexcept=default

Protected Attributes

- apiv::VDynamicQuantizeLayer * mImpl

9.56.1 Detailed Description

A network layer to perform dynamic quantization.

This layer accepts a floating-point input tensor and computes the block scale factors needed to quantize the input's data. It outputs the quantized tensor as its first output and the scale factors as its second output.

Use ILayer::setInput to add an input for the double-quantization scale factor.

Note

Only symmetric quantization is supported.

The input tensor for this layer must not be a scalar.

Warning

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

9.56.2 Constructor & Destructor Documentation

9.56.2.1 ~IDynamicQuantizeLayer()

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

9.56.3 Member Function Documentation

9.56.3.1 getAxis()

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

Get the axis along which blocking occurs.

See also

setAxis()

9.56.3.2 getBlockSize()

```
int32_t nvinfer1::IDynamicQuantizeLayer::getBlockSize ( ) const  [inline], [noexcept]
```

Get the size of the quantization block.

See also

setBlockSize()

9.56.3.3 getScaleType()

```
DataType nvinfer1::IDynamicQuantizeLayer::getScaleType ( ) const  [inline], [noexcept]
```

Return the scale factors data type.

Returns

scaleType parameter set during layer creation or by setScaleType().

The return value is the type of the scale factors used to quantize the dynamic data. The default value is DataType::kFP8.

9.56.3.4 getToType()

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

Return DynamicQuantizeLayer's quantized output type.

Returns

toType parameter set during layer creation or by setToType().

The return value is the type of the quantized output tensor. The default value is DataType::kFP4.

9.56.3.5 setAxis()

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

Set the axis along which block quantization occurs.

The axis must be the last dimension or second to last dimension. The input's shape along the axis must be constant.

See also

getAxis()

9.56.3.6 setBlockSize()

```
void nvinfer1::IDynamicQuantizeLayer::setBlockSize (
            int32_t size )  [inline], [noexcept]
```

Set the size of the quantization block.

Note: The block size must divide the input in the blocked axis without remainder. Currently only 16-element blocks are supported.

See also

getBlockSize()

9.56.3.7 setInput()

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

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor

Input 0 is the input activation tensor. Input 1 is the double-quantization scale factor. This scale is used to quantize the dynamically computed high-precision scale factors that are used to quantize the activation data. Currently this input

must be a positive scalar (a 0D tensor).

9.56.3.8 setScaleType()

```
void nvinfer1::IDynamicQuantizeLayer::setScaleType (
            DataType scaleType )  [inline], [noexcept]
```

Set the data type of the scale factors used to quantize the data.

Parameters

scaleType	The scale factors data type.

Set the scale-factors type. Currently the only valid value is DataType::kFP8.

9.56.3.9 setToType()

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

Set DynamicQuantizeLayer's quantized output type.

Parameters

toType	The data type of the quantized output tensor.

Set the type of the dynamic quantization layer's quantized output. Currently the only valid value is DataType::kFP4. If the network is strongly typed, setToType must be used to set the output type, and use of setOutputType is an error. Otherwise, types passed to setOutputType and setToType must be the same.

See also

NetworkDefinitionCreationFlag::kSTRONGLY_TYPED

9.56.4 Member Data Documentation

9.56.4.1 mImpl

```
apiv::VDynamicQuantizeLayer* nvinfer1::IDynamicQuantizeLayer::mImpl  [protected]
```

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

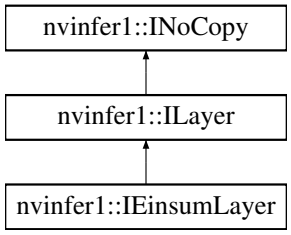
- NvInfer.h

9.57 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

9.57.1 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.

9.57.2 Constructor & Destructor Documentation

9.57.2.1 ~IEinsumLayer()

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

9.57.3 Member Function Documentation

9.57.3.1 getEquation()

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

Return the equation.

See also

setEquation()

9.57.3.2 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()

9.57.4 Member Data Documentation

9.57.4.1 mImpl

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

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

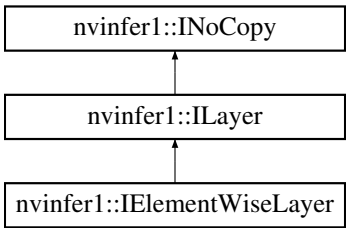
- NvInfer.h

9.58 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

9.58.1 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.

9.58.2 Constructor & Destructor Documentation

9.58.2.1 ~IElementWiseLayer()

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

9.58.3 Member Function Documentation

9.58.3.1 getOperation()

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

Get the binary operation for the layer.

See also

setOperation(), ElementWiseOperation

setBiasWeights()

9.58.3.2 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()

9.58.4 Member Data Documentation

9.58.4.1 mImpl

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

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

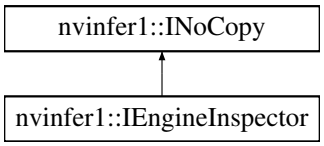
- NvInfer.h

9.59 nvinfer1::IEngineInspector Class Reference

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

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

Inheritance diagram for nvinfer1::IEngineInspector:



Public Member Functions

- virtual ~IEngineInspector () noexcept=default
- bool setExecutionContext (IExecutionContext const *context) noexcept

  Set an execution context as the inspection source.
- IExecutionContext const * getExecutionContext () const noexcept

  Get the context currently being inspected.
- char const * getLayerInformation (int32_t layerIndex, LayerInformationFormat format) const noexcept

  Get a string describing the information about a specific layer in the current engine or the execution context.
- char const * getEngineInformation (LayerInformationFormat format) const noexcept

  Get a string describing the information about all the layers in the current engine or the execution context.
- void setErrorRecorder (IErrorRecorder *recorder) noexcept

  Set the ErrorRecorder for this interface.
- IErrorRecorder * getErrorRecorder () const noexcept

  Get the ErrorRecorder assigned to this interface.

Protected Attributes

- apiv::VEngineInspector * mImpl

Additional Inherited Members

9.59.1 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

9.59.2 Constructor & Destructor Documentation

9.59.2.1 ~IEngineInspector()

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

9.59.3 Member Function Documentation

9.59.3.1 getEngineInformation()

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

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

Parameters

format	the format the layer information should be printed in.

Returns

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

Warning

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

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

See also

LayerInformationFormat

9.59.3.2 getErrorRecorder()

```
IErrorRecorder * nvinfer1::IEngineInspector::getErrorRecorder ( ) const  [inline], [noexcept]
```

Get the ErrorRecorder assigned to this interface.

Retrieves the assigned error recorder object for the given class. A nullptr will be returned if an error handler has not been set.

Returns

A pointer to the IErrorRecorder object that has been registered.

See also

setErrorRecorder()

9.59.3.3 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()

9.59.3.4 getLayerInformation()

```
char const * nvinfer1::IEngineInspector::getLayerInformation (
            int32_t layerIndex,
            LayerInformationFormat format ) const  [inline], [noexcept]
```

Get a string describing the information about a specific layer in the current engine or the execution context.

Parameters

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

Returns

A null-terminated C-style string describing the information about a specific layer in the current engine or the execution context.

Warning

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

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

See also

LayerInformationFormat

9.59.3.5 setErrorRecorder()

```
void nvinfer1::IEngineInspector::setErrorRecorder (
            IErrorRecorder * recorder )  [inline], [noexcept]
```

Set the ErrorRecorder for this interface.

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

If an error recorder is not set, messages will be sent to the global log stream.

Parameters

recorder	The error recorder to register with this interface.

See also

getErrorRecorder()

9.59.3.6 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.

9.59.4 Member Data Documentation

9.59.4.1 mImpl

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

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.60 IErrorRecorder Class Reference

Reference counted application-implemented error reporting interface for TensorRT objects.

```
#include <NvInferRuntimeBase.h>
```

9.60.1 Detailed Description

Reference counted application-implemented error reporting interface for TensorRT objects.

The error reporting mechanism is a user-defined object that interacts with the internal state of the object that it is assigned to in order to determine information about abnormalities in execution. The error recorder gets both an error enum that is more descriptive than pass/fail and also a string description that gives more detail on the exact failure modes. In the safety context, the error strings are all limited to 128 bytes or less in length, including the NULL terminator.

The ErrorRecorder gets passed along to any class that is created from another class that has an ErrorRecorder assigned to it. For example, assigning an ErrorRecorder to an IBuilder allows all INetwork's, ILayer's, and ITensor's to use the same error recorder. For functions that have their own ErrorRecorder accessor functions. This allows registering a different error recorder or de-registering of the error recorder for that specific object.

ErrorRecorder objects that are used in the safety runtime must define an implementation-dependent upper limit of errors whose information can be stored, and drop errors above this upper limit. The limit must fit in int32_t. The IErrorRecorder::hasOverflowed() method is used to signal that one or more errors have been dropped.

The ErrorRecorder object implementation must be thread safe. All locking and synchronization is pushed to the interface implementation and TensorRT does not hold any synchronization primitives when calling the interface functions.

The lifetime of the ErrorRecorder object must exceed the lifetime of all TensorRT objects that use it.

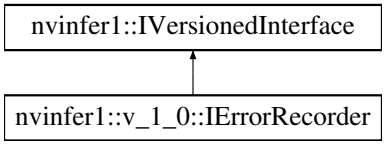
The documentation for this class was generated from the following file:

- NvInferRuntimeBase.h

9.61 nvinfer1::v_1_0::IErrorRecorder Class Reference

`#include <NvInferRuntimeBase.h>`

Inheritance diagram for nvinfer1::v_1_0::IErrorRecorder:



Public Types

- using ErrorDesc = char const *

  A typedef of a C-style string for reporting error descriptions.
- using RefCount = int32_t

  A typedef of a 32-bit integer for reference counting.

Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- IErrorRecorder ()=default
- ~IErrorRecorder () noexcept override=default
- virtual int32_t getNbErrors () const noexcept=0

  Return the number of errors.
- virtual ErrorCode getErrorCode (int32_t errorIdx) const noexcept=0

  Returns the ErrorCode enumeration.
- virtual ErrorDesc getErrorDesc (int32_t errorIdx) const noexcept=0

  Returns a null-terminated C-style string description of the error.
- virtual bool hasOverflowed () const noexcept=0

  Determine if the error stack has overflowed.
- virtual void clear () noexcept=0

  Clear the error stack on the error recorder.
- virtual bool reportError (ErrorCode val, ErrorDesc desc) noexcept=0

  Report an error to the error recorder with the corresponding enum and description.
- virtual RefCount incRefCount () noexcept=0

  Increments the refcount for the current ErrorRecorder.
- virtual RefCount decRefCount () noexcept=0

  Decrements the refcount for the current ErrorRecorder.

Static Public Attributes

- static constexpr size_t kMAX_DESC_LENGTH {127U}

  The length limit for an error description in bytes, excluding the '\0' string terminator. Only applicable to safe runtime. General error recorder implementation can use any size appropriate for the use case.

Additional Inherited Members

9.61.1 Member Typedef Documentation

9.61.1.1 ErrorDesc

```
using nvinfer1::v_1_0::IErrorRecorder::ErrorDesc = char const*
```

A typedef of a C-style string for reporting error descriptions.

9.61.1.2 RefCount

```
using nvinfer1::v_1_0::IErrorRecorder::RefCount = int32_t
```

A typedef of a 32-bit integer for reference counting.

9.61.2 Constructor & Destructor Documentation

9.61.2.1 IErrorRecorder()

```
nvinfer1::v_1_0::IErrorRecorder::IErrorRecorder ( )  [default]
```

9.61.2.2 ~IErrorRecorder()

```
nvinfer1::v_1_0::IErrorRecorder::~IErrorRecorder ( )  [override], [default], [noexcept]
```

9.61.3 Member Function Documentation

9.61.3.1 clear()

```
virtual void nvinfer1::v_1_0::IErrorRecorder::clear ( )  [pure virtual], [noexcept]
```

Clear the error stack on the error recorder.

Removes all the tracked errors by the error recorder. The implementation must guarantee that after this function is called, and as long as no error occurs, the next call to getNbErrors will return zero and hasOverflowed will return false.

See also

getNbErrors(), hasOverflowed()

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.

9.61.3.2 decRefCount()

```
virtual RefCount nvinfer1::v_1_0::IErrorRecorder::decRefCount ( )  [pure virtual], [noexcept]
```

Decrements the refcount for the current ErrorRecorder.

Decrements the reference count for the object by one and returns the current value. This reference count allows the application to know that an object inside of TensorRT has taken a reference to the ErrorRecorder. TensorRT guarantees that every call to IErrorRecorder::decRefCount() will be preceded by a call to IErrorRecorder::incRefCount(). It is undefined behavior to destruct the ErrorRecorder when incRefCount() has been called without a corresponding decRefCount().

Returns

The reference counted value after the decrement completes.

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.

9.61.3.3 getErrorCode()

```
virtual ErrorCode nvinfer1::v_1_0::IErrorRecorder::getErrorCode (
            int32_t errorIdx ) const  [pure virtual], [noexcept]
```

Returns the ErrorCode enumeration.

Parameters

errorIdx	A 32-bit integer that indexes into the error array.

The errorIdx specifies what error code from 0 to getNbErrors()-1 that the application wants to analyze and return the error code enum.

Returns

Returns the enum corresponding to errorIdx if errorIdx is in range (between 0 and getNbErrors()-1). ErrorCode::kUNSPECIFIED_ERROR must be returned if errorIdx is not in range.

See also

getErrorDesc(), ErrorCode

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.

9.61.3.4 getErrorDesc()

```
virtual ErrorDesc nvinfer1::v_1_0::IErrorRecorder::getErrorDesc (
            int32_t errorIdx ) const  [pure virtual], [noexcept]
```

Returns a null-terminated C-style string description of the error.

Parameters

errorIdx	A 32-bit integer that indexes into the error array.

For the error specified by the idx value, return the string description of the error. The error string is a null-terminated C-style string. In the safety context there is a constant length requirement to remove any dynamic memory allocations and the error message will be truncated if it exceeds kMAX_DESC_LENGTH bytes. The format of the string is "<EnumAsStr> - <Description>".

Returns

Returns a string representation of the error along with a description of the error if errorIdx is in range (between 0 and getNbErrors()-1). An empty string will be returned if errorIdx is not in range.

See also

getErrorCode()

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.

9.61.3.5 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IErrorRecorder::getInterfaceInfo ( ) const  [inline], [override], [virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.61.3.6 getNbErrors()

```
virtual int32_t nvinfer1::v_1_0::IErrorRecorder::getNbErrors ( ) const  [pure virtual], [noexcept]
```

Return the number of errors.

Determines the number of errors that occurred between the current point in execution and the last time that the clear() was executed. Due to the possibility of asynchronous errors occurring, a TensorRT API can return correct results, but still register errors with the Error Recorder. The value of getNbErrors() must increment by 1 after each reportError() call until clear() is called, or the maximum number of errors that can be stored is exceeded.

Returns

Returns the number of errors detected, or 0 if there are no errors. If the upper bound of errors that can be stored is exceeded, the upper bound value must be returned.

For example, if the error recorder can store up to 16 error descriptions but reportError() has been called 20 times, getNbErrors() must return 16.

See also

clear(), hasOverflowed()

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.

9.61.3.7 hasOverflowed()

```
virtual bool nvinfer1::v_1_0::IErrorRecorder::hasOverflowed ( ) const  [pure virtual], [noexcept]
```

Determine if the error stack has overflowed.

In the case when the number of errors is large, this function is used to query if one or more errors have been dropped due to lack of storage capacity. This is especially important in the automotive safety case where the internal error handling mechanisms cannot allocate memory.

Returns

true if errors have been dropped due to overflowing the error stack.

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.

9.61.3.8 incRefCount()

```
virtual RefCount nvinfer1::v_1_0::IErrorRecorder::incRefCount ( )  [pure virtual], [noexcept]
```

Increments the refcount for the current ErrorRecorder.

Increments the reference count for the object by one and returns the current value. This reference count allows the application to know that an object inside of TensorRT has taken a reference to the ErrorRecorder. TensorRT guarantees that every call to IErrorRecorder::incRefCount() will be paired with a call to IErrorRecorder::decRefCount() when the reference is released. It is undefined behavior to destruct the ErrorRecorder when incRefCount() has been called without a corresponding decRefCount().

Returns

The reference counted value after the increment completes.

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.

9.61.3.9 reportError()

```
virtual bool nvinfer1::v_1_0::IErrorRecorder::reportError (
            ErrorCode val,
            ErrorDesc desc )  [pure virtual], [noexcept]
```

Report an error to the error recorder with the corresponding enum and description.

Parameters

val	The error code enum that is being reported.
desc	The string description of the error, which will be a NULL-terminated string. For safety use cases its length is limited to kMAX_DESC_LENGTH bytes (excluding the NULL terminator) and descriptions that exceed this limit will be silently truncated.

Report an error to the user that has a given value and human readable description. The function returns false if processing can continue, which implies that the reported error is not fatal. This does not guarantee that processing continues, but provides a hint to TensorRT. The desc C-string data is only valid during the call to reportError and may be immediately deallocated by the caller when reportError returns. The implementation must not store the desc pointer in the ErrorRecorder object or otherwise access the data from desc after reportError returns.

Returns

True if the error is determined to be fatal and processing of the current function must end.

Warning

If the error recorder's maximum number of storable errors is exceeded, the error description will be silently dropped and the value returned by getNbErrors() will not be incremented. However, the return value will still signal whether the error must be considered fatal.

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.

9.61.4 Member Data Documentation

9.61.4.1 kMAX_DESC_LENGTH

```
constexpr size_t nvinfer1::v_1_0::IErrorRecorder::kMAX_DESC_LENGTH {127U}  [static], [constexpr]
```

The length limit for an error description in bytes, excluding the '\0' string terminator. Only applicable to safe runtime. General error recorder implementation can use any size appropriate for the use case.

The documentation for this class was generated from the following file:

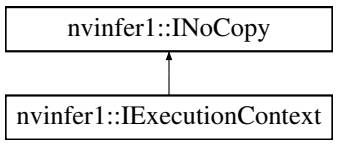
- NvInferRuntimeBase.h

9.62 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
- void setDebugSync (bool sync) noexcept

  Set the debug sync flag.
- bool getDebugSync () const noexcept

  Get the debug sync flag.
- void setProfiler (IProfiler *profiler) noexcept

  Set the profiler.
- IProfiler * getProfiler () const noexcept

  Get the profiler.
- ICudaEngine const & getEngine () const noexcept

  Get the associated engine.
- void setName (char const *name) noexcept

  Set the name of the execution context.
- char const * getName () const noexcept

  Return the name of the execution context.
- void setDeviceMemory (void *memory) noexcept

  Set the device memory for use by this execution context.
- void setDeviceMemoryV2 (void *memory, int64_t size) noexcept

  Set the device memory and its corresponding size for use by this execution context.
- Dims getTensorStrides (char const *tensorName) const noexcept

  Return the strides of the buffer for the given tensor name.
- int32_t getOptimizationProfile () const noexcept

  Get the index of the currently selected optimization profile.
- bool setInputShape (char const *tensorName, Dims const &dims) noexcept

  Set shape of given input.
- Dims getTensorShape (char const *tensorName) const noexcept

  Return the shape of the given input or output.
- bool allInputDimensionsSpecified () const noexcept

  Whether all dynamic dimensions of input tensors have been specified.
- TRT_DEPRECATED 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 a network.
- bool setOptimizationProfileAsync (int32_t profileIndex, cudaStream_t stream) noexcept

  Select an optimization profile for the current context with async semantics.
- void setEnqueueEmitsProfile (bool enqueueEmitsProfile) noexcept

  Set whether enqueue emits layer timing to the profiler.
- bool getEnqueueEmitsProfile () const noexcept

  Get the enqueueEmitsProfile state.
- bool reportToProfiler () const noexcept

  Calculate layer timing info for the current optimization profile in IExecutionContext and update the profiler after one iteration of inference launch.
- bool setTensorAddress (char const *tensorName, void *data) noexcept

  Set memory address for given input or output tensor.
- void const * getTensorAddress (char const *tensorName) const noexcept

  Get memory address bound to given input or output tensor, or nullptr if the provided name does not map to an input or output tensor.
- bool setOutputTensorAddress (char const *tensorName, void *data) noexcept

  Set the memory address for a given output tensor.
- bool setInputTensorAddress (char const *tensorName, void const *data) noexcept

  Set memory address for given input.
- void * getOutputTensorAddress (char const *tensorName) const noexcept

  Get memory address for given output.
- int32_t inferShapes (int32_t nbMaxNames, char const **tensorNames) noexcept

  Run shape calculations.
- size_t updateDeviceMemorySizeForShapes () noexcept

  Recompute the internal activation buffer sizes based on the current input shapes, and return the total amount of memory required.
- bool setInputConsumedEvent (cudaEvent_t event) noexcept

  Mark input as consumed.
- cudaEvent_t getInputConsumedEvent () const noexcept

  The event associated with consuming the input.
- bool setOutputAllocator (char const *tensorName, IOutputAllocator *outputAllocator) noexcept

  Set output allocator to use for output tensor of given name. Pass nullptr to outputAllocator to unset. The allocator is called by enqueueV3().
- IOutputAllocator * getOutputAllocator (char const *tensorName) const noexcept

  Get output allocator associated with output tensor of given name, or nullptr if the provided name does not map to an output tensor.
- int64_t getMaxOutputSize (char const *tensorName) const noexcept

  Get upper bound on an output tensor's size, in bytes, based on the current optimization profile and input dimensions.
- bool setTemporaryStorageAllocator (IGpuAllocator *allocator) noexcept

  Specify allocator to use for internal temporary storage.
- IGpuAllocator * getTemporaryStorageAllocator () const noexcept

  Get allocator set by setTemporaryStorageAllocator.
- bool enqueueV3 (cudaStream_t stream) noexcept

Enqueue inference on a stream.

- void setPersistentCacheLimit (size_t size) noexcept

  Set the maximum size for persistent cache usage.
- size_t getPersistentCacheLimit () const noexcept

  Get the maximum size for persistent cache usage.
- bool setNvtxVerbosity (ProfilingVerbosity verbosity) noexcept

  Set the verbosity of the NVTX markers in the execution context.
- ProfilingVerbosity getNvtxVerbosity () const noexcept

  Get the NVTX verbosity of the execution context.
- void setAuxStreams (cudaStream_t *auxStreams, int32_t nbStreams) noexcept

  Set the auxiliary streams that TensorRT should launch kernels on in the next enqueueV3() call.
- bool setDebugListener (IDebugListener *listener) noexcept

  Set DebugListener for this execution context.
- IDebugListener * getDebugListener () noexcept

  Get the DebugListener of this execution context.
- bool setTensorDebugState (char const *name, bool flag) noexcept

  Set debug state of tensor given the tensor name.
- bool setAllTensorsDebugState (bool flag) noexcept
- bool getDebugState (char const *name) const noexcept

Protected Attributes

- apiv::VExecutionContext * mImpl

Additional Inherited Members

9.62.1 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.

9.62.2 Constructor & Destructor Documentation

9.62.2.1 ~IExecutionContext()

```
virtual nvinfer1::IExecutionContext::~IExecutionContext ( )  [virtual], [default], [noexcept]
```

9.62.3 Member Function Documentation

9.62.3.1 allInputDimensionsSpecified()

```
bool nvinfer1::IExecutionContext::allInputDimensionsSpecified ( ) const  [inline], [noexcept]
```

Whether all dynamic dimensions of input tensors have been specified.

Returns

True if all dynamic dimensions of input tensors have been specified by calling setInputShape().

Trivially true if network has no dynamically shaped input tensors.

Does not work with name-base interfaces eg. IExecutionContext::setInputShape(). Use IExecutionContext::inferShapes() instead.

9.62.3.2 allInputShapesSpecified()

```
TRT_DEPRECATED bool nvinfer1::IExecutionContext::allInputShapesSpecified ( ) const  [inline],
[noexcept]
```

Whether all input shape bindings have been specified.

Returns

True if all input shape bindings have been specified by setInputShapeBinding().

Trivially true if network has no input shape bindings.

Does not work with name-base interfaces eg. IExecutionContext::setInputShape(). Use IExecutionContext::inferShapes() instead.

Deprecated Deprecated in TensorRT 10.0. setInputShapeBinding() is removed since TensorRT 10.0.

9.62.3.3 enqueueV3()

```
bool nvinfer1::IExecutionContext::enqueueV3 (
            cudaStream_t stream )  [inline], [noexcept]
```

Enqueue inference on a stream.

Parameters

stream	A CUDA stream on which the inference kernels will be enqueued.

Returns

True if the kernels were enqueued successfully, false otherwise.

Modifying or releasing memory that has been registered for the tensors before stream synchronization or the event passed to setInputConsumedEvent has been being triggered results in undefined behavior. Input tensor can be released after the setInputConsumedEvent whereas output tensors require stream synchronization.

Warning

Using default stream may lead to performance issues due to additional cudaDeviceSynchronize() calls by TensorRT to ensure correct synchronizations. Please use non-default stream instead.

If the Engine is streaming weights, enqueueV3 will become synchronous, and the graph will not be capturable.

9.62.3.4 executeV2()

```
bool nvinfer1::IExecutionContext::executeV2 (
            void *const * bindings )  [inline], [noexcept]
```

Synchronously execute a network.

This method requires an array of input and output buffers. The mapping from indices to tensor names can be queried using ICudaEngine::getIOTensorName().

Parameters

bindings	An array of pointers to input and output buffers for the network.

Returns

True if execution succeeded.

See also

ICudaEngine::getIOTensorName()

9.62.3.5 getDebugListener()

```
IDebugListener * nvinfer1::IExecutionContext::getDebugListener ( )  [inline], [noexcept]
```

Get the DebugListener of this execution context.

Returns

DebugListener of this execution context.

9.62.3.6 getDebugState()

```
bool nvinfer1::IExecutionContext::getDebugState (
            char const * name ) const  [inline], [noexcept]
```

Get the debug state.

Returns

true if there is a debug tensor with the given name and it has debug state turned on.

9.62.3.7 getDebugSync()

```
bool nvinfer1::IExecutionContext::getDebugSync ( ) const  [inline], [noexcept]
```

Get the debug sync flag.

See also

setDebugSync()

9.62.3.8 getEngine()

```
ICudaEngine const & nvinfer1::IExecutionContext::getEngine ( ) const  [inline], [noexcept]
```

Get the associated engine.

See also

ICudaEngine

9.62.3.9 getEnqueueEmitsProfile()

```
bool nvinfer1::IExecutionContext::getEnqueueEmitsProfile ( ) const  [inline], [noexcept]
```

Get the enqueueEmitsProfile state.

Returns

The enqueueEmitsProfile state.

See also

IExecutionContext::setEnqueueEmitsProfile()

9.62.3.10 getErrorRecorder()

```
IErrorRecorder * nvinfer1::IExecutionContext::getErrorRecorder ( ) const  [inline], [noexcept]
```

Get the ErrorRecorder assigned to this interface.

Retrieves the assigned error recorder object for the given class. A nullptr will be returned if an error handler has not been set.

Returns

A pointer to the IErrorRecorder object that has been registered.

See also

setErrorRecorder()

9.62.3.11 getInputConsumedEvent()

```
cudaEvent_t nvinfer1::IExecutionContext::getInputConsumedEvent ( ) const  [inline], [noexcept]
```

The event associated with consuming the input.

Returns

The CUDA event. Nullptr will be returned if the event is not set yet.

9.62.3.12 getMaxOutputSize()

```
int64_t nvinfer1::IExecutionContext::getMaxOutputSize (
            char const * tensorName ) const  [inline], [noexcept]
```

Get upper bound on an output tensor's size, in bytes, based on the current optimization profile and input dimensions.

If the profile or input dimensions are not yet set, or the provided name does not map to an output, returns -1.

Parameters

tensorName	The name of an output tensor.

Returns

Upper bound in bytes.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

9.62.3.13 getName()

```
char const * nvinfer1::IExecutionContext::getName ( ) const  [inline], [noexcept]
```

Return the name of the execution context.

See also

setName()

9.62.3.14 getNvtxVerbosity()

```
ProfilingVerbosity nvinfer1::IExecutionContext::getNvtxVerbosity ( ) const  [inline], [noexcept]
```

Get the NVTX verbosity of the execution context.

Returns

The current NVTX verbosity of the execution context.

See also

setNvtxVerbosity()

9.62.3.15 getOptimizationProfile()

```
int32_t nvinfer1::IExecutionContext::getOptimizationProfile ( ) const  [inline], [noexcept]
```

Get the index of the currently selected optimization profile.

If the profile index has not been set yet (implicitly to 0 if no other execution context has been set to profile 0, or explicitly for all subsequent contexts), an invalid value of -1 will be returned and all calls to enqueueV3()/executeV2() will fail until a valid profile index has been set. This behavior is deprecated in TensorRT 8.6, all profiles will default to optimization profile 0 and -1 will no longer be returned.

9.62.3.16 getOutputAllocator()

```
IOutputAllocator * nvinfer1::IExecutionContext::getOutputAllocator (
            char const * tensorName ) const  [inline], [noexcept]
```

Get output allocator associated with output tensor of given name, or nullptr if the provided name does not map to an output tensor.

> Warning
>
> The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

See also

IOutputAllocator

9.62.3.17 getOutputTensorAddress()

```
void * nvinfer1::IExecutionContext::getOutputTensorAddress (
            char const * tensorName ) const  [inline], [noexcept]
```

Get memory address for given output.

Parameters

tensorName	The name of an output tensor.

Returns

Raw output data pointer (void∗) for given output tensor, or nullptr if the provided name does not map to an output tensor.

If only a (void const∗) pointer is needed, an alternative is to call method getTensorAddress().

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

See also

getTensorAddress()

9.62.3.18 getPersistentCacheLimit()

```
size_t nvinfer1::IExecutionContext::getPersistentCacheLimit ( ) const  [inline], [noexcept]
```

Get the maximum size for persistent cache usage.

Returns

The size of the persistent cache limit

See also

setPersistentCacheLimit

9.62.3.19 getProfiler()

```
IProfiler * nvinfer1::IExecutionContext::getProfiler ( ) const  [inline], [noexcept]
```

Get the profiler.

See also

IProfiler setProfiler()

9.62.3.20 getTemporaryStorageAllocator()

```
IGpuAllocator * nvinfer1::IExecutionContext::getTemporaryStorageAllocator ( ) const  [inline],
[noexcept]
```

Get allocator set by setTemporaryStorageAllocator.

Returns a nullptr if a nullptr was passed with setTemporaryStorageAllocator().

9.62.3.21 getTensorAddress()

```
void const * nvinfer1::IExecutionContext::getTensorAddress (
            char const * tensorName ) const  [inline], [noexcept]
```

Get memory address bound to given input or output tensor, or nullptr if the provided name does not map to an input or output tensor.

Parameters

tensorName	The name of an input or output tensor.

Use method getOutputTensorAddress() if a non-const pointer for an output tensor is required.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

See also

getOutputTensorAddress()

9.62.3.22 getTensorShape()

```
Dims nvinfer1::IExecutionContext::getTensorShape (
            char const * tensorName ) const  [inline], [noexcept]
```

Return the shape of the given input or output.

Parameters

tensorName	The name of an input or output tensor.

Return Dims{-1, {}} if the provided name does not map to an input or output tensor. Otherwise return the shape of the input or output tensor.

A dimension in an input tensor will have a -1 wildcard value if all the following are true:

- setInputShape() has not yet been called for this tensor
- The dimension is a runtime dimension that is not implicitly constrained to be a single value.

A dimension in an output tensor will have a -1 wildcard value if the dimension depends on values of execution tensors OR if all the following are true:

- It is a runtime dimension.
- setInputShape() has NOT been called for some input tensor(s) with a runtime shape.
- setTensorAddress() has NOT been called for some input tensor(s) with isShapeInferenceIO() = true.

An output tensor may also have -1 wildcard dimensions if its shape depends on values of tensors supplied to enqueueV3().

If the request is for the shape of an output tensor with runtime dimensions, all input tensors with isShapeInferenceIO() = true should have their value already set, since these values might be needed to compute the output shape.

Examples of an input dimension that is implicitly constrained to a single value:

- The optimization profile specifies equal min and max values.
- The dimension is named and only one value meets the optimization profile requirements for dimensions with that name.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

9.62.3.23 getTensorStrides()

```
Dims nvinfer1::IExecutionContext::getTensorStrides (
            char const * tensorName ) const  [inline], [noexcept]
```

Return the strides of the buffer for the given tensor name.

The strides are in units of elements, not components or bytes. For example, for TensorFormat::kHWC8, a stride of one spans 8 scalars.

Note that strides can be different for different execution contexts with dynamic shapes.

If the provided name does not map to an input or output tensor, or there are dynamic dimensions that have not been set yet, return Dims{-1, {}}

Parameters

tensorName	The name of an input or output tensor.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

9.62.3.24 inferShapes()

```
int32_t nvinfer1::IExecutionContext::inferShapes (
            int32_t nbMaxNames,
            char const ** tensorNames )  [inline], [noexcept]
```

Run shape calculations.

Parameters

nbMaxNames	Maximum number of names to write to tensorNames. When the return value is a positive value n and tensorNames != nullptr, the names of min(n,nbMaxNames) insufficiently specified input tensors are written to tensorNames.
tensorNames	Buffer in which to place names of insufficiently specified input tensors.

Returns

0 on success. Positive value n if n input tensors were not sufficiently specified. -1 for other errors.

An input tensor is insufficiently specified if either of the following is true:

- It has dynamic dimensions and its runtime dimensions have not yet been specified via IExecutionContext::setInputShape.
- isShapeInferenceIO(t)=true and the tensor's address has not yet been set.

If an output tensor has isShapeInferenceIO(t)=true and its address has been specified, then its value is written.

Returns -1 if tensorNames == nullptr and nbMaxNames != 0. Returns -1 if nbMaxNames < 0. Returns -1 if a tensor's dimensions are invalid, e.g. a tensor ends up with a negative dimension.

9.62.3.25 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()

9.62.3.26 setAllTensorsDebugState()

```
bool nvinfer1::IExecutionContext::setAllTensorsDebugState (
            bool flag )  [inline], [noexcept]
```

Turn the debug state of all debug tensors on or off.

Parameters

flag	true if turning on debug state, false if turning off debug state.

Returns

true if successful, false otherwise.

The default is off.

9.62.3.27 setAuxStreams()

```
void nvinfer1::IExecutionContext::setAuxStreams (
            cudaStream_t * auxStreams,
            int32_t nbStreams )  [inline], [noexcept]
```

Set the auxiliary streams that TensorRT should launch kernels on in the next enqueueV3() call.

If set, TensorRT will launch the kernels that are supposed to run on the auxiliary streams using the streams provided by the user with this API. If this API is not called before the enqueueV3() call, then TensorRT will use the auxiliary streams created by TensorRT internally.

TensorRT will always insert event synchronizations between the main stream provided via enqueueV3() call and the auxiliary streams:

- At the beginning of the enqueueV3() call, TensorRT will make sure that all the auxiliary streams wait on the activities on the main stream.
- At the end of the enqueueV3() call, TensorRT will make sure that the main stream wait on the activities on all the auxiliary streams.

Parameters

auxStreams	The pointer to an array of cudaStream_t with the array length equal to nbStreams.
nbStreams	The number of auxiliary streams provided. If nbStreams is greater than `engine->getNbAuxStreams()`, then only the first `engine->getNbAuxStreams()` streams will be used. If `nbStreams` is less than `engine->getNbAuxStreams()`, such as setting `nbStreams` to 0, then TensorRT will use the provided streams for the first `nbStreams` auxiliary streams, and will create additional streams internally for the rest of the auxiliary streams.

Note

The provided auxiliary streams must not be the default stream and must all be different to avoid deadlocks.

See also

enqueueV3(), IBuilderConfig::setMaxAuxStreams(), ICudaEngine::getNbAuxStreams()

9.62.3.28 setDebugListener()

```
bool nvinfer1::IExecutionContext::setDebugListener (
            IDebugListener * listener )  [inline], [noexcept]
```

Set DebugListener for this execution context.

Parameters

listener	DebugListener for this execution context.

Returns

true if succeed, false if failure.

9.62.3.29 setDebugSync()

```
void nvinfer1::IExecutionContext::setDebugSync (
            bool sync )  [inline], [noexcept]
```

Set the debug sync flag.

If this flag is set to true, the engine will log the successful execution for each kernel during executeV2(). It has no effect when using enqueueV3().

See also

getDebugSync()

9.62.3.30 setDeviceMemory()

```
void nvinfer1::IExecutionContext::setDeviceMemory (
            void * memory )  [inline], [noexcept]
```

Set the device memory for use by this execution context.

The memory must be aligned with CUDA memory alignment property (using cudaGetDeviceProperties()), and its size must be large enough for performing inference with the given network inputs. getDeviceMemorySize() and get↩ DeviceMemorySizeForProfile() report upper bounds of the size. Setting memory to nullptr is acceptable if the reported size is 0. If using enqueueV3() to run the network, the memory is in use from the invocation of enqueueV3() until network execution is complete. If using executeV2(), it is in use until executeV2() returns. Releasing or otherwise using the memory for other purposes, including using it in another execution context running in parallel, during this time will result in undefined behavior.

Deprecated Deprecated in TensorRT 10.1. Superceded by setDeviceMemoryV2().

Warning

Weight streaming related scratch memory will be allocated by TensorRT if the memory is set by this API. Please use setDeviceMemoryV2() instead.

See also

ICudaEngine::getDeviceMemorySize()

ICudaEngine::getDeviceMemorySizeForProfile()

ExecutionContextAllocationStrategy

ICudaEngine::createExecutionContext()

ICudaEngine::createExecutionContextWithoutDeviceMemory()

9.62.3.31 setDeviceMemoryV2()

```
void nvinfer1::IExecutionContext::setDeviceMemoryV2 (
            void * memory,
            int64_t size )  [inline], [noexcept]
```

Set the device memory and its corresponding size for use by this execution context.

The memory must be aligned with CUDA memory alignment property (using cudaGetDeviceProperties()), and its size must be large enough for performing inference with the given network inputs. getDeviceMemorySize() and get↩ DeviceMemorySizeForProfile() report upper bounds of the size. Setting memory to nullptr is acceptable if the reported size is 0. If using enqueueV3() to run the network, the memory is in use from the invocation of enqueueV3() until network execution is complete. If using executeV2(), it is in use until executeV2() returns. Releasing or otherwise using the memory for other purposes, including using it in another execution context running in parallel, during this time will result in undefined behavior.

See also

ICudaEngine::getDeviceMemorySizeV2()

ICudaEngine::getDeviceMemorySizeForProfileV2()

ExecutionContextAllocationStrategy

ICudaEngine::createExecutionContext()

ICudaEngine::createExecutionContextWithoutDeviceMemory()

9.62.3.32 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()

9.62.3.33 setErrorRecorder()

```
void nvinfer1::IExecutionContext::setErrorRecorder (
            IErrorRecorder * recorder )  [inline], [noexcept]
```

Set the ErrorRecorder for this interface.

Assigns the ErrorRecorder to this interface. The ErrorRecorder will track all errors during execution. This function will call incRefCount of the registered ErrorRecorder at least once. Setting recorder to nullptr unregisters the recorder with the interface, resulting in a call to decRefCount if a recorder has been registered.

If an error recorder is not set, messages will be sent to the global log stream.

Parameters

recorder	The error recorder to register with this interface.

See also

getErrorRecorder()

9.62.3.34 setInputConsumedEvent()

```
bool nvinfer1::IExecutionContext::setInputConsumedEvent (
            cudaEvent_t event )  [inline], [noexcept]
```

Mark input as consumed.

Parameters

event	The CUDA event that is triggered after all input tensors have been consumed.

Warning

The set event must be valid during the inferece.

Returns

True on success, false if error occurred.

Passing event==nullptr removes whatever event was set, if any.

9.62.3.35 setInputShape()

```
bool nvinfer1::IExecutionContext::setInputShape (
            char const * tensorName,
            Dims const & dims )  [inline], [noexcept]
```

Set shape of given input.

Parameters

tensorName	The name of an input tensor.
dims	The shape of an input tensor.

Returns

True on success, false if the provided name does not map to an input tensor, or if some other error occurred.

Each dimension must agree with the network dimension unless the latter was -1.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

9.62.3.36 setInputTensorAddress()

```
bool nvinfer1::IExecutionContext::setInputTensorAddress (
            char const * tensorName,
            void const * data )  [inline], [noexcept]
```

Set memory address for given input.

Parameters

tensorName	The name of an input tensor.
data	The pointer (void const∗) to the const data owned by the user.

Returns

True on success, false if the provided name does not map to an input tensor, does not meet alignment requirements, or some other error occurred.

Input addresses can also be set using method setTensorAddress, which requires a (void∗).

See description of method setTensorAddress() for alignment and data type constraints.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

See also

setTensorAddress()

9.62.3.37 setName()

```
void nvinfer1::IExecutionContext::setName (
            char const * name )  [inline], [noexcept]
```

Set the name of the execution context.

This method copies the name string.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

See also

getName()

9.62.3.38 setNvtxVerbosity()

```
bool nvinfer1::IExecutionContext::setNvtxVerbosity (
            ProfilingVerbosity verbosity )  [inline], [noexcept]
```

Set the verbosity of the NVTX markers in the execution context.

Building with kDETAILED verbosity will generally increase latency in enqueueV3(). Call this method to select NVTX verbosity in this execution context at runtime.

The default is the verbosity with which the engine was built, and the verbosity may not be raised above that level.

This function does not affect how IEngineInspector interacts with the engine.

Parameters

verbosity	The verbosity of the NVTX markers.

Returns

True if the NVTX verbosity is set successfully. False if the provided verbosity level is higher than the profiling verbosity of the corresponding engine.

See also

getNvtxVerbosity()

ICudaEngine::getProfilingVerbosity()

9.62.3.39 setOptimizationProfileAsync()

```
bool nvinfer1::IExecutionContext::setOptimizationProfileAsync (
            int32_t profileIndex,
            cudaStream_t stream )  [inline], [noexcept]
```

Select an optimization profile for the current context with async semantics.

Parameters

profileIndex	Index of the profile. The value must lie between 0 and getEngine().getNbOptimizationProfiles() - 1
stream	A CUDA stream on which the cudaMemcpyAsyncs may be enqueued

When an optimization profile is switched via this API, TensorRT may require that data is copied via cudaMemcpy↩ Async. It is the application's responsibility to guarantee that synchronization between the profile sync stream and the enqueue stream occurs.

The selected profile will be used in subsequent calls to executeV2()/enqueueV3(). If the associated CUDA engine has inputs with dynamic shapes, the optimization profile must be set with its corresponding profileIndex before calling execute or enqueue. The newly created execution context will be assigned optimization profile 0.

If the associated CUDA engine does not have inputs with dynamic shapes, this method need not be called, in which case the default profile index of 0 will be used.

setOptimizationProfileAsync() must be called before calling setInputShape() for all dynamic input tensors or input shape tensors, which in turn must be called before executeV2()/enqueueV3().

Warning

This function will trigger layer resource updates on the next call of executeV2()/enqueueV3(), possibly resulting in performance bottlenecks.

Not synchronizing the stream used at enqueue with the stream used to set optimization profile asynchronously using this API will result in undefined behavior.

Returns

true if the call succeeded, else false (e.g. input out of range)

See also

ICudaEngine::getNbOptimizationProfiles()

9.62.3.40 setOutputAllocator()

```
bool nvinfer1::IExecutionContext::setOutputAllocator (
            char const * tensorName,
            IOutputAllocator * outputAllocator )  [inline], [noexcept]
```

Set output allocator to use for output tensor of given name. Pass nullptr to outputAllocator to unset. The allocator is called by enqueueV3().

Parameters

tensorName	The name of an output tensor.
outputAllocator	IOutputAllocator for the tensors.

Returns

True if success, false if the provided name does not map to an output or, if some other error occurred.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

See also

enqueueV3() IOutputAllocator

9.62.3.41 setOutputTensorAddress()

```
bool nvinfer1::IExecutionContext::setOutputTensorAddress (
            char const * tensorName,
            void * data )  [inline], [noexcept]
```

Set the memory address for a given output tensor.

Parameters

tensorName	The name of an output tensor.
data	The pointer to the buffer to which to write the output.

Returns

True on success, false if the provided name does not map to an output tensor, does not meet alignment requirements, or some other error occurred.

Output addresses can also be set using method setTensorAddress. This method is provided for applications which prefer to use different methods for setting input and output tensors.

See setTensorAddress() for alignment and data type constraints.

> Warning
>
> The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

See also

setTensorAddress()

9.62.3.42 setPersistentCacheLimit()

```
void nvinfer1::IExecutionContext::setPersistentCacheLimit (
            size_t size )  [inline], [noexcept]
```

Set the maximum size for persistent cache usage.

This function sets the maximum persistent L2 cache that this execution context may use for activation caching. Activation caching is not supported on all architectures - see "How TensorRT uses Memory" in the developer guide for details

Parameters

size	the size of persistent cache limitation in bytes. The default is 0 Bytes.

See also

getPersistentCacheLimit

9.62.3.43 setProfiler()

```
void nvinfer1::IExecutionContext::setProfiler (
            IProfiler * profiler )  [inline], [noexcept]
```

Set the profiler.

See also

IProfiler getProfiler()

9.62.3.44 setTemporaryStorageAllocator()

```
bool nvinfer1::IExecutionContext::setTemporaryStorageAllocator (
            IGpuAllocator * allocator )  [inline], [noexcept]
```

Specify allocator to use for internal temporary storage.

This allocator is used only by enqueueV3() for temporary storage whose size cannot be predicted ahead of enqueueV3(). It is not used for output tensors, because memory allocation for those is allocated by the allocator set by setOutputAllocator(). All memory allocated is freed by the time enqueueV3() returns.

Parameters

allocator	pointer to allocator to use. Pass nullptr to revert to using TensorRT's default allocator.

Returns

True on success, false if error occurred.

See also

enqueueV3() setOutputAllocator()

9.62.3.45 setTensorAddress()

```
bool nvinfer1::IExecutionContext::setTensorAddress (
            char const * tensorName,
            void * data )  [inline], [noexcept]
```

Set memory address for given input or output tensor.

Parameters

tensorName	The name of an input or output tensor.
data	The pointer (void∗) to the data owned by the user.

Returns

True on success, false if error occurred.

An address defaults to nullptr. Pass data=nullptr to reset to the default state.

Return false if the provided name does not map to an input or output tensor.

If an input pointer has type (void const∗), use setInputTensorAddress() instead.

Before calling enqueueV3(), each input must have a non-null address and each output must have a non-null address or an IOutputAllocator to set it later.

If the TensorLocation of the tensor is kHOST:

- The pointer must point to a host buffer of sufficient size.
- Data representing shape values is not copied until enqueueV3 is invoked.

If the TensorLocation of the tensor is kDEVICE:

- The pointer must point to a device buffer of sufficient size and alignment, or
- Be nullptr if the tensor is an output tensor that will be allocated by IOutputAllocator.

If getTensorShape(name) reports a -1 for any dimension of an output after all input shapes have been set, use setOutputAllocator() to associate an IOutputAllocator to which the dimensions will be reported when known.

Calling both setTensorAddress and setOutputAllocator() for the same output is allowed, and can be useful for preallocating memory, and then reallocating if it's not big enough.

The pointer must have at least 256-byte alignment.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

See also

setInputTensorAddress() setOutputTensorAddress() getTensorShape() setOutputAllocator() IOutputAllocator

9.62.3.46 setTensorDebugState()

```
bool nvinfer1::IExecutionContext::setTensorDebugState (
            char const * name,
            bool flag )  [inline], [noexcept]
```

Set debug state of tensor given the tensor name.

Turn the debug state of a tensor on or off. A tensor with the parameter tensor name must exist in the network, and the tensor must have been marked as a debug tensor during build time. Otherwise, an error is thrown.

Parameters

name	Name of target tensor.
flag	True if turning on debug state, false if turning off debug state of tensor The default is off.

Returns

True if successful, false otherwise.

9.62.3.47 updateDeviceMemorySizeForShapes()

```
size_t nvinfer1::IExecutionContext::updateDeviceMemorySizeForShapes ( )  [inline], [noexcept]
```

Recompute the internal activation buffer sizes based on the current input shapes, and return the total amount of memory required.

Users can allocate the device memory based on the size returned and provided the memory to TRT with IExecutionContext::setDeviceMemory(). Must specify all input shapes and the optimization profile to use before calling this function, otherwise the partition will be invalidated.

Returns

Total amount of memory required on success, 0 if error occurred.

See also

IExecutionContext::setDeviceMemory()

9.62.4 Member Data Documentation

9.62.4.1 mImpl

```
apiv::VExecutionContext* nvinfer1::IExecutionContext::mImpl  [protected]
```

The documentation for this class was generated from the following file:

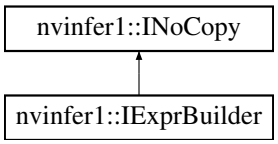
- NvInferRuntime.h

9.63 nvinfer1::IExprBuilder Class Reference

Object for constructing IDimensionExpr.

```
#include <NvInferRuntime.h>
```

Inheritance diagram for nvinfer1::IExprBuilder:



Public Member Functions

- IDimensionExpr const * constant (int64_t value) noexcept

  Return pointer to IDimensionExpr for given value.
- IDimensionExpr const * operation (DimensionOperation op, IDimensionExpr const &first, IDimensionExpr const &second) noexcept

  Get the operation.
- IDimensionExpr const * declareSizeTensor (int32_t outputIndex, IDimensionExpr const &opt, IDimensionExpr const &upper)

  Declare a size tensor at the given output index, with the specified auto-tuning formula and upper bound.

Protected Member Functions

- virtual ~IExprBuilder () noexcept=default

Protected Attributes

- apiv::VExprBuilder * mImpl

9.63.1 Detailed Description

Object for constructing IDimensionExpr.

There is no public way to construct an IExprBuilder. It appears as an argument to method IPluginV2DynamicExt::getOutputDimensions() and IPluginV3OneBuild::getOutputShapes(). Overrides of that method can use that IExprBuilder argument to construct expressions that define output dimensions in terms of input dimensions.

Clients should assume that any values constructed by the IExprBuilder are destroyed after IPluginV2DynamicExt::getOutputDimensions(or IPluginV3OneBuild::getOutputShapes() returns.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

See also

IDimensionExpr

9.63.2 Constructor & Destructor Documentation

9.63.2.1 ~IExprBuilder()

```
virtual nvinfer1::IExprBuilder::~IExprBuilder ( )  [protected], [virtual], [default], [noexcept]
```

9.63.3 Member Function Documentation

9.63.3.1 constant()

```
IDimensionExpr const * nvinfer1::IExprBuilder::constant (
            int64_t value )  [inline], [noexcept]
```

Return pointer to IDimensionExpr for given value.

9.63.3.2 declareSizeTensor()

```
IDimensionExpr const * nvinfer1::IExprBuilder::declareSizeTensor (
            int32_t outputIndex,
            IDimensionExpr const & opt,
            IDimensionExpr const & upper )  [inline]
```

Declare a size tensor at the given output index, with the specified auto-tuning formula and upper bound.

A size tensor allows a plugin to have output dimensions that cannot be computed solely from input dimensions. For example, suppose a plugin implements the equivalent of INonZeroLayer for 2D input. The plugin can have one output for the indices of non-zero elements, and a second output containing the number of non-zero elements. Suppose the input has size [M,N] and has K non-zero elements. The plugin can write K to the second output. When telling TensorRT that the first output has shape [2,K], plugin uses IExprBuilder::constant() and IExprBuilder::declareSizeTensor(1,...) to create the IDimensionExpr that respectively denote 2 and K.

TensorRT also needs to know the value of K to use for auto-tuning and an upper bound on K so that it can allocate memory for the output tensor. In the example, supposed typically half of the plugin's input elements are non-zero, and all the elements might be nonzero. then using M∗N/2 might be a good expression for the opt parameter, and M∗N for the upper bound. IDimensionsExpr for these expressions can be constructed from IDimensionsExpr for the input dimensions.

Parameters

outputIndex	index of a plugin output that is a size tensor.
opt	formula for computing auto-tuning value. Must not depend on a size tensor.
upper	Upper bound on the size tensor.

Returns

IDimensionExpr denoting the value of the size tensor.

See also

IPluginV3OneBuild::getOutputShapes()

9.63.3.3 operation()

```
IDimensionExpr const * nvinfer1::IExprBuilder::operation (
            DimensionOperation op,
            IDimensionExpr const & first,
            IDimensionExpr const & second )  [inline], [noexcept]
```

Get the operation.

Return pointer to IDimensionExpr that represents the given operation applied to first and second. Returns nullptr if op is not a valid DimensionOperation.

9.63.4 Member Data Documentation

9.63.4.1 mImpl

```
apiv::VExprBuilder* nvinfer1::IExprBuilder::mImpl  [protected]
```

The documentation for this class was generated from the following file:

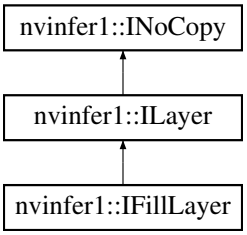
- NvInferRuntime.h

9.64 nvinfer1::IFillLayer Class Reference

Generate a tensor according to a specified mode.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IFillLayer:



Public Member Functions

- void setDimensions (Dims const &dimensions) noexcept

  Set the output tensor's dimensions.
- Dims getDimensions () const noexcept

  Get the output tensor's dimensions.
- void setOperation (FillOperation op) noexcept

  Set the fill operation for the layer.
- FillOperation getOperation () const noexcept

  Get the fill operation for the layer.
- void setAlpha (double alpha) noexcept

  Set the alpha parameter.
- double getAlpha () const noexcept

  Get the value of alpha parameter.
- void setBeta (double beta) noexcept

  Set the beta parameter.

- double getBeta () const noexcept

  Get the value of beta parameter.
- void setAlphaInt64 (int64_t alpha) noexcept

  Set the alpha parameter with int64 datatype.
- int64_t getAlphaInt64 () const noexcept

  Get the value of alpha parameter with int64 datatype.
- void setBetaInt64 (int64_t beta) noexcept

  Set the beta parameter with int64 datatype.
- int64_t getBetaInt64 () const noexcept

  Get the value of beta parameter with int64 datatype.
- bool isAlphaBetaInt64 () const noexcept

  Return true if alpha/beta have type int64, false if they have type double.
- void setToType (DataType toType) noexcept

  Set the fill layer output type.
- DataType getToType () const noexcept

  Get the fill layer output type.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~IFillLayer () noexcept=default

Protected Attributes

- apiv::VFillLayer * mImpl

9.64.1 Detailed Description

Generate a tensor according to a specified mode.

The fill layer generates a tensor with values that are drawn from a random distribution or an affine function of their indices, as specified by the FillMode.

When an IFillLayer is initially added to a network, all of its parameters are static. Each parameter may be changed to dynamic by setting a corresponding input. A parameter is considered dynamic even if that input is the output of an IConstantLayer. The inputs for each parameter are:

- 0: Dimensions
- 1: Alpha
- 2: Beta

The parameter Dimensions describes the shape of the output. If the Dimensions input is provided, it must be a 1D tensor of type Int32 or Int64 whose length is computable by constant folding.

The meanings of Alpha and Beta depend on the mode, as described in IFillLayer::setAlpha(), IFillLayer::setBeta(), and IFillLayer::setInput(). Parameters Alpha and Beta must both be static or both be dynamic.

An IFillLayer can produce a shape tensor if the following restrictions are met:

- The FillOperation is kLINSPACE.
- The output has type Int32, Int64, or Float.
- The volume of the output is within the volume limit imposed on shape tensors.
- If input 0 exists, the values of input 0 must be computable by constant folding.

See also

FillOperation

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.64.2 Constructor & Destructor Documentation

9.64.2.1 ~IFillLayer()

```
virtual nvinfer1::IFillLayer::~IFillLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.64.3 Member Function Documentation

9.64.3.1 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

9.64.3.2 getAlphaInt64()

```
int64_t nvinfer1::IFillLayer::getAlphaInt64 ( ) const  [inline], [noexcept]
```

Get the value of alpha parameter with int64 datatype.

Returns

A int64 value of alpha.

If the second input is present and non-null, this function returns -1.

See also

setAlphaInt64

9.64.3.3 getBeta()

```
double nvinfer1::IFillLayer::getBeta ( ) const  [inline], [noexcept]
```

Get the value of beta parameter.

Returns

A double value of beta.

If the third input is present and non-null, this function returns -1.0.

See also

setBeta, setBetaInt64

9.64.3.4 getBetaInt64()

```
int64_t nvinfer1::IFillLayer::getBetaInt64 ( ) const  [inline], [noexcept]
```

Get the value of beta parameter with int64 datatype.

Returns

A int64 value of beta.

If the third input is present and non-null, this function returns -1.0.

See also

setBetaInt64

9.64.3.5 getDimensions()

```
Dims nvinfer1::IFillLayer::getDimensions ( ) const  [inline], [noexcept]
```

Get the output tensor's dimensions.

Returns

The output tensor's dimensions, or an invalid Dims structure.

If the first input is present and non-null, this function returns a Dims with nbDims = -1.

See also

setDimensions

9.64.3.6 getOperation()

```
FillOperation nvinfer1::IFillLayer::getOperation ( ) const  [inline], [noexcept]
```

Get the fill operation for the layer.

See also

setOperation(), FillOperation

9.64.3.7 getToType()

```
DataType nvinfer1::IFillLayer::getToType ( ) const  [inline], [noexcept]
```

Get the fill layer output type.

Returns

toType parameter set during layer creation or by setToType(). The return value is the output type of the fill layer. The default value is DataType::kFLOAT.

9.64.3.8 isAlphaBetaInt64()

```
bool nvinfer1::IFillLayer::isAlphaBetaInt64 ( ) const  [inline], [noexcept]
```

Return true if alpha/beta have type int64, false if they have type double.

9.64.3.9 setAlpha()

```
void nvinfer1::IFillLayer::setAlpha (
            double alpha )  [inline], [noexcept]
```

Set the alpha parameter.

Parameters

alpha	has different meanings for each operator:

Operation | Usage kLINSPACE | the start value, defaults to 0.0; kRANDOM_UNIFORM | the minimum value, defaults to 0.0; kRANDOM_NORMAL | the mean of the normal distribution, default is 0.0;

If input 1 exists, it is reset to null by this method.

See also

getAlpha, setAlphaInt64

9.64.3.10 setAlphaInt64()

```
void nvinfer1::IFillLayer::setAlphaInt64 (
            int64_t alpha )  [inline], [noexcept]
```

Set the alpha parameter with int64 datatype.

Parameters

alpha	has different meanings for each operator:

Operation | Usage kLINSPACE | the start value, defaults to 0; kRANDOM_UNIFORM | the minimum value, defaults to 0; kRANDOM_NORMAL | the mean of the normal distribution, default is 0;

If a third input had been used to create this layer, that input is reset to null by this method.

See also

getAlphaInt64

9.64.3.11 setBeta()

```
void nvinfer1::IFillLayer::setBeta (
            double beta )  [inline], [noexcept]
```

Set the beta parameter.

Parameters

beta	has different meanings for each operator:

Operation | Usage kLINSPACE | the delta value, defaults to 1.0; kRANDOM_UNIFORM | the maximal value, defaults to 1.0; kRANDOM_NORMAL | the standard deviation of the normal distribution, default is 1.0;

If input 2 exists, it is reset to null by this method.

See also

getBeta

9.64.3.12 setBetaInt64()

```
void nvinfer1::IFillLayer::setBetaInt64 (
            int64_t beta )  [inline], [noexcept]
```

Set the beta parameter with int64 datatype.

Parameters

beta	has different meanings for each operator:

Operation | Usage kLINSPACE | the delta value, defaults to 1; kRANDOM_UNIFORM | the maximal value, defaults to 1; kRANDOM_NORMAL | the standard deviation of the normal distribution, default is 1;

If a third input had been used to create this layer, that input is reset to null by this method.

See also

getBetaInt64

9.64.3.13 setDimensions()

```
void nvinfer1::IFillLayer::setDimensions (
            Dims const & dimensions )  [inline], [noexcept]
```

Set the output tensor's dimensions.

Parameters

dimensions	The output tensor's dimensions.

If the first input had been used to create this layer, that input is reset to null by this method.

See also

getDimensions

9.64.3.14 setInput()

```
void nvinfer1::ILayer::setInput (
             int32_t index,
             ITensor & tensor )  [inline], [noexcept]
```

Replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to set.
tensor	the new input tensor

The three inputs correspond to these setters of IFillLayer:

- 0: setDimensions
- 1: setAlpha
- 2: setBeta

The following descriptions give more intuitive names for the inputs.

Indices for kLINSPACE are:

- 0: Shape, a 1D shape tensor, specifies the output tensor's dimensions.
- 1: Start, a scalar, specifies the start value.
- 2: Delta, a 1D tensor, specifies the delta value for each dimension.

Indices for kRANDOM_UNIFORM are:

- 0: Shape, a 1D shape tensor, specifies the output tensor's dimensions.

- 1: Minimum, a scalar, specifies the minimum random value.
- 2: Maximum, a scalar, specifies the maximal random value.

Indices for kRANDOM_NORMAL are:

- 0: Shape, a 1D shape tensor, specifies the output tensor's dimensions.
- 1: Mean, a scalar, specifies the mean of the normal distribution,.
- 2: Scale, a scalar, specifies the standard deviation of the normal distribution.

Using the corresponding setter resets the input to null.

If either inputs 1 or 2 is non-null, then both must be non-null and have the same data type.

If this function is called for an index greater or equal to getNbInputs(), then afterwards getNbInputs() returns index + 1, and any missing intervening inputs are set to null.

9.64.3.15 setOperation()

```
void nvinfer1::IFillLayer::setOperation (
             FillOperation op )  [inline], [noexcept]
```

Set the fill operation for the layer.

See also

getOperation(), FillOperation

9.64.3.16 setToType()

```
void nvinfer1::IFillLayer::setToType (
             DataType toType )  [inline], [noexcept]
```

Set the fill layer output type.

Parameters

toType	The DataType of the output tensor.

Set the output type of the fill layer. Valid values are DataType::kFLOAT, DataType::kINT32, and DataType::kINT64. If the network is strongly typed, setToType must be used to set the output type, and use of setOutputType is an error. Otherwise, types passed to setOutputType and setToType must be the same.

See also

NetworkDefinitionCreationFlag::kSTRONGLY_TYPED

9.64.4 Member Data Documentation

9.64.4.1 mImpl

```
apiv::VFillLayer* nvinfer1::IFillLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

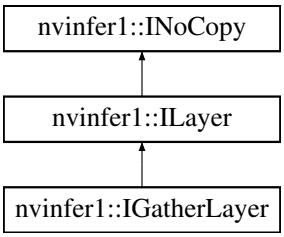
- NvInfer.h

9.65 nvinfer1::IGatherLayer Class Reference

A Gather layer in a network definition. Supports several kinds of gathering.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IGatherLayer:



Public Member Functions

- void setGatherAxis (int32_t axis) noexcept

  Set the axis used by GatherMode::kELEMENTS and GatherMode::kDEFAULT The axis must be less than the number of dimensions in the data input. The axis defaults to 0.
- int32_t getGatherAxis () const noexcept

  Get the axis to gather on.
- void setNbElementWiseDims (int32_t elementWiseDims) noexcept

  Set the number of leading dimensions of indices tensor to be handled elementwise.
- int32_t getNbElementWiseDims () const noexcept

  Get the number of leading dimensions of indices tensor to be handled elementwise.
- void setMode (GatherMode mode) noexcept

  Set the gather mode.
- GatherMode getMode () const noexcept

  Get the gather mode.

Protected Member Functions

- virtual ~IGatherLayer () noexcept=default

Protected Attributes

- apiv::VGatherLayer * mImpl

9.65.1 Detailed Description

A Gather layer in a network definition. Supports several kinds of gathering.

The Gather layer has two input tensors, Data and Indices, and an output tensor Output. Additionally, there are three parameters: mode, nbElementwiseDims, and axis that control how the indices are interpreted.

- Data is a tensor of rank r >= 1 that stores the values to be gathered in Output.
- Indices is a tensor of rank q that determines which locations in Data to gather.
  - GatherMode::kDEFAULT: q >= 0
  - GatherMode::kND: q >= 1 and the last dimension of Indices must be a build time constant.
  - GatherMode::kELEMENT: q = r
- Output stores the gathered results. Its rank s depends on the mode:
  - GatherMode::kDEFAULT: s = q + r - 1 - nbElementwiseDims
  - GatherMode::kND: s = q + r - indices.d[q-1] - 1 - nbElementwiseDims
  - GatherMode::kELEMENT: s = q = r.

The dimensions of the output likewise depends on the mode:

```
GatherMode::kDEFAULT:

    First nbElementwiseDims of output are computed by applying broadcast rules to
    first nbElementwiseDims of indices and data. Note that nbElementwiseDims <= 1.
    Rest of dimensions are computed by copying dimensions of Data, and replacing
    the dimension for axis gatherAxis with the dimensions of indices.

GatherMode::kND:
    If indices.d[q-1] = r - nbElementwiseDims
        output.d = [indices.d[0], ... , indices.d[q-2]]
    Else if indices.d[q-1] < r - nbElementWiseDims
        output.d = [indices.d[0], ... , indices.d[q-1], data.d[nbElementwiseDims + indices.d[q-1] + q],
        data.d[r-1]]
    Else
        This is build time error

GatherMode::kELEMENT:
    The output dimensions match the dimensions of the indices tensor.
```

The types of Data and Output must be the same, and Indices shall be DataType::kINT32 or DataType::kINT64.

How the elements of Data are gathered depends on the mode:

```
GatherMode::kDEFAULT:
    Each index in indices is used to index Data along axis gatherAxis.

GatherMode::kND:
    Indices is a rank q integer tensor, best thought of as a rank (q-1) tensor of
    indices into data, where each element defines a slice of data
    The operation can be formulated as output[i_1, ..., i_{q-1}] = data[indices[i_1, ..., i_{q-1}]]

GatherMode::kELEMENT:

    Here "axis" denotes the result of getGatherAxis().
    For each element X of indices:
        Let J denote a sequence for the subscripts of X
        Let K = sequence J with element [axis] replaced by X
        output[J] = data[K]
```

The handling of nbElementWiseDims depends on the mode:

- GatherMode::kDEFAULT: nbElementWiseDims <= 1. Broadcast is supported across the elementwise dimension if present.
- GatherMode::kND: 0 <= nbElementWiseDims < rank(Data)-1. Broadcast is not supported across the elementwise dimensions.
- GatherMode::kELEMENT: nbElementWiseDims = 0

Notes:

- For modes GatherMode::kND and GatherMode::kELEMENT, the first nbElementWiseDims dimensions of data and index must be equal. If not, an error will be reported at build time or run time.
- If an axis of Data has dynamic length, using a negative index for it has undefined behavior.
- No DLA support
- Zero will be stored for OOB access

> **Warning**
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.65.2 Constructor & Destructor Documentation

9.65.2.1 ~IGatherLayer()

```
virtual nvinfer1::IGatherLayer::~IGatherLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.65.3 Member Function Documentation

9.65.3.1 getGatherAxis()

```
int32_t nvinfer1::IGatherLayer::getGatherAxis ( ) const  [inline], [noexcept]
```

Get the axis to gather on.

> Warning
>
> Undefined behavior when used with GatherMode::kND.

See also

setGatherAxis()

9.65.3.2 getMode()

```
GatherMode nvinfer1::IGatherLayer::getMode ( ) const  [inline], [noexcept]
```

Get the gather mode.

See also

setMode()

9.65.3.3 getNbElementWiseDims()

```
int32_t nvinfer1::IGatherLayer::getNbElementWiseDims ( ) const  [inline], [noexcept]
```

Get the number of leading dimensions of indices tensor to be handled elementwise.

See also

setNbElementWiseDims()

9.65.3.4 setGatherAxis()

```
void nvinfer1::IGatherLayer::setGatherAxis (
            int32_t axis )  [inline], [noexcept]
```

Set the axis used by GatherMode::kELEMENTS and GatherMode::kDEFAULT The axis must be less than the number of dimensions in the data input. The axis defaults to 0.

Warning

Undefined behavior when used with GatherMode::kND.

See also

getGatherAxis()

9.65.3.5 setMode()

```
void nvinfer1::IGatherLayer::setMode (
            GatherMode mode )  [inline], [noexcept]
```

Set the gather mode.

See also

getMode()

9.65.3.6 setNbElementWiseDims()

```
void nvinfer1::IGatherLayer::setNbElementWiseDims (
            int32_t elementWiseDims )  [inline], [noexcept]
```

Set the number of leading dimensions of indices tensor to be handled elementwise.

The gathering of indexing starts from the dimension of data[NbElementWiseDims:]. The NbElementWiseDims must be less than the Rank of the data input.

Parameters

elementWiseDims	number of dims to be handled as elementwise.

Default: 0

The value of nbElementWiseDims and GatherMode are checked during network validation:

GatherMode::kDEFAULT: nbElementWiseDims can be 0 or 1. GatherMode::kND: nbElementWiseDims can be between 0 and one less than rank(data). GatherMode::kELEMENT: nbElementWiseDims must be 0

See also

getNbElementWiseDims()

9.65.4 Member Data Documentation

9.65.4.1 mImpl

```
apiv::VGatherLayer* nvinfer1::IGatherLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.66 IGpuAllocator Class Reference

Application-implemented class for controlling allocation on the GPU.

```
#include <NvInferRuntime.h>
```

9.66.1 Detailed Description

Application-implemented class for controlling allocation on the GPU.

Warning

The lifetime of an IGpuAllocator object must exceed that of all objects that use it.

This class is intended as a base class for allocators that implement synchronous allocation. If you want the benefits of asynchronous allocation, you can do either of:

- Derive your class from IGpuAllocator and override all four of its virtual methods for allocation/deallocation, including the two deprecated methods.
- Derive your class from IGpuAsyncAllocator and override its two pure virtual methods for allocation/deallocation.

The latter style is preferred because it does not tie code to deprecated methods.

See also

IGpuAsyncAllocator.

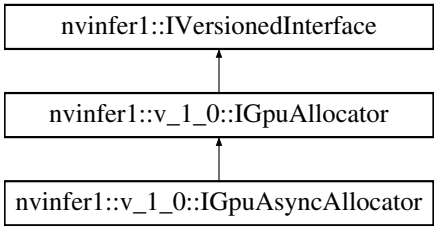
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.67 nvinfer1::v_1_0::IGpuAllocator Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IGpuAllocator:



Public Member Functions

- virtual TRT_DEPRECATED void * allocate (uint64_t const size, uint64_t const alignment, AllocatorFlags const flags) noexcept=0

  A thread-safe callback implemented by the application to handle acquisition of GPU memory.
- ~IGpuAllocator () override=default
- IGpuAllocator ()=default
- virtual void * reallocate (void *const, uint64_t, uint64_t) noexcept

  A thread-safe callback implemented by the application to resize an existing allocation.
- virtual TRT_DEPRECATED bool deallocate (void *const memory) noexcept=0

  A thread-safe callback implemented by the application to handle release of GPU memory.
- virtual void * allocateAsync (uint64_t const size, uint64_t const alignment, AllocatorFlags const flags, cuda↩ Stream_t) noexcept

  A thread-safe callback implemented by the application to handle stream-ordered acquisition of GPU memory.
- virtual bool deallocateAsync (void *const memory, cudaStream_t) noexcept

  A thread-safe callback implemented by the application to handle stream-ordered release of GPU memory.
- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.

Additional Inherited Members

9.67.1 Constructor & Destructor Documentation

9.67.1.1 ~IGpuAllocator()

`nvinfer1::v_1_0::IGpuAllocator::~IGpuAllocator ( ) [override], [default]`

9.67.1.2 IGpuAllocator()

```
nvinfer1::v_1_0::IGpuAllocator::IGpuAllocator ( )  [default]
```

9.67.2 Member Function Documentation

9.67.2.1 allocate()

```
virtual TRT_DEPRECATED void * nvinfer1::v_1_0::IGpuAllocator::allocate (
            uint64_t const size,
            uint64_t const alignment,
            AllocatorFlags const flags )  [pure virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle acquisition of GPU memory.

Parameters

size	The size of the memory block required (in bytes).
alignment	The required alignment of memory. Alignment will be zero or a power of 2 not exceeding the alignment guaranteed by cudaMalloc. Thus this allocator can be safely implemented with cudaMalloc/cudaFree. An alignment value of zero indicates any alignment is acceptable.
flags	Reserved for future use. In the current release, 0 will be passed.

Returns

If the allocation was successful, the start address of a device memory block of the requested size. If an allocation request of size 0 is made, nullptr must be returned. If an allocation request cannot be satisfied, nullptr must be returned. If a non-null address is returned, it is guaranteed to have the specified alignment.

Note

The implementation must guarantee thread safety for concurrent allocate/reallocate/deallocate requests.

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.

Deprecated Deprecated in TensorRT 10.0. Superseded by allocateAsync

Implemented in nvinfer1::v_1_0::IGpuAsyncAllocator.

9.67.2.2 allocateAsync()

```
virtual void * nvinfer1::v_1_0::IGpuAllocator::allocateAsync (
            uint64_t const size,
            uint64_t const alignment,
            AllocatorFlags const flags,
            cudaStream_t  )  [inline], [virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle stream-ordered acquisition of GPU memory.

The default behavior is to call method allocate(), which is synchronous and thus loses any performance benefits of asynchronous allocation. If you want the benefits of asynchronous allocation, see discussion of IGpuAsyncAllocator vs. IGpuAllocator in the documentation for nvinfer1::IGpuAllocator.

Parameters

size	The size of the memory block required (in bytes).
alignment	The required alignment of memory. Alignment will be zero or a power of 2 not exceeding the alignment guaranteed by cudaMalloc. Thus this allocator can be safely implemented with cudaMalloc/cudaFree. An alignment value of zero indicates any alignment is acceptable.
flags	Reserved for future use. In the current release, 0 will be passed.
stream	specifies the cudaStream for asynchronous usage.

Returns

If the allocation was successful, the start address of a device memory block of the requested size. If an allocation request of size 0 is made, nullptr must be returned. If an allocation request cannot be satisfied, nullptr must be returned. If a non-null address is returned, it is guaranteed to have the specified alignment.

Note

The implementation must guarantee thread safety for concurrent allocate/reallocate/deallocate requests.

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.

Reimplemented in nvinfer1::v_1_0::IGpuAsyncAllocator.

9.67.2.3 deallocate()

```
virtual TRT_DEPRECATED bool nvinfer1::v_1_0::IGpuAllocator::deallocate (
            void *const memory )  [pure virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle release of GPU memory.

TensorRT may pass a nullptr to this function if it was previously returned by allocate().

Parameters

memory	A memory address that was previously returned by an allocate() or reallocate() call of the same allocator object.

Returns

True if the acquired memory is released successfully.

Note

The implementation must guarantee thread safety for concurrent allocate/reallocate/deallocate requests.

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.

    Deprecated Deprecated in TensorRT 10.0. Superseded by deallocateAsync

Implemented in nvinfer1::v_1_0::IGpuAsyncAllocator.

9.67.2.4 deallocateAsync()

```
virtual bool nvinfer1::v_1_0::IGpuAllocator::deallocateAsync (
            void *const memory,
            cudaStream_t  )  [inline], [virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle stream-ordered release of GPU memory.

The default behavior is to call method deallocate(), which is synchronous and thus loses any performance benefits of asynchronous deallocation. If you want the benefits of asynchronous deallocation, see discussion of IGpuAsyncAllocator vs. IGpuAllocator in the documentation for nvinfer1::IGpuAllocator.

TensorRT may pass a nullptr to this function if it was previously returned by allocate().

Parameters

memory	A memory address that was previously returned by an allocate() or reallocate() call of the same allocator object.
stream	specifies the cudaStream for asynchronous usage.

Returns

True if the acquired memory is released successfully.

Note

The implementation must guarantee thread safety for concurrent allocate/reallocate/deallocate requests.

The implementation is not required to be asynchronous. It is permitted to synchronize, albeit doing so will lose the performance advantage of asynchronous deallocation. Either way, it is critical that it not actually free the memory until the current stream position is reached.

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.

Reimplemented in nvinfer1::v_1_0::IGpuAsyncAllocator.

9.67.2.5 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IGpuAllocator::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

Reimplemented in nvinfer1::v_1_0::IGpuAsyncAllocator.

9.67.2.6 reallocate()

```
virtual void * nvinfer1::v_1_0::IGpuAllocator::reallocate (
            void * const ,
            uint64_t ,
            uint64_t  )  [inline], [virtual], [noexcept]
```

A thread-safe callback implemented by the application to resize an existing allocation.

Only allocations which were allocated with AllocatorFlag::kRESIZABLE will be resized.

Options are one of:

- resize in place leaving min(oldSize, newSize) bytes unchanged and return the original address
- move min(oldSize, newSize) bytes to a new location of sufficient size and return its address
- return nullptr, to indicate that the request could not be fulfilled.

If nullptr is returned, TensorRT will assume that resize() is not implemented, and that the allocation at baseAddr is still valid.

This method is made available for use cases where delegating the resize strategy to the application provides an opportunity to improve memory management. One possible implementation is to allocate a large virtual device buffer and progressively commit physical memory with cuMemMap. CU_MEM_ALLOC_GRANULARITY_RECOMMENDED is suggested in this case.

TensorRT may call realloc to increase the buffer by relatively small amounts.

Parameters

baseAddr	the address of the original allocation, which will have been returned by previously calling allocate() or reallocate() on the same object.
alignment	The alignment used by the original allocation. This will be the same value that was previously passed to the allocate() or reallocate() call that returned baseAddr.
newSize	The new memory size required (in bytes).

Returns

The address of the reallocated memory, or nullptr. If a non-null address is returned, it is guaranteed to have the specified alignment.

Note

The implementation must guarantee thread safety for concurrent allocate/reallocate/deallocate requests.

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:

- NvInferRuntime.h

9.68 IGpuAsyncAllocator Class Reference

Application-implemented class for controlling asynchronous (stream ordered) memory allocation on the GPU.

`#include <NvInferRuntime.h>`

9.68.1 Detailed Description

Application-implemented class for controlling asynchronous (stream ordered) memory allocation on the GPU.

Warning

The lifetime of an IGpuAsyncAllocator object must exceed that of all objects that use it.

The advantage of deriving from IGpuAsyncAllocator instead of IGpuAllocator is that you only have to override two methods: allocateAsync() and deallocateAsync() to implement an allocator with asynchronous capability, whereas deriving from IGpuAllocator requires overriding four methods, including two deprecated methods.

See also

IGpuAllocator

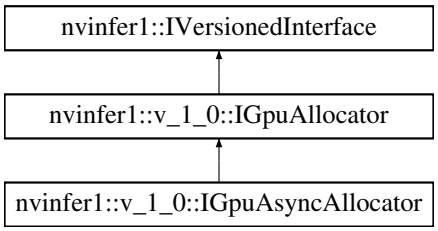
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.69 nvinfer1::v_1_0::IGpuAsyncAllocator Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IGpuAsyncAllocator:



Public Member Functions

- IGpuAsyncAllocator ()=default
- ~IGpuAsyncAllocator () override=default
- void * allocateAsync (uint64_t const size, uint64_t const alignment, AllocatorFlags const flags, cudaStream_t) noexcept override=0

  A thread-safe callback implemented by the application to handle stream-ordered asynchronous acquisition of GPU memory.
- bool deallocateAsync (void *const memory, cudaStream_t) noexcept override=0

  A thread-safe callback implemented by the application to handle stream-ordered asynchronous release of GPU memory.
- TRT_DEPRECATED void * allocate (uint64_t const size, uint64_t const alignment, AllocatorFlags const flags) noexcept override

  A thread-safe callback implemented by the application to handle acquisition of GPU memory.
- TRT_DEPRECATED bool deallocate (void *const memory) noexcept override

  A thread-safe callback implemented by the application to handle release of GPU memory.
- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.

Additional Inherited Members

9.69.1 Constructor & Destructor Documentation

9.69.1.1 IGpuAsyncAllocator()

`nvinfer1::v_1_0::IGpuAsyncAllocator::IGpuAsyncAllocator ( ) [default]`

9.69.1.2 ~IGpuAsyncAllocator()

```
nvinfer1::v_1_0::IGpuAsyncAllocator::~IGpuAsyncAllocator ( )  [override], [default]
```

9.69.2 Member Function Documentation

9.69.2.1 allocate()

```
TRT_DEPRECATED void * nvinfer1::v_1_0::IGpuAsyncAllocator::allocate (
            uint64_t const size,
            uint64_t const alignment,
            AllocatorFlags const flags )  [inline], [override], [virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle acquisition of GPU memory.

Parameters

size	The size of the memory block required (in bytes).
alignment	The required alignment of memory. Alignment will be zero or a power of 2 not exceeding the alignment guaranteed by cudaMalloc. Thus this allocator can be safely implemented with cudaMalloc/cudaFree. An alignment value of zero indicates any alignment is acceptable.
flags	Reserved for future use. In the current release, 0 will be passed.

Returns

If the allocation was successful, the start address of a device memory block of the requested size. If an allocation request of size 0 is made, nullptr must be returned. If an allocation request cannot be satisfied, nullptr must be returned. If a non-null address is returned, it is guaranteed to have the specified alignment.

Note

The implementation must guarantee thread safety for concurrent allocateAsync/deallocateAsync/reallocate requests.

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.

  Deprecated Deprecated in TensorRT 10.0. Superseded by allocateAsync

Implements nvinfer1::v_1_0::IGpuAllocator.

9.69.2.2 allocateAsync()

```
void * nvinfer1::v_1_0::IGpuAsyncAllocator::allocateAsync (
            uint64_t const size,
            uint64_t const alignment,
            AllocatorFlags const flags,
            cudaStream_t  )  [override], [pure virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle stream-ordered asynchronous acquisition of GPU memory.

Parameters

size	The size of the memory block required (in bytes).
alignment	The required alignment of memory. Alignment will be zero or a power of 2 not exceeding the alignment guaranteed by cudaMalloc. Thus this allocator can be safely implemented with cudaMalloc/cudaFree. An alignment value of zero indicates any alignment is acceptable.
flags	Reserved for future use. In the current release, 0 will be passed.
stream	Specifies the cudastream for the asynchronous allocation. If nullptr or 0 is passed, the default stream will be used.

Returns

If the allocation was successful, the start address of a device memory block of the requested size. If an allocation request of size 0 is made, nullptr must be returned. If an allocation request cannot be satisfied, nullptr must be returned. If a non-null address is returned, it is guaranteed to have the specified alignment.

Note

The implementation must guarantee thread safety for concurrent allocateAsync/deallocateAsync requests.

The implementation is not required to be asynchronous. It is permitted to synchronize, albeit doing so will lose the performance advantage of asynchronous allocation.

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.

Reimplemented from nvinfer1::v_1_0::IGpuAllocator.

9.69.2.3 deallocate()

```
TRT_DEPRECATED bool nvinfer1::v_1_0::IGpuAsyncAllocator::deallocate (
            void *const memory )  [inline], [override], [virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle release of GPU memory.

TensorRT may pass a nullptr to this function if it was previously returned by allocate().

Parameters

memory	A memory address that was previously returned by an allocate() or reallocate() call of the same allocator object.

Returns

True if the acquired memory is released successfully.

Note

The implementation must guarantee thread safety for concurrent allocate/reallocate/deallocate requests.

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.

    Deprecated Deprecated in TensorRT 10.0. Superseded by deallocateAsync

Implements nvinfer1::v_1_0::IGpuAllocator.

9.69.2.4 deallocateAsync()

```
bool nvinfer1::v_1_0::IGpuAsyncAllocator::deallocateAsync (
            void *const memory,
            cudaStream_t  )  [override], [pure virtual], [noexcept]
```

A thread-safe callback implemented by the application to handle stream-ordered asynchronous release of GPU memory.

TensorRT may pass a nullptr to this function if it was previously returned by allocate().

Parameters

memory	A memory address that was previously returned by an allocate() or reallocate() call of the same allocator object.
stream	Specifies the cudastream for the asynchronous deallocation. If nullptr or 0 is passed, the default stream will be used.

Returns

True if the acquired memory is released successfully.

Note

The implementation must guarantee thread safety for concurrent allocateAsync/deallocateAsync requests.

The implementation is not required to be asynchronous. It is permitted to synchronize, albeit doing so will lose the performance advantage of asynchronous deallocation. Either way, it is critical that it not actually free the memory until the current stream position is reached.

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.

Reimplemented from nvinfer1::v_1_0::IGpuAllocator.

9.69.2.5 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IGpuAsyncAllocator::getInterfaceInfo ( ) const  [inline], [override], [virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Reimplemented from nvinfer1::v_1_0::IGpuAllocator.

The documentation for this class was generated from the following file:

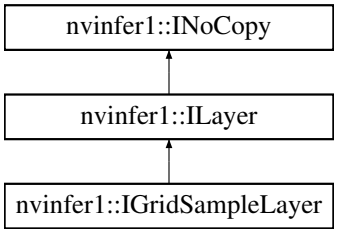
- NvInferRuntime.h

9.70 nvinfer1::IGridSampleLayer Class Reference

A GridSample layer in a network definition.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IGridSampleLayer:



Public Member Functions

- void setInterpolationMode (InterpolationMode mode) noexcept

  Set the grid sample interpolation mode.
- InterpolationMode getInterpolationMode () const noexcept

  Get the grid sample interpolation mode.
- void setAlignCorners (bool alignCorners) noexcept

  Set the align corners mode.
- bool getAlignCorners () const noexcept

  Get the align corners mode.
- bool setSampleMode (SampleMode mode) noexcept

  Set the sample mode.
- SampleMode getSampleMode () const noexcept

  Get the sample mode.

Protected Member Functions

- virtual ~IGridSampleLayer () noexcept=default

Protected Attributes

- apiv::VGridSampleLayer * mImpl

9.70.1 Detailed Description

A GridSample layer in a network definition.

This layer uses an input tensor and a grid tensor to produce an interpolated output tensor. The input and grid tensors must be shape tensors of rank 4. The only supported SampleMode values are SampleMode::kCLAMP, SampleMode::kFILL, and SampleMode::kREFLECT.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.70.2 Constructor & Destructor Documentation

9.70.2.1 ~IGridSampleLayer()

```
virtual nvinfer1::IGridSampleLayer::~IGridSampleLayer ( )  [protected], [virtual], [default],
[noexcept]
```

9.70.3 Member Function Documentation

9.70.3.1 getAlignCorners()

```
bool nvinfer1::IGridSampleLayer::getAlignCorners ( ) const  [inline], [noexcept]
```

Get the align corners mode.

See also

setAlignCorners()

Returns

The value specified by setAlignCorners(), or false otherwise.

9.70.3.2 getInterpolationMode()

```
InterpolationMode nvinfer1::IGridSampleLayer::getInterpolationMode ( ) const  [inline], [noexcept]
```

Get the grid sample interpolation mode.

See also

setInterpolationMode()

Returns

The value specified by setInterpolationMode, or InterpolationMode::kLINEAR otherwise.

9.70.3.3 getSampleMode()

```
SampleMode nvinfer1::IGridSampleLayer::getSampleMode ( ) const  [inline], [noexcept]
```

Get the sample mode.

See also

setSampleMode()

Returns

the value specified by a successful call to setSampleMode(), or SampleMode::kFILL otherwise.

9.70.3.4 setAlignCorners()

```
void nvinfer1::IGridSampleLayer::setAlignCorners (
            bool alignCorners )  [inline], [noexcept]
```

Set the align corners mode.

See also

getAlignCorners()

9.70.3.5 setInterpolationMode()

```
void nvinfer1::IGridSampleLayer::setInterpolationMode (
            InterpolationMode mode )  [inline], [noexcept]
```

Set the grid sample interpolation mode.

See also

getInterpolationMode()

9.70.3.6 setSampleMode()

```
bool nvinfer1::IGridSampleLayer::setSampleMode (
            SampleMode mode )  [inline], [noexcept]
```

Set the sample mode.

See also

getSampleMode()

Returns

true if layer's sample mode was set to mode, false otherwise.

9.70.4 Member Data Documentation

9.70.4.1 mImpl

```
apiv::VGridSampleLayer* nvinfer1::IGridSampleLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

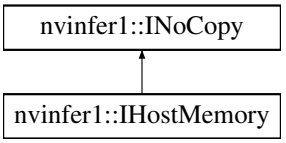
- NvInfer.h

9.71 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.

Protected Attributes

- apiv::VHostMemory * mImpl

Additional Inherited Members

9.71.1 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.

9.71.2 Constructor & Destructor Documentation

9.71.2.1 ~IHostMemory()

```
virtual nvinfer1::IHostMemory::~IHostMemory ( )  [virtual], [default], [noexcept]
```

9.71.3 Member Function Documentation

9.71.3.1 data()

```
void * nvinfer1::IHostMemory::data ( ) const  [inline], [noexcept]
```

A pointer to the raw data that is owned by the library.

9.71.3.2 size()

```
std::size_t nvinfer1::IHostMemory::size ( ) const  [inline], [noexcept]
```

The size in bytes of the data that was allocated.

9.71.3.3 type()

```
DataType nvinfer1::IHostMemory::type ( ) const  [inline], [noexcept]
```

The type of the memory that was allocated.

9.71.4 Member Data Documentation

9.71.4.1 mImpl

```
apiv::VHostMemory* nvinfer1::IHostMemory::mImpl  [protected]
```

The documentation for this class was generated from the following file:

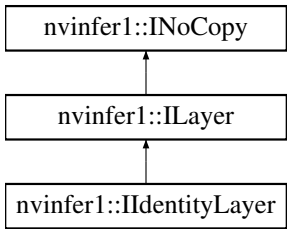
- NvInferRuntime.h

9.72 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

9.72.1 Detailed Description

A layer that represents the identity function.

If the output type is explicitly specified via setOutputType, IIdentityLayer can be used to convert from one type to another. Other than conversions between the same type (kFLOAT -> kFLOAT for example), the only valid conversions are:

```
(kFLOAT | kHALF | kINT32 | kBOOL) -> (kFLOAT | kHALF | kINT32 | kBOOL)

(kFLOAT | kHALF) -> kUINT8

kUINT8 -> (kFLOAT | kHALF)
```

Conversion also happens implicitly, without calling setOutputType, if the output tensor is a network output.

Two types are compatible if they are identical, or are both in {kFLOAT, kHALF}. Implicit conversion between incompatible types, i.e. without using setOutputType, is recognized as incorrect as of TensorRT 8.4, but is retained for API compatibility within TensorRT 8.x releases. TensorRT 10.0 onwards it is an error if the network output tensor type is incompatible with the layer output type. E.g., implicit conversion from kFLOAT to kINT32 is not allowed, Use setOutputType(DataType::kINT32) to explict convert kFLOAT to kINT32.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.72.2 Constructor & Destructor Documentation

9.72.2.1 ~IIdentityLayer()

```
virtual nvinfer1::IIdentityLayer::~IIdentityLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.72.3 Member Data Documentation

9.72.3.1 mImpl

```
apiv::VIdentityLayer* nvinfer1::IIdentityLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

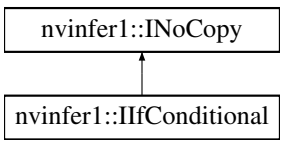
- NvInfer.h

9.73 nvinfer1::IIfConditional Class Reference

Helper for constructing conditionally-executed subgraphs.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IIfConditional:



Public Member Functions

- IConditionLayer * setCondition (ITensor &condition) noexcept

  Set the condition tensor for this If-Conditional construct.
- IIfConditionalOutputLayer * addOutput (ITensor &trueSubgraphOutput, ITensor &falseSubgraphOutput) noexcept

  Add an If-conditional output.
- IIfConditionalInputLayer * addInput (ITensor &input) noexcept

  Add an If-conditional input.
- void setName (char const *name) noexcept

  Set the name of the conditional.
- char const * getName () const noexcept

  Return the name of the conditional.

Protected Member Functions

- virtual ~IIfConditional () noexcept=default

Protected Attributes

- apiv::VIfConditional * mImpl

9.73.1 Detailed Description

Helper for constructing conditionally-executed subgraphs.

An If-conditional conditionally executes part of the network according to the following pseudo-code:

If condition is true then: output = trueSubgraph(trueInputs); Else output = falseSubgraph(falseInputs); Emit output

Condition is a 0D boolean tensor (representing a scalar). trueSubgraph represents a network subgraph that is executed when condition evaluates to True. falseSubgraph represents a network subgraph that is executed when condition evaluates to False.

The following constraints apply to If-conditionals:

- Both the trueSubgraph and falseSubgraph must be defined.
- The number of output tensors in both subgraphs is the same.
- Corresponding output tensors from the true/false subgraphs have the same type and shape.

9.73.2 Constructor & Destructor Documentation

9.73.2.1 ~IIfConditional()

```
virtual nvinfer1::IIfConditional::~IIfConditional ( )  [protected], [virtual], [default], [noexcept]
```

9.73.3 Member Function Documentation

9.73.3.1 addInput()

```
IIfConditionalInputLayer * nvinfer1::IIfConditional::addInput (
            ITensor & input )  [inline], [noexcept]
```

Add an If-conditional input.

Parameters

input	An input to the conditional that can be used by either or both of the conditional's subgraphs.

See also

IIfConditionalInputLayer

9.73.3.2 addOutput()

```
IIfConditionalOutputLayer * nvinfer1::IIfConditional::addOutput (
            ITensor & trueSubgraphOutput,
            ITensor & falseSubgraphOutput )  [inline], [noexcept]
```

Add an If-conditional output.

Parameters

trueSubgraphOutput	The output of the subgraph executed when the conditional evaluates to true.
falseSubgraphOutput	The output of the subgraph executed when the conditional evaluates to false.

Each output layer of an IIfConditional represents a single output of either the true-subgraph or the false-subgraph of an IIfConditional, depending on which subgraph was executed.

The shapes of the two tensors must be equal unless the condition is a build-time constant.

See also

IIfConditionalOutputLayer

9.73.3.3 getName()

```
char const * nvinfer1::IIfConditional::getName ( ) const  [inline], [noexcept]
```

Return the name of the conditional.

See also

setName()

9.73.3.4 setCondition()

```
IConditionLayer * nvinfer1::IIfConditional::setCondition (
            ITensor & condition )  [inline], [noexcept]
```

Set the condition tensor for this If-Conditional construct.

Parameters

condition	The condition tensor that will determine which subgraph to execute.

`condition` tensor must be a 0D execution tensor (scalar) with type DataType::kBOOL.

See also

IConditionLayer

9.73.3.5 setName()

```
void nvinfer1::IIfConditional::setName (
            char const * name )  [inline], [noexcept]
```

Set the name of the conditional.

The name is used in error diagnostics. This method copies the name string.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

See also

getName()

9.73.4 Member Data Documentation

9.73.4.1 mImpl

```
apiv::VIfConditional* nvinfer1::IIfConditional::mImpl  [protected]
```

The documentation for this class was generated from the following file:

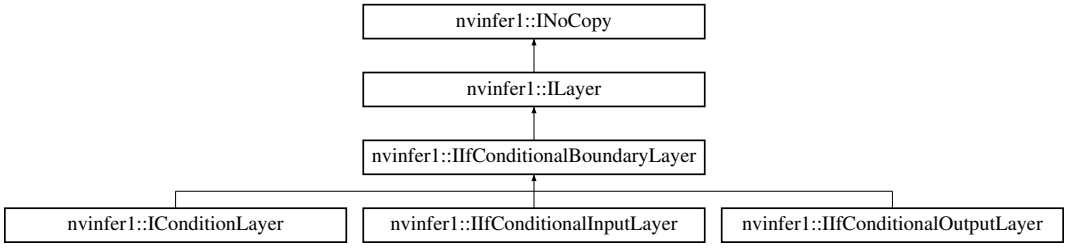
- NvInfer.h

9.74 nvinfer1::IIfConditionalBoundaryLayer Class Reference

This is a base class for Conditional boundary layers.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IIfConditionalBoundaryLayer:



Public Member Functions

- IIfConditional * getConditional () const noexcept

  Get a pointer to the IIfConditional associated with this boundary layer.

Protected Member Functions

- virtual ~IIfConditionalBoundaryLayer () noexcept=default

Protected Attributes

- apiv::VConditionalBoundaryLayer * mBoundary

9.74.1 Detailed Description

This is a base class for Conditional boundary layers.

Boundary layers are used to demarcate the boundaries of Conditionals.

9.74.2 Constructor & Destructor Documentation

9.74.2.1 ~IIfConditionalBoundaryLayer()

```
virtual nvinfer1::IIfConditionalBoundaryLayer::~IIfConditionalBoundaryLayer ( )  [protected],
[virtual], [default], [noexcept]
```

9.74.3 Member Function Documentation

9.74.3.1 getConditional()

```
IIfConditional * nvinfer1::IIfConditionalBoundaryLayer::getConditional ( ) const  [inline], [noexcept]
```

Get a pointer to the IIfConditional associated with this boundary layer.

9.74.4 Member Data Documentation

9.74.4.1 mBoundary

```
apiv::VConditionalBoundaryLayer* nvinfer1::IIfConditionalBoundaryLayer::mBoundary  [protected]
```

The documentation for this class was generated from the following file:

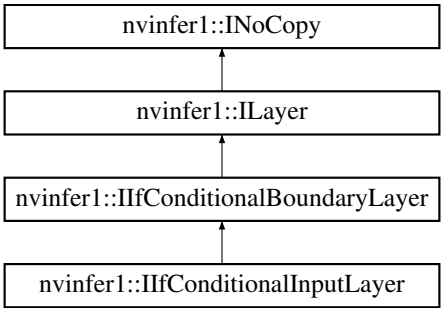
- NvInfer.h

9.75 nvinfer1::IIfConditionalInputLayer Class Reference

This layer represents an input to an IIfConditional.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IIfConditionalInputLayer:



Protected Member Functions

- virtual ~IIfConditionalInputLayer () noexcept=default

Protected Attributes

- apiv::VConditionalInputLayer * mImpl

Additional Inherited Members

9.75.1 Detailed Description

This layer represents an input to an IIfConditional.

9.75.2 Constructor & Destructor Documentation

9.75.2.1 ~IIfConditionalInputLayer()

```
virtual nvinfer1::IIfConditionalInputLayer::~IIfConditionalInputLayer ( )  [protected], [virtual],
[default], [noexcept]
```

9.75.3 Member Data Documentation

9.75.3.1 mImpl

```
apiv::VConditionalInputLayer* nvinfer1::IIfConditionalInputLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

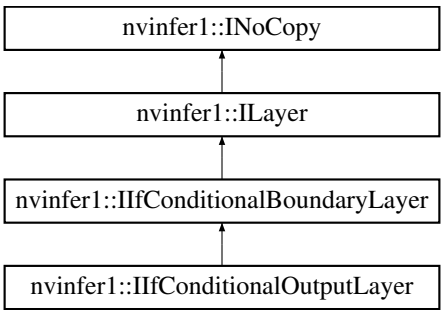
- NvInfer.h

9.76 nvinfer1::IIfConditionalOutputLayer Class Reference

This layer represents an output of an IIfConditional.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IIfConditionalOutputLayer:



Protected Member Functions

- virtual ~IIfConditionalOutputLayer () noexcept=default

Protected Attributes

- apiv::VConditionalOutputLayer * mImpl

Additional Inherited Members

9.76.1 Detailed Description

This layer represents an output of an IIfConditional.

An IIfConditionalOutputLayer has two inputs and one output.

See also

IIfConditional::addOutput

9.76.2 Constructor & Destructor Documentation

9.76.2.1 ~IIfConditionalOutputLayer()

```
virtual nvinfer1::IIfConditionalOutputLayer::~IIfConditionalOutputLayer ( )  [protected], [virtual],
[default], [noexcept]
```

9.76.3 Member Data Documentation

9.76.3.1 mImpl

```
apiv::VConditionalOutputLayer* nvinfer1::IIfConditionalOutputLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

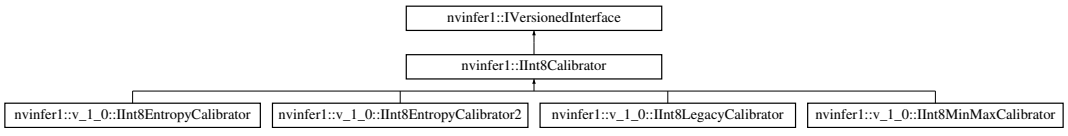
- NvInfer.h

9.77 nvinfer1::IInt8Calibrator Class Reference

Application-implemented interface for calibration.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IInt8Calibrator:



Public Member Functions

- virtual TRT_DEPRECATED 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.
- ∼IInt8Calibrator () noexcept override=default

Additional Inherited Members

9.77.1 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.

Deprecated Deprecated in TensorRT 10.1. Superseded by explicit quantization.

9.77.2 Constructor & Destructor Documentation

9.77.2.1 ∼IInt8Calibrator()

```
nvinfer1::IInt8Calibrator::∼IInt8Calibrator ( )  [override], [default], [noexcept]
```

9.77.3 Member Function Documentation

9.77.3.1 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::v_1_0::IInt8EntropyCalibrator, nvinfer1::v_1_0::IInt8EntropyCalibrator2, nvinfer1::v_1_0::IInt8MinMaxCalibr and nvinfer1::v_1_0::IInt8LegacyCalibrator.

9.77.3.2 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

bindings	An array of pointers to device memory that must be updated to point to device memory containing each network input data.
names	The names of the network input for each pointer in the binding array.
nbBindings	The number of pointers in the bindings array.

Returns

False if there are no more batches for calibration.

See also

getBatchSize()

9.77.3.3 getBatchSize()

```
virtual TRT_DEPRECATED int32_t nvinfer1::IInt8Calibrator::getBatchSize ( ) const  [pure virtual],
[noexcept]
```

Get the batch size used for calibration batches.

Returns

The batch size.

Deprecated Deprecated in TensorRT 10.0. Implicit batch support is removed in TensorRT 10.0.

9.77.3.4 readCalibrationCache()

```
virtual void const * nvinfer1::IInt8Calibrator::readCalibrationCache (
            std::size_t & length )  [pure virtual], [noexcept]
```

Load a calibration cache.

Calibration is potentially expensive, so it can be useful to generate the calibration data once, then use it on subsequent builds of the network. The cache includes the regression cutoff and quantile values used to generate it, and will not be used if these do not batch the settings of the current calibrator. However, the network should also be recalibrated if its structure changes, or the input data set changes, and it is the responsibility of the application to ensure this.

Parameters

length	The length of the cached data, that should be set by the called function. If there is no data, this should be zero.

Returns

A pointer to the cache, or nullptr if there is no data.

9.77.3.5 writeCalibrationCache()

```
virtual void nvinfer1::IInt8Calibrator::writeCalibrationCache (
            void const * ptr,
            std::size_t length )  [pure virtual], [noexcept]
```

Save a calibration cache.

Parameters

ptr	A pointer to the data to cache.
length	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

9.78 IInt8EntropyCalibrator Class Reference

Entropy calibrator.

`#include <NvInfer.h>`

9.78.1 Detailed Description

Entropy calibrator.

This is the Legacy Entropy calibrator. It is less complicated than the legacy calibrator and produces better results.

Note

To ensure compatibility of source code with future versions of TensorRT, use IEntropyCalibrator, not v_1_0::IEntropyCalibrator

Deprecated Deprecated in TensorRT 10.1. Superseded by explicit quantization.

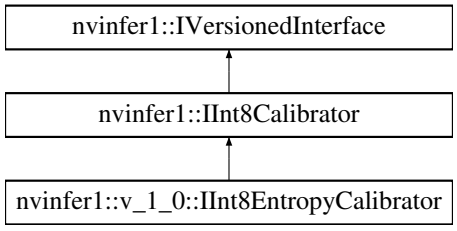
The documentation for this class was generated from the following file:

- NvInfer.h

9.79 nvinfer1::v_1_0::IInt8EntropyCalibrator Class Reference

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::v_1_0::IInt8EntropyCalibrator:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- CalibrationAlgoType getAlgorithm () noexcept override
- ~IInt8EntropyCalibrator () noexcept override=default

Additional Inherited Members

9.79.1 Constructor & Destructor Documentation

9.79.1.1 ~IInt8EntropyCalibrator()

```
nvinfer1::v_1_0::IInt8EntropyCalibrator::~IInt8EntropyCalibrator ( )  [override], [default], [noexcept]
```

9.79.2 Member Function Documentation

9.79.2.1 getAlgorithm()

```
CalibrationAlgoType nvinfer1::v_1_0::IInt8EntropyCalibrator::getAlgorithm ( )  [inline], [override],
[virtual], [noexcept]
```

Signal that this is the entropy calibrator.

Implements nvinfer1::IInt8Calibrator.

9.79.2.2 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IInt8EntropyCalibrator::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

The documentation for this class was generated from the following file:

- NvInfer.h

9.80 IInt8EntropyCalibrator2 Class Reference

Entropy calibrator 2.

`#include <NvInfer.h>`

9.80.1 Detailed Description

Entropy calibrator 2.

This is the preferred calibrator. This is the required calibrator for DLA, as it supports per activation tensor scaling.

Note

To ensure compatibility of source code with future versions of TensorRT, use IEntropyCalibrator2, not v_1_0::↩ IEntropyCalibrator2

Deprecated Deprecated in TensorRT 10.1. Superseded by explicit quantization.

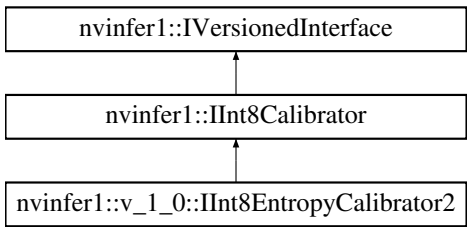
The documentation for this class was generated from the following file:

- NvInfer.h

9.81 nvinfer1::v_1_0::IInt8EntropyCalibrator2 Class Reference

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::v_1_0::IInt8EntropyCalibrator2:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- CalibrationAlgoType getAlgorithm () noexcept override
- ~IInt8EntropyCalibrator2 () noexcept override=default

Additional Inherited Members

9.81.1 Constructor & Destructor Documentation

9.81.1.1 ~IInt8EntropyCalibrator2()

```
nvinfer1::v_1_0::IInt8EntropyCalibrator2::~IInt8EntropyCalibrator2 ( )  [override], [default],
[noexcept]
```

9.81.2 Member Function Documentation

9.81.2.1 getAlgorithm()

```
CalibrationAlgoType nvinfer1::v_1_0::IInt8EntropyCalibrator2::getAlgorithm ( )  [inline], [override],
[virtual], [noexcept]
```

Signal that this is the entropy calibrator 2.

Implements nvinfer1::IInt8Calibrator.

9.81.2.2 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IInt8EntropyCalibrator2::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

The documentation for this class was generated from the following file:

- NvInfer.h

9.82 IInt8LegacyCalibrator Class Reference

Legacy calibrator.

```
#include <NvInfer.h>
```

9.82.1 Detailed Description

Legacy calibrator.

This calibrator requires user parameterization, and is provided as a fallback option if the other calibrators yield poor results.

Note

To ensure compatibility of source code with future versions of TensorRT, use ILegacyCalibrator, not v_1_0::↩ ILegacyCalibrator

Deprecated Deprecated in TensorRT 10.1. Superseded by explicit quantization.

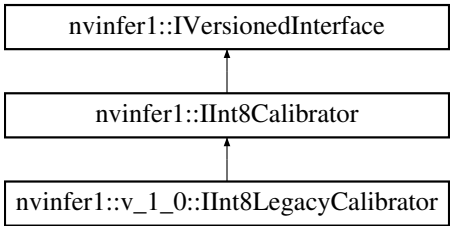
The documentation for this class was generated from the following file:

- NvInfer.h

9.83 nvinfer1::v_1_0::IInt8LegacyCalibrator Class Reference

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::v_1_0::IInt8LegacyCalibrator:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- 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.
- ~IInt8LegacyCalibrator () noexcept override=default

Additional Inherited Members

9.83.1 Constructor & Destructor Documentation

9.83.1.1 ~IInt8LegacyCalibrator()

```
nvinfer1::v_1_0::IInt8LegacyCalibrator::~IInt8LegacyCalibrator ( )  [override], [default], [noexcept]
```

9.83.2 Member Function Documentation

9.83.2.1 getAlgorithm()

```
CalibrationAlgoType nvinfer1::v_1_0::IInt8LegacyCalibrator::getAlgorithm ( )  [inline], [override],
[virtual], [noexcept]
```

Signal that this is the legacy calibrator.

Implements nvinfer1::IInt8Calibrator.

9.83.2.2 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IInt8LegacyCalibrator::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.83.2.3 getQuantile()

```
virtual double nvinfer1::v_1_0::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.

9.83.2.4 getRegressionCutoff()

```
virtual double nvinfer1::v_1_0::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

9.83.2.5 readHistogramCache()

```
virtual void const * nvinfer1::v_1_0::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

length	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.

9.83.2.6 writeHistogramCache()

```
virtual void nvinfer1::v_1_0::IInt8LegacyCalibrator::writeHistogramCache (
            void const * ptr,
            std::size_t length )  [pure virtual], [noexcept]
```

Save a histogram cache.

Parameters

ptr	A pointer to the data to cache.
length	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

9.84 IInt8MinMaxCalibrator Class Reference

MinMax Calibrator.

`#include <NvInfer.h>`

9.84.1 Detailed Description

MinMax Calibrator.

It supports per activation tensor scaling.

Note

To ensure compatibility of source code with future versions of TensorRT, use IMinMaxCalibrator>, not v_1_0↩ ::IMinMaxCalibrator

Deprecated Deprecated in TensorRT 10.1. Superseded by explicit quantization.

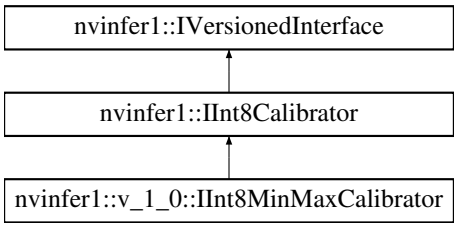
The documentation for this class was generated from the following file:

- NvInfer.h

9.85 nvinfer1::v_1_0::IInt8MinMaxCalibrator Class Reference

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::v_1_0::IInt8MinMaxCalibrator:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- CalibrationAlgoType getAlgorithm () noexcept override
- ~IInt8MinMaxCalibrator () noexcept override=default

Additional Inherited Members

9.85.1 Constructor & Destructor Documentation

9.85.1.1 ~IInt8MinMaxCalibrator()

```
nvinfer1::v_1_0::IInt8MinMaxCalibrator::~IInt8MinMaxCalibrator ( )  [override], [default], [noexcept]
```

9.85.2 Member Function Documentation

9.85.2.1 getAlgorithm()

```
CalibrationAlgoType nvinfer1::v_1_0::IInt8MinMaxCalibrator::getAlgorithm ( )  [inline], [override],
[virtual], [noexcept]
```

Signal that this is the MinMax Calibrator.

Implements nvinfer1::IInt8Calibrator.

9.85.2.2 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IInt8MinMaxCalibrator::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

The documentation for this class was generated from the following file:

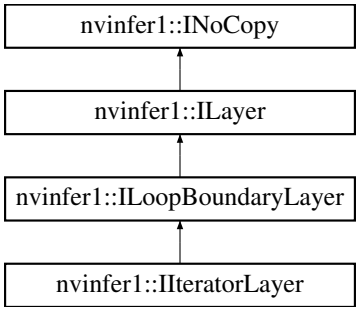
- NvInfer.h

9.86 nvinfer1::IIteratorLayer Class Reference

A layer to do iterations.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IIteratorLayer:



Public Member Functions

- void setAxis (int32_t axis) noexcept

  Set axis to iterate over.
- int32_t getAxis () const noexcept

  Get axis being iterated over.
- void setReverse (bool reverse) noexcept

  Set iteration order to be reverse.
- bool getReverse () const noexcept

  Check if the iteration order is reverse.

Protected Member Functions

- virtual ~IIteratorLayer () noexcept=default

Protected Attributes

- apiv::VIteratorLayer * mImpl

9.86.1 Detailed Description

A layer to do iterations.

The iterator layer iterates over a tensor along the given axis and in the given direction. It enables each loop iteration to inspect a different slice of the tensor.

See also

ILoop::addIterator()

9.86.2 Constructor & Destructor Documentation

9.86.2.1 ~IIteratorLayer()

```
virtual nvinfer1::IIteratorLayer::~IIteratorLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.86.3 Member Function Documentation

9.86.3.1 getAxis()

```
int32_t nvinfer1::IIteratorLayer::getAxis ( ) const  [inline], [noexcept]
```

Get axis being iterated over.

9.86.3.2 getReverse()

```
bool nvinfer1::IIteratorLayer::getReverse ( ) const  [inline], [noexcept]
```

Check if the iteration order is reverse.

Returns

True if and only if reversing input.

9.86.3.3 setAxis()

```
void nvinfer1::IIteratorLayer::setAxis (
            int32_t axis )  [inline], [noexcept]
```

Set axis to iterate over.

9.86.3.4 setReverse()

```
void nvinfer1::IIteratorLayer::setReverse (
            bool reverse )  [inline], [noexcept]
```

Set iteration order to be reverse.

For reverse=false, the layer is equivalent to addGather(tensor, I, 0) where I is a scalar tensor containing the loop iteration number. For reverse=true, the layer is equivalent to addGather(tensor, M-1-I, 0) where M is the trip count computed from TripLimits of kind kCOUNT. The default is reverse=false.

9.86.4 Member Data Documentation

9.86.4.1 mImpl

```
apiv::VIteratorLayer* nvinfer1::IIteratorLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

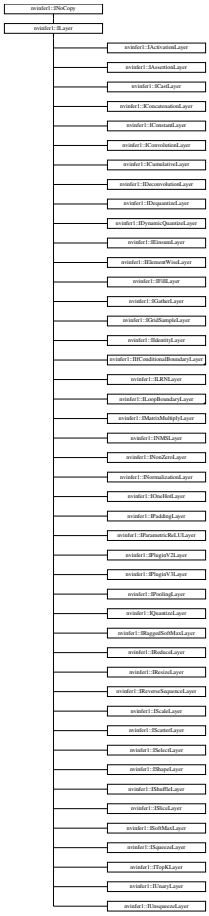
- NvInfer.h

9.87 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 preferred or required computational precision of this layer in a weakly-typed network.

- 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 in a weakly-typed network.
- DataType getOutputType (int32_t index) const noexcept

  get the output type of this layer
- bool outputTypeIsSet (int32_t index) const noexcept

  whether the output type has been set for this layer
- void resetOutputType (int32_t index) noexcept

  reset the output type for this layer
- void setMetadata (char const *metadata) noexcept

  Set the metadata for this layer.
- char const * getMetadata () const noexcept

  Get the metadata of the layer.

Protected Member Functions

- virtual ~ILayer () noexcept=default

Protected Attributes

- apiv::VLayer * mLayer

9.87.1 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.

9.87.2 Constructor & Destructor Documentation

9.87.2.1 ~ILayer()

```
virtual nvinfer1::ILayer::~ILayer ( )  [protected], [virtual], [default], [noexcept]
```

9.87.3 Member Function Documentation

9.87.3.1 getInput()

```
ITensor * nvinfer1::ILayer::getInput (
            int32_t index ) const  [inline], [noexcept]
```

Get the layer input corresponding to the given index.

Parameters

index	The index of the input tensor.

Returns

The input tensor, or nullptr if the index is out of range or the tensor is optional (ISliceLayer).

9.87.3.2 getMetadata()

```
char const * nvinfer1::ILayer::getMetadata ( ) const  [inline], [noexcept]
```

Get the metadata of the layer.

Returns

The metadata as a null-terminated C-style string. If setMetadata() has not been called, an empty string "" will be returned as a default value.

See also

setMetadata()

9.87.3.3 getName()

```
char const * nvinfer1::ILayer::getName ( ) const  [inline], [noexcept]
```

Return the name of a layer.

See also

setName()

9.87.3.4 getNbInputs()

```
int32_t nvinfer1::ILayer::getNbInputs ( ) const  [inline], [noexcept]
```

Get the number of inputs of a layer.

9.87.3.5 getNbOutputs()

```
int32_t nvinfer1::ILayer::getNbOutputs ( ) const  [inline], [noexcept]
```

Get the number of outputs of a layer.

9.87.3.6 getOutput()

```
ITensor * nvinfer1::ILayer::getOutput (
            int32_t index ) const  [inline], [noexcept]
```

Get the layer output corresponding to the given index.

Returns

The indexed output tensor, or nullptr if the index is out of range or the tensor is optional.

9.87.3.7 getOutputType()

```
DataType nvinfer1::ILayer::getOutputType (
            int32_t index ) const  [inline], [noexcept]
```

get the output type of this layer

Parameters

index	the index of the output

Returns

the 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()

9.87.3.8 getPrecision()

```
DataType nvinfer1::ILayer::getPrecision ( ) const  [inline], [noexcept]
```

get the computational precision of this layer

Returns

the computational precision

See also

setPrecision() precisionIsSet() resetPrecision()

9.87.3.9 getType()

```
LayerType nvinfer1::ILayer::getType ( ) const  [inline], [noexcept]
```

Return the type of a layer.

See also

LayerType

9.87.3.10 outputTypeIsSet()

```
bool nvinfer1::ILayer::outputTypeIsSet (
            int32_t index ) const  [inline], [noexcept]
```

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

See also

setOutputType() getOutputType() resetOutputType()

9.87.3.11 precisionIsSet()

```
bool nvinfer1::ILayer::precisionIsSet ( ) const  [inline], [noexcept]
```

whether the computational precision has been set for this layer

Returns

whether the computational precision has been explicitly set

See also

setPrecision() getPrecision() resetPrecision()

9.87.3.12 resetOutputType()

```
void nvinfer1::ILayer::resetOutputType (
            int32_t index )  [inline], [noexcept]
```

reset the output type for this layer

Parameters

index	the index of the output

See also

setOutputType() getOutputType() outputTypeIsSet()

9.87.3.13 resetPrecision()

```
void nvinfer1::ILayer::resetPrecision ( )  [inline], [noexcept]
```

reset the computational precision for this layer

See also

setPrecision() getPrecision() precisionIsSet()

9.87.3.14 setInput()

```
void nvinfer1::ILayer::setInput (
             int32_t index,
             ITensor & tensor )  [inline], [noexcept]
```

Replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor

Except for IFillLayer, ILoopOutputLayer, INMSLayer, IResizeLayer, IShuffleLayer, and ISliceLayer, this method cannot change the number of inputs to a layer. The index argument must be less than the value of getNbInputs().

See comments for overloads of setInput() for layers with special behavior.

9.87.3.15 setMetadata()

```
void nvinfer1::ILayer::setMetadata (
             char const * metadata )  [inline], [noexcept]
```

Set the metadata for this layer.

The metadata is emitted in the JSON returned by IEngineInspector with ProfilingVerbosity set to kDETAILED.

Parameters

metadata	The per-layer metadata.

Warning

The string name must be null-terminated and be at most 4096 bytes including the terminator.

See also

getMetadata()

getLayerInformation()

9.87.3.16 setName()

```
void nvinfer1::ILayer::setName (
            char const * name )  [inline], [noexcept]
```

Set the name of a layer.

This method copies the name string.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

See also

getName()

9.87.3.17 setOutputType()

```
void nvinfer1::ILayer::setOutputType (
            int32_t index,
            DataType dataType )  [inline], [noexcept]
```

Set the output type of this layer in a weakly-typed network.

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.

Strongly-typed networks reject calls to method setOutputType. Instead, the output type can be set only for layers that define method setToType(). Those layers are:

- ICastLayer
- IDequantizeLayer
- IFillLayer
- IQuantizeLayer

Parameters

index	the index of the output to set
dataType	the type of the output

See also

getOutputType() outputTypeIsSet() resetOutputType()

9.87.3.18 setPrecision()

```
void nvinfer1::ILayer::setPrecision (
            DataType dataType )  [inline], [noexcept]
```

Set the preferred or required computational precision of this layer in a weakly-typed network.

Setting the precision directs TensorRT to choose an implementation that runs at this computational precision. TensorRT could still choose a non-conforming fastest implementation that ignores the requested precision. To force choosing an implementation with the requested precision, set exactly one of the following flags, which differ in what happens if no such implementation exists:

- BuilderFlag::kOBEY_PRECISION_CONSTRAINTS - build fails with an error message.

- BuilderFlag::kPREFER_PRECISION_CONSTRAINTS - TensorRT falls back to an implementation without the requested precision.

If precision is not set, or falling back, TensorRT will select the layer computational precision and layer input type based on global performance considerations and the flags specified to the builder.

For a IIdentityLayer: If it casts to/from float/half/int8/uint8, the precision must be one of those types, otherwise it must be either the input or output type.

Strongly-typed networks reject calls to method setPrecision. In strongly-typed networks, the computation precision is typically controlled by casting the input tensors to the desired type.

Parameters

dataType	the computational precision.

See also

getPrecision() precisionIsSet() resetPrecision()

9.87.4 Member Data Documentation

9.87.4.1 mLayer

```
apiv::VLayer* nvinfer1::ILayer::mLayer  [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.88 nvinfer1::ILogger Class Reference

Application-implemented logging interface for the builder, refitter and runtime.

```
#include <NvInferRuntime.h>
```

Public Types

- enum class Severity : int32_t {
  kINTERNAL_ERROR = 0 , kERROR = 1 , kWARNING = 2 , kINFO = 3 ,
  kVERBOSE = 4 }

  The severity corresponding to a log message.

Public Member Functions

- virtual void log (Severity severity, AsciiChar const *msg) noexcept=0

  A callback implemented by the application to handle logging messages;.
- ILogger ()=default
- virtual ~ILogger ()=default

9.88.1 Detailed Description

Application-implemented logging interface for the builder, refitter and runtime.

DO NOT REFER TO namespace v_1_0 IN CODE. ALWAYS USE nvinfer1 INSTEAD. The name v_1_0 may change in future versions of TensoRT.

The logger used to create an instance of IBuilder, IRuntime or IRefitter is used for all objects created through that interface. The logger must be valid until all objects created are released.

The Logger object implementation must be thread safe. All locking and synchronization is pushed to the interface implementation and TensorRT does not hold any synchronization primitives when calling the interface functions.

9.88.2 Member Enumeration Documentation

9.88.2.1 Severity

```
enum class nvinfer1::ILogger::Severity :  int32_t  [strong]
```

The severity corresponding to a log message.

Enumerator

kINTERNAL_ERROR	An internal error has occurred. Execution is unrecoverable.
kERROR	An application error has occurred.
kWARNING	An application error has been discovered, but TensorRT has recovered or fallen back to a default.
kINFO	Informational messages with instructional information.
kVERBOSE	Verbose messages with debugging information.

9.88.3 Constructor & Destructor Documentation

9.88.3.1 ILogger()

```
nvinfer1::ILogger::ILogger ( )  [default]
```

9.88.3.2 ~ILogger()

```
virtual nvinfer1::ILogger::~ILogger ( )  [virtual], [default]
```

9.88.4 Member Function Documentation

9.88.4.1 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

severity	The severity of the message.
msg	A null-terminated log message.

Warning

Loggers used in the safety certified runtime must set a maximum message length and truncate messages exceeding this length. It is up to the implementer of the derived class to define a suitable limit that will prevent buffer overruns, resource exhaustion, and other security vulnerabilities in their implementation. The TensorRT safety certified runtime will never emit messages longer than 1024 bytes.

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:

- NvInferRuntime.h

9.89 nvinfer1::ILoggerFinder Class Reference

A virtual base class to find a logger. Allows a plugin to find an instance of a logger if it needs to emit a log message. A pointer to an instance of this class is passed to a plugin shared library on initialization when that plugin is serialized as part of a version-compatible plan. See the plugin chapter in the developer guide for details.

`#include <NvInferRuntime.h>`

Public Member Functions

- virtual ILogger * findLogger ()=0

  Get the logger used by the engine or execution context which called the plugin method.

Protected Member Functions

- virtual ~ILoggerFinder ()=default

9.89.1 Detailed Description

A virtual base class to find a logger. Allows a plugin to find an instance of a logger if it needs to emit a log message. A pointer to an instance of this class is passed to a plugin shared library on initialization when that plugin is serialized as part of a version-compatible plan. See the plugin chapter in the developer guide for details.

9.89.2 Constructor & Destructor Documentation

9.89.2.1 ~ILoggerFinder()

```
virtual nvinfer1::ILoggerFinder::~ILoggerFinder ( )  [protected], [virtual], [default]
```

9.89.3 Member Function Documentation

9.89.3.1 findLogger()

```
virtual ILogger * nvinfer1::ILoggerFinder::findLogger ( )  [pure virtual]
```

Get the logger used by the engine or execution context which called the plugin method.

Warning

Must be called from the thread in which the plugin method was called.

Returns

A pointer to the logger.

The documentation for this class was generated from the following file:

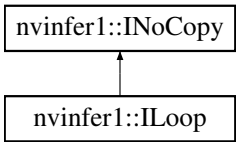
- NvInferRuntime.h

9.90 nvinfer1::ILoop Class Reference

Helper for creating a recurrent subgraph.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::ILoop:



Public Member Functions

- IRecurrenceLayer * addRecurrence (ITensor &initialValue) noexcept

  Create a recurrence layer for this loop with initialValue as its first input.
- ITripLimitLayer * addTripLimit (ITensor &tensor, TripLimit limit) noexcept

  Add a trip-count limiter, based on the given tensor.
- IIteratorLayer * addIterator (ITensor &tensor, int32_t axis=0, bool reverse=false) noexcept

  Return layer that subscripts tensor by loop iteration.
- ILoopOutputLayer * addLoopOutput (ITensor &tensor, LoopOutput outputKind, int32_t axis=0) noexcept

  Make an output for this loop, based on the given tensor.
- void setName (char const *name) noexcept

  Set the name of the loop.
- char const * getName () const noexcept

  Return the name of the loop.

Protected Member Functions

- virtual ~ILoop () noexcept=default

Protected Attributes

- apiv::VLoop * mImpl

9.90.1 Detailed Description

Helper for creating a recurrent subgraph.

An ILoop defines a loop within a network. It supports the implementation of recurrences, which are crucial for iterative computations, such as RNNs for natural language processing and time-series analysis.

9.90.2 Constructor & Destructor Documentation

9.90.2.1 ~ILoop()

```
virtual nvinfer1::ILoop::~ILoop ( )  [protected], [virtual], [default], [noexcept]
```

9.90.3 Member Function Documentation

9.90.3.1 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.

9.90.3.2 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.

9.90.3.3 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.

9.90.3.4 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.

9.90.3.5 getName()

```
char const * nvinfer1::ILoop::getName ( ) const  [inline], [noexcept]
```

Return the name of the loop.

See also

setName()

9.90.3.6 setName()

```
void nvinfer1::ILoop::setName (
            char const * name )  [inline], [noexcept]
```

Set the name of the loop.

The name is used in error diagnostics. This method copies the name string.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

See also

getName()

9.90.4 Member Data Documentation

9.90.4.1 mImpl

```
apiv::VLoop* nvinfer1::ILoop::mImpl  [protected]
```

The documentation for this class was generated from the following file:

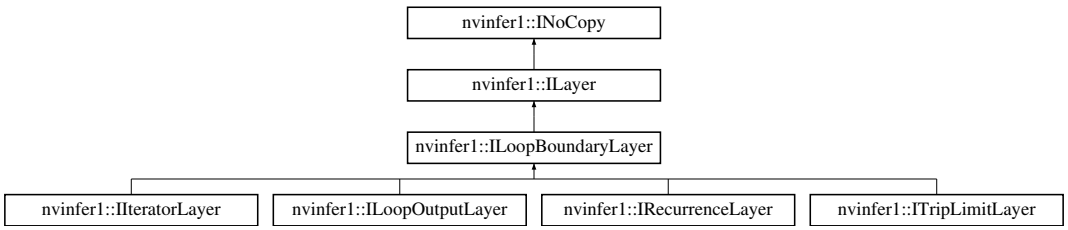
- NvInfer.h

9.91 nvinfer1::ILoopBoundaryLayer Class Reference

This is a base class for Loop boundary layers.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::ILoopBoundaryLayer:



Public Member Functions

- ILoop * getLoop () const noexcept

  Get a pointer to ILoop associated with this boundary layer.

Protected Member Functions

- virtual ~ILoopBoundaryLayer () noexcept=default

Protected Attributes

- apiv::VLoopBoundaryLayer * mBoundary

9.91.1 Detailed Description

This is a base class for Loop boundary layers.

The loop boundary layers are used to define loops within a network, enabling the implementation of recurrences. The boundary layers for a loop are created by class ILoop.

There are four kinds of boundary layers.

- ITripLimitLayer: controls the number of loop iterations.
- IIterationLayer: iterates over an input tensor.
- IRecurrenceLayer: returns an initial value or value from the previous loop iteration.
- ILoopOutputLayer: generates an output tensor from the loop iterations.

9.91.2 Constructor & Destructor Documentation

9.91.2.1 ~ILoopBoundaryLayer()

```
virtual nvinfer1::ILoopBoundaryLayer::~ILoopBoundaryLayer ( )  [protected], [virtual], [default],
[noexcept]
```

9.91.3 Member Function Documentation

9.91.3.1 getLoop()

```
ILoop * nvinfer1::ILoopBoundaryLayer::getLoop ( ) const  [inline], [noexcept]
```

Get a pointer to ILoop associated with this boundary layer.

9.91.4 Member Data Documentation

9.91.4.1 mBoundary

```
apiv::VLoopBoundaryLayer* nvinfer1::ILoopBoundaryLayer::mBoundary  [protected]
```

The documentation for this class was generated from the following file:

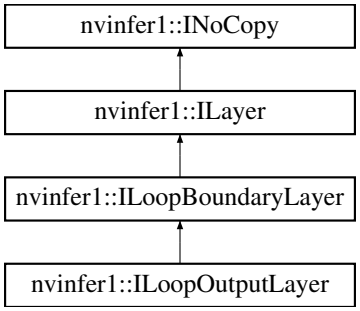
- NvInfer.h

9.92 nvinfer1::ILoopOutputLayer Class Reference

An ILoopOutputLayer is the sole way to get output from a loop.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::ILoopOutputLayer:



Public Member Functions

- LoopOutput getLoopOutput () const noexcept

  Get which kind a loop output has.
- void setAxis (int32_t axis) noexcept

  Set where to insert the contenation axis. Ignored if getLoopOutput() is kLAST_VALUE.
- int32_t getAxis () const noexcept

  Get axis being concatenated over.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~ILoopOutputLayer () noexcept=default

Protected Attributes

- apiv::VLoopOutputLayer * mImpl

9.92.1 Detailed Description

An ILoopOutputLayer is the sole way to get output from a loop.

The first input tensor must be defined inside the loop; the output tensor is outside the loop. The second input tensor, if present, must be defined outside the loop.

If getLoopOutput() is kLAST_VALUE, a single input must be provided, and that input must be from an IRecurrenceLayer in the same loop.

If getLoopOutput() is kCONCATENATE or kREVERSE, a second input must be provided. The second input must be a 0D shape tensor, defined before the loop commences, that specifies the concatenation length of the output.

The output tensor has j more dimensions than the input tensor, where j == 0 if getLoopOutput() is kLAST_VALUE j == 1 if getLoopOutput() is kCONCATENATE or kREVERSE.

9.92.2 Constructor & Destructor Documentation

9.92.2.1 ~ILoopOutputLayer()

```
virtual nvinfer1::ILoopOutputLayer::~ILoopOutputLayer ( )  [protected], [virtual], [default],
[noexcept]
```

9.92.3 Member Function Documentation

9.92.3.1 getAxis()

```
int32_t nvinfer1::ILoopOutputLayer::getAxis ( ) const  [inline], [noexcept]
```

Get axis being concatenated over.

9.92.3.2 getLoopOutput()

```
LoopOutput nvinfer1::ILoopOutputLayer::getLoopOutput ( ) const  [inline], [noexcept]
```

Get which kind a loop output has.

9.92.3.3 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.

9.92.3.4 setInput()

```
void nvinfer1::ILayer::setInput (
            int32_t index,
            ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor Sets the input tensor for the given index. The index must be 0 for a kLAST_VALUE loop output layer. Loop output layer is converted to a kCONCATENATE or kREVERSE loop output layer by calling setInput with an index 1. A kCONCATENATE or kREVERSE loop output layer cannot be converted back to a kLAST_VALUE loop output layer.

For a kCONCATENATE or kREVERSE loop output layer, the values 0 and 1 are valid. The indices in the 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 shape tensor of type Int32 or Int64.

If this function is called with the value 1, then the function getNbInputs() changes from returning 1 to 2.

9.92.4 Member Data Documentation

9.92.4.1 mImpl

```
apiv::VLoopOutputLayer* nvinfer1::ILoopOutputLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

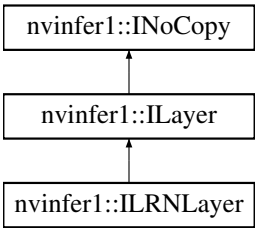
- NvInfer.h

9.93 nvinfer1::ILRNLayer Class Reference

A LRN layer in a network definition.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::ILRNLayer:



Public Member Functions

- void setWindowSize (int64_t windowSize) noexcept
  Set the LRN window size.
- int64_t getWindowSize () const noexcept
  Get the LRN window size.
- void setAlpha (float alpha) noexcept
  Set the LRN alpha value.
- float getAlpha () const noexcept
  Get the LRN alpha value.
- void setBeta (float beta) noexcept
  Set the LRN beta value.
- float getBeta () const noexcept
  Get the LRN beta value.
- void setK (float k) noexcept
  Set the LRN K value.
- float getK () const noexcept
  Get the LRN K value.

Protected Member Functions

- virtual ~ILRNLayer () noexcept=default

Protected Attributes

- apiv::VLRNLayer * mImpl

9.93.1 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.

9.93.2 Constructor & Destructor Documentation

9.93.2.1 ~ILRNLayer()

```
virtual nvinfer1::ILRNLayer::~ILRNLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.93.3 Member Function Documentation

9.93.3.1 getAlpha()

```
float nvinfer1::ILRNLayer::getAlpha ( ) const  [inline], [noexcept]
```

Get the LRN alpha value.

See also

setAlpha()

9.93.3.2 getBeta()

```
float nvinfer1::ILRNLayer::getBeta ( ) const  [inline], [noexcept]
```

Get the LRN beta value.

See also

setBeta()

9.93.3.3 getK()

```
float nvinfer1::ILRNLayer::getK ( ) const  [inline], [noexcept]
```

Get the LRN K value.

See also

setK()

9.93.3.4 getWindowSize()

```
int64_t nvinfer1::ILRNLayer::getWindowSize ( ) const  [inline], [noexcept]
```

Get the LRN window size.

See also

getWindowStride()

9.93.3.5 setAlpha()

```
void nvinfer1::ILRNLayer::setAlpha (
            float alpha )  [inline], [noexcept]
```

Set the LRN alpha value.

The valid range is [-1e20, 1e20].

See also

getAlpha()

9.93.3.6 setBeta()

```
void nvinfer1::ILRNLayer::setBeta (
            float beta )  [inline], [noexcept]
```

Set the LRN beta value.

The valid range is [0.01, 1e5f].

See also

getBeta()

9.93.3.7 setK()

```
void nvinfer1::ILRNLayer::setK (
            float k )  [inline], [noexcept]
```

Set the LRN K value.

The valid range is [1e-5, 1e10].

See also

getK()

9.93.3.8 setWindowSize()

```
void nvinfer1::ILRNLayer::setWindowSize (
            int64_t windowSize )  [inline], [noexcept]
```

Set the LRN window size.

The window size must be odd and in the range of [1, 15].

If executing this layer on the DLA, only values in the set, [3, 5, 7, 9], are valid.

See also

setWindowStride()

9.93.4 Member Data Documentation

9.93.4.1 mImpl

```
apiv::VLRNLayer* nvinfer1::ILRNLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

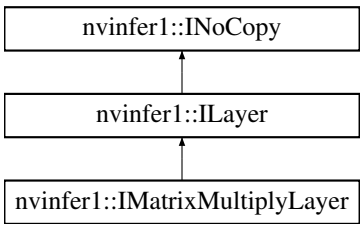
- NvInfer.h

9.94 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

9.94.1 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.

9.94.2 Constructor & Destructor Documentation

9.94.2.1 ~IMatrixMultiplyLayer()

```
virtual nvinfer1::IMatrixMultiplyLayer::~IMatrixMultiplyLayer ( )  [protected], [virtual], [default],
[noexcept]
```

9.94.3 Member Function Documentation

9.94.3.1 getOperation()

```
MatrixOperation nvinfer1::IMatrixMultiplyLayer::getOperation (
            int32_t index ) const  [inline], [noexcept]
```

Get the operation for an input tensor.

Parameters

index	Input tensor number (0 or 1).

See also

setOperation()

9.94.3.2 setOperation()

```
void nvinfer1::IMatrixMultiplyLayer::setOperation (
            int32_t index,
            MatrixOperation op )  [inline], [noexcept]
```

Set the operation for an input tensor.

Parameters

index	Input tensor number (0 or 1).
op	New operation.

See also

getOperation()

9.94.4 Member Data Documentation

9.94.4.1 mImpl

```
apiv::VMatrixMultiplyLayer* nvinfer1::IMatrixMultiplyLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

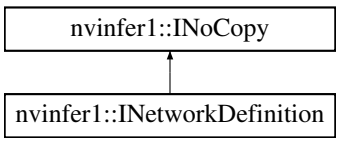
- NvInfer.h

9.95 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 const &dimensions) noexcept

  Add an input tensor to the network.
- void markOutput (ITensor &tensor) noexcept

  Mark a tensor as a network output.
- bool markDebug (ITensor &tensor) noexcept

  Mark a tensor as a debug tensor.
- bool unmarkDebug (ITensor &tensor) noexcept

  Unmark a tensor as a debug tensor.
- bool isDebugTensor (nvinfer1::ITensor const &tensor) const noexcept

  Check if a tensor is marked as debug tensor.
- IActivationLayer * addActivation (ITensor &input, ActivationType type) noexcept

  Add an activation layer to the network.
- ILRNLayer * addLRN (ITensor &input, int64_t window, float alpha, float beta, float k) noexcept

  Add a LRN layer to the network.
- IScaleLayer * addScale (ITensor &input, ScaleMode mode, Weights shift, Weights scale, Weights power) noexcept

  Add a Scale layer to the network.
- ISoftMaxLayer * addSoftMax (ITensor &input) noexcept

  Add a SoftMax layer to the network.
- IConcatenationLayer * addConcatenation (ITensor *const *inputs, int32_t nbInputs) noexcept

  Add a concatenation layer to the network.
- IElementWiseLayer * addElementWise (ITensor &input1, ITensor &input2, ElementWiseOperation op) noexcept

  Add an elementwise layer to the network.
- IUnaryLayer * addUnary (ITensor &input, UnaryOperation operation) noexcept

  Add a unary layer to the network.
- IShuffleLayer * addShuffle (ITensor &input) noexcept

  Add a shuffle layer to the network.
- IOneHotLayer * addOneHot (ITensor &indices, ITensor &values, ITensor &depth, int32_t axis) noexcept

  Add a OneHot layer to the network.
- int32_t getNbLayers () const noexcept

  Get the number of layers in the network.
- ILayer * getLayer (int32_t index) const noexcept

  Get the layer specified by the given index.
- int32_t getNbInputs () const noexcept

  Get the number of inputs in the network.
- ITensor * getInput (int32_t index) const noexcept

  Get the input tensor specified by the given index.
- int32_t getNbOutputs () const noexcept

  Get the number of outputs in the network.
- ITensor * getOutput (int32_t index) const noexcept

  Get the output tensor specified by the given index.
- IReduceLayer * addReduce (ITensor &input, ReduceOperation operation, uint32_t reduceAxes, bool keep↩ Dimensions) noexcept

  Add a reduce layer to the network.

- ITopKLayer * addTopK (ITensor &input, TopKOperation op, int32_t k, uint32_t reduceAxes) noexcept

  Add a TopK layer to the network.
- IGatherLayer * addGather (ITensor &data, ITensor &indices, int32_t axis) noexcept

  Add gather with mode GatherMode::kDEFAULT and specified axis and nbElementWiseDims=0.
- IGatherLayer * addGatherV2 (ITensor &data, ITensor &indices, GatherMode mode) noexcept

  Add gather with specified mode, axis=0 and nbElementWiseDims=0.
- IRaggedSoftMaxLayer * addRaggedSoftMax (ITensor &input, ITensor &bounds) noexcept

  Add a RaggedSoftMax layer to the network.
- IMatrixMultiplyLayer * addMatrixMultiply (ITensor &input0, MatrixOperation op0, ITensor &input1, MatrixOperation op1) noexcept

  Add a MatrixMultiply layer to the network.
- INonZeroLayer * addNonZero (ITensor &input) noexcept

  Add a nonzero layer to the network.
- IConstantLayer * addConstant (Dims const &dimensions, Weights weights) noexcept

  Add a constant layer to the network.
- IIdentityLayer * addIdentity (ITensor &input) noexcept

  Add an identity layer.
- ICastLayer * addCast (ITensor &input, DataType toType) noexcept

  Add a cast layer.
- void removeTensor (ITensor &tensor) noexcept

  remove a tensor from the network definition.
- void unmarkOutput (ITensor &tensor) noexcept

  unmark a tensor as a network output.
- TRT_DEPRECATED IPluginV2Layer * addPluginV2 (ITensor *const *inputs, int32_t nbInputs, IPluginV2 &plugin) noexcept

  Add a plugin layer to the network using the IPluginV2 interface.
- IPluginV3Layer * addPluginV3 (ITensor *const *inputs, int32_t nbInputs, ITensor *const *shapeInputs, int32_t nbShapeInputs, IPluginV3 &plugin) noexcept

  Add a plugin layer implementing the IPluginV3 interface to the network.
- ISliceLayer * addSlice (ITensor &input, Dims const &start, Dims const &size, Dims const &stride) noexcept

  Add a slice layer to the network.
- void setName (char const *name) noexcept

  Sets the name of the network.
- char const * getName () const noexcept

  Returns the name associated with the network.
- IShapeLayer * addShape (ITensor &input) noexcept

  Add a shape layer to the network.
- TRT_DEPRECATED bool hasImplicitBatchDimension () const noexcept

  Query whether the network was created with an implicit batch dimension.
- NetworkDefinitionCreationFlags getFlags () const noexcept

  Get the network definition creation flags for this network definition object. Defaults to 0.
- bool getFlag (NetworkDefinitionCreationFlag networkDefinitionCreationFlag) const noexcept

  Returns true if the network definition creation flag is set.
- bool markOutputForShapes (ITensor &tensor) noexcept

  Enable tensor's value to be computed by IExecutionContext::getShapeBinding.
- bool unmarkOutputForShapes (ITensor &tensor) noexcept

  Undo markOutputForShapes.

- IParametricReLULayer * addParametricReLU (ITensor &input, ITensor &slope) noexcept

  Add a parametric ReLU layer to the network.
- IConvolutionLayer * addConvolutionNd (ITensor &input, int64_t nbOutputMaps, Dims const &kernelSize, Weights kernelWeights, Weights biasWeights) noexcept

  Add a multi-dimension convolution layer to the network.
- IPoolingLayer * addPoolingNd (ITensor &input, PoolingType type, Dims const &windowSize) noexcept

  Add a multi-dimension pooling layer to the network.
- IDeconvolutionLayer * addDeconvolutionNd (ITensor &input, int64_t nbOutputMaps, Dims kernelSize, Weights kernelWeights, Weights biasWeights) noexcept

  Add a multi-dimension deconvolution layer to the network.
- IScaleLayer * addScaleNd (ITensor &input, ScaleMode mode, Weights shift, Weights scale, Weights power, int32_t channelAxis) noexcept

  Add a multi-dimension scale layer to the network.
- IResizeLayer * addResize (ITensor &input) noexcept

  Add a resize layer to the network.
- ILoop * addLoop () noexcept

  Add a loop to the network.
- IIfConditional * addIfConditional () noexcept

  Add an if-then-else to the network.
- ISelectLayer * addSelect (ITensor &condition, ITensor &thenInput, ITensor &elseInput) noexcept

  Add a select layer to the network.
- IAssertionLayer * addAssertion (ITensor &condition, char const *message) noexcept

  Add an assertion layer to the network.
- TRT_DEPRECATED IFillLayer * addFill (Dims const &dimensions, FillOperation op) noexcept

  Add a fill layer to the network.
- IFillLayer * addFill (Dims const &dimensions, FillOperation op, DataType outputType) noexcept

  Add a fill layer to the network.
- IPaddingLayer * addPaddingNd (ITensor &input, Dims const &prePadding, Dims const &postPadding) noexcept

  Add a padding layer to the network. Only 2D padding is currently supported.
- bool setWeightsName (Weights weights, char const *name) noexcept

  Associate a name with all current uses of the given weights.
- void setErrorRecorder (IErrorRecorder *recorder) noexcept

  Set the ErrorRecorder for this interface.
- IErrorRecorder * getErrorRecorder () const noexcept

  get the ErrorRecorder assigned to this interface.
- TRT_DEPRECATED IDequantizeLayer * addDequantize (ITensor &input, ITensor &scale) noexcept

  Add a dequantization layer to the network.
- IDequantizeLayer * addDequantize (ITensor &input, ITensor &scale, DataType outputType) noexcept

  Add a dequantization layer to the network.
- IScatterLayer * addScatter (ITensor &data, ITensor &indices, ITensor &updates, ScatterMode mode) noexcept

  Add a Scatter layer to the network with specified mode and axis=0.
- TRT_DEPRECATED IQuantizeLayer * addQuantize (ITensor &input, ITensor &scale) noexcept

  Add a quantization layer to the network.
- IQuantizeLayer * addQuantize (ITensor &input, ITensor &scale, DataType outputType) noexcept

  Add a quantization layer to the network.
- IDynamicQuantizeLayer * addDynamicQuantize (ITensor &input, int32_t axis, int32_t blockSize, DataType outputType, DataType scaleType) noexcept

Add a dynamic quantization layer to the network.

- IEinsumLayer * addEinsum (ITensor *const *inputs, int32_t nbInputs, char const *equation) noexcept

  Add an Einsum layer to the network.
- IGridSampleLayer * addGridSample (ITensor &input, ITensor &grid) noexcept

  Add a GridSample layer to the network.
- INMSLayer * addNMS (ITensor &boxes, ITensor &scores, ITensor &maxOutputBoxesPerClass) noexcept

  Add a non-maximum suppression layer to the network.
- IReverseSequenceLayer * addReverseSequence (ITensor &input, ITensor &sequenceLens) noexcept

  Add a ReverseSequence layer to the network.
- INormalizationLayer * addNormalization (ITensor &input, ITensor &scale, ITensor &bias, uint32_t axesMask) noexcept

  Add a normalization layer to the network.
- ICumulativeLayer * addCumulative (ITensor &input, ITensor &axis, CumulativeOperation operation, bool exclusive, bool reverse) noexcept

  Add a cumulative layer to the network.
- virtual IBuilder & getBuilder () const noexcept

  Return the builder from which this INetworkDefinition was created.
- bool markWeightsRefittable (char const *name) noexcept

  Mark weights as refittable when the builder flag kREFIT_INDIVIDUAL is set.
- bool unmarkWeightsRefittable (char const *name) noexcept

  Unmark weights as refittable when the builder flag kREFIT_INDIVIDUAL is set.
- bool areWeightsMarkedRefittable (char const *name) const noexcept

  Whether the weight has been marked as refittable.
- ISqueezeLayer * addSqueeze (ITensor &input, ITensor &axes) noexcept

  Add a squeeze layer to the network.
- IUnsqueezeLayer * addUnsqueeze (ITensor &input, ITensor &axes) noexcept

  Add an unsqueeze layer to the network.

Protected Attributes

- apiv::VNetworkDefinition * mImpl

Additional Inherited Members

9.95.1 Detailed Description

A network definition for input to the builder.

A network definition defines the structure of the network, and combined with a IBuilderConfig, is built into an engine using an IBuilder. An INetworkDefinition can have all dimensions explicit, full dims mode, in the network definition. The former mode, i.e. the implicit batch size mode, has been deprecated.

A network with implicit batch dimensions returns the dimensions of a layer without the implicit dimension, and instead the batch is specified at execute/enqueue time. If the network has all dimensions specified, then the first dimension follows elementwise broadcast rules: if it is 1 for some inputs and is some value N for all other inputs, then the first dimension of each output is N, and the inputs with 1 for the first dimension are broadcast. Having divergent batch sizes across inputs to a layer is not supported.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.95.2 Constructor & Destructor Documentation

9.95.2.1 ~INetworkDefinition()

```
virtual nvinfer1::INetworkDefinition::~INetworkDefinition ( )  [virtual], [default], [noexcept]
```

9.95.3 Member Function Documentation

9.95.3.1 addActivation()

```
IActivationLayer * nvinfer1::INetworkDefinition::addActivation (
            ITensor & input,
            ActivationType type )  [inline], [noexcept]
```

Add an activation layer to the network.

Parameters

input	The input tensor to the layer.
type	The type of activation function to apply.

Note that the setAlpha() and setBeta() methods must be used on the output for activations that require these parameters.

See also

IActivationLayer ActivationType

Warning

Int32 and Int64 are valid only for activation type kRELU.

Returns

The new activation layer, or nullptr if it could not be created.

9.95.3.2 addAssertion()

```
IAssertionLayer * nvinfer1::INetworkDefinition::addAssertion (
            ITensor & condition,
            char const * message )  [inline], [noexcept]
```

Add an assertion layer to the network.

Parameters

condition	The input tensor to the layer.
message	A message to print if the assertion fails.

See also

IAssertionLayer

Returns

The new assertion layer, or nullptr if it could not be created.

The input tensor must be a boolean shape tensor.

9.95.3.3 addCast()

```
ICastLayer * nvinfer1::INetworkDefinition::addCast (
            ITensor & input,
            DataType toType )  [inline], [noexcept]
```

Add a cast layer.

Parameters

input	The input tensor to the layer.
toType	The DataType of the output tensor

See also

ICastLayer

Returns

The new cast layer, or nullptr if it could not be created.

9.95.3.4 addConcatenation()

```
IConcatenationLayer * nvinfer1::INetworkDefinition::addConcatenation (
            ITensor *const * inputs,
            int32_t nbInputs )  [inline], [noexcept]
```

Add a concatenation layer to the network.

Parameters

inputs	The input tensors to the layer.
nbInputs	The number of input tensors.

See also

IConcatenationLayer

Returns

The new concatenation layer, or nullptr if it could not be created.

Warning

All tensors must have the same dimensions except along the concatenation axis.

9.95.3.5 addConstant()

```
IConstantLayer * nvinfer1::INetworkDefinition::addConstant (
            Dims const & dimensions,
            Weights weights )  [inline], [noexcept]
```

Add a constant layer to the network.

Parameters

dimensions	The dimensions of the constant.
weights	The constant value, represented as weights.

See also

IConstantLayer

Returns

The new constant layer, or nullptr if it could not be created.

If weights.type is DataType::kINT32, the output is a tensor of 32-bit indices. Otherwise the output is a tensor of real values and the output type will be follow TensorRT's normal precision rules.

If a wildcard dimension is used, the volume of the runtime dimensions must equal the number of weights specified.

Warning

DataType::kUINT8 not supported.

9.95.3.6 addConvolutionNd()

```
IConvolutionLayer * nvinfer1::INetworkDefinition::addConvolutionNd (
            ITensor & input,
            int64_t nbOutputMaps,
            Dims const & kernelSize,
            Weights kernelWeights,
            Weights biasWeights )  [inline], [noexcept]
```

Add a multi-dimension convolution layer to the network.

Parameters

input	The input tensor to the convolution.
nbOutputMaps	The number of output feature maps for the convolution.
kernelSize	The multi-dimensions of the convolution kernel.
kernelWeights	The kernel weights for the convolution.
biasWeights	The bias weights for the convolution. Weights{} represents no bias.

See also

IConvolutionLayer

Warning

It is an error to specify a wildcard value for the 'C' dimension of the input tensor.

Int32 tensors are not valid input tensors.

Only 2D or 3D convolution is supported.

Returns

The new convolution layer, or nullptr if it could not be created.

9.95.3.7 addCumulative()

```
ICumulativeLayer * nvinfer1::INetworkDefinition::addCumulative (
            ITensor & input,
            ITensor & axis,
            CumulativeOperation operation,
            bool exclusive,
            bool reverse )  [inline], [noexcept]
```

Add a cumulative layer to the network.

Parameters

input	The input tensor to the layer.
axis	The axis tensor to apply the cumulative operation on. Currently, it must be a build-time constant 0D shape tensor and must be in the range [-rank(input), rank(input)-1]. Negative value means counting dimensions from the back.
operation	The reduction operation to perform.
exclusive	The boolean that specifies whether it is an exclusive cumulative or inclusive cumulative.
reverse	The boolean that specifies whether the cumulative operation should be applied backward.

The cumulative layer works by performing the specified cumulative `operation` to the tensor `input` on the axis specified by `axis`.

See also

ICumulativeLayer

Returns

The new cumulative layer, or nullptr if it could not be created.

9.95.3.8 addDeconvolutionNd()

```
IDeconvolutionLayer * nvinfer1::INetworkDefinition::addDeconvolutionNd (
            ITensor & input,
            int64_t nbOutputMaps,
            Dims kernelSize,
            Weights kernelWeights,
            Weights biasWeights )  [inline], [noexcept]
```

Add a multi-dimension deconvolution layer to the network.

Parameters

input	The input tensor to the layer.
nbOutputMaps	The number of output feature maps.
kernelSize	The multi-dimensions of the deconvolution kernel.
kernelWeights	The kernel weights for the deconvolution.
biasWeights	The bias weights for the deconvolution. Weights{} represents no bias.

See also

IDeconvolutionLayer

Warning

It is an error to specify a wildcard value for the 'C' dimension of the input tensor.

Int32 tensors are not valid input tensors.

Only 2D or 3D deconvolution is supported.

Returns

The new deconvolution layer, or nullptr if it could not be created.

9.95.3.9 addDequantize() [1/2]

```
TRT_DEPRECATED IDequantizeLayer * nvinfer1::INetworkDefinition::addDequantize (
            ITensor & input,
            ITensor & scale ) [inline], [noexcept]
```

Add a dequantization layer to the network.

Parameters

input	The input tensor to be quantized.
scale	A tensor with the scale value.

See also

IDequantizeLayer

`input` tensor data type must be DataType::kINT8 or DataType::kFP8. `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.

Deprecated Deprecated in TensorRT 9.0. Superseded by three-argument addDequantize.

9.95.3.10 addDequantize() [2/2]

```
IDequantizeLayer * nvinfer1::INetworkDefinition::addDequantize (
            ITensor & input,
            ITensor & scale,
            DataType outputType )  [inline], [noexcept]
```

Add a dequantization layer to the network.

Parameters

input	The input tensor to be dequantized.
scale	A tensor with the scale value.
outputType	Output tensor data type.

See also

IDequantizeLayer

`input` tensor data type must be DataType::kINT8, DataType::kFP8 or DataType::kINT4. `scale` tensor data type defaults to DataType::kFLOAT. For strongly typed networks, it must be the same as the output data type. The subgraph which terminates with the `scale` tensor must be a build-time constant. `outputType` output tensor data type, default value is DataType::kFLOAT. Future calls to set output type using setToType or setOutputType must be consistent. For strongly typed networks, it must be the same as the scale data type.

Returns

The new quantization layer, or nullptr if it could not be created.

9.95.3.11 addDynamicQuantize()

```
IDynamicQuantizeLayer * nvinfer1::INetworkDefinition::addDynamicQuantize (
            ITensor & input,
            int32_t axis,
            int32_t blockSize,
            DataType outputType,
            DataType scaleType )  [inline], [noexcept]
```

Add a dynamic quantization layer to the network.

This layer performs dynamic block quantization of its input tensor and outputs the quantized data and the computed block scale-factors. The block size is currently limited to 16 and the size of the blocked axis must be divisible by 16.

Parameters

input	The input tensor to be quantized. Its data type must be one of DataType::kFLOAT, DataType::kHALF, or DataType::kBF16. Currently only 2D and 3D inputs are supported.
axis	The axis that is sliced into blocks. The axis must be the last or second to last dimension.
blockSize	The number of elements that are quantized using a shared scale factor. Currently only blocks of 16 elements are supported.
outputType	The data type of the quantized output tensor, must be DataType::kFP4. Future calls to set output type using setToType or setOutputType must be consistent.
scaleType	The data type of the scale factor used for quantizing the input data, must be DataType::kFP8.

Returns

The new dynamic quantization layer, or nullptr if it could not be created.

See also

IDynamicQuantizeLayer

9.95.3.12 addEinsum()

```
IEinsumLayer * nvinfer1::INetworkDefinition::addEinsum (
            ITensor *const * inputs,
            int32_t nbInputs,
            char const * equation )  [inline], [noexcept]
```

Add an Einsum layer to the network.

Parameters

inputs	The input tensors to the layer.
nbInputs	The number of input tensors.
equation	The equation of the layer

See also

IEinsumLayer

Returns

The new Einsum layer, or nullptr if it could not be created.

9.95.3.13 addElementWise()

```
IElementWiseLayer * nvinfer1::INetworkDefinition::addElementWise (
            ITensor & input1,
            ITensor & input2,
            ElementWiseOperation op )  [inline], [noexcept]
```

Add an elementwise layer to the network.

Parameters

input1	The first input tensor to the layer.
input2	The second input tensor to the layer.
op	The binary operation that the layer applies.

The input tensors must have the same rank and compatible type. Two types are compatible if they are the same type or are both in the set {kFLOAT, kHALF}. For each dimension, their lengths must match, or one of them must be one. In the latter case, the tensor is broadcast along that axis.

The output tensor has the same rank as the inputs. For each dimension, its length is the maximum of the lengths of the corresponding input dimension.

The inputs are shape tensors if the output is a shape tensor.

See also

IElementWiseLayer

Returns

The new elementwise layer, or nullptr if it could not be created.

9.95.3.14 addFill() [1/2]

```
TRT_DEPRECATED IFillLayer * nvinfer1::INetworkDefinition::addFill (
            Dims const & dimensions,
            FillOperation op )  [inline], [noexcept]
```

Add a fill layer to the network.

Parameters

dimensions	The output tensor dimensions if input 0 is missing.
op	The fill operation that the layer applies.

Warning

For FillOperation::kLINSPACE, dimensions.nbDims must be 1 for static start/delta. If delta is provided as a 1D tensor, the length of delta must match dimensions.nbDims.

This layer is non-deterministic across subsequent calls as the same inputs will produce different output tensors if `op` is either FillOperation::kRANDOM_UNIFORM or FillOperation::kRANDOM_NORMAL due to random state being shared across calls. The output tensors generated are determinstic when starting from the same initial state.

See also

IFillLayer

Returns

The new fill layer, or nullptr if it could not be created.

Deprecated Deprecated in TensorRT 9.0. Superseded by three-argument addFill.

9.95.3.15 addFill() [2/2]

```
IFillLayer * nvinfer1::INetworkDefinition::addFill (
            Dims const & dimensions,
            FillOperation op,
            DataType outputType )  [inline], [noexcept]
```

Add a fill layer to the network.

Parameters

dimensions	The output tensor dimensions if input 0 is missing.
op	The fill operation that the layer applies.
outputType	Optional output tensor data type, must be DataType::kFLOAT, DataType::kHALF, DataType::kINT32, or DataType::kINT64. This parameter is only used for static alpha/beta. Future calls to set output type using setToType or setOutputType must be consistent.

Warning

For FillOperation::kLINSPACE, dimensions.nbDims must be 1 for static start/delta. If delta is provided as a 1D tensor, the length of delta must match dimensions.nbDims.

This layer is non-deterministic across subsequent calls as the same inputs will produce different output tensors if `op` is either FillOperation::kRANDOM_UNIFORM or FillOperation::kRANDOM_NORMAL due to random state being shared across calls. The output tensors generated are deterministic when starting from the same initial state.

See also

IFillLayer

Returns

The new fill layer, or nullptr if it could not be created.

9.95.3.16 addGather()

```
IGatherLayer * nvinfer1::INetworkDefinition::addGather (
            ITensor & data,
            ITensor & indices,
            int32_t axis )  [inline], [noexcept]
```

Add gather with mode GatherMode::kDEFAULT and specified axis and nbElementWiseDims=0.

Parameters

data	The tensor to gather values from.
indices	The tensor to get indices from to populate the output tensor.
axis	The axis in the data tensor to gather on.

See also

IGatherLayer

Returns

The new gather layer, or nullptr if it could not be created.

9.95.3.17 addGatherV2()

```
IGatherLayer * nvinfer1::INetworkDefinition::addGatherV2 (
            ITensor & data,
            ITensor & indices,
            GatherMode mode )  [inline], [noexcept]
```

Add gather with specified mode, axis=0 and nbElementWiseDims=0.

Parameters

data	The tensor to gather values from.
indices	The tensor to get indices from to populate the output tensor.
mode	The gather mode.

See also

IGatherLayer

Returns

The new gather layer, or nullptr if it could not be created.

9.95.3.18 addGridSample()

```
IGridSampleLayer * nvinfer1::INetworkDefinition::addGridSample (
            ITensor & input,
            ITensor & grid )  [inline], [noexcept]
```

Add a GridSample layer to the network.

Parameters

input	The input tensor to the layer.
grid	The grid tensor to the layer.

See also

IGridSampleLayer

Creates a GridSample layer with a InterpolationMode::kLINEAR, unaligned corners, and SampleMode::kFILL for 4d-shape input tensors.

Returns

The new GridSample layer, or nullptr if it could not be created.

9.95.3.19 addIdentity()

```
IIdentityLayer * nvinfer1::INetworkDefinition::addIdentity (
            ITensor & input )  [inline], [noexcept]
```

Add an identity layer.

Parameters

input	The input tensor to the layer.

See also

IIdentityLayer

Returns

The new identity layer, or nullptr if it could not be created.

9.95.3.20 addIfConditional()

```
IIfConditional * nvinfer1::INetworkDefinition::addIfConditional ( )  [inline], [noexcept]
```

Add an if-then-else to the network.

An IIfConditional provides a way to conditionally execute parts of the network.

Returns

Pointer to the IIfConditional that can be used to add conditional-boundary layers for the if-then-else.

See also

IIfConditional

9.95.3.21 addInput()

```
ITensor * nvinfer1::INetworkDefinition::addInput (
            char const * name,
            DataType type,
            Dims const & dimensions )  [inline], [noexcept]
```

Add an input tensor to the network.

Each input and output tensor must have a unique name. The volume must be less than 2^31 elements.

For networks with wildcard dimensions, the volume is based on the maxima specified by an IOptimizationProfile.↩ Dimensions are normally non-negative integers. The exception is that in networks with all explicit dimensions, -1 can be used as a wildcard for a dimension to be specified at runtime. Input tensors with such a wildcard must have a corresponding entry in the IOptimizationProfiles indicating the permitted extrema, and the input dimensions must be set by IExecutionContext::setInputShape. Different IExecutionContext instances can have different dimensions. Wildcard dimensions are only supported for EngineCapability::kSTANDARD. They are not supported in safety contexts. DLA does not support Wildcard dimensions.

Tensor dimensions are specified independent of format. For example, if a tensor is formatted in "NHWC" or a vectorized format, the dimensions are still specified in the order{N, C, H, W}. For 2D images with a channel dimension, the last three dimensions are always {C,H,W}. For 3D images with a channel dimension, the last four dimensions are always {C,D,H,W}.

Parameters

name	The name of the tensor.
type	The type of the data held in the tensor.
dimensions	The dimensions of the tensor.

Warning

It is an error to specify a wildcard value on a dimension that is determined by trained parameters.

If run on DLA with explicit dimensions, only leading dimension can be a wildcard. And provided profile must have same minimum, optimum, and maximum dimensions.

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

See also

ITensor

Returns

The new tensor or nullptr if there is an error.

9.95.3.22 addLoop()

```
ILoop * nvinfer1::INetworkDefinition::addLoop ( )  [inline], [noexcept]
```

Add a loop to the network.

An ILoop provides a way to specify a recurrent subgraph.

Returns

Pointer to ILoop that can be used to add loop-boundary layers for the loop.

See also

ILoop

9.95.3.23 addLRN()

```
ILRNLayer * nvinfer1::INetworkDefinition::addLRN (
            ITensor & input,
            int64_t window,
            float alpha,
            float beta,
            float k )  [inline], [noexcept]
```

Add a LRN layer to the network.

Parameters

input	The input tensor to the layer.
window	The size of the window.
alpha	The alpha value for the LRN computation.
beta	The beta value for the LRN computation.
k	The k value for the LRN computation.

See also

ILRNLayer

Warning

Int32 tensors are not valid input tensors.

Returns

The new LRN layer, or nullptr if it could not be created.

9.95.3.24 addMatrixMultiply()

```
IMatrixMultiplyLayer * nvinfer1::INetworkDefinition::addMatrixMultiply (
            ITensor & input0,
            MatrixOperation op0,
            ITensor & input1,
            MatrixOperation op1 )  [inline], [noexcept]
```

Add a MatrixMultiply layer to the network.

Parameters

input0	The first input tensor (commonly A).
op0	The operation to apply to input0.
input1	The second input tensor (commonly B).
op1	The operation to apply to input1.

The inputs are shape tensors if the output is a shape tensor.

See also

IMatrixMultiplyLayer

Warning

Int32 tensors are not valid input tensors.

Returns

The new matrix multiply layer, or nullptr if it could not be created.

9.95.3.25 addNMS()

```
INMSLayer * nvinfer1::INetworkDefinition::addNMS (
            ITensor & boxes,
            ITensor & scores,
            ITensor & maxOutputBoxesPerClass )  [inline], [noexcept]
```

Add a non-maximum suppression layer to the network.

Parameters

boxes	The input boxes tensor to the layer.
scores	The input scores tensor to the layer.
maxOutputBoxesPerClass	The input maxOutputBoxesPerClass tensor to the layer.

See also

INMSLayer

Returns

The new NMS layer, or nullptr if it could not be created.

9.95.3.26 addNonZero()

```
INonZeroLayer * nvinfer1::INetworkDefinition::addNonZero (
            ITensor & input )  [inline], [noexcept]
```

Add a nonzero layer to the network.

Parameters

input	The input tensor to the layer.

See also

INonZeroLayer

Returns

The new nonzero layer, or nullptr if it could be created.

9.95.3.27 addNormalization()

```
INormalizationLayer * nvinfer1::INetworkDefinition::addNormalization (
            ITensor & input,
            ITensor & scale,
            ITensor & bias,
            uint32_t axesMask )  [inline], [noexcept]
```

Add a normalization layer to the network.

Parameters

input	The input tensor to the layer.
scale	The scale tensor used to scale the normalized output.
bias	The bias tensor used to scale the normalized output.
axesMask	The axes on which to perform mean calculations. The bit in position i of bitmask axesMask corresponds to explicit dimension i of the result. E.g., the least significant bit corresponds to the first explicit dimension and the next to least significant bit corresponds to the second explicit dimension.

The normalization layer works by performing normalization of the tensor `input` on the specified `axesMask`. The result is then scaled by multiplying with `scale` and adding `bias`.

The shape of `scale` and `bias` are expected the be the same, and must have the same rank and be unidirectionally broadcastable to the shape of `input`.

See also

INormalizationLayer

Returns

The new normalization layer, or nullptr if it could not be created.

9.95.3.28 addOneHot()

```
IOneHotLayer * nvinfer1::INetworkDefinition::addOneHot (
            ITensor & indices,
            ITensor & values,
            ITensor & depth,
            int32_t axis )  [inline], [noexcept]
```

Add a OneHot layer to the network.

Parameters

indices	- tensor containing indices where on_value should be set.
values	- a 2-element tensor, consisting of [off_value, on_value].
depth	- a shape tensor containing the width of the added one-hot dimension.
axis	- the axis to add the one-hot encoding to.

See also

IOneHotLayer

Returns

The new OneHot layer, or nullptr if it could not be created.

9.95.3.29 addPaddingNd()

```
IPaddingLayer * nvinfer1::INetworkDefinition::addPaddingNd (
            ITensor & input,
            Dims const & prePadding,
            Dims const & postPadding )  [inline], [noexcept]
```

Add a padding layer to the network. Only 2D padding is currently supported.

Parameters

input	The input tensor to the layer.
prePadding	The padding to apply to the start of the tensor.
postPadding	The padding to apply to the end of the tensor.

See also

IPaddingLayer

Returns

The new padding layer, or nullptr if it could not be created.

9.95.3.30 addParametricReLU()

```
IParametricReLULayer * nvinfer1::INetworkDefinition::addParametricReLU (
            ITensor & input,
            ITensor & slope )  [inline], [noexcept]
```

Add a parametric ReLU layer to the network.

Parameters

input	The input tensor to the layer.
slope	The slope tensor to the layer. This tensor should be unidirectionally broadcastable to the input tensor.

See also

IParametricReLULayer

Warning

Tensors of type Int32, Int64, Bool, or UInt8 are not allowed as inputs.

Returns

The new parametric ReLU layer, or nullptr if it could not be created.

9.95.3.31 addPluginV2()

```
TRT_DEPRECATED IPluginV2Layer * nvinfer1::INetworkDefinition::addPluginV2 (
            ITensor *const * inputs,
            int32_t nbInputs,
            IPluginV2 & plugin )  [inline], [noexcept]
```

Add a plugin layer to the network using the IPluginV2 interface.

Parameters

inputs	The input tensors to the layer.
nbInputs	The number of input tensors.
plugin	The layer plugin.

See also

IPluginV2Layer

Warning

Dimension wildcard are only supported with IPluginV2DynamicExt or IPluginV2IOExt plugins.

Int32 tensors are not valid input tensors.

Returns

The new plugin layer, or nullptr if it could not be created.

Deprecated Deprecated in TensorRT 10.8. Superseded by addPluginV3.

9.95.3.32 addPluginV3()

```
IPluginV3Layer * nvinfer1::INetworkDefinition::addPluginV3 (
            ITensor *const * inputs,
            int32_t nbInputs,
            ITensor *const * shapeInputs,
            int32_t nbShapeInputs,
            IPluginV3 & plugin )  [inline], [noexcept]
```

Add a plugin layer implementing the IPluginV3 interface to the network.

Parameters

inputs	The input tensors to the layer.
nbInputs	The number of input tensors.
shapeInputs	Shape tensor inputs to the layer.
nbShapeInputs	The number of shape tensor inputs.
plugin	The layer plugin.

See also

IPluginV3Layer

Returns

The new plugin layer, or nullptr if it could not be created.

9.95.3.33 addPoolingNd()

```
IPoolingLayer * nvinfer1::INetworkDefinition::addPoolingNd (
            ITensor & input,
            PoolingType type,
            Dims const & windowSize )  [inline], [noexcept]
```

Add a multi-dimension pooling layer to the network.

Parameters

input	The input tensor to the layer.
type	The type of pooling to apply.
windowSize	The size of the pooling window.

See also

IPoolingLayer PoolingType

Warning

Int32 tensors are not valid input tensors.

Only 2D or 3D pooling is supported.

Returns

The new pooling layer, or nullptr if it could not be created.

9.95.3.34 addQuantize() [1/2]

```
TRT_DEPRECATED IQuantizeLayer * nvinfer1::INetworkDefinition::addQuantize (
            ITensor & input,
            ITensor & scale )  [inline], [noexcept]
```

Add a quantization layer to the network.

Parameters

input	The input tensor to be quantized.
scale	A tensor with the scale value.

See also

IQuantizeLayer

`input` tensor data type must be DataType::kFLOAT or DataType::kHALF. `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.

Deprecated Deprecated in TensorRT 9.0. Superseded by three-argument addQuantize.

9.95.3.35 addQuantize() [2/2]

```
IQuantizeLayer * nvinfer1::INetworkDefinition::addQuantize (
            ITensor & input,
            ITensor & scale,
            DataType outputType )  [inline], [noexcept]
```

Add a quantization layer to the network.

Parameters

input	The input tensor to be quantized.
scale	A tensor with the scale value.
outputType	Output tensor data type.

See also

IQuantizeLayer

`input` tensor data type must be DataType::kFLOAT, DataType::kHALF or DataType::kBF16. `scale` tensor data type defaults to DataType::kFLOAT. For strongly typed networks, it must have the same data type as the input. The subgraph which terminates with the `scale` tensor must be a build-time constant. `outputType` output tensor data type, must be DataType::kINT8 (default), DataType::kFP8 or DataType::kINT4. Future calls to set output type using setToType or setOutputType must be consistent.

Returns

The new quantization layer, or nullptr if it could not be created.

9.95.3.36 addRaggedSoftMax()

```
IRaggedSoftMaxLayer * nvinfer1::INetworkDefinition::addRaggedSoftMax (
            ITensor & input,
            ITensor & bounds )  [inline], [noexcept]
```

Add a RaggedSoftMax layer to the network.

Parameters

input	The ZxS input tensor.
bounds	The Zx1 bounds tensor.

See also

IRaggedSoftMaxLayer

Warning

The bounds tensor cannot have the last dimension be the wildcard character.

Int32 tensors are not valid input tensors.

The input and bounds tensors should be 3D tensors.

Returns

The new RaggedSoftMax layer, or nullptr if it could not be created.

9.95.3.37 addReduce()

```
IReduceLayer * nvinfer1::INetworkDefinition::addReduce (
            ITensor & input,
            ReduceOperation operation,
            uint32_t reduceAxes,
            bool keepDimensions )  [inline], [noexcept]
```

Add a reduce layer to the network.

Parameters

input	The input tensor to the layer.
operation	The reduction operation to perform.
reduceAxes	The reduction dimensions. The bit in position i of bitmask reduceAxes corresponds to explicit dimension i if result. E.g., the least significant bit corresponds to the first explicit dimension and the next to least significant bit corresponds to the second explicit dimension.
keepDimensions	The boolean that specifies whether or not to keep the reduced dimensions in the output of the layer.

The reduce layer works by performing an operation specified by `operation` to reduce the tensor `input` across the axes specified by `reduceAxes`.

See also

IReduceLayer

Warning

If output is an Int32 or Int64 shape tensor, ReduceOperation::kAVG is unsupported.

Returns

The new reduce layer, or nullptr if it could not be created.

9.95.3.38 addResize()

```
IResizeLayer * nvinfer1::INetworkDefinition::addResize (
            ITensor & input ) [inline], [noexcept]
```

Add a resize layer to the network.

Parameters

input	The input tensor to the layer.

See also

IResizeLayer

Warning

Int32 tensors are not valid input tensors.

Returns

The new resize layer, or nullptr if it could not be created.

9.95.3.39 addReverseSequence()

```
IReverseSequenceLayer * nvinfer1::INetworkDefinition::addReverseSequence (
            ITensor & input,
            ITensor & sequenceLens ) [inline], [noexcept]
```

Add a ReverseSequence layer to the network.

Parameters

input	The input tensor to the layer. Must have rank >= 2.
sequenceLens	1D tensor specifying lengths of sequences to reverse in a batch. The length of the sequenceLens tensor must be equal to the size of the dimension in input tensor specified by batchAxis.

See also

IReverseSequenceLayer

Returns

The new ReverseSequence layer, or nullptr if it could not be created.

9.95.3.40 addScale()

```
IScaleLayer * nvinfer1::INetworkDefinition::addScale (
            ITensor & input,
            ScaleMode mode,
            Weights shift,
            Weights scale,
            Weights power )  [inline], [noexcept]
```

Add a Scale layer to the network.

Parameters

input	The input tensor to the layer. This tensor must have at least 4 dimensions.
mode	The scaling mode.
shift	The shift value.
scale	The scale value.
power	The power value.

If the weights are available, then the size of weights are dependent on the ScaleMode. For ScaleMode::kUNIFORM, the number of weights equals 1. For ScaleMode::kCHANNEL, the number of weights equals the channel dimension. For ScaleMode::kELEMENTWISE, the number of weights equals the product of the last three dimensions of the input.

See also

addScaleNd

IScaleLayer

Warning

Int32 tensors are not valid input tensors.

Returns

The new Scale layer, or nullptr if it could not be created.

9.95.3.41 addScaleNd()

```
IScaleLayer * nvinfer1::INetworkDefinition::addScaleNd (
            ITensor & input,
            ScaleMode mode,
            Weights shift,
            Weights scale,
            Weights power,
            int32_t channelAxis )  [inline], [noexcept]
```

Add a multi-dimension scale layer to the network.

Parameters

input	The input tensor to the layer.
mode	The scaling mode.
shift	The shift value.
scale	The scale value.
power	The power value.
channelAxis	The channel axis.

If the weights are available, then the size of weights are dependent on the ScaleMode. For ScaleMode::kUNIFORM, the number of weights equals 1. For ScaleMode::kCHANNEL, the number of weights equals the channel dimension. For ScaleMode::kELEMENTWISE, the number of weights equals the product of all input dimensions at channelAxis and beyond.

For example, if the inputs dimensions are [A,B,C,D,E,F], and channelAxis=2: For ScaleMode::kUNIFORM, the number of weights is equal to 1. For ScaleMode::kCHANNEL, the number of weights is C. For ScaleMode::kELEMENTWISE, the number of weights is C∗D∗E∗F.

channelAxis can also be set explicitly using setChannelAxis().

See also

IScaleLayer

setChannelAxis()

Warning

Int32 tensors are not valid input tensors.

Only 2D or 3D scale is supported.

Returns

The new Scale layer, or nullptr if it could not be created.

9.95.3.42 addScatter()

```
IScatterLayer * nvinfer1::INetworkDefinition::addScatter (
            ITensor & data,
            ITensor & indices,
            ITensor & updates,
            ScatterMode mode )  [inline], [noexcept]
```

Add a Scatter layer to the network with specified mode and axis=0.

Parameters

data	The input tensor to be updated with additional values.
indices	indices of the elements to be updated.
updates	values to be used for updates.
mode	scatter mode.

See also

IScatterLayer

`indices` tensor data type must be DataType::kINT32. `updates` tensor data type must be the same as `data`

Returns

The new Scatter layer, or nullptr if it could not be created.

9.95.3.43 addSelect()

```
ISelectLayer * nvinfer1::INetworkDefinition::addSelect (
            ITensor & condition,
            ITensor & thenInput,
            ITensor & elseInput )  [inline], [noexcept]
```

Add a select layer to the network.

Parameters

condition	The condition tensor to the layer. Must have type DataType::kBOOL.
thenInput	The ”then” input tensor to the layer.
elseInput	The ”else” input tensor to the layer.

All three input tensors must have the same rank, and along each axis must have the same length or a length of one. If the length is one, the tensor is broadcast along that axis. The output tensor has the dimensions of the inputs AFTER the broadcast rule is applied. For example, given:

dimensions of condition: [1,1,5,9] dimensions of thenInput: [1,1,5,9] dimensions of elseInput: [1,3,1,9]

the output dimensions are [1,3,5,9], and the output contents are defined by:

```
output[0,i,j,k] = condition[0,0,j,k] ? thenInput[0,0,j,k] : elseInput[0,i,0,k]
```

The output dimensions are not necessarily the max of the input dimensions if any input is an empty tensor. For example, if in the preceding example, 5 is changed to 0:

dimensions of condition: [1,1,0,9] dimensions of thenInput: [1,1,0,9] dimensions of elseInput: [1,3,1,9]

then the output dimensions are [1,3,0,9].

The inputs are shape tensors if the output is a shape tensor.

See also

ISelectLayer

Returns

The new select layer, or nullptr if it could not be created.

9.95.3.44 addShape()

```
IShapeLayer * nvinfer1::INetworkDefinition::addShape (
            ITensor & input )  [inline], [noexcept]
```

Add a shape layer to the network.

Parameters

input	The input tensor to the layer.

See also

IShapeLayer

Warning

addShape is only supported when hasImplicitBatchDimensions is false.

Returns

The new shape layer, or nullptr if it could not be created.

9.95.3.45 addShuffle()

```
IShuffleLayer * nvinfer1::INetworkDefinition::addShuffle (
            ITensor & input )  [inline], [noexcept]
```

Add a shuffle layer to the network.

Parameters

input	The input tensor to the layer.

See also

IShuffleLayer

Returns

The new shuffle layer, or nullptr if it could not be created.

9.95.3.46 addSlice()

```
ISliceLayer * nvinfer1::INetworkDefinition::addSlice (
            ITensor & input,
            Dims const & start,
            Dims const & size,
            Dims const & stride )  [inline], [noexcept]
```

Add a slice layer to the network.

Parameters

input	The input tensor to the layer.
start	The start offset
size	The output dimension
stride	The slicing stride

Positive, negative, zero stride values, and combinations of them in different dimensions are allowed.

See also

ISliceLayer

Returns

The new slice layer, or nullptr if it could not be created.

9.95.3.47 addSoftMax()

```
ISoftMaxLayer * nvinfer1::INetworkDefinition::addSoftMax (
            ITensor & input )  [inline], [noexcept]
```

Add a SoftMax layer to the network.

See also

ISoftMaxLayer

Warning

Int32 tensors are not valid input tensors.

Returns

The new SoftMax layer, or nullptr if it could not be created.

9.95.3.48 addSqueeze()

```
ISqueezeLayer * nvinfer1::INetworkDefinition::addSqueeze (
            ITensor & input,
            ITensor & axes )  [inline], [noexcept]
```

Add a squeeze layer to the network.

Parameters

input	The input tensor to the layer.
axes	The axes to remove unit dimensions on.

See also

ISqueezeLayer

Axes must be resolvable to a constant Int32 or Int64 1D shape tensor. Values in axes must be unique and in the range of [-r, r-1], where r is the rank of the input tensor. For each axis value, the corresponding dimension in the input tensor must be one.

Returns

The new Squeeze layer, or nullptr if it could not be created.

9.95.3.49 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

input	The input tensor to the layer.
op	Operation to perform.
k	The number of elements to keep. For dynamic k, use the setInput() method to pass in k as a tensor instead, which will override the static k value passed here in calculations.
reduceAxes	The reduction dimensions. The bit in position i of bitmask reduceAxes corresponds to explicit dimension i of the result. E.g., the least significant bit corresponds to the first explicit dimension and the next to least significant bit corresponds to the second explicit dimension.

Currently reduceAxes must specify exactly one dimension, and it must be one of the last four dimensions.

See also

ITopKLayer

Returns

The new TopK layer, or nullptr if it could not be created.

9.95.3.50 addUnary()

```
IUnaryLayer * nvinfer1::INetworkDefinition::addUnary (
            ITensor & input,
            UnaryOperation operation )  [inline], [noexcept]
```

Add a unary layer to the network.

Parameters

input	The input tensor to the layer.
operation	The operation to apply.

See also

IUnaryLayer

Generally the input must have a floating-point type (or kINT8 as a quantized float), except for the following operations:

- kSIGN accepts a floating-point or Int32 tensor.
- kNOT requires a Bool tensor.

The input is a shape tensor if the output is a shape tensor.

Returns

The new unary layer, or nullptr if it could not be created

9.95.3.51 addUnsqueeze()

```
IUnsqueezeLayer * nvinfer1::INetworkDefinition::addUnsqueeze (
            ITensor & input,
            ITensor & axes )  [inline], [noexcept]
```

Add an unsqueeze layer to the network.

Parameters

input	The input tensor to the layer.
axes	The axes to add unit dimensions.

See also

IUnsqueezeLauyer

Axes must be resolvable to a constant Int32 or Int64 shape tensor. Values in axes must be unique and in the range of [-r_final, r_final-1], where r_final is the sum of rank(input) and len(axes).

r_final must be less than Dims::MAX_DIMS.

Returns

The new Unsqueeze layer, or nullptr if it could not be created

9.95.3.52 areWeightsMarkedRefittable()

```
bool nvinfer1::INetworkDefinition::areWeightsMarkedRefittable (
            char const * name ) const  [inline], [noexcept]
```

Whether the weight has been marked as refittable.

Parameters

name	The name of the weights to check.

Returns

True if the weights are marked as refittable, false if the weights do not exist or are marked as non-refittable.

9.95.3.53 getBuilder()

```
virtual IBuilder & nvinfer1::INetworkDefinition::getBuilder ( ) const  [inline], [virtual], [noexcept]
```

Return the builder from which this INetworkDefinition was created.

See also

IBuilder::createNetworkV2

Returns

the builder

9.95.3.54 getErrorRecorder()

```
IErrorRecorder * nvinfer1::INetworkDefinition::getErrorRecorder ( ) const  [inline], [noexcept]
```

get the ErrorRecorder assigned to this interface.

Retrieves the assigned error recorder object for the given class. A nullptr will be returned if setErrorRecorder has not been called.

Returns

A pointer to the IErrorRecorder object that has been registered.

See also

setErrorRecorder()

9.95.3.55 getFlag()

```
bool nvinfer1::INetworkDefinition::getFlag (
            NetworkDefinitionCreationFlag networkDefinitionCreationFlag ) const  [inline], [noexcept]
```

Returns true if the network definition creation flag is set.

See also

getFlags()

Returns

True if flag is set, false if unset.

9.95.3.56 getFlags()

```
NetworkDefinitionCreationFlags nvinfer1::INetworkDefinition::getFlags ( ) const  [inline], [noexcept]
```

Get the network definition creation flags for this network definition object. Defaults to 0.

Returns

The network definition creation options as a bitmask.

9.95.3.57 getInput()

```
ITensor * nvinfer1::INetworkDefinition::getInput (
            int32_t index ) const  [inline], [noexcept]
```

Get the input tensor specified by the given index.

Parameters

index	The index of the input tensor.

Returns

The input tensor, or nullptr if the index is out of range.

Note

adding inputs invalidates indexing here

See also

getNbInputs()

9.95.3.58 getLayer()

```
ILayer * nvinfer1::INetworkDefinition::getLayer (
            int32_t index ) const  [inline], [noexcept]
```

Get the layer specified by the given index.

Parameters

index	The index of the layer.

Returns

The layer, or nullptr if the index is out of range.

See also

getNbLayers()

9.95.3.59 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.

9.95.3.60 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()

9.95.3.61 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()

9.95.3.62 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()

9.95.3.63 getOutput()

```
ITensor * nvinfer1::INetworkDefinition::getOutput (
            int32_t index ) const  [inline], [noexcept]
```

Get the output tensor specified by the given index.

Parameters

index	The index of the output tensor.

Returns

The output tensor, or nullptr if the index is out of range.

Note

adding inputs invalidates indexing here

See also

getNbOutputs()

9.95.3.64 hasImplicitBatchDimension()

```
TRT_DEPRECATED bool nvinfer1::INetworkDefinition::hasImplicitBatchDimension ( ) const  [inline],
[noexcept]
```

Query whether the network was created with an implicit batch dimension.

Returns

Always false since TensorRT 10.0 does not support an implicit batch dimension.

See also

createNetworkV2

Deprecated Deprecated in TensorRT 10.0. Implicit batch is not supported since TensorRT 10.0.

9.95.3.65 isDebugTensor()

```
bool nvinfer1::INetworkDefinition::isDebugTensor (
            nvinfer1::ITensor const & tensor ) const  [inline], [noexcept]
```

Check if a tensor is marked as debug tensor.

Returns

true if tensor is marked as debug tensor, false otherwise.

9.95.3.66 markDebug()

```
bool nvinfer1::INetworkDefinition::markDebug (
            ITensor & tensor )  [inline], [noexcept]
```

Mark a tensor as a debug tensor.

A debug tensor can be optionally emitted at runtime. Note that tensor names are required to specify debug tensors at runtime.

Parameters

tensor	Tensor to be marked as debug

Returns

True if tensor successfully marked (or was already marked), false otherwise.

See also

unmarkDebug(), IExecutionContext::setDebugListener(), ITensor::setName()

9.95.3.67 markOutput()

```
void nvinfer1::INetworkDefinition::markOutput (
             ITensor & tensor )  [inline], [noexcept]
```

Mark a tensor as a network output.

Parameters

tensor	The tensor to mark as an output tensor.

Warning

It is an error to mark a network input as an output.

It is an error to mark a tensor inside an ILoop or an IIfConditional as an output.

9.95.3.68 markOutputForShapes()

```
bool nvinfer1::INetworkDefinition::markOutputForShapes (
             ITensor & tensor )  [inline], [noexcept]
```

Enable tensor's value to be computed by IExecutionContext::getShapeBinding.

Returns

True if successful, false if tensor is already marked as an output.

The tensor must be of type DataType::kINT32 and have no more than one dimension.

Warning

The tensor must have dimensions that can be determined to be constants at build time.

It is an error to mark a network input as a shape output.

9.95.3.69 markWeightsRefittable()

```
bool nvinfer1::INetworkDefinition::markWeightsRefittable (
            char const * name )  [inline], [noexcept]
```

Mark weights as refittable when the builder flag kREFIT_INDIVIDUAL is set.

Parameters

name	The name of the weights.

Returns

True if the weights were successfully marked as refittable, false if the weights do not exist or cannot be refitted.

9.95.3.70 removeTensor()

```
void nvinfer1::INetworkDefinition::removeTensor (
            ITensor & tensor )  [inline], [noexcept]
```

remove a tensor from the network definition.

Parameters

tensor	the tensor to remove

It is illegal to remove a tensor that is the input or output of a layer. if this method is called with such a tensor, a warning will be emitted on the log and the call will be ignored. Its intended use is to remove detached tensors after e.g. concatenating two networks with Layer::setInput().

9.95.3.71 setErrorRecorder()

```
void nvinfer1::INetworkDefinition::setErrorRecorder (
            IErrorRecorder * recorder )  [inline], [noexcept]
```

Set the ErrorRecorder for this interface.

Assigns the ErrorRecorder to this interface. The ErrorRecorder will track all errors during execution. This function will call incRefCount of the registered ErrorRecorder at least once. Setting recorder to nullptr unregisters the recorder with the interface, resulting in a call to decRefCount if a recorder has been registered.

If an error recorder is not set, messages will be sent to the global log stream.

Parameters

recorder	The error recorder to register with this interface.

See also

getErrorRecorder()

9.95.3.72 setName()

```
void nvinfer1::INetworkDefinition::setName (
            char const * name )  [inline], [noexcept]
```

Sets the name of the network.

Parameters

name	The name to assign to this network.

Set the name of the network so that it can be associated with a built engine. The `name` must be a null-terminated C-style string. TensorRT makes no use of this string except storing it as part of the engine so that it may be retrieved at runtime. A name unique to the builder will be generated by default.

This method copies the name string.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

See also

INetworkDefinition::getName(), ISafeCudaEngine::getName()

Returns

none

9.95.3.73 setWeightsName()

```
bool nvinfer1::INetworkDefinition::setWeightsName (
            Weights weights,
            char const * name )  [inline], [noexcept]
```

Associate a name with all current uses of the given weights.

The name must be set after the Weights are used in the network. Lookup is associative. The name applies to all Weights with matching type, value pointer, and count. If Weights with a matching value pointer, but different type or count exists in the network, an error message is issued, the name is rejected, and return false. If the name has already been used for other weights, return false. A nullptr causes the weights to become unnamed, i.e. clears any previous name.

Parameters

weights	The weights to be named.
name	The name to associate with the weights.

Returns

true on success.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

9.95.3.74 unmarkDebug()

```
bool nvinfer1::INetworkDefinition::unmarkDebug (
            ITensor & tensor )  [inline], [noexcept]
```

Unmark a tensor as a debug tensor.

Remove the marking of a tensor as a debug tensor.

Parameters

tensor	Tensor to be unmarked as debug.

Returns

True if tensor successfully unmarked (or was already unmarked), false otherwise.

See also

markDebug(), IExecutionContext::setDebugListener()

9.95.3.75 unmarkOutput()

```
void nvinfer1::INetworkDefinition::unmarkOutput (
            ITensor & tensor )  [inline], [noexcept]
```

unmark a tensor as a network output.

Parameters

tensor	The tensor to unmark as an output tensor.

see markOutput()

9.95.3.76 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.

9.95.3.77 unmarkWeightsRefittable()

```
bool nvinfer1::INetworkDefinition::unmarkWeightsRefittable (
            char const * name )  [inline], [noexcept]
```

Unmark weights as refittable when the builder flag kREFIT_INDIVIDUAL is set.

Parameters

name	The name of the weights.

Returns

True if the weights were successfully marked as unrefittable, false if the weights do not exist.

9.95.4 Member Data Documentation

9.95.4.1 mImpl

```
apiv::VNetworkDefinition* nvinfer1::INetworkDefinition::mImpl  [protected]
```

The documentation for this class was generated from the following file:

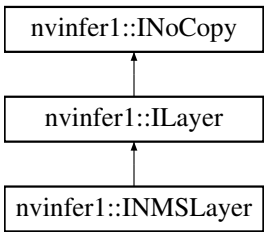
- NvInfer.h

9.96 nvinfer1::INMSLayer Class Reference

A non-maximum suppression layer in a network definition.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::INMSLayer:



Public Member Functions

- void setBoundingBoxFormat (BoundingBoxFormat fmt) noexcept

  Set the bounding box format parameter for the layer.
- BoundingBoxFormat getBoundingBoxFormat () const noexcept

  Get the bounding box format parameter for the layer.
- void setTopKBoxLimit (int32_t limit) noexcept

  Set the TopK box limit parameter for the layer.
- int32_t getTopKBoxLimit () const noexcept

  Get the TopK box limit parameter for the layer.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~INMSLayer () noexcept=default

Protected Attributes

- apiv::VNMSLayer * mImpl

9.96.1 Detailed Description

A non-maximum suppression layer in a network definition.

The NMS algorithm iterates through a set of bounding boxes and their confidence scores, in decreasing order of score. Boxes are selected if their score is above a given threshold, and their intersection-over-union (IoU) with previously selected boxes is less than or equal to a given threshold. This layer implements NMS per batch item and per class.

Per batch item, boxes are initially sorted by their scores without regard to class. Only boxes up to a maximum of the TopK limit are considered for selection (per batch). During selection, only overlapping boxes of the same class are compared, so that overlapping boxes of different classes do not suppress each other.

For each batch item, the ordering of candidate bounding boxes with the same score is unspecified, but the ordering will be consistent across different runs for the same inputs.

The layer has the following inputs, in order of input index:

- Boxes contains the input bounding boxes. It is a linear tensor of type kFLOAT or kHALF. It has shape [batchSize, numInputBoundingBoxes, numClasses, 4] if the boxes are per class, or [batchSize, numInputBoundingBoxes, 4] if the same boxes are to be used for each class.
- Scores contains the per-box scores. It is a linear tensor of the same type as Boxes. It has shape [batchSize, numInputBoundingBoxes, numClasses].
- MaxOutputBoxesPerClass is the maximum number of output boxes per batch item per class. It is a scalar (0D tensor) of type kINT32.
- IoUThreshold is the maximum IoU for selected boxes. It is a scalar (0D tensor) of type kFLOAT in the range [0.0f, 1.0f]. It is an optional input with default 0.0f.
- ScoreThreshold is the value that a box score must exceed in order to be selected. It is a scalar (0D tensor) of type kFLOAT. It is an optional input with default 0.0f.

The layer has the following outputs, in order of output index:

- SelectedIndices contains the indices of the selected boxes. It is a linear tensor of type kINT32. It has shape [NumOutputBoxes, 3]. Each row contains a (batchIndex, classIndex, boxIndex) tuple. The output boxes are sorted in order of increasing batchIndex and then in order of decreasing score within each batchIndex. For each batchIndex, the ordering of output boxes with the same score is unspecified. If MaxOutputBoxesPerClass is a constant input, the maximum number of output boxes is batchSize * numClasses * min(numInputBounding↩ Boxes, MaxOutputBoxesPerClass). Otherwise, the maximum number of output boxes is batchSize * numClasses * numInputBoundingBoxes. The maximum number of output boxes is used to determine the upper-bound on allocated memory for this output tensor.
- NumOutputBoxes is the number of output boxes in SelectedIndices. It is a scalar (0D tensor) of type kINT32.

Warning

There is a hardware-dependent limit K such that only the K highest scoring boxes in each batch item will be considered for selection. The value of K is 2000 for SM 5.3 and 6.2 devices, and 5000 otherwise.

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.96.2 Constructor & Destructor Documentation

9.96.2.1 ~INMSLayer()

```
virtual nvinfer1::INMSLayer::~INMSLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.96.3 Member Function Documentation

9.96.3.1 getBoundingBoxFormat()

```
BoundingBoxFormat nvinfer1::INMSLayer::getBoundingBoxFormat ( ) const  [inline], [noexcept]
```

Get the bounding box format parameter for the layer.

See also

BoundingBoxFormat

setBoundingBoxFormat()

9.96.3.2 getTopKBoxLimit()

```
int32_t nvinfer1::INMSLayer::getTopKBoxLimit ( ) const  [inline], [noexcept]
```

Get the TopK box limit parameter for the layer.

See also

setTopKBoxLimit()

9.96.3.3 setBoundingBoxFormat()

```
void nvinfer1::INMSLayer::setBoundingBoxFormat (
            BoundingBoxFormat fmt )  [inline], [noexcept]
```

Set the bounding box format parameter for the layer.

The default value for the bounding box format parameter is kCORNER_PAIRS.

See also

BoundingBoxFormat

getBoundingBoxFormat()

9.96.3.4 setInput()

```
void nvinfer1::ILayer::setInput (
            int32_t index,
            ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor

The indices are as follows:

- 0: The required Boxes tensor.
- 1: The required Scores tensor.
- 2: The required MaxOutputBoxesPerClass tensor.
- 3: The optional IoUThreshold tensor.
- 4: The optional ScoreThreshold tensor.

If this function is called for an index greater or equal to getNbInputs(), then afterwards getNbInputs() returns index + 1, and any missing intervening inputs are set to null. Note that only optional inputs can be missing.

9.96.3.5 setTopKBoxLimit()

```
void nvinfer1::INMSLayer::setTopKBoxLimit (
            int32_t limit )  [inline], [noexcept]
```

Set the TopK box limit parameter for the layer.

The TopK box limit is the maximum number of filtered boxes considered for selection per batch item. The default value for the TopK box limit parameter is 2000 for SM 5.3 and 6.2 devices, and 5000 otherwise. The TopK box limit must be less than or equal to {2000 for SM 5.3 and 6.2 devices, 5000 otherwise}.

See also

getTopKBoxLimit()

9.96.4 Member Data Documentation

9.96.4.1 mImpl

```
apiv::VNMSLayer* nvinfer1::INMSLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

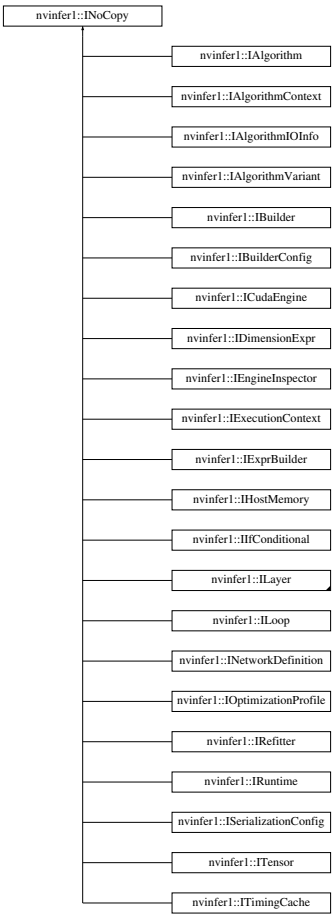
- NvInfer.h

9.97 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

9.97.1 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.

9.97.2 Constructor & Destructor Documentation

9.97.2.1 INoCopy() [1/3]

```
nvinfer1::INoCopy::INoCopy ( )  [protected], [default]
```

9.97.2.2 ~INoCopy()

```
virtual nvinfer1::INoCopy::~INoCopy ( )  [protected], [virtual], [default]
```

9.97.2.3 INoCopy() [2/3]

```
nvinfer1::INoCopy::INoCopy (
            INoCopy const & other )  [protected], [delete]
```

9.97.2.4 INoCopy() [3/3]

```
nvinfer1::INoCopy::INoCopy (
            INoCopy && other )  [protected], [delete]
```

9.97.3 Member Function Documentation

9.97.3.1 operator=() [1/2]

```
INoCopy & nvinfer1::INoCopy::operator= (
            INoCopy && other )  [protected], [delete]
```

9.97.3.2 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

9.98 INonZero Class Reference

A NonZero layer in a network.

`#include <NvInfer.h>`

9.98.1 Detailed Description

A NonZero layer in a network.

This layer gets the positions of elements that are non-zero in the input. For boolean input, ”non-zero” means ”true”. Semantics are similar to ONNX NonZero.

The input may have type kFLOAT, kHALF, kINT32, or kBOOL.

The output is a matrix of type kINT32. For an input with dimensions [L1, L2, ..., Lm], the output has dimensions [m,n], where n is the number of non-zero elements. I.e., each column denotes a m-D position.

The columns are lexically ordered. E.g., a column with [3,2,4,7] precedes a column with [3,2,5,6].

Tip: ”compress” can be implemented with INonZero+IShuffle+Gather. For example, to compress a tensor x over axis k using mask vector v, use nonzero(v) to compute the subscripts, shuffle with reshape dimensions = [-1] to make the subscripts 1D, and then gather with the subscripts.

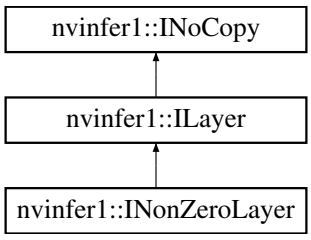
The documentation for this class was generated from the following file:

- NvInfer.h

9.99 nvinfer1::INonZeroLayer Class Reference

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::INonZeroLayer:



Protected Member Functions

- virtual ~INonZeroLayer () noexcept=default

Protected Attributes

- apiv::VNonZeroLayer * mImpl

Additional Inherited Members

9.99.1 Constructor & Destructor Documentation

9.99.1.1 ~INonZeroLayer()

```
virtual nvinfer1::INonZeroLayer::~INonZeroLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.99.2 Member Data Documentation

9.99.2.1 mImpl

```
apiv::VNonZeroLayer* nvinfer1::INonZeroLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

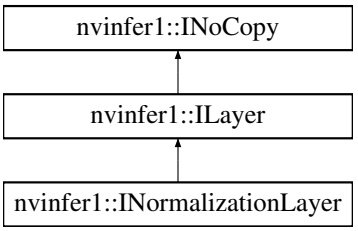
- NvInfer.h

9.100 nvinfer1::INormalizationLayer Class Reference

A normalization layer in a network definition.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::INormalizationLayer:



Public Member Functions

- void setEpsilon (float eps) noexcept

  Set the epsilon value used for the normalization calculation.
- float getEpsilon () const noexcept

  Get the epsilon value used for the normalization calculation.
- void setAxes (uint32_t axesMask) noexcept

  Set the reduction axes for the normalization calculation.
- uint32_t getAxes () const noexcept

  Get the axes value used for the normalization calculation.
- void setNbGroups (int64_t nbGroups) noexcept

  Set the number of groups used to split the channels in the normalization calculation.
- int64_t getNbGroups () const noexcept

  Get the number of groups used to split the channels for the normalization calculation.
- void setComputePrecision (DataType type) noexcept

  Set the compute precision of this layer.
- DataType getComputePrecision () const noexcept

  Get the compute precision of this layer.

Protected Member Functions

- virtual ~INormalizationLayer () noexcept=default

Protected Attributes

- apiv::VNormalizationLayer * mImpl

9.100.1 Detailed Description

A normalization layer in a network definition.

The normalization layer performs the following operation:

X - input Tensor Y - output Tensor S - scale Tensor B - bias Tensor

Y = (X - Mean(X, axes)) / Sqrt(Variance(X) + epsilon) * S + B

Where Mean(X, axes) is a reduction over a set of axes, and Variance(X) = Mean((X - Mean(X, axes)) ^ 2, axes).

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.100.2 Constructor & Destructor Documentation

9.100.2.1 ~INormalizationLayer()

```
virtual nvinfer1::INormalizationLayer::~INormalizationLayer ( )  [protected], [virtual], [default],
[noexcept]
```

9.100.3 Member Function Documentation

9.100.3.1 getAxes()

```
uint32_t nvinfer1::INormalizationLayer::getAxes ( ) const  [inline], [noexcept]
```

Get the axes value used for the normalization calculation.

Returns

The axes used for the normalization calculation.

9.100.3.2 getComputePrecision()

```
DataType nvinfer1::INormalizationLayer::getComputePrecision ( ) const  [inline], [noexcept]
```

Get the compute precision of this layer.

Returns

The datatype used for the compute precision of this layer.

9.100.3.3 getEpsilon()

```
float nvinfer1::INormalizationLayer::getEpsilon ( ) const  [inline], [noexcept]
```

Get the epsilon value used for the normalization calculation.

Returns

The epsilon value used for the normalization calculation.

9.100.3.4 getNbGroups()

```
int64_t nvinfer1::INormalizationLayer::getNbGroups ( ) const  [inline], [noexcept]
```

Get the number of groups used to split the channels for the normalization calculation.

Returns

The number of groups used to split the channel used for the normalization calculation.

9.100.3.5 setAxes()

```
void nvinfer1::INormalizationLayer::setAxes (
             uint32_t axesMask )  [inline], [noexcept]
```

Set the reduction axes for the normalization calculation.

Parameters

axesMask	The axes used for the normalization calculation.

9.100.3.6 setComputePrecision()

```
void nvinfer1::INormalizationLayer::setComputePrecision (
             DataType type )  [inline], [noexcept]
```

Set the compute precision of this layer.

Parameters

type	The datatype used for the compute precision of this layer.

The method is used to avoid overflow errors by controlling the normalization computation in mixed precision mode. The compute precision defaults to DataType::kFLOAT32. To override this default, use this method to set the desired compute precision.

For a weakly typed network:

- Method setOutputType() can still be called to control the output data type.
- Method setPrecision() can still be called. The input data is cast to that precision before being cast to the compute precision.

Strongly typed network rejects calls to this method since the compute precision is typically controlled by casting the input tensors to the desired type.

Only DataType::kFLOAT32 and DataType::kHALF are valid types for `type`.

9.100.3.7 setEpsilon()

```
void nvinfer1::INormalizationLayer::setEpsilon (
             float eps )  [inline], [noexcept]
```

Set the epsilon value used for the normalization calculation.

The default value of `eps` is 1e-5F.

Parameters

eps	The epsilon value used for the normalization calculation.

9.100.3.8 setNbGroups()

```
void nvinfer1::INormalizationLayer::setNbGroups (
             int64_t nbGroups )  [inline], [noexcept]
```

Set the number of groups used to split the channels in the normalization calculation.

The input tensor channels are divided into `nbGroups` groups, and normalization is performed per group. The channel dimension is considered to be the second dimension in a [N, C, H, W, ...] formatted tensor.

The default `nbGroups` is 1.

Warning

It is an error to set `nbGroups` to a value that does not evenly divide into the number of channels of the input tensor.

When `nbGroups` is != 1, it is expected that the provided axesMask will have all bits corresponding to dimensions after the channel dimension set to 1, with all other bits set to 0.

Parameters

nbGroups	The number of groups to split the channels into for the normalization calculation.

9.100.4 Member Data Documentation

9.100.4.1 mImpl

```
apiv::VNormalizationLayer* nvinfer1::INormalizationLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.101 nvinfer1::InterfaceInfo Class Reference

Version information associated with a TRT interface.

```
#include <NvInferRuntimeBase.h>
```

Public Attributes

- InterfaceKind kind
- int32_t major
- int32_t minor

9.101.1 Detailed Description

Version information associated with a TRT interface.

9.101.2 Member Data Documentation

9.101.2.1 kind

```
InterfaceKind nvinfer1::InterfaceInfo::kind
```

9.101.2.2 major

```
int32_t nvinfer1::InterfaceInfo::major
```

9.101.2.3 minor

```
int32_t nvinfer1::InterfaceInfo::minor
```

The documentation for this class was generated from the following file:

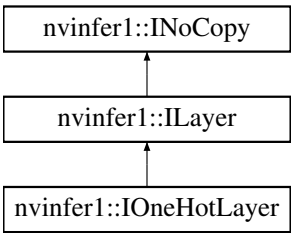
- NvInferRuntimeBase.h

9.102 nvinfer1::IOneHotLayer Class Reference

A OneHot layer in a network definition.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IOneHotLayer:



Public Member Functions

- void setAxis (int32_t axis) noexcept

  Set the axis parameter.
- int32_t getAxis () const noexcept

  Get the value of the axis parameter.

Protected Member Functions

- virtual ~IOneHotLayer () noexcept=default

Protected Attributes

- apiv::VOneHotLayer * mImpl

9.102.1 Detailed Description

A OneHot layer in a network definition.

The OneHot layer has three input tensors: Indices, Values, and Depth, one output tensor: Output, and an axis attribute.

- Indices is an Int32 tensor that determines which locations in Output to set as on_value.
- Values is a two-element (rank=1) tensor that consists of [off_value, on_value]
- Depth is an 0D tensor of type Int32 or Int64, which contains the depth (number of classes) of the one-hot encoding. The depth tensor must be a positive build-time constant.
- Output is a tensor with rank = rank(indices)+1, where the added dimension contains the one-hot encoding. The data types of Output is equal to the Values data type.
- Axis is a scalar specifying to which dimension of the output one-hot encoding is added. Valid range for axis is -rank(indices)-1 <= axis <= rank(indices).

The output is computed by copying off_values to all output elements, then setting on_value on the indices specified by the indices tensor. when axis = 0: output[indices[i, j, k], i, j, k] = on_value for all i, j, k and off_value otherwise.

when axis = -1: output[i, j, k, indices[i, j, k]] = on_value for all i, j, k and off_value otherwise.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.102.2 Constructor & Destructor Documentation

9.102.2.1 ~IOneHotLayer()

```
virtual nvinfer1::IOneHotLayer::~IOneHotLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.102.3 Member Function Documentation

9.102.3.1 getAxis()

```
int32_t nvinfer1::IOneHotLayer::getAxis ( ) const  [inline], [noexcept]
```

Get the value of the axis parameter.

9.102.3.2 setAxis()

```
void nvinfer1::IOneHotLayer::setAxis (
            int32_t axis )  [inline], [noexcept]
```

Set the axis parameter.

See also

IOneHotLayer

9.102.4 Member Data Documentation

9.102.4.1 mImpl

```
apiv::VOneHotLayer* nvinfer1::IOneHotLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.103 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.

9.103.1 Detailed Description

Configuration Manager Class.

9.103.2 Member Typedef Documentation

9.103.2.1 Verbosity

`nvonnxparser::IOnnxConfig::Verbosity`

Defines Verbosity level.

9.103.3 Constructor & Destructor Documentation

9.103.3.1 ~IOnnxConfig()

```
virtual nvonnxparser::IOnnxConfig::~IOnnxConfig ( )  [virtual], [default], [noexcept]
```

9.103.4 Member Function Documentation

9.103.4.1 addVerbosity()

```
virtual void nvonnxparser::IOnnxConfig::addVerbosity ( )  [pure virtual], [noexcept]
```

Increase the Verbosity Level.

Returns

The Verbosity Level.

See also

reduceVerbosity(), setVerbosity(Verbosity)

9.103.4.2 getFullTextFileName()

```
virtual char const * nvonnxparser::IOnnxConfig::getFullTextFileName ( ) const  [pure virtual],
[noexcept]
```

Get the File Name of the Network Description as a Text File, including the weights.

Returns

Return the name of the file containing the network description converted to a plain text, used for debugging purposes.

See also

setFullTextFilename()

9.103.4.3 getModelDtype()

```
virtual nvinfer1::DataType nvonnxparser::IOnnxConfig::getModelDtype ( ) const  [pure virtual],
[noexcept]
```

Get the Model Data Type.

Returns

the data type of the model.

See also

setModelDtype() and DataType

9.103.4.4 getModelFileName()

```
virtual char const * nvonnxparser::IOnnxConfig::getModelFileName ( ) const  [pure virtual], [noexcept]
```

Get the Model FileName.

Returns

Return the Model Filename, as a null-terminated C-style string.

See also

setModelFileName()

9.103.4.5 getPrintLayerInfo()

```
virtual bool nvonnxparser::IOnnxConfig::getPrintLayerInfo ( ) const  [pure virtual], [noexcept]
```

Get whether the layer information will be printed.

Returns

Returns whether the layer information will be printed.

See also

setPrintLayerInfo()

9.103.4.6 getTextFileName()

```
virtual char const * nvonnxparser::IOnnxConfig::getTextFileName ( ) const  [pure virtual], [noexcept]
```

Returns the File Name of the Network Description as a Text File.

Returns

Return the name of the file containing the network description converted to a plain text, used for debugging purposes.

See also

setTextFilename()

9.103.4.7 getVerbosityLevel()

```
virtual Verbosity nvonnxparser::IOnnxConfig::getVerbosityLevel ( ) const  [pure virtual], [noexcept]
```

Get the Verbosity Level.

Returns

The Verbosity Level.

See also

addVerbosity(), reduceVerbosity()

9.103.4.8 reduceVerbosity()

```
virtual void nvonnxparser::IOnnxConfig::reduceVerbosity ( )  [pure virtual], [noexcept]
```

Reduce the Verbosity Level.

See also

addVerbosity(), setVerbosity(Verbosity)

9.103.4.9 setFullTextFileName()

```
virtual void nvonnxparser::IOnnxConfig::setFullTextFileName (
            char const * fullTextFileName )  [pure virtual], [noexcept]
```

Set the File Name of the Network Description as a Text File, including the weights.

This API allows setting a file name for the network description in plain text, equivalent of the ONNX protobuf.

This method copies the name string.

Parameters

fullTextFileName	Name of the file.

See also

getFullTextFilename()

9.103.4.10 setModelDtype()

```
virtual void nvonnxparser::IOnnxConfig::setModelDtype (
            const nvinfer1::DataType  )  [pure virtual], [noexcept]
```

Set the Model Data Type.

Sets the Model DataType, one of the following: float -d 32 (default), half precision -d 16, and int8 -d 8 data types.

See also

getModelDtype()

9.103.4.11 setModelFileName()

```
virtual void nvonnxparser::IOnnxConfig::setModelFileName (
            char const * onnxFilename )  [pure virtual], [noexcept]
```

Set the Model File Name.

The Model File name contains the Network Description in ONNX pb format.

This method copies the name string.

Parameters

onnxFilename	The name.

See also

getModelFileName()

9.103.4.12 setPrintLayerInfo()

```
virtual void nvonnxparser::IOnnxConfig::setPrintLayerInfo (
            bool  )  [pure virtual], [noexcept]
```

Set whether the layer information will be printed.

See also

getPrintLayerInfo()

9.103.4.13 setTextFileName()

```
virtual void nvonnxparser::IOnnxConfig::setTextFileName (
            char const * textFileName )  [pure virtual], [noexcept]
```

Set the File Name of the Network Description as a Text File.

This API allows setting a file name for the network description in plain text, equivalent of the ONNX protobuf.

This method copies the name string.

Parameters

textFileName	Name of the file.

See also

getTextFilename()

9.103.4.14 setVerbosityLevel()

```
virtual void nvonnxparser::IOnnxConfig::setVerbosityLevel (
            Verbosity  )  [pure virtual], [noexcept]
```

Set to specific verbosity Level.

See also

addVerbosity(), reduceVerbosity()

The documentation for this class was generated from the following file:

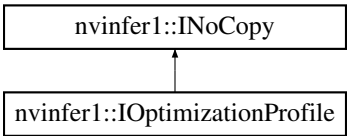
- NvOnnxConfig.h

9.104 nvinfer1::IOptimizationProfile Class Reference

Optimization profile for dynamic input dimensions and shape tensors.

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::IOptimizationProfile:



Public Member Functions

- bool setDimensions (char const *inputName, OptProfileSelector select, Dims const &dims) noexcept

  Set the minimum / optimum / maximum dimensions for a dynamic input tensor.
- Dims getDimensions (char const *inputName, OptProfileSelector select) const noexcept

  Get the minimum / optimum / maximum dimensions for a dynamic input tensor.
- 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

9.104.1 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()

9.104.2 Constructor & Destructor Documentation

9.104.2.1 ~IOptimizationProfile()

```
virtual nvinfer1::IOptimizationProfile::~IOptimizationProfile ( )  [protected], [virtual], [default],
[noexcept]
```

9.104.3 Member Function Documentation

9.104.3.1 getDimensions()

```
Dims nvinfer1::IOptimizationProfile::getDimensions (
            char const * inputName,
            OptProfileSelector select ) const  [inline], [noexcept]
```

Get the minimum / optimum / maximum dimensions for a dynamic input tensor.

If the dimensions have not been previously set via setDimensions(), return an invalid Dims with nbDims == -1.

Warning

The string inputName must be null-terminated, and be at most 4096 bytes including the terminator.

9.104.3.2 getExtraMemoryTarget()

```
float nvinfer1::IOptimizationProfile::getExtraMemoryTarget ( ) const  [inline], [noexcept]
```

Get the extra memory target that has been defined for this profile.

This defaults to 1.0F.

Returns

the valid value set by setExtraMemoryTarget or 1.0F.

9.104.3.3 getNbShapeValues()

```
int32_t nvinfer1::IOptimizationProfile::getNbShapeValues (
            char const * inputName ) const  [inline], [noexcept]
```

Get the number of values for an input shape tensor.

This will return the number of shape values if setShapeValues() has been called before for this input tensor. Otherwise, return -1.

Warning

The string inputName must be null-terminated, and be at most 4096 bytes including the terminator.

9.104.3.4 getShapeValues()

```
int32_t const * nvinfer1::IOptimizationProfile::getShapeValues (
            char const * inputName,
            OptProfileSelector select ) const  [inline], [noexcept]
```

Get the minimum / optimum / maximum values for an input shape tensor.

If the shape values have not been set previously with setShapeValues(), this returns nullptr.

Warning

The string inputName must be null-terminated, and be at most 4096 bytes including the terminator.

9.104.3.5 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.

9.104.3.6 setDimensions()

```
bool nvinfer1::IOptimizationProfile::setDimensions (
            char const * inputName,
            OptProfileSelector select,
            Dims const & dims )  [inline], [noexcept]
```

Set the minimum / optimum / maximum dimensions for a dynamic input tensor.

This function must be called three times (for the minimum, optimum, and maximum) for any network input tensor that has dynamic dimensions. If minDims, optDims, and maxDims are the minimum, optimum, and maximum dimensions, and networkDims are the dimensions for this input tensor that are provided to the INetworkDefinition object, then the following conditions must all hold:

(1) minDims.nbDims == optDims.nbDims == maxDims.nbDims == networkDims.nbDims (2) 0 <= minDims.d[i] <= optDims.d[i] <= maxDims.d[i] for i = 0, ..., networkDims.nbDims-1 (3) if networkDims.d[i] != -1, then minDims.d[i] == optDims.d[i] == maxDims.d[i] == networkDims.d[i]

This function may (but need not be) called for an input tensor that does not have dynamic dimensions. In this case, the third argument must always equal networkDims.

Parameters

inputName	The input tensor name
select	Whether to set the minimum, optimum, or maximum dimensions
dims	The minimum, optimum, or maximum dimensions for this input tensor

Returns

false if an inconsistency was detected (e.g. the rank does not match another dimension that was previously set for the same input), true if no inconsistency was detected. Note that inputs can be validated only partially; a full validation is performed at engine build time.

Warning

If run on DLA, minimum, optimum, and maximum dimensions must to be the same.

The string inputName must be null-terminated, and be at most 4096 bytes including the terminator.

9.104.3.7 setExtraMemoryTarget()

```
bool nvinfer1::IOptimizationProfile::setExtraMemoryTarget (
            float target )  [inline], [noexcept]
```

Set a target for extra GPU memory that may be used by this profile.

Parameters

target	Additional memory that the builder should aim to maximally allocate for this profile, as a fraction of the memory it would use if the user did not impose any constraints on memory. This unconstrained case is the default; it corresponds to target == 1.0. If target == 0.0, the builder aims to create the new optimization profile without allocating any additional weight memory. Valid inputs lie between 0.0 and 1.0. This parameter is only a hint, and TensorRT does not guarantee that the target will be reached. This parameter is ignored for the first (default) optimization profile that is defined.

Returns

true if the input is in the valid range (between 0 and 1 inclusive), else false.

9.104.3.8 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 dimensions of t are fixed at network definition time and the volume does not exceed 64. This function must not be called for any input tensor that is not a shape tensor.

Each time this function is called for the same input tensor, the same nbValues must be supplied (either 1 if the tensor rank is 0, or dims.d[0] if the rank is 1). Furthermore, if minVals, optVals, maxVals are the minimum, optimum, and maximum values, it must be true that minVals[i] <= optVals[i] <= maxVals[i] for i = 0, ..., nbValues - 1. Execution of the network must be valid for the optVals.

Shape tensors are tensors that contribute to shape calculations in some way. While input shape tensors can be type kINT32 or kINT64, the values used to set the minimum, optimium, and maximum values must fit in int32_t.

Examples:

- A shape tensor used as the second input to IShuffleLayer can contain a -1 wildcard. The corresponding minVal[i] should be -1.
- A shape tensor used as the stride input to ISliceLayer can contain any valid strides. The values could be positive, negative, or zero.
- A shape tensor subtracted from zero to compute the size input of an ISliceLayer can contain any non-positive values that yield a valid slice operation.

Tightening the minVals and maxVals bounds to cover only values that are necessary may help optimization.

Parameters

inputName	The input tensor name
select	Whether to set the minimum, optimum, or maximum input values.
values	An array of length nbValues containing the minimum, optimum, or maximum shape tensor elements. For multidimensional tensors, the array is in row-major order.
nbValues	The length of the value array, which must equal the number of shape tensor elements (>= 1)

Returns

false if an inconsistency was detected (e.g. nbValues does not match a previous call for the same tensor), else true. As for setDimensions(), a full validation can only be performed at engine build time.

Warning

If run on DLA, minimum, optimum, and maximum shape values must to be the same.

The string inputName must be null-terminated, and be at most 4096 bytes including the terminator.

9.104.4 Member Data Documentation

9.104.4.1 mImpl

```
apiv::VOptimizationProfile* nvinfer1::IOptimizationProfile::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.105 IOutputAllocator Class Reference

Callback from ExecutionContext::enqueueV3()

`#include <NvInferRuntime.h>`

9.105.1 Detailed Description

Callback from ExecutionContext::enqueueV3()

See also

IExecutionContext::enqueueV3()

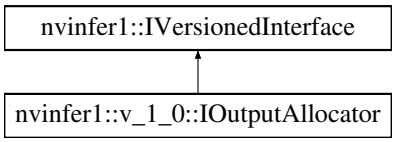
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.106 nvinfer1::v_1_0::IOutputAllocator Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IOutputAllocator:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual TRT_DEPRECATED void * reallocateOutput (char const *tensorName, void *currentMemory, uint64_t size, uint64_t alignment) noexcept

  Return a pointer to memory for an output tensor, or nullptr if memory cannot be allocated. If the requested memory size exceeds the currentMemory size, the currentMemory can be freed as well. If currentMemory is known to be big enough, one option is to return currentMemory.
- virtual void * reallocateOutputAsync (char const *tensorName, void *currentMemory, uint64_t size, uint64_t alignment, cudaStream_t)

  Return a pointer to memory for an output tensor, or nullptr if memory cannot be allocated. If the requested memory size exceeds the currentMemory size, the currentMemory can be freed as well. If currentMemory is known to be big enough, one option is to return currentMemory.
- virtual void notifyShape (char const *tensorName, Dims const &dims) noexcept=0

  Called by TensorRT when the shape of the output tensor is known.

Additional Inherited Members

9.106.1 Member Function Documentation

9.106.1.1 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IOutputAllocator::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.106.1.2 notifyShape()

```
virtual void nvinfer1::v_1_0::IOutputAllocator::notifyShape (
            char const * tensorName,
            Dims const & dims )  [pure virtual], [noexcept]
```

Called by TensorRT when the shape of the output tensor is known.

Called by TensorRT sometime between when it calls reallocateOutput and enqueueV3 returns.

Parameters

dims	dimensions of the output
tensorName	name of the tensor

9.106.1.3 reallocateOutput()

```
virtual TRT_DEPRECATED void * nvinfer1::v_1_0::IOutputAllocator::reallocateOutput (
            char const * tensorName,
            void * currentMemory,
            uint64_t size,
            uint64_t alignment )  [inline], [virtual], [noexcept]
```

Return a pointer to memory for an output tensor, or nullptr if memory cannot be allocated. If the requested memory size exceeds the currentMemory size, the currentMemory can be freed as well. If currentMemory is known to be big enough, one option is to return currentMemory.

Parameters

tensorName	name of the output tensor.
currentMemory	points to the address set by IExectionContext::setTensorAddress.
size	number of bytes required. Always positive, even for an empty tensor.
alignment	required alignment of the allocation.

Returns

A pointer to memory to use for the output tensor or nullptr.

To preallocate memory and have the engine fail if the preallocation is not big enough, use IExecutionContext::setTensorAddress to set a pointer to the preallocated memory, and have reallocateOutput return nullptr if that memory is not big enough.

Deprecated Deprecated in TensorRT 10.0. Superseded by reallocateOutputAsync with cudaStream_t argument

9.106.1.4 reallocateOutputAsync()

```
virtual void * nvinfer1::v_1_0::IOutputAllocator::reallocateOutputAsync (
             char const * tensorName,
             void * currentMemory,
             uint64_t size,
             uint64_t alignment,
             cudaStream_t  )  [inline], [virtual]
```

Return a pointer to memory for an output tensor, or nullptr if memory cannot be allocated. If the requested memory size exceeds the currentMemory size, the currentMemory can be freed as well. If currentMemory is known to be big enough, one option is to return currentMemory.

Parameters

tensorName	name of the output tensor.
currentMemory	points to the address set by IExectionContext::setTensorAddress.
size	number of bytes required. Always positive, even for an empty tensor.
alignment	required alignment of the allocation.
stream	The stream in which to execute the kernels.

Returns

A pointer to memory to use for the output tensor or nullptr.

To preallocate memory and have the engine fail if the preallocation is not big enough, use IExecutionContext::setTensorAddress to set a pointer to the preallocated memory, and have reallocateOutputAsync return nullptr if that memory is not big enough.

The default definition exists for sake of backward compatibility with earlier versions of TensorRT. Eventually this method will become a pure virtual method that requires an override, and method reallocateOutput() will disappear. Code moving away from TensorRT 9.x should override method reallocateOutputAsync() and NOT override method reallocateOutput().

The documentation for this class was generated from the following file:

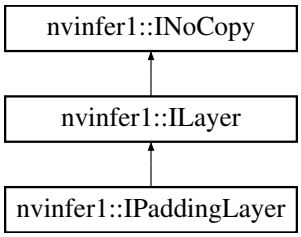
- NvInferRuntime.h

9.107 nvinfer1::IPaddingLayer Class Reference

Layer that represents a padding operation.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IPaddingLayer:



Public Member Functions

- void setPrePaddingNd (Dims const &padding) noexcept

  Set the padding that is applied at the start of the tensor.
- Dims getPrePaddingNd () const noexcept

  Get the padding that is applied at the start of the tensor.
- void setPostPaddingNd (Dims const &padding) noexcept

  Set the padding that is applied at the end of the tensor.
- Dims getPostPaddingNd () const noexcept

  Get the padding that is applied at the end of the tensor.

Protected Member Functions

- virtual ~IPaddingLayer () noexcept=default

Protected Attributes

- apiv::VPaddingLayer * mImpl

9.107.1 Detailed Description

Layer that represents a padding operation.

The padding layer adds zero-padding at the start and end of the input tensor. It supports padding only the last two dimensions. Applying negative padding results in cropping of the input.

To pad across any subset of dimensions, use ISliceLayer with SampleMode::kFILL.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.107.2 Constructor & Destructor Documentation

9.107.2.1 ~IPaddingLayer()

```
virtual nvinfer1::IPaddingLayer::~IPaddingLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.107.3 Member Function Documentation

9.107.3.1 getPostPaddingNd()

```
Dims nvinfer1::IPaddingLayer::getPostPaddingNd ( ) const  [inline], [noexcept]
```

Get the padding that is applied at the end of the tensor.

> Warning
>
> Only 2 dimensional padding is currently supported.

See also

setPostPaddingNd

9.107.3.2 getPrePaddingNd()

```
Dims nvinfer1::IPaddingLayer::getPrePaddingNd ( ) const  [inline], [noexcept]
```

Get the padding that is applied at the start of the tensor.

Warning

Only 2 dimensional padding is currently supported.

See also

setPrePaddingNd

9.107.3.3 setPostPaddingNd()

```
void nvinfer1::IPaddingLayer::setPostPaddingNd (
            Dims const & padding )  [inline], [noexcept]
```

Set the padding that is applied at the end of the tensor.

Negative padding results in trimming the edge by the specified amount

Warning

Only 2 dimensional padding is currently supported.

See also

getPostPaddingNd

9.107.3.4 setPrePaddingNd()

```
void nvinfer1::IPaddingLayer::setPrePaddingNd (
            Dims const & padding )  [inline], [noexcept]
```

Set the padding that is applied at the start of the tensor.

Negative padding results in trimming the edge by the specified amount.

Warning

Only 2 dimensional padding is currently supported.

See also

getPrePaddingNd

9.107.4 Member Data Documentation

9.107.4.1 mImpl

```
apiv::VPaddingLayer* nvinfer1::IPaddingLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

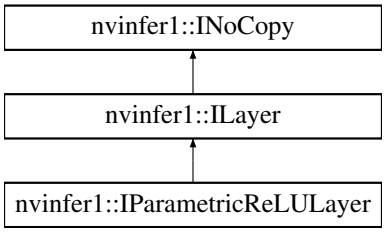
- NvInfer.h

9.108 nvinfer1::IParametricReLULayer Class Reference

Layer that represents a parametric ReLU operation.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IParametricReLULayer:



Protected Member Functions

- virtual ~IParametricReLULayer () noexcept=default

Protected Attributes

- apiv::VParametricReLULayer * mImpl

Additional Inherited Members

9.108.1 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.

9.108.2 Constructor & Destructor Documentation

9.108.2.1 ~IParametricReLULayer()

```
virtual nvinfer1::IParametricReLULayer::~IParametricReLULayer ( )  [protected], [virtual], [default],
[noexcept]
```

9.108.3 Member Data Documentation

9.108.3.1 mImpl

```
apiv::VParametricReLULayer* nvinfer1::IParametricReLULayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.109 nvonnxparser::IParser Class Reference

an object for parsing ONNX models into a TensorRT network definition

```
#include <NvOnnxParser.h>
```

Public Member Functions

- virtual bool parse (void const *serialized_onnx_model, size_t serialized_onnx_model_size, const char *model_path=nullptr) noexcept=0

  Parse a serialized ONNX model into the TensorRT network. This method has very limited diagnostics. If parsing the serialized model fails for any reason (e.g. unsupported IR version, unsupported opset, etc.) it the user responsibility to intercept and report the error. To obtain a better diagnostic, use the parseFromFile method below.
- virtual bool parseFromFile (const char *onnxModelFile, int verbosity) noexcept=0

  Parse an onnx model file, which can be a binary protobuf or a text onnx model calls parse method inside.
- virtual TRT_DEPRECATED bool supportsModel (void const *serialized_onnx_model, size_t serialized_onnx_model_size, SubGraphCollection_t &sub_graph_collection, const char *model_path=nullptr) noexcept=0

  Check whether TensorRT supports a particular ONNX model. If the function returns True, one can proceed to engine building without having to call `parse` or `parseFromFile`.
- virtual bool parseWithWeightDescriptors (void const *serialized_onnx_model, size_t serialized_onnx_model_size) noexcept=0

  Parse a serialized ONNX model into the TensorRT network with consideration of user provided weights.
- virtual bool supportsOperator (const char *op_name) const noexcept=0

  Returns whether the specified operator may be supported by the parser.
- virtual int getNbErrors () const noexcept=0

  Get the number of errors that occurred during prior calls to `parse`.
- virtual IParserError const * getError (int index) const noexcept=0

  Get an error that occurred during prior calls to `parse`.
- virtual void clearErrors () noexcept=0

  Clear errors from prior calls to `parse`.
- virtual ~IParser () noexcept=default
- virtual char const *const * getUsedVCPluginLibraries (int64_t &nbPluginLibs) const noexcept=0

  Query the plugin libraries needed to implement operations used by the parser in a version-compatible engine.
- virtual void setFlags (OnnxParserFlags onnxParserFlags) noexcept=0

  Set the parser flags.
- virtual OnnxParserFlags getFlags () const noexcept=0

  Get the parser flags. Defaults to 0.
- virtual void clearFlag (OnnxParserFlag onnxParserFlag) noexcept=0

  clear a parser flag.
- virtual void setFlag (OnnxParserFlag onnxParserFlag) noexcept=0

  Set a single parser flag.
- virtual bool getFlag (OnnxParserFlag onnxParserFlag) const noexcept=0

  Returns true if the parser flag is set.
- virtual nvinfer1::ITensor const * getLayerOutputTensor (char const *name, int64_t i) noexcept=0

  Return the i-th output ITensor object for the ONNX layer "name".
- virtual bool supportsModelV2 (void const *serializedOnnxModel, size_t serializedOnnxModelSize, char const *modelPath=nullptr) noexcept=0

  Check whether TensorRT supports a particular ONNX model. If the function returns True, one can proceed to engine building without having to call `parse` or `parseFromFile`. Results can be queried through `getNbSubgraphs`, `isSubgraphSupported`, `getSubgraphNodes`.
- virtual int64_t getNbSubgraphs () noexcept=0

  Get the number of subgraphs. Calling this function before calling `supportsModelV2` results in undefined behavior.
- virtual bool isSubgraphSupported (int64_t const index) noexcept=0

  Returns whether the subgraph is supported. Calling this function before calling `supportsModelV2` results in undefined behavior.
- virtual int64_t * getSubgraphNodes (int64_t const index, int64_t &subgraphLength) noexcept=0

  Get the nodes of the specified subgraph. Calling this function before calling `supportsModelV2` results in undefined behavior.

9.109.1 Detailed Description

an object for parsing ONNX models into a TensorRT network definition

> Warning
>
> If the ONNX model has a graph output with the same name as a graph input, the output will be renamed by prepending "__".
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.109.2 Constructor & Destructor Documentation

9.109.2.1 ~IParser()

```
virtual nvonnxparser::IParser::~IParser ( )  [virtual], [default], [noexcept]
```

9.109.3 Member Function Documentation

9.109.3.1 clearErrors()

```
virtual void nvonnxparser::IParser::clearErrors ( )  [pure virtual], [noexcept]
```

Clear errors from prior calls to `parse`.

See also

getNbErrors() getError() IParserError

9.109.3.2 clearFlag()

```
virtual void nvonnxparser::IParser::clearFlag (
            OnnxParserFlag onnxParserFlag )  [pure virtual], [noexcept]
```

clear a parser flag.

clears the parser flag from the enabled flags.

See also

setFlags()

9.109.3.3 getError()

```
virtual IParserError const * nvonnxparser::IParser::getError (
            int index ) const  [pure virtual], [noexcept]
```

Get an error that occurred during prior calls to `parse`.

See also

getNbErrors() clearErrors() IParserError

9.109.3.4 getFlag()

```
virtual bool nvonnxparser::IParser::getFlag (
            OnnxParserFlag onnxParserFlag ) const  [pure virtual], [noexcept]
```

Returns true if the parser flag is set.

See also

getFlags()

Returns

True if flag is set, false if unset.

9.109.3.5 getFlags()

```
virtual OnnxParserFlags nvonnxparser::IParser::getFlags ( ) const  [pure virtual], [noexcept]
```

Get the parser flags. Defaults to 0.

Returns

The parser flags as a bitmask.

See also

setFlags()

9.109.3.6 getLayerOutputTensor()

```
virtual nvinfer1::ITensor const * nvonnxparser::IParser::getLayerOutputTensor (
            char const * name,
            int64_t i )  [pure virtual], [noexcept]
```

Return the i-th output ITensor object for the ONNX layer ”name”.

Return the i-th output ITensor object for the ONNX layer ”name”. If ”name” is not found or i is out of range, return nullptr. In the case of multiple nodes sharing the same name this function will return the output tensors of the first instance of the node in the ONNX graph.

Parameters

name	The name of the ONNX layer.
i	The index of the output. i must be in range [0, layer.num_outputs).

9.109.3.7 getNbErrors()

```
virtual int nvonnxparser::IParser::getNbErrors ( ) const  [pure virtual], [noexcept]
```

Get the number of errors that occurred during prior calls to `parse`.

See also

getError() clearErrors() IParserError

9.109.3.8 getNbSubgraphs()

```
virtual int64_t nvonnxparser::IParser::getNbSubgraphs ( )  [pure virtual], [noexcept]
```

Get the number of subgraphs. Calling this function before calling `supportsModelV2` results in undefined behavior.

Returns

Number of subgraphs.

9.109.3.9 getSubgraphNodes()

```
virtual int64_t * nvonnxparser::IParser::getSubgraphNodes (
            int64_t const index,
            int64_t & subgraphLength )  [pure virtual], [noexcept]
```

Get the nodes of the specified subgraph. Calling this function before calling `supportsModelV2` results in undefined behavior.

Parameters

index	Index of the subgraph.
subgraphLength	Returns the length of the subgraph as reference.

Returns

Pointer to the subgraph nodes array. This pointer is owned by the Parser.

9.109.3.10 getUsedVCPluginLibraries()

```
virtual char const *const * nvonnxparser::IParser::getUsedVCPluginLibraries (
            int64_t & nbPluginLibs ) const  [pure virtual], [noexcept]
```

Query the plugin libraries needed to implement operations used by the parser in a version-compatible engine.

This provides a list of plugin libraries on the filesystem needed to implement operations in the parsed network. If you are building a version-compatible engine using this network, provide this list to IBuilderConfig::setPluginsToSerialize to serialize these plugins along with the version-compatible engine, or, if you want to ship these plugin libraries externally to the engine, ensure that IPluginRegistry::loadLibrary is used to load these libraries in the appropriate runtime before deserializing the corresponding engine.

Parameters

out	*nbPluginLibs*	Returns the number of plugin libraries in the array, or -1 if there was an error.

Returns

Array of `nbPluginLibs` C-strings describing plugin library paths on the filesystem if nbPluginLibs > 0, or nullptr otherwise. This array is owned by the IParser, and the pointers in the array are only valid until the next call to parse(), supportsModel(), parseFromFile(), or parseWithWeightDescriptors().

9.109.3.11 isSubgraphSupported()

```
virtual bool nvonnxparser::IParser::isSubgraphSupported (
            int64_t const index )  [pure virtual], [noexcept]
```

Returns whether the subgraph is supported. Calling this function before calling `supportsModelV2` results in undefined behavior.

Parameters

index	Index of the subgraph.

Returns

Whether the subgraph is supported.

9.109.3.12 parse()

```
virtual bool nvonnxparser::IParser::parse (
            void const * serialized_onnx_model,
            size_t serialized_onnx_model_size,
            const char * model_path = nullptr )  [pure virtual], [noexcept]
```

Parse a serialized ONNX model into the TensorRT network. This method has very limited diagnostics. If parsing the serialized model fails for any reason (e.g. unsupported IR version, unsupported opset, etc.) it the user responsibility to intercept and report the error. To obtain a better diagnostic, use the parseFromFile method below.

Parameters

serialized_onnx_model	Pointer to the serialized ONNX model
serialized_onnx_model_size	Size of the serialized ONNX model in bytes
model_path	Absolute path to the model file for loading external weights if required

Returns

true if the model was parsed successfully

See also

getNbErrors() getError()

9.109.3.13 parseFromFile()

```
virtual bool nvonnxparser::IParser::parseFromFile (
            const char * onnxModelFile,
            int verbosity )  [pure virtual], [noexcept]
```

Parse an onnx model file, which can be a binary protobuf or a text onnx model calls parse method inside.

Parameters

onnxModelFile	name
verbosity	Level

Returns

true if the model was parsed successfully

9.109.3.14 parseWithWeightDescriptors()

```
virtual bool nvonnxparser::IParser::parseWithWeightDescriptors (
            void const * serialized_onnx_model,
            size_t serialized_onnx_model_size )  [pure virtual], [noexcept]
```

Parse a serialized ONNX model into the TensorRT network with consideration of user provided weights.

Parameters

serialized_onnx_model	Pointer to the serialized ONNX model
serialized_onnx_model_size	Size of the serialized ONNX model in bytes

Returns

true if the model was parsed successfully

See also

getNbErrors() getError()

9.109.3.15 setFlag()

```
virtual void nvonnxparser::IParser::setFlag (
            OnnxParserFlag onnxParserFlag )  [pure virtual], [noexcept]
```

Set a single parser flag.

Add the input parser flag to the already enabled flags.

See also

setFlags()

9.109.3.16 setFlags()

```
virtual void nvonnxparser::IParser::setFlags (
            OnnxParserFlags onnxParserFlags )  [pure virtual], [noexcept]
```

Set the parser flags.

The flags are listed in the OnnxParserFlag enum.

Parameters

OnnxParserFlags	The flags used when parsing an ONNX model.

Note

This function will override the previous set flags, rather than bitwise ORing the new flag.

See also

getFlags()

9.109.3.17 supportsModel()

```
virtual TRT_DEPRECATED 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], [noexcept]
```

Check whether TensorRT supports a particular ONNX model. If the function returns True, one can proceed to engine building without having to call `parse` or `parseFromFile`.

[DEPRECATED] Deprecated in TensorRT 10.1. See supportsModelV2.

Parameters

serialized_onnx_model	Pointer to the serialized ONNX model
serialized_onnx_model_size	Size of the serialized ONNX model in bytes
sub_graph_collection	Container to hold supported subgraphs
model_path	Absolute path to the model file for loading external weights if required

Returns

true if the model is supported

9.109.3.18 supportsModelV2()

```
virtual bool nvonnxparser::IParser::supportsModelV2 (
            void const * serializedOnnxModel,
```

```
        size_t serializedOnnxModelSize,
        char const * modelPath = nullptr )  [pure virtual], [noexcept]
```

Check whether TensorRT supports a particular ONNX model. If the function returns True, one can proceed to engine building without having to call `parse` or `parseFromFile`. Results can be queried through `getNbSubgraphs`, `isSubgraphSupported`, `getSubgraphNodes`.

Parameters

serializedOnnxModel	Pointer to the serialized ONNX model
serializedOnnxModelSize	Size of the serialized ONNX model in bytes
modelPath	Absolute path to the model file for loading external weights if required

Returns

true if the model is supported

9.109.3.19 supportsOperator()

```
virtual bool nvonnxparser::IParser::supportsOperator (
        const char * op_name ) const  [pure virtual], [noexcept]
```

Returns whether the specified operator may be supported by the parser.

Note that a result of true does not guarantee that the operator will be supported in all cases (i.e., this function may return false-positives).

Parameters

op_name	The name of the ONNX operator to check for support

The documentation for this class was generated from the following file:

- NvOnnxParser.h

9.110 nvonnxparser::IParserError Class Reference

an object containing information about an error

```
#include <NvOnnxParser.h>
```

Public Member Functions

- virtual ErrorCode code () const =0

  the error code.
- virtual char const * desc () const =0

  description of the error.
- virtual char const * file () const =0

  source file in which the error occurred.
- virtual int line () const =0

  source line at which the error occurred.
- virtual char const * func () const =0

  source function in which the error occurred.
- virtual int node () const =0

  index of the ONNX model node in which the error occurred.
- virtual char const * nodeName () const =0

  name of the node in which the error occurred.
- virtual char const * nodeOperator () const =0

  name of the node operation in which the error occurred.
- virtual char const *const * localFunctionStack () const =0

  A list of the local function names, from the top level down, constituting the current stack trace in which the error occurred. A top-level node that is not inside any local function would return a nullptr.
- virtual int32_t localFunctionStackSize () const =0

  The size of the stack of local functions at the point where the error occurred. A top-level node that is not inside any local function would correspond to.

Protected Member Functions

- virtual ~IParserError ()

9.110.1 Detailed Description

an object containing information about an error

9.110.2 Constructor & Destructor Documentation

9.110.2.1 ~IParserError()

```
virtual nvonnxparser::IParserError::~IParserError ( )  [inline], [protected], [virtual]
```

9.110.3 Member Function Documentation

9.110.3.1 code()

```
virtual ErrorCode nvonnxparser::IParserError::code ( ) const  [pure virtual]
```

the error code.

9.110.3.2 desc()

```
virtual char const * nvonnxparser::IParserError::desc ( ) const  [pure virtual]
```

description of the error.

9.110.3.3 file()

```
virtual char const * nvonnxparser::IParserError::file ( ) const  [pure virtual]
```

source file in which the error occurred.

9.110.3.4 func()

```
virtual char const * nvonnxparser::IParserError::func ( ) const  [pure virtual]
```

source function in which the error occurred.

9.110.3.5 line()

```
virtual int nvonnxparser::IParserError::line ( ) const  [pure virtual]
```

source line at which the error occurred.

9.110.3.6 localFunctionStack()

```
virtual char const *const * nvonnxparser::IParserError::localFunctionStack ( ) const  [pure virtual]
```

A list of the local function names, from the top level down, constituting the current stack trace in which the error occurred. A top-level node that is not inside any local function would return a nullptr.

9.110.3.7 localFunctionStackSize()

```
virtual int32_t nvonnxparser::IParserError::localFunctionStackSize ( ) const  [pure virtual]
```

The size of the stack of local functions at the point where the error occurred. A top-level node that is not inside any local function would correspond to.

9.110.3.8 node()

```
virtual int nvonnxparser::IParserError::node ( ) const  [pure virtual]
```

index of the ONNX model node in which the error occurred.

9.110.3.9 nodeName()

```
virtual char const * nvonnxparser::IParserError::nodeName ( ) const  [pure virtual]
```

name of the node in which the error occurred.

9.110.3.10 nodeOperator()

```
virtual char const * nvonnxparser::IParserError::nodeOperator ( ) const  [pure virtual]
```

name of the node operation in which the error occurred.

The documentation for this class was generated from the following file:

- NvOnnxParser.h

9.111 nvonnxparser::IParserRefitter Class Reference

An interface designed to refit weights from an ONNX model.

```
#include <NvOnnxParser.h>
```

Public Member Functions

- virtual bool refitFromBytes (void const *serializedOnnxModel, size_t serializedOnnxModelSize, char const *modelPath=nullptr) noexcept=0

  Load a serialized ONNX model from memory and perform weight refit.
- virtual bool refitFromFile (char const *onnxModelFile) noexcept=0

  Load and parse a ONNX model from disk and perform weight refit.
- virtual int32_t getNbErrors () const noexcept=0

  Get the number of errors that occurred during prior calls to `refitFromBytes` or `refitFromFile`.
- virtual IParserError const * getError (int32_t index) const noexcept=0

  Get an error that occurred during prior calls to `refitFromBytes` or `refitFromFile`.
- virtual void clearErrors ()=0

  Clear errors from prior calls to `refitFromBytes` or `refitFromFile`.
- virtual ~IParserRefitter () noexcept=default

9.111.1 Detailed Description

An interface designed to refit weights from an ONNX model.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.111.2 Constructor & Destructor Documentation

9.111.2.1 ~IParserRefitter()

```
virtual nvonnxparser::IParserRefitter::~IParserRefitter ( )  [virtual], [default], [noexcept]
```

9.111.3 Member Function Documentation

9.111.3.1 clearErrors()

```
virtual void nvonnxparser::IParserRefitter::clearErrors ( )  [pure virtual]
```

Clear errors from prior calls to `refitFromBytes` or `refitFromFile`.

See also

getNbErrors() getError() IParserError

9.111.3.2 getError()

```
virtual IParserError const * nvonnxparser::IParserRefitter::getError (
            int32_t index ) const  [pure virtual], [noexcept]
```

Get an error that occurred during prior calls to `refitFromBytes` or `refitFromFile`.

See also

getNbErrors() IParserError

9.111.3.3 getNbErrors()

```
virtual int32_t nvonnxparser::IParserRefitter::getNbErrors ( ) const  [pure virtual], [noexcept]
```

Get the number of errors that occurred during prior calls to `refitFromBytes` or `refitFromFile`.

See also

getError() IParserError

9.111.3.4 refitFromBytes()

```
virtual bool nvonnxparser::IParserRefitter::refitFromBytes (
            void const * serializedOnnxModel,
            size_t serializedOnnxModelSize,
            char const * modelPath = nullptr )  [pure virtual], [noexcept]
```

Load a serialized ONNX model from memory and perform weight refit.

Parameters

serializedOnnxModel	Pointer to the serialized ONNX model
serializedOnnxModelSize	Size of the serialized ONNX model in bytes
modelPath	Absolute path to the model file for loading external weights if required

Returns

true if all the weights in the engine were refit successfully.

The serialized ONNX model must be identical to the one used to generate the engine that will be refit.

9.111.3.5 refitFromFile()

```
virtual bool nvonnxparser::IParserRefitter::refitFromFile (
            char const * onnxModelFile )  [pure virtual], [noexcept]
```

Load and parse a ONNX model from disk and perform weight refit.

Parameters

onnxModelFile	Path to the ONNX model to load from disk.

Returns

true if the model was loaded successfully, and if all the weights in the engine were refit successfully.

The provided ONNX model must be identical to the one used to generate the engine that will be refit.

The documentation for this class was generated from the following file:

- NvOnnxParser.h

9.112 IPluginCapability Class Reference

Base class for plugin capability interfaces.

```
#include <NvInferPluginBase.h>
```

9.112.1 Detailed Description

Base class for plugin capability interfaces.

IPluginCapability represents a split in TensorRT V3 plugins to sub-objects that expose different types of capabilites a plugin may have, as opposed to a single interface which defines all capabilities and behaviors of a plugin.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

See also

PluginCapabilityType

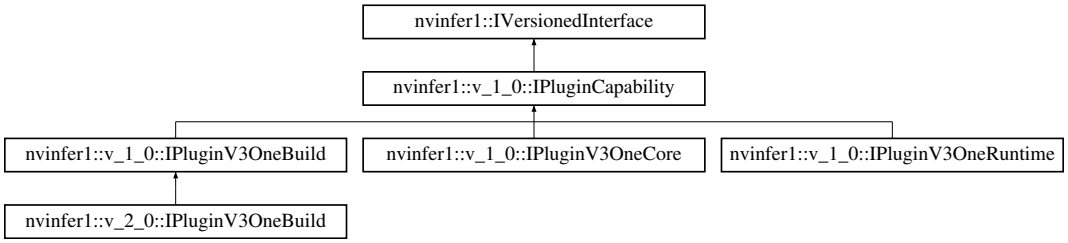
The documentation for this class was generated from the following file:

- NvInferPluginBase.h

9.113 nvinfer1::v_1_0::IPluginCapability Class Reference

`#include <NvInferPluginBase.h>`

Inheritance diagram for nvinfer1::v_1_0::IPluginCapability:



Additional Inherited Members

The documentation for this class was generated from the following file:

- NvInferPluginBase.h

9.114 IPluginCreator Class Reference

Plugin creator class for user implemented layers.

`#include <NvInferRuntimePlugin.h>`

9.114.1 Detailed Description

Plugin creator class for user implemented layers.

See also

IPlugin and IPluginFactory

Deprecated Deprecated in TensorRT 10.0. Please implement IPluginCreatorV3One along with IPluginV3 plugins instead.

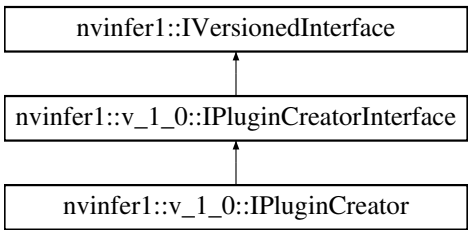
The documentation for this class was generated from the following file:

- NvInferRuntimePlugin.h

9.115 nvinfer1::v_1_0::IPluginCreator Class Reference

`#include <NvInferRuntimePlugin.h>`

Inheritance diagram for nvinfer1::v_1_0::IPluginCreator:



Public Member Functions

- virtual AsciiChar const * getPluginName () const noexcept=0

  Return the plugin name.
- virtual AsciiChar const * getPluginVersion () const noexcept=0

  Return the plugin version.
- virtual PluginFieldCollection const * getFieldNames () noexcept=0

  Return a list of fields that need to be passed to createPlugin.
- virtual IPluginV2 * createPlugin (AsciiChar const *name, PluginFieldCollection const *fc) noexcept=0

  Return a plugin object. Return nullptr in case of error.
- virtual IPluginV2 * deserializePlugin (AsciiChar const *name, void const *serialData, size_t serialLength) noexcept=0

  Called during deserialization of plugin layer. Return a plugin object.
- virtual void setPluginNamespace (AsciiChar const *pluginNamespace) noexcept=0

  Set the namespace of the plugin creator based on the plugin library it belongs to. This can be set while registering the plugin creator.
- virtual AsciiChar const * getPluginNamespace () const noexcept=0

  Return the namespace of the plugin creator object.
- IPluginCreator ()=default
- ~IPluginCreator () override=default
- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.

Additional Inherited Members

9.115.1 Constructor & Destructor Documentation

9.115.1.1 IPluginCreator()

```
nvinfer1::v_1_0::IPluginCreator::IPluginCreator ( )  [default]
```

9.115.1.2 ~IPluginCreator()

```
nvinfer1::v_1_0::IPluginCreator::~IPluginCreator ( )  [override], [default]
```

9.115.2 Member Function Documentation

9.115.2.1 createPlugin()

```
virtual IPluginV2 * nvinfer1::v_1_0::IPluginCreator::createPlugin (
            AsciiChar const * name,
            PluginFieldCollection const * fc )  [pure virtual], [noexcept]
```

Return a plugin object. Return nullptr in case of error.

Parameters

name	A NULL-terminated name string of length 1024 or less, including the NULL terminator.
fc	A pointer to a collection of fields needed for constructing the plugin.

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.

9.115.2.2 deserializePlugin()

```
virtual IPluginV2 * nvinfer1::v_1_0::IPluginCreator::deserializePlugin (
            AsciiChar const * name,
            void const * serialData,
            size_t serialLength )  [pure virtual], [noexcept]
```

Called during deserialization of plugin layer. Return a plugin object.

Parameters

name	A NULL-terminated name string of length 1024 or less, including the NULL terminator.
serialData	The start address of a byte array with the serialized plugin representation.
serialLength	The length in bytes of the byte array with the serialized plugin representation.

Returns

A deserialized plugin object

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.

9.115.2.3 getFieldNames()

```
virtual PluginFieldCollection const * nvinfer1::v_1_0::IPluginCreator::getFieldNames ( )  [pure
virtual], [noexcept]
```

Return a list of fields that need to be passed to createPlugin.

See also

PluginFieldCollection

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.

9.115.2.4 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IPluginCreator::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.115.2.5 getPluginName()

```
virtual AsciiChar const * nvinfer1::v_1_0::IPluginCreator::getPluginName ( ) const  [pure virtual],
[noexcept]
```

Return the plugin name.

> Warning
>
> The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

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.

9.115.2.6 getPluginNamespace()

```
virtual AsciiChar const * nvinfer1::v_1_0::IPluginCreator::getPluginNamespace ( ) const  [pure
virtual], [noexcept]
```

Return the namespace of the plugin creator object.

> Warning
>
> The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

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.

9.115.2.7 getPluginVersion()

```
virtual AsciiChar const * nvinfer1::v_1_0::IPluginCreator::getPluginVersion ( ) const  [pure virtual],
[noexcept]
```

Return the plugin version.

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

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.

9.115.2.8 setPluginNamespace()

```
virtual void nvinfer1::v_1_0::IPluginCreator::setPluginNamespace (
            AsciiChar const * pluginNamespace )  [pure virtual], [noexcept]
```

Set the namespace of the plugin creator based on the plugin library it belongs to. This can be set while registering the plugin creator.

Parameters

pluginNamespace	A NULL-terminated namespace string of length 1024 or less, including the NULL terminator

See also

IPluginRegistry::registerCreator()

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when deserializing multiple engines concurrently sharing plugins.

The documentation for this class was generated from the following file:

- NvInferRuntimePlugin.h

9.116 IPluginCreatorInterface Class Reference

Base class for all plugin creator versions.

`#include <NvInferPluginBase.h>`

9.116.1 Detailed Description

Base class for all plugin creator versions.

See also

IPluginCreator and IPluginRegistry

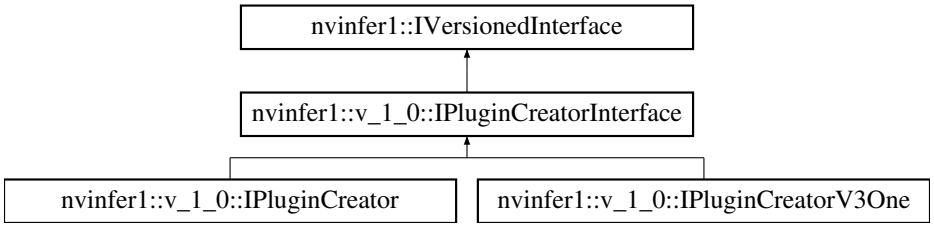
The documentation for this class was generated from the following file:

- NvInferPluginBase.h

9.117 nvinfer1::v_1_0::IPluginCreatorInterface Class Reference

`#include <NvInferPluginBase.h>`

Inheritance diagram for nvinfer1::v_1_0::IPluginCreatorInterface:



Public Member Functions

- ~IPluginCreatorInterface () noexcept override=default

Protected Member Functions

- IPluginCreatorInterface ()=default
- IPluginCreatorInterface (IPluginCreatorInterface const &)=default
- IPluginCreatorInterface (IPluginCreatorInterface &&)=default
- IPluginCreatorInterface & operator= (IPluginCreatorInterface const &) &=default
- IPluginCreatorInterface & operator= (IPluginCreatorInterface &&) &=default

9.117.1 Constructor & Destructor Documentation

9.117.1.1 ~IPluginCreatorInterface()

```
nvinfer1::v_1_0::IPluginCreatorInterface::~IPluginCreatorInterface ( )  [override], [default],
[noexcept]
```

9.117.1.2 IPluginCreatorInterface() [1/3]

```
nvinfer1::v_1_0::IPluginCreatorInterface::IPluginCreatorInterface ( )  [protected], [default]
```

9.117.1.3 IPluginCreatorInterface() [2/3]

```
nvinfer1::v_1_0::IPluginCreatorInterface::IPluginCreatorInterface (
            IPluginCreatorInterface const &  )  [protected], [default]
```

9.117.1.4 IPluginCreatorInterface() [3/3]

```
nvinfer1::v_1_0::IPluginCreatorInterface::IPluginCreatorInterface (
            IPluginCreatorInterface &&  )  [protected], [default]
```

9.117.2 Member Function Documentation

9.117.2.1 operator=() [1/2]

```
IPluginCreatorInterface & nvinfer1::v_1_0::IPluginCreatorInterface::operator= (
            IPluginCreatorInterface &&  ) &  [protected], [default]
```

9.117.2.2 operator=() [2/2]

```
IPluginCreatorInterface & nvinfer1::v_1_0::IPluginCreatorInterface::operator= (
            IPluginCreatorInterface const &  ) &  [protected], [default]
```

The documentation for this class was generated from the following file:

- NvInferPluginBase.h

9.118 IPluginCreatorV3One Class Reference

A plugin creator class capable of producing IPluginV3 objects.

`#include <NvInferRuntime.h>`

9.118.1 Detailed Description

A plugin creator class capable of producing IPluginV3 objects.

See also

IPluginV3

IPluginRegistry

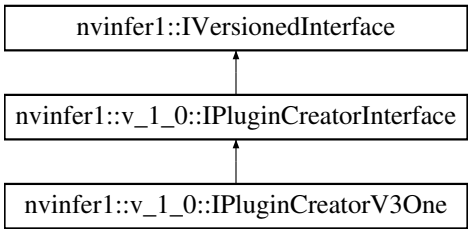
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.119 nvinfer1::v_1_0::IPluginCreatorV3One Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IPluginCreatorV3One:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual IPluginV3 * createPlugin (AsciiChar const *name, PluginFieldCollection const *fc, TensorRTPhase phase) noexcept=0

  Return a plugin object. Return nullptr in case of error.
- virtual PluginFieldCollection const * getFieldNames () noexcept=0

  Return a list of fields that need to be passed to createPlugin() when creating a plugin for use in the TensorRT build phase.
- virtual AsciiChar const * getPluginName () const noexcept=0

  Return the plugin name.
- virtual AsciiChar const * getPluginVersion () const noexcept=0

  Return the plugin version.
- virtual AsciiChar const * getPluginNamespace () const noexcept=0

  Return the plugin namespace.
- IPluginCreatorV3One ()=default
- virtual ~IPluginCreatorV3One ()=default

Protected Member Functions

- IPluginCreatorV3One (IPluginCreatorV3One const &)=default
- IPluginCreatorV3One (IPluginCreatorV3One &&)=default
- IPluginCreatorV3One & operator= (IPluginCreatorV3One const &) &=default
- IPluginCreatorV3One & operator= (IPluginCreatorV3One &&) &=default

9.119.1 Constructor & Destructor Documentation

9.119.1.1 IPluginCreatorV3One() [1/3]

```
nvinfer1::v_1_0::IPluginCreatorV3One::IPluginCreatorV3One ( )  [default]
```

9.119.1.2 ~IPluginCreatorV3One()

```
virtual nvinfer1::v_1_0::IPluginCreatorV3One::~IPluginCreatorV3One ( )  [virtual], [default]
```

9.119.1.3 IPluginCreatorV3One() [2/3]

```
nvinfer1::v_1_0::IPluginCreatorV3One::IPluginCreatorV3One (
            IPluginCreatorV3One const &  )  [protected], [default]
```

9.119.1.4 IPluginCreatorV3One() [3/3]

```
nvinfer1::v_1_0::IPluginCreatorV3One::IPluginCreatorV3One (
            IPluginCreatorV3One &&  )  [protected], [default]
```

9.119.2 Member Function Documentation

9.119.2.1 createPlugin()

```
virtual IPluginV3 * nvinfer1::v_1_0::IPluginCreatorV3One::createPlugin (
            AsciiChar const * name,
            PluginFieldCollection const * fc,
            TensorRTPhase phase )  [pure virtual], [noexcept]
```

Return a plugin object. Return nullptr in case of error.

Parameters

name	A NULL-terminated name string of length 1024 or less, including the NULL terminator.
fc	A pointer to a collection of fields needed for constructing the plugin.
phase	The TensorRT phase in which the plugin is being created

When the phase is TensorRTPhase::kRUNTIME, the PluginFieldCollection provided for serialization by the plugin's runtime interface will be passed as fc.

Note

The returned plugin object must be in an initialized state

If invoked by the user (e.g. with TensorRTPhase::kBUILD, to add to the network defintion with addPlugin↩ V3()), it is the user's responsibility to delete the plugin object. If invoked by TensorRT (e.g. during engine deserialization), TensorRT will delete any objects it creates.

9.119.2.2 getFieldNames()

```
virtual PluginFieldCollection const * nvinfer1::v_1_0::IPluginCreatorV3One::getFieldNames ( )  [pure virtual], [noexcept]
```

Return a list of fields that need to be passed to createPlugin() when creating a plugin for use in the TensorRT build phase.

See also

PluginFieldCollection

9.119.2.3 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IPluginCreatorV3One::getInterfaceInfo ( ) const  [inline], [override], [virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.119.2.4 getPluginName()

```
virtual AsciiChar const * nvinfer1::v_1_0::IPluginCreatorV3One::getPluginName ( ) const  [pure virtual], [noexcept]
```

Return the plugin name.

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

9.119.2.5 getPluginNamespace()

```
virtual AsciiChar const * nvinfer1::v_1_0::IPluginCreatorV3One::getPluginNamespace ( ) const  [pure virtual], [noexcept]
```

Return the plugin namespace.

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

9.119.2.6 getPluginVersion()

```
virtual AsciiChar const * nvinfer1::v_1_0::IPluginCreatorV3One::getPluginVersion ( ) const  [pure virtual], [noexcept]
```

Return the plugin version.

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

9.119.2.7 operator=() [1/2]

```
IPluginCreatorV3One & nvinfer1::v_1_0::IPluginCreatorV3One::operator= (
            IPluginCreatorV3One &&  ) &  [protected], [default]
```

9.119.2.8 operator=() [2/2]

```
IPluginCreatorV3One & nvinfer1::v_1_0::IPluginCreatorV3One::operator= (
            IPluginCreatorV3One const &  ) &  [protected], [default]
```

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.120 nvinfer1::IPluginRegistry Class Reference

Single registration point for all plugins in an application. It is used to find plugin implementations during engine deserialization. Internally, the plugin registry is considered to be a singleton so all plugins in an application are part of the same global registry. Note that the plugin registry is only supported for plugins of type IPluginV2 and should also have a corresponding IPluginCreator implementation.

```
#include <NvInferRuntimeCommon.h>
```

Public Types

- using PluginLibraryHandle = void *

  Pointer for plugin library handle.

Public Member Functions

- virtual TRT_DEPRECATED bool registerCreator (IPluginCreator &creator, AsciiChar const *const pluginNamespace) noexcept=0

  Register a plugin creator implementing IPluginCreator. Returns false if any plugin creator with the same name, version or namespace is already registered.
- virtual TRT_DEPRECATED 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 TRT_DEPRECATED IPluginCreator * getPluginCreator (AsciiChar const *const pluginName, AsciiChar const *const pluginVersion, AsciiChar const *const pluginNamespace="") noexcept=0

  Return plugin creator based on plugin name, version, and namespace associated with plugin during network creation.
- virtual void setErrorRecorder (IErrorRecorder *const recorder) noexcept=0

  Set the ErrorRecorder for this interface.
- virtual IErrorRecorder * getErrorRecorder () const noexcept=0

  Get the ErrorRecorder assigned to this interface.
- virtual TRT_DEPRECATED bool deregisterCreator (IPluginCreator const &creator) noexcept=0

  Deregister a previously registered plugin creator implementing IPluginCreator.
- virtual bool isParentSearchEnabled () const =0

  Return whether the parent registry will be searched if a plugin is not found in this registry default: true.
- virtual void setParentSearchEnabled (bool const enabled)=0

  Set whether the parent registry will be searched if a plugin is not found in this registry.
- virtual PluginLibraryHandle loadLibrary (AsciiChar const *pluginPath) noexcept=0

  Load and register a shared library of plugins.
- virtual void deregisterLibrary (PluginLibraryHandle handle) noexcept=0

  Deregister plugins associated with a library. Any resources acquired when the library was loaded will be released.
- virtual bool registerCreator (IPluginCreatorInterface &creator, AsciiChar const *const pluginNamespace) noexcept=0

  Register a plugin creator. Returns false if a plugin creator with the same type is already registered.
- virtual IPluginCreatorInterface *const * getAllCreators (int32_t *const numCreators) const noexcept=0

  Return all registered plugin creators. Returns nullptr if none found.
- virtual IPluginCreatorInterface * getCreator (AsciiChar const *const pluginName, AsciiChar const *const pluginVersion, AsciiChar const *const pluginNamespace="") noexcept=0

  Return a registered plugin creator based on plugin name, version, and namespace associated with the plugin during network creation.
- virtual bool deregisterCreator (IPluginCreatorInterface const &creator) noexcept=0

  Deregister a previously registered plugin creator.
- virtual IPluginResource * acquirePluginResource (AsciiChar const *key, IPluginResource *resource) noexcept=0

  Get a plugin resource.
- virtual int32_t releasePluginResource (AsciiChar const *key) noexcept=0

  Decrement reference count for the resource with this key If reference count goes to zero after decrement, release() will be invoked on the resource, the key will be deregistered and the resource object will be deleted.
- virtual IPluginCreatorInterface *const * getAllCreatorsRecursive (int32_t *const numCreators) noexcept=0

  Return all registered plugin creators by searching starting from the current registry and following parent registries recursively as long as isParentSearchEnabled() returns true.

Protected Member Functions

- virtual ~IPluginRegistry () noexcept=default

9.120.1 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.

9.120.2 Member Typedef Documentation

9.120.2.1 PluginLibraryHandle

```
using nvinfer1::IPluginRegistry::PluginLibraryHandle = void*
```

Pointer for plugin library handle.

9.120.3 Constructor & Destructor Documentation

9.120.3.1 ~IPluginRegistry()

```
virtual nvinfer1::IPluginRegistry::~IPluginRegistry ( )  [protected], [virtual], [default], [noexcept]
```

9.120.4 Member Function Documentation

9.120.4.1 acquirePluginResource()

```
virtual IPluginResource * nvinfer1::IPluginRegistry::acquirePluginResource (
            AsciiChar const * key,
            IPluginResource * resource )  [pure virtual], [noexcept]
```

Get a plugin resource.

Parameters

key	Key for identifying the resource. Cannot be null.
resource	A plugin resource object. The object will only need to be valid until this method returns, as only a clone of this object will be registered by TRT. Cannot be null.

Returns

Registered plugin resource object

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes; calls to this method will be synchronized by a mutex.

9.120.4.2 deregisterCreator() [1/2]

```
virtual TRT_DEPRECATED bool nvinfer1::IPluginRegistry::deregisterCreator (
            IPluginCreator const & creator )  [pure virtual], [noexcept]
```

Deregister a previously registered plugin creator implementing IPluginCreator.

Since there may be a desire to limit the number of plugins, this function provides a mechanism for removing plugin creators registered in TensorRT. The plugin creator that is specified by `creator` is removed from TensorRT and no longer tracked.

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

Deprecated Deprecated in TensorRT 10.0. Superseded by IPluginRegistry::deregisterCreator(IPluginCreatorInterface const&).

9.120.4.3 deregisterCreator() [2/2]

```
virtual bool nvinfer1::IPluginRegistry::deregisterCreator (
            IPluginCreatorInterface const & creator )  [pure virtual], [noexcept]
```

Deregister a previously registered plugin creator.

Since there may be a desire to limit the number of plugins, this function provides a mechanism for removing plugin creators registered in TensorRT. The plugin creator that is specified by `creator` is removed from TensorRT and no longer tracked.

Returns

True if the plugin creator was deregistered, false if it was not found in the registry or otherwise could not be deregistered.

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes

9.120.4.4 deregisterLibrary()

```
virtual void nvinfer1::IPluginRegistry::deregisterLibrary (
            PluginLibraryHandle handle )  [pure virtual], [noexcept]
```

Deregister plugins associated with a library. Any resources acquired when the library was loaded will be released.

Parameters

handle	the plugin library handle to deregister.

9.120.4.5 getAllCreators()

```
virtual IPluginCreatorInterface *const * nvinfer1::IPluginRegistry::getAllCreators (
            int32_t *const numCreators ) const  [pure virtual], [noexcept]
```

Return all registered plugin creators. Returns nullptr if none found.

Warning

If any plugin creators are registered or deregistered after calling this function, the returned pointer is not guaranteed to be valid thereafter.

Usage considerations

- Allowed context for the API call
  - Thread-safe: No

9.120.4.6 getAllCreatorsRecursive()

```
virtual IPluginCreatorInterface *const * nvinfer1::IPluginRegistry::getAllCreatorsRecursive (
            int32_t *const numCreators ) [pure virtual], [noexcept]
```

Return all registered plugin creators by searching starting from the current registry and following parent registries recursively as long as isParentSearchEnabled() returns true.

Parameters

out	*numCreators*	Pointer to an integer where the number of registered plugin creators will be stored.

Returns

A pointer to an array of IPluginCreatorInterface pointers. Returns nullptr if no creators are found.

Warning

If any plugin creators are registered or deregistered after calling this function, the returned pointer is not guaranteed to remain valid.

Usage considerations

- Allowed context for the API call
  - Thread-safe: No

9.120.4.7 getCreator()

```
virtual IPluginCreatorInterface * nvinfer1::IPluginRegistry::getCreator (
            AsciiChar const *const pluginName,
            AsciiChar const *const pluginVersion,
            AsciiChar const *const pluginNamespace = "" )  [pure virtual], [noexcept]
```

Return a registered plugin creator based on plugin name, version, and namespace associated with the plugin during network creation.

> Warning
>
> The strings pluginName, pluginVersion, and pluginNamespace must be 1024 bytes or less including the NULL terminator and must be NULL terminated.

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes

9.120.4.8 getErrorRecorder()

```
virtual IErrorRecorder * nvinfer1::IPluginRegistry::getErrorRecorder ( ) const  [pure virtual],
[noexcept]
```

Get the ErrorRecorder assigned to this interface.

Retrieves the assigned error recorder object for the given class. A default error recorder does not exist, so a nullptr will be returned if setErrorRecorder has not been called, or an ErrorRecorder has not been inherited.

Returns

A pointer to the IErrorRecorder object that has been registered.

See also

setErrorRecorder()

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes

9.120.4.9 getPluginCreator()

```
virtual TRT_DEPRECATED 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.

Returns nullptr if a plugin creator with matching name, version, and namespace is found, but is not a descendent of IPluginCreator

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes

Deprecated Deprecated in TensorRT 10.0. Superseded by IPluginRegistry::getCreator(AsciiChar const* const, AsciiChar const* const, AsciiChar const* const).

9.120.4.10 getPluginCreatorList()

```
virtual TRT_DEPRECATED 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.

Warning

If any plugin creators are registered or deregistered after calling this function, the returned pointer is not guaranteed to be valid thereafter.

Usage considerations

- Allowed context for the API call
  - Thread-safe: No

Deprecated Deprecated in TensorRT 10.0. Superseded by IPluginRegistry::getAllCreators(int32_t* const).

9.120.4.11 isParentSearchEnabled()

```
virtual bool nvinfer1::IPluginRegistry::isParentSearchEnabled ( ) const  [pure virtual]
```

Return whether the parent registry will be searched if a plugin is not found in this registry default: true.

Returns

bool variable indicating whether parent search is enabled.

See also

setParentSearchEnabled

9.120.4.12 loadLibrary()

```
virtual PluginLibraryHandle nvinfer1::IPluginRegistry::loadLibrary (
            AsciiChar const * pluginPath )  [pure virtual], [noexcept]
```

Load and register a shared library of plugins.

Parameters

pluginPath	the plugin library path.

Returns

The loaded plugin library handle. The call will fail and return nullptr if any of the plugins are already registered.

9.120.4.13 registerCreator() [1/2]

```
virtual TRT_DEPRECATED bool nvinfer1::IPluginRegistry::registerCreator (
            IPluginCreator & creator,
            AsciiChar const *const pluginNamespace )  [pure virtual], [noexcept]
```

Register a plugin creator implementing IPluginCreator. Returns false if any plugin creator with the same name, version or namespace 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.

Deprecated Deprecated in TensorRT 10.0. Superseded by IPluginRegistry::registerCreator(IPluginCreatorInterface&, AsciiChar

9.120.4.14 registerCreator() [2/2]

```
virtual bool nvinfer1::IPluginRegistry::registerCreator (
            IPluginCreatorInterface & creator,
            AsciiChar const *const pluginNamespace )  [pure virtual], [noexcept]
```

Register a plugin creator. Returns false if a plugin creator with the same type is already registered.

> Warning
>
> The string pluginNamespace must be 1024 bytes or less including the NULL terminator and must be NULL terminated.

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes; calls to this method will be synchronized by a mutex.

9.120.4.15 releasePluginResource()

```
virtual int32_t nvinfer1::IPluginRegistry::releasePluginResource (
            AsciiChar const * key )  [pure virtual], [noexcept]
```

Decrement reference count for the resource with this key If reference count goes to zero after decrement, release() will be invoked on the resource, the key will be deregistered and the resource object will be deleted.

Parameters

key	Key that was used to register the resource. Cannot be null.

Returns

0 for success, else non-zero

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes; calls to this method will be synchronized by a mutex.

9.120.4.16 setErrorRecorder()

```
virtual void nvinfer1::IPluginRegistry::setErrorRecorder (
            IErrorRecorder *const recorder )  [pure virtual], [noexcept]
```

Set the ErrorRecorder for this interface.

Assigns the ErrorRecorder to this interface. The ErrorRecorder will track all errors during execution. This function will call incRefCount of the registered ErrorRecorder at least once. Setting recorder to nullptr unregisters the recorder with the interface, resulting in a call to decRefCount if a recorder has been registered.

Parameters

recorder	The error recorder to register with this interface.

See also

getErrorRecorder()

Usage considerations

- Allowed context for the API call
  - Thread-safe: No

9.120.4.17 setParentSearchEnabled()

```
virtual void nvinfer1::IPluginRegistry::setParentSearchEnabled (
            bool const enabled )  [pure virtual]
```

Set whether the parent registry will be searched if a plugin is not found in this registry.

Parameters

enabled	The bool variable indicating whether parent search is enabled.

See also

isParentSearchEnabled

The documentation for this class was generated from the following file:

- NvInferRuntimeCommon.h

9.121 IPluginResource Class Reference

Interface for plugins to define custom resources that could be shared through the plugin registry.

`#include <NvInferPluginBase.h>`

9.121.1 Detailed Description

Interface for plugins to define custom resources that could be shared through the plugin registry.

See also

IPluginRegistry::acquirePluginResource
IPluginRegistry::releasePluginResource

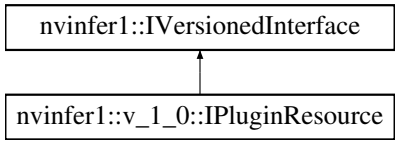
The documentation for this class was generated from the following file:

- NvInferPluginBase.h

9.122 nvinfer1::v_1_0::IPluginResource Class Reference

`#include <NvInferPluginBase.h>`

Inheritance diagram for nvinfer1::v_1_0::IPluginResource:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual int32_t release () noexcept=0

  Free the underlying resource.
- virtual IPluginResource * clone () noexcept=0

  Clone the resource object.
- ~IPluginResource () noexcept override=default
- IPluginResource ()=default
- IPluginResource (IPluginResource const &)=default
- IPluginResource (IPluginResource &&)=default
- IPluginResource & operator= (IPluginResource const &) &=default
- IPluginResource & operator= (IPluginResource &&) &=default

Additional Inherited Members

9.122.1 Constructor & Destructor Documentation

9.122.1.1 ~IPluginResource()

```
nvinfer1::v_1_0::IPluginResource::~IPluginResource ( )  [override], [default], [noexcept]
```

9.122.1.2 IPluginResource() [1/3]

```
nvinfer1::v_1_0::IPluginResource::IPluginResource ( )  [default]
```

9.122.1.3 IPluginResource() [2/3]

```
nvinfer1::v_1_0::IPluginResource::IPluginResource (
            IPluginResource const &  )  [default]
```

9.122.1.4 IPluginResource() [3/3]

```
nvinfer1::v_1_0::IPluginResource::IPluginResource (
            IPluginResource &&  )  [default]
```

9.122.2 Member Function Documentation

9.122.2.1 clone()

```
virtual IPluginResource * nvinfer1::v_1_0::IPluginResource::clone ( )  [pure virtual], [noexcept]
```

Clone the resource object.

Note

Resource initialization (if any) may be skipped for non-cloned objects since only clones will be registered by TensorRT

Returns

Pointer to cloned object. nullptr if there was an issue.

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.

9.122.2.2 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IPluginResource::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.122.2.3 operator=() [1/2]

```
IPluginResource & nvinfer1::v_1_0::IPluginResource::operator= (
            IPluginResource &&  ) &  [default]
```

9.122.2.4 operator=() [2/2]

```
IPluginResource & nvinfer1::v_1_0::IPluginResource::operator= (
            IPluginResource const &  ) &  [default]
```

9.122.2.5 release()

```
virtual int32_t nvinfer1::v_1_0::IPluginResource::release ( )  [pure virtual], [noexcept]
```

Free the underlying resource.

This will only be called for IPluginResource objects that were produced from IPluginResource::clone()

The IPluginResource object on which release() is called must still be in a clone-able state after release() returns

Returns

0 for success, else non-zero
Usage considerations

- Allowed context for the API call
  - Thread-safe: No; this method is not required to be thread-safe

The documentation for this class was generated from the following file:

- NvInferPluginBase.h

9.123 nvinfer1::IPluginResourceContext Class Reference

Interface for plugins to access per context resources provided by TensorRT.

```
#include <NvInferRuntime.h>
```

Public Member Functions

- virtual IGpuAllocator * getGpuAllocator () const noexcept=0
  Get the GPU allocator associated with the resource context.
- virtual IErrorRecorder * getErrorRecorder () const noexcept=0
  Get the error recorder associated with the resource context.
- virtual ~IPluginResourceContext () noexcept=default

Protected Member Functions

- IPluginResourceContext ()=default
- IPluginResourceContext (IPluginResourceContext const &)=default
- IPluginResourceContext (IPluginResourceContext &&)=default
- IPluginResourceContext & operator= (IPluginResourceContext const &) &=default
- IPluginResourceContext & operator= (IPluginResourceContext &&) &=default

9.123.1 Detailed Description

Interface for plugins to access per context resources provided by TensorRT.

There is no public way to construct an IPluginResourceContext. It appears as an argument to IPluginV3OneRuntime::attachToContext(). Overrides of that method can use the IPluginResourceContext object to access any available per context resources.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

See also

IPluginV3OneRuntime::attachToContext()

9.123.2 Constructor & Destructor Documentation

9.123.2.1 ~IPluginResourceContext()

```
virtual nvinfer1::IPluginResourceContext::~IPluginResourceContext ( )  [virtual], [default], [noexcept]
```

9.123.2.2 IPluginResourceContext() [1/3]

```
nvinfer1::IPluginResourceContext::IPluginResourceContext ( )  [protected], [default]
```

9.123.2.3 IPluginResourceContext() [2/3]

```
nvinfer1::IPluginResourceContext::IPluginResourceContext (
            IPluginResourceContext const &  )  [protected], [default]
```

9.123.2.4 IPluginResourceContext() [3/3]

```
nvinfer1::IPluginResourceContext::IPluginResourceContext (
            IPluginResourceContext &&  )  [protected], [default]
```

9.123.3 Member Function Documentation

9.123.3.1 getErrorRecorder()

```
virtual IErrorRecorder * nvinfer1::IPluginResourceContext::getErrorRecorder ( ) const  [pure virtual],
[noexcept]
```

Get the error recorder associated with the resource context.

See also

IPluginV3OneRuntime::attachToContext()

9.123.3.2 getGpuAllocator()

```
virtual IGpuAllocator * nvinfer1::IPluginResourceContext::getGpuAllocator ( ) const  [pure virtual],
[noexcept]
```

Get the GPU allocator associated with the resource context.

See also

IPluginV3OneRuntime::attachToContext()

9.123.3.3 operator=() [1/2]

```
IPluginResourceContext & nvinfer1::IPluginResourceContext::operator= (
            IPluginResourceContext &&  ) &  [protected], [default]
```

9.123.3.4 operator=() [2/2]

```
IPluginResourceContext & nvinfer1::IPluginResourceContext::operator= (
            IPluginResourceContext const &  ) &  [protected], [default]
```

The documentation for this class was generated from the following file:

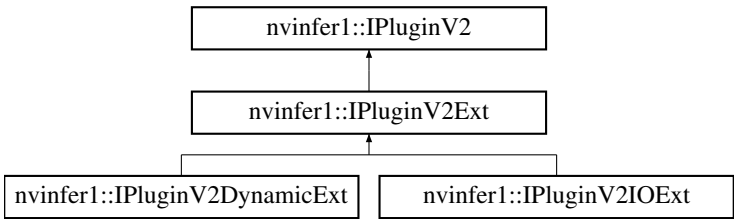
- NvInferRuntime.h

9.124 nvinfer1::IPluginV2 Class Reference

Plugin class for user-implemented layers.

```
#include <NvInferRuntimePlugin.h>
```

Inheritance diagram for nvinfer1::IPluginV2:



Public Member Functions

- virtual int32_t getTensorRTVersion () const noexcept

  Return the API version with which this plugin was built.
- virtual AsciiChar const * getPluginType () const noexcept=0

  Return the plugin type. Should match the plugin name returned by the corresponding plugin creator.
- virtual AsciiChar const * getPluginVersion () const noexcept=0

  Return the plugin version. Should match the plugin version returned by the corresponding plugin creator.
- virtual int32_t getNbOutputs () const noexcept=0

  Get the number of outputs from the layer.
- virtual Dims getOutputDimensions (int32_t index, Dims const *inputs, int32_t nbInputDims) noexcept=0

  Get the dimension of an output tensor.
- virtual bool supportsFormat (DataType type, PluginFormat format) const noexcept=0

  Check format support.
- virtual void configureWithFormat (Dims const *inputDims, int32_t nbInputs, Dims const *outputDims, int32_t nbOutputs, DataType type, PluginFormat format, int32_t maxBatchSize) noexcept=0

  Configure the layer.
- virtual int32_t initialize () noexcept=0

  Initialize the layer for execution. This is called when the engine is created.
- virtual void terminate () noexcept=0

Release resources acquired during plugin layer initialization. This is called when the engine is destroyed.

- virtual size_t getWorkspaceSize (int32_t maxBatchSize) const noexcept=0

  Find the workspace size required by the layer.

- virtual int32_t enqueue (int32_t batchSize, void const *const *inputs, void *const *outputs, void *workspace, cudaStream_t stream) noexcept=0

  Execute the layer.

- virtual size_t getSerializationSize () const noexcept=0

  Find the size of the serialization buffer required to store the plugin configuration in a binary file.

- virtual void serialize (void *buffer) const noexcept=0

  Serialize the layer.

- virtual void destroy () noexcept=0

  Destroy the plugin object. This will be called when the network, builder or engine is destroyed.

- virtual IPluginV2 * clone () const noexcept=0

  Clone the plugin object. This copies over internal plugin parameters and returns a new plugin object with these parameters.

- virtual void setPluginNamespace (AsciiChar const *pluginNamespace) noexcept=0

  Set the namespace that this plugin object belongs to. Ideally, all plugin objects from the same plugin library must have the same namespace.

- virtual AsciiChar const * getPluginNamespace () const noexcept=0

  Return the namespace of the plugin object.

9.124.1 Detailed Description

Plugin class for user-implemented layers.

Plugins are a mechanism for applications to implement custom layers. When combined with IPluginCreator it provides a mechanism to register plugins and look up the Plugin Registry during de-serialization.

See also

IPluginCreator

IPluginRegistry

Deprecated Deprecated in TensorRT 8.5. Implement IPluginV3 instead.

9.124.2 Member Function Documentation

9.124.2.1 clone()

```
virtual IPluginV2 * nvinfer1::IPluginV2::clone ( ) const  [pure virtual], [noexcept]
```

Clone the plugin object. This copies over internal plugin parameters and returns a new plugin object with these parameters.

The TensorRT runtime calls clone() to clone the plugin when an execution context is created for an engine, after the engine has been created. The runtime does not call initialize() on the cloned plugin, so the cloned plugin must be created in an initialized state.

Returns

A cloned plugin object in an initialized state with the same parameters as the current object. nullptr must be returned if the cloning fails, e.g. because of resource exhaustion.

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when creating multiple execution contexts.

Implemented in nvinfer1::IPluginV2DynamicExt, and nvinfer1::IPluginV2Ext.

9.124.2.2 configureWithFormat()

```
virtual void nvinfer1::IPluginV2::configureWithFormat (
             Dims const * inputDims,
             int32_t nbInputs,
             Dims const * outputDims,
             int32_t nbOutputs,
             DataType type,
             PluginFormat format,
             int32_t maxBatchSize )  [pure virtual], [noexcept]
```

Configure the layer.

This function is called by the builder prior to initialize(). It provides an opportunity for the layer to make algorithm choices on the basis of its weights, dimensions, and maximum batch size.

Parameters

inputDims	The input tensor dimensions. Will be the start address of a Dims array of length nbInputs.
nbInputs	The number of inputs. Will be a non-negative integer.
outputDims	The output tensor dimensions. Will be the start address of a Dims array of length nbOutputs.
nbOutputs	The number of outputs. Will be a positive integer identical to the return value of getNbOutputs().
type	The data type selected for the engine.
format	The format selected for the engine.
maxBatchSize	The maximum batch size. Will be a positive integer.

The dimensions passed here do not include the outermost batch size (i.e. for 2D image networks, they will be 3-dimensional CHW dimensions).

Warning

for the format field, the values PluginFormat::kCHW4, PluginFormat::kCHW16, and PluginFormat::k↩ CHW32 will not be passed in, this is to keep backward compatibility with TensorRT 5.x series. Use Plugin↩ V2IOExt or PluginV2DynamicExt for other PluginFormats.

DataType:kBOOL and DataType::kUINT8 are not supported.

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.

9.124.2.3 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.

9.124.2.4 enqueue()

```
virtual int32_t nvinfer1::IPluginV2::enqueue (
            int32_t batchSize,
            void const *const * inputs,
            void *const * outputs,
            void * workspace,
            cudaStream_t stream )  [pure virtual], [noexcept]
```

Execute the layer.

Parameters

batchSize	The number of inputs in the batch.
inputs	The memory for the input tensors. Will be an array of device addresses corresponding to input tensors of length nbInputs, where nbInputs is the second parameter passed to configureWithFormat(). The i-th input tensor will have the dimensions inputDims[i], where inputDims is the first parameter that was passed to configureWithFormat().
outputs	The memory for the output tensors. Will be an array of device addresses corresponding to output tensors of length getNbOutputs().
workspace	Workspace for execution. Will be the start address of a device buffer whose length will be at least getWorkspaceSize(batchSize).
stream	The stream in which to execute the kernels. This will be a valid CUDA stream.

Returns

0 for success, else non-zero (which will cause engine termination).

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.

9.124.2.5 getNbOutputs()

```
virtual int32_t nvinfer1::IPluginV2::getNbOutputs ( ) const  [pure virtual], [noexcept]
```

Get the number of outputs from the layer.

Returns

The number of outputs, which is a positive integer.

This function is called by the implementations of INetworkDefinition and IBuilder. In particular, it is called prior to any call to initialize().

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.124.2.6 getOutputDimensions()

```
virtual Dims nvinfer1::IPluginV2::getOutputDimensions (
            int32_t index,
            Dims const * inputs,
            int32_t nbInputDims )  [pure virtual], [noexcept]
```

Get the dimension of an output tensor.

Parameters

index	The index of the output tensor. Will lie in the valid range (between 0 and getNbOutputs()-1 inclusive).
inputs	The input tensor dimensions. Will be the start address of a Dims array of length nbInputDims.
nbInputDims	The number of input tensors. Will be a non-negative integer.

Returns

The output tensor dimensions if the index is in the valid range. An invalid value of Dims{-1, {}} must be returned if the index is not in the valid range.

This function is called by the implementations of INetworkDefinition and IBuilder. In particular, it is called prior to any call to initialize().

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

Note

In any non-IPluginV2DynamicExt plugin, batch size must not be included in the returned dimensions, even if the plugin is expected to be run in a network with explicit batch mode enabled. Please see the TensorRT Developer Guide for more details on how plugin inputs and outputs behave.

9.124.2.7 getPluginNamespace()

```
virtual AsciiChar const * nvinfer1::IPluginV2::getPluginNamespace ( ) const  [pure virtual], [noexcept]
```

Return the namespace of the plugin object.

Returns

The namespace string that was passed to setPluginNamespace(), possibly after truncation to 1024 bytes if a longer string was passed. An empty string must be returned as default value.

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.124.2.8 getPluginType()

```
virtual AsciiChar const * nvinfer1::IPluginV2::getPluginType ( ) const  [pure virtual], [noexcept]
```

Return the plugin type. Should match the plugin name returned by the corresponding plugin creator.

See also

IPluginCreator::getPluginName()

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.124.2.9 getPluginVersion()

```
virtual AsciiChar const * nvinfer1::IPluginV2::getPluginVersion ( ) const  [pure virtual], [noexcept]
```

Return the plugin version. Should match the plugin version returned by the corresponding plugin creator.

See also

IPluginCreator::getPluginVersion()

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.124.2.10 getSerializationSize()

```
virtual size_t nvinfer1::IPluginV2::getSerializationSize ( ) const  [pure virtual], [noexcept]
```

Find the size of the serialization buffer required to store the plugin configuration in a binary file.

Returns

The size of the serialization buffer in bytes.

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.124.2.11 getTensorRTVersion()

```
virtual int32_t nvinfer1::IPluginV2::getTensorRTVersion ( ) const  [inline], [virtual], [noexcept]
```

Return the API version with which this plugin was built.

Do not override this method as it is used by the TensorRT library to maintain backwards-compatibility with plugins.

Returns

The TensorRT version in the format (major $*$ 100 + minor) $*$ 100 + patch.

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.

9.124.2.12 getWorkspaceSize()

```
virtual size_t nvinfer1::IPluginV2::getWorkspaceSize (
            int32_t maxBatchSize ) const  [pure virtual], [noexcept]
```

Find the workspace size required by the layer.

This function is called during engine startup, after initialize(). The workspace size returned must be sufficient for any batch size up to the maximum.

Parameters

maxBatchSize	The maximum batch size, which will be a positive integer.

Returns

The workspace size in bytes, i.e. the device memory size that the plugin requires for its internal computations.

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.

9.124.2.13 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.

9.124.2.14 serialize()

```
virtual void nvinfer1::IPluginV2::serialize (
            void * buffer ) const  [pure virtual], [noexcept]
```

Serialize the layer.

Parameters

buffer	A pointer to a host buffer to serialize data. Size of buffer will be at least as large as the value returned by getSerializationSize.

See also

getSerializationSize()

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.

9.124.2.15 setPluginNamespace()

```
virtual void nvinfer1::IPluginV2::setPluginNamespace (
            AsciiChar const * pluginNamespace )  [pure virtual], [noexcept]
```

Set the namespace that this plugin object belongs to. Ideally, all plugin objects from the same plugin library must have the same namespace.

Parameters

pluginNamespace	The namespace for the plugin object.

Warning

The string pluginNamespace will be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

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.

9.124.2.16 supportsFormat()

```
virtual bool nvinfer1::IPluginV2::supportsFormat (
            DataType type,
            PluginFormat format ) const  [pure virtual], [noexcept]
```

Check format support.

Parameters

type	DataType requested.
format	PluginFormat requested.

Returns

true if the plugin supports the type-format combination.

This function is called by the implementations of INetworkDefinition, IBuilder, and safe::ICudaEngine/ICudaEngine. In particular, it is called when creating an engine and when deserializing an engine.

Warning

for the format field, the values PluginFormat::kCHW4, PluginFormat::kCHW16, and PluginFormat::kCHW32 will not be passed in, this is to keep backward compatibility with TensorRT 5.x series. Use PluginV2IOExt or PluginV2DynamicExt for other PluginFormats.

DataType:kBOOL and DataType::kUINT8 are not supported.

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.

9.124.2.17 terminate()

```
virtual void nvinfer1::IPluginV2::terminate ( )  [pure virtual], [noexcept]
```

Release resources acquired during plugin layer initialization. This is called when the engine is destroyed.

See also

initialize()

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin or when using multiple execution contexts using this plugin. However, TensorRT will not call this method from two threads simultaneously on a given clone of a plugin.

The documentation for this class was generated from the following file:

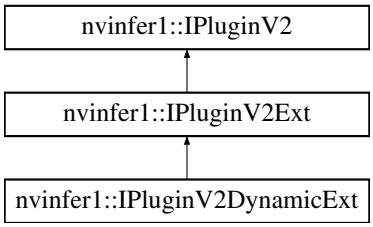
- NvInferRuntimePlugin.h

9.125 nvinfer1::IPluginV2DynamicExt Class Reference

Similar to IPluginV2Ext, but with support for dynamic shapes.

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::IPluginV2DynamicExt:



Public Member Functions

- IPluginV2DynamicExt * clone () const noexcept override=0

  Clone the plugin object. This copies over internal plugin parameters as well and returns a new plugin object with these parameters. If the source plugin is pre-configured with configurePlugin(), the returned object must also be pre-configured. The returned object must allow attachToContext() with a new execution context. Cloned plugin objects can share the same per-engine immutable resource (e.g. weights) with the source object (e.g. via ref-counting) to avoid duplication.
- virtual DimsExprs getOutputDimensions (int32_t outputIndex, DimsExprs const *inputs, int32_t nbInputs, IExprBuilder &exprBuilder) noexcept=0

  Get expressions for computing dimensions of an output tensor from dimensions of the input tensors.
- virtual bool supportsFormatCombination (int32_t pos, PluginTensorDesc const *inOut, int32_t nbInputs, int32_t nbOutputs) noexcept=0

Return true if plugin supports the format and datatype for the input/output indexed by pos.

- virtual void configurePlugin (DynamicPluginTensorDesc const ∗in, int32_t nbInputs, DynamicPluginTensorDesc const ∗out, int32_t nbOutputs) noexcept=0

  Configure the plugin.

- virtual size_t getWorkspaceSize (PluginTensorDesc const ∗inputs, int32_t nbInputs, PluginTensorDesc const ∗outputs, int32_t nbOutputs) const noexcept=0

  Find the workspace size required by the layer.

- virtual int32_t enqueue (PluginTensorDesc const ∗inputDesc, PluginTensorDesc const ∗outputDesc, void const ∗const ∗inputs, void ∗const ∗outputs, void ∗workspace, cudaStream_t stream) noexcept=0

  Execute the layer.

Static Public Attributes

- static constexpr int32_t kFORMAT_COMBINATION_LIMIT = 100

  Limit on number of format combinations accepted.

Protected Member Functions

- int32_t getTensorRTVersion () const noexcept override

  Return the API version with which this plugin was built. The upper byte reserved by TensorRT and is used to differentiate this from IPluginV2.

- virtual ∼IPluginV2DynamicExt () noexcept

9.125.1 Detailed Description

Similar to IPluginV2Ext, but with support for dynamic shapes.

Clients should override the public methods, including the following inherited methods:

- virtual int32_t getNbOutputs() const noexcept = 0;
- virtual DataType getOutputDataType(int32_t index, DataType const∗ inputTypes, int32_t nbInputs) const noexcept = 0;
- virtual size_t getSerializationSize() const noexcept = 0;
- virtual void serialize(void∗ buffer) const noexcept = 0;
- virtual void destroy() noexcept = 0;
- virtual void setPluginNamespace(char const∗ pluginNamespace) noexcept = 0;
- virtual char const∗ getPluginNamespace() const noexcept = 0;

For weakly typed networks, the inputTypes will always be DataType::kFLOAT or DataType::kINT32, and the returned type is canonicalized to DataType::kFLOAT if it is DataType::kHALF or DataType:kINT8. For strongly typed networks, inputTypes are inferred from previous operations, and getOutputDataType specifies the returned type based on the inputTypes. Details about the floating-point precision are elicited later by method supportsFormatCombination.

Deprecated Deprecated in TensorRT 10.0. Please implement IPluginV3 instead.

9.125.2 Constructor & Destructor Documentation

9.125.2.1 ~IPluginV2DynamicExt()

```
virtual nvinfer1::IPluginV2DynamicExt::~IPluginV2DynamicExt ( )  [inline], [protected], [virtual],
[noexcept]
```

9.125.3 Member Function Documentation

9.125.3.1 clone()

```
IPluginV2DynamicExt * nvinfer1::IPluginV2DynamicExt::clone ( ) const  [override], [pure virtual],
[noexcept]
```

Clone the plugin object. This copies over internal plugin parameters as well and returns a new plugin object with these parameters. If the source plugin is pre-configured with configurePlugin(), the returned object must also be pre-configured. The returned object must allow attachToContext() with a new execution context. Cloned plugin objects can share the same per-engine immutable resource (e.g. weights) with the source object (e.g. via ref-counting) to avoid duplication.

Returns

A pointer to a cloned plugin object if cloning was successful, otherwise nullptr.

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

Implements nvinfer1::IPluginV2Ext.

9.125.3.2 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 setOptimizationProfileAsync().
  - An input execution binding is changed via setInputShape().

> Warning
>
> The execution phase is timing critical during IExecutionContext but is not part of the timing loop when called from IBuilder. Performance bottlenecks of configurePlugin won't show up during engine building but will be visible during execution after calling functions that trigger layer resource updates.

Parameters

in	The input tensors attributes that are used for configuration.
nbInputs	Number of input tensors.
out	The output tensors attributes that are used for configuration.
nbOutputs	Number of output tensors.

9.125.3.3 enqueue()

```
virtual int32_t nvinfer1::IPluginV2DynamicExt::enqueue (
            PluginTensorDesc const * inputDesc,
            PluginTensorDesc const * outputDesc,
            void const *const * inputs,
            void *const * outputs,
            void * workspace,
            cudaStream_t stream )  [pure virtual], [noexcept]
```

Execute the layer.

Parameters

inputDesc	how to interpret the memory for the input tensors.
outputDesc	how to interpret the memory for the output tensors.
inputs	The memory for the input tensors.
outputs	The memory for the output tensors.
workspace	Workspace for execution.
stream	The stream in which to execute the kernels.

Returns

0 for success, else non-zero (which will cause engine termination).

9.125.3.4 getOutputDimensions()

```
virtual DimsExprs nvinfer1::IPluginV2DynamicExt::getOutputDimensions (
            int32_t outputIndex,
            DimsExprs const * inputs,
            int32_t nbInputs,
            IExprBuilder & exprBuilder )  [pure virtual], [noexcept]
```

Get expressions for computing dimensions of an output tensor from dimensions of the input tensors.

Parameters

outputIndex	The index of the output tensor
inputs	Expressions for dimensions of the input tensors
nbInputs	The number of input tensors
exprBuilder	Object for generating new expressions

This function is called by the implementations of IBuilder during analysis of the network.

Example #1: A plugin has a single output that transposes the last two dimensions of the plugin's single input. The body of the override of getOutputDimensions can be:

```
DimsExprs output(inputs[0]);
std::swap(output.d[output.nbDims-1], output.d[output.nbDims-2]);
return output;
```

Example #2: A plugin concatenates its two inputs along the first dimension. The body of the override of getOutput↩ Dimensions can be:

```
DimsExprs output(inputs[0]);
output.d[0] = exprBuilder.operation(DimensionOperation::kSUM, *inputs[0].d[0], *inputs[1].d[0]);
return output;
```

9.125.3.5 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.

9.125.3.6 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.

9.125.3.7 supportsFormatCombination()

```
virtual bool nvinfer1::IPluginV2DynamicExt::supportsFormatCombination (
            int32_t pos,
            PluginTensorDesc const * inOut,
            int32_t nbInputs,
            int32_t nbOutputs )  [pure virtual], [noexcept]
```

Return true if plugin supports the format and datatype for the input/output indexed by pos.

For this method inputs are numbered 0..(nbInputs-1) and outputs are numbered nbInputs..(nbInputs+nbOutputs-1). Using this numbering, pos is an index into InOut, where 0 <= pos < nbInputs+nbOutputs.

TensorRT invokes this method to ask if the input/output indexed by pos supports the format/datatype specified by inOut[pos].format and inOut[pos].type. The override should return true if that format/datatype at inOut[pos] are supported by the plugin. If support is conditional on other input/output formats/datatypes, the plugin can make its result conditional on the formats/datatypes in inOut[0..pos-1], which will be set to values that the plugin supports. The override should not inspect inOut[pos+1..nbInputs+nbOutputs-1], which will have invalid values. In other words, the decision for pos must be based on inOut[0..pos] only.

Some examples:

- A definition for a plugin that supports only FP16 NCHW:

  ```
  return inOut[pos].format == TensorFormat::kLINEAR && inOut[pos].type == DataType::kHALF;
  ```

- A definition for a plugin that supports only FP16 NCHW for its two inputs, and FP32 NCHW for its single output:

  ```
  return inOut[pos].format == TensorFormat::kLINEAR && (inOut[pos].type == (pos < 2 ? DataType::kHALF :
  DataType::kFLOAT));
  ```

- A definition for a "polymorphic" plugin with two inputs and one output that supports any format or type, but the inputs and output must have the same format and type:

  ```
  return pos == 0 || (inOut[pos].format == inOut.format[0] && inOut[pos].type == inOut[0].type);
  ```

  Warning: TensorRT will stop asking for formats once it finds kFORMAT_COMBINATION_LIMIT on combinations.

9.125.4 Member Data Documentation

9.125.4.1 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:

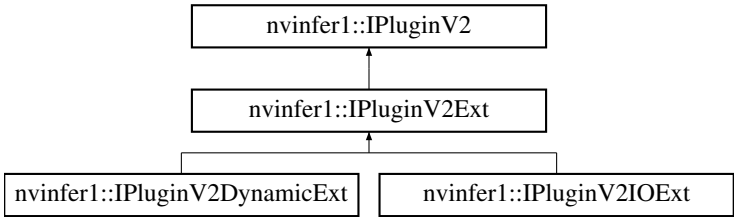
- NvInferRuntime.h

9.126 nvinfer1::IPluginV2Ext Class Reference

Plugin class for user-implemented layers.

`#include <NvInferRuntimePlugin.h>`

Inheritance diagram for nvinfer1::IPluginV2Ext:



Public Member Functions

- virtual nvinfer1::DataType getOutputDataType (int32_t index, nvinfer1::DataType const *inputTypes, int32_t nbInputs) const noexcept=0

  Return the DataType of the plugin output at the requested index.
- virtual TRT_DEPRECATED bool isOutputBroadcastAcrossBatch (int32_t outputIndex, bool const *inputIsBroadcasted, int32_t nbInputs) const noexcept=0

  Return true if the output tensor is broadcast across a batch.
- virtual TRT_DEPRECATED bool canBroadcastInputAcrossBatch (int32_t inputIndex) const noexcept=0

  Return true if the plugin can use an input tensor that is broadcast across batch without replication.
- virtual void configurePlugin (Dims const *inputDims, int32_t nbInputs, Dims const *outputDims, int32_t nbOutputs, DataType const *inputTypes, DataType const *outputTypes, bool const *inputIsBroadcast, bool const *outputIsBroadcast, PluginFormat floatFormat, int32_t maxBatchSize) noexcept=0

  Configure the layer with input and output data types.
- IPluginV2Ext ()=default
- ~IPluginV2Ext () override=default
- virtual void attachToContext (cudnnContext *, cublasContext *, IGpuAllocator *) noexcept

  Attach the plugin object to an execution context and grant the plugin the access to some context resources.
- virtual void detachFromContext () noexcept

  Detach the plugin object from its execution context.
- IPluginV2Ext * clone () const noexcept override=0

  Clone the plugin object. This copies over internal plugin parameters as well and returns a new plugin object with these parameters. If the source plugin is pre-configured with configurePlugin(), the returned object must also be pre-configured. The returned object must allow attachToContext() with a new execution context. Cloned plugin objects can share the same per-engine immutable resource (e.g. weights) with the source object (e.g. via ref-counting) to avoid duplication.

Protected Member Functions

- int32_t getTensorRTVersion () const noexcept override

  Return the API version with which this plugin was built. The upper byte reserved by TensorRT and is used to differentiate this from IPluginV2.
- void configureWithFormat (Dims const *, int32_t, Dims const *, int32_t, DataType, PluginFormat, int32_t) noexcept override

  Derived classes must not implement this. In a C++11 API it would be override final.

9.126.1 Detailed Description

Plugin class for user-implemented layers.

Plugins are a mechanism for applications to implement custom layers. This interface provides additional capabilities to the IPluginV2 interface by supporting different output data types and broadcast across batches.

See also

IPluginV2

Deprecated Deprecated in TensorRT 8.5. Implement IPluginV3 instead.

9.126.2 Constructor & Destructor Documentation

9.126.2.1 IPluginV2Ext()

```
nvinfer1::IPluginV2Ext::IPluginV2Ext ( )  [default]
```

9.126.2.2 ~IPluginV2Ext()

```
nvinfer1::IPluginV2Ext::~IPluginV2Ext ( )  [override], [default]
```

9.126.3 Member Function Documentation

9.126.3.1 attachToContext()

```
virtual void nvinfer1::IPluginV2Ext::attachToContext (
            cudnnContext * ,
            cublasContext * ,
            IGpuAllocator *  )  [inline], [virtual], [noexcept]
```

Attach the plugin object to an execution context and grant the plugin the access to some context resources.

Parameters

cudnn	The cuDNN context handle of the execution context. Will be a valid cuDNN context handle, or nullptr if TacticSource::kCUDNN is disabled.
cublas	The cuBLAS context handle of the execution context. Will be a valid cuBLAS context handle, or nullptr if TacticSource::kCUBLAS is disabled.
allocator	The allocator used by the execution context

This function is called automatically for each plugin when a new execution context is created. If the context was created without resources, this method is not called until the resources are assigned. It is also called if new resources are assigned to the context.

If the plugin needs per-context resource, it can be allocated here. The plugin can also get context-owned cuDNN and cuBLAS context here.

Note

The TacticSource::kCUDNN and TacticSource::kCUBLAS flag is disabled by default. The allocator pointer is unique to each building or execution context instance having overlapping lifetimes. It can be used as a key to manage resources across plugin instances sharing the same context. Plugins attached to different contexts will have different handles as their execution will not overlap.

See also

TacticSources

getPluginCudnnHandle(void* executionContextIdentifier)

getPluginCublasHandle(void* excecutionContextIdentifier)

Note

In the automotive safety context, the cuDNN and cuBLAS parameters will be nullptr because cuDNN and cuBLAS are not used by the safe runtime.

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.

9.126.3.2 canBroadcastInputAcrossBatch()

```
virtual TRT_DEPRECATED bool nvinfer1::IPluginV2Ext::canBroadcastInputAcrossBatch (
            int32_t inputIndex ) const  [pure virtual], [noexcept]
```

Return true if the plugin can use an input tensor that is broadcast across batch without replication.

Parameters

inputIndex	Index of input that could be broadcast. Will be in the valid range between 0 and nbInputs - 1 where nbInputs is the maximum number of input tensors supported by this plugin.

Returns

true if the index is in the valid range and the plugin is able to broadcast a single copy of this input tensor across the batch. False otherwise.

For each input whose tensor is semantically broadcast across a batch, TensorRT calls this method before calling configurePlugin. If canBroadcastInputAcrossBatch returns true, TensorRT will not replicate the input tensor; i.e., there will be a single copy that the plugin must share across the batch. If it returns false, TensorRT will replicate the input tensor so that it appears like a non-broadcasted tensor.

This method is called only for inputs that can be broadcast.

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

Deprecated Deprecated in TensorRT 10.0. Implicit batch support is removed in TensorRT 10.0.

9.126.3.3 clone()

```
IPluginV2Ext * nvinfer1::IPluginV2Ext::clone ( ) const  [override], [pure virtual], [noexcept]
```

Clone the plugin object. This copies over internal plugin parameters as well and returns a new plugin object with these parameters. If the source plugin is pre-configured with configurePlugin(), the returned object must also be pre-configured. The returned object must allow attachToContext() with a new execution context. Cloned plugin objects can share the same per-engine immutable resource (e.g. weights) with the source object (e.g. via ref-counting) to avoid duplication.

Returns

A pointer to a cloned plugin object if cloning was successful, otherwise nullptr.

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

Implements nvinfer1::IPluginV2.

Implemented in nvinfer1::IPluginV2DynamicExt.

9.126.3.4 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

inputDims	The input tensor dimensions. Will be an array of length nbInputs.
nbInputs	The number of inputs. Will be a non-negative integer.
outputDims	The output tensor dimensions. Will be an array of length nbOutputs.
nbOutputs	The number of outputs. Will be a positive integer.
inputTypes	The data types selected for the plugin inputs. Will be an array of length nbInputs.
outputTypes	The data types selected for the plugin outputs. Will be an array of length nbOutputs.
inputIsBroadcast	True for each input that the plugin must broadcast across the batch. Will be an array of length nbInputs.
outputIsBroadcast	True for each output that TensorRT will broadcast across the batch. Will be an array of length nbOutputs.
floatFormat	The format selected for the engine for the floating point inputs/outputs.
maxBatchSize	The maximum batch size. Will be a positive integer.

The dimensions passed here do not include the outermost batch size (i.e. for 2D image networks, they will be 3-dimensional CHW dimensions). When inputIsBroadcast or outputIsBroadcast is true, the outermost batch size for that input or output must be treated as if it is one. Index 'i' of inputIsBroadcast is true only if the input is semantically broadcast across the batch and calling canBroadcastInputAcrossBatch with argument 'i' returns true. Index 'i' of outputIsBroadcast is true only if calling isOutputBroadcastAcrossBatch with argument 'i' returns true.

Warning

for the floatFormat field, the values PluginFormat::kCHW4, PluginFormat::kCHW16, and PluginFormat↩ ::kCHW32 will not be passed in, this is to keep backward compatibility with TensorRT 5.x series. Use PluginV2IOExt or PluginV2DynamicExt for other PluginFormats.

Usage considerations

- Allowed context for the API call

- Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin. However, TensorRT will not call this method from two threads simultaneously on a given clone of a plugin.

9.126.3.5 configureWithFormat()

```
void nvinfer1::IPluginV2Ext::configureWithFormat (
            Dims const * ,
            int32_t ,
            Dims const * ,
            int32_t ,
            DataType ,
            PluginFormat ,
            int32_t  )  [inline], [override], [protected], [virtual], [noexcept]
```

Derived classes must not implement this. In a C++11 API it would be override final.

IPluginV2Ext::configureWithFormat() is a NOP operation for all classes derived from IPluginV2Ext. These classes call configurePlugin() instead.

Implements nvinfer1::IPluginV2.

9.126.3.6 detachFromContext()

```
virtual void nvinfer1::IPluginV2Ext::detachFromContext ( )  [inline], [virtual], [noexcept]
```

Detach the plugin object from its execution context.

This function is called automatically for each plugin when an execution context is destroyed or the context resources are unassigned from the context.

If the plugin owns per-context resource, it can be released here.

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

9.126.3.7 getOutputDataType()

```
virtual nvinfer1::DataType nvinfer1::IPluginV2Ext::getOutputDataType (
            int32_t index,
            nvinfer1::DataType const * inputTypes,
            int32_t nbInputs ) const  [pure virtual], [noexcept]
```

Return the DataType of the plugin output at the requested index.

Parameters

index	The output tensor index in the valid range between 0 and getNbOutputs()-1.
inputTypes	The data types of the input tensors, stored in an array of length nbInputs.
nbInputs	The number of input tensors. Will be a non-negative integer.

Returns

The data type of the output tensor with the provided index if the input tensors have the data types provided in inputTypes, provided the output tensor index is in the valid range. DataType::kFLOAT must be returned if the index is not in the valid range.

The default behavior must be to return the type of the first input, or DataType::kFLOAT if the layer has no inputs. The returned data type must have a format that is supported by the plugin.

See also

supportsFormat()

Warning

DataType:kBOOL and DataType::kUINT8 are not supported.

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.

9.126.3.8 getTensorRTVersion()

```
int32_t nvinfer1::IPluginV2Ext::getTensorRTVersion ( ) const  [inline], [override], [protected], [virtual], [noexcept]
```

Return the API version with which this plugin was built. The upper byte reserved by TensorRT and is used to differentiate this from IPluginV2.

Returns

In the lower three bytes, the TensorRT version in the format (major $*$ 100 + minor) $*$ 100 + patch. In the upper byte, the value 1.

Do not override this method as it is used by the TensorRT library to maintain backwards-compatibility with plugins.

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.

9.126.3.9 isOutputBroadcastAcrossBatch()

```
virtual TRT_DEPRECATED bool nvinfer1::IPluginV2Ext::isOutputBroadcastAcrossBatch (
            int32_t outputIndex,
            bool const * inputIsBroadcasted,
            int32_t nbInputs ) const  [pure virtual], [noexcept]
```

Return true if the output tensor is broadcast across a batch.

Parameters

outputIndex	The index of the output tensor, which will be in the valid range between 0 and nbOutputs()-1.
inputIsBroadcasted	A boolean array of length nbInputs. The i-th element will be true if and only if the tensor for the ith input is broadcast across a batch.
nbInputs	The number of inputs. Will be a non-negative integer.

The values in inputIsBroadcasted refer to broadcasting at the semantic level, i.e. are unaffected by whether method canBroadcastInputAcrossBatch requests physical replication of the values.

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

Deprecated Deprecated in TensorRT 10.0. Implicit batch support is removed in TensorRT 10.0.

The documentation for this class was generated from the following file:

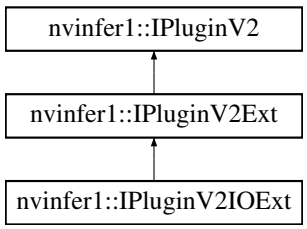
- NvInferRuntimePlugin.h

9.127 nvinfer1::IPluginV2IOExt Class Reference

Plugin class for user-implemented layers.

```
#include <NvInferRuntimePlugin.h>
```

Inheritance diagram for nvinfer1::IPluginV2IOExt:



Public Member Functions

- virtual void configurePlugin (PluginTensorDesc const *in, int32_t nbInput, PluginTensorDesc const *out, int32_t nbOutput) noexcept=0

  Configure the layer.
- virtual bool supportsFormatCombination (int32_t pos, PluginTensorDesc const *inOut, int32_t nbInputs, int32_t nbOutputs) const noexcept=0

  Return true if plugin supports the format and datatype for the input/output indexed by pos.

Protected Member Functions

- int32_t getTensorRTVersion () const noexcept override

  Return the API version with which this plugin was built. The upper byte is reserved by TensorRT and is used to differentiate this from IPluginV2 and IPluginV2Ext.

9.127.1 Detailed Description

Plugin class for user-implemented layers.

Plugins are a mechanism for applications to implement custom layers. This interface provides additional capabilities to the IPluginV2Ext interface by extending different I/O data types and tensor formats.

See also

IPluginV2Ext

Deprecated Deprecated in TensorRT 10.0. Implement IPluginV3 instead.

9.127.2 Member Function Documentation

9.127.2.1 configurePlugin()

```
virtual void nvinfer1::IPluginV2IOExt::configurePlugin (
            PluginTensorDesc const * in,
            int32_t nbInput,
            PluginTensorDesc const * out,
            int32_t nbOutput )  [pure virtual], [noexcept]
```

Configure the layer.

This function is called by the builder prior to initialize(). It provides an opportunity for the layer to make algorithm choices on the basis of the provided I/O PluginTensorDesc.

Parameters

in	The input tensors attributes that are used for configuration.
nbInput	Number of input tensors.
out	The output tensors attributes that are used for configuration.
nbOutput	Number of output tensors.

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.

9.127.2.2 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.

9.127.2.3 supportsFormatCombination()

```
virtual bool nvinfer1::IPluginV2IOExt::supportsFormatCombination (
            int32_t pos,
            PluginTensorDesc const * inOut,
            int32_t nbInputs,
            int32_t nbOutputs ) const  [pure virtual], [noexcept]
```

Return true if plugin supports the format and datatype for the input/output indexed by pos.

For this method inputs are numbered 0..(nbInputs-1) and outputs are numbered nbInputs..(nbInputs+nbOutputs-1). Using this numbering, pos is an index into InOut, where 0 <= pos < nbInputs+nbOutputs.

TensorRT invokes this method to ask if the input/output indexed by pos supports the format/datatype specified by in↩ Out[pos].format and inOut[pos].type. The override must return true if that format/datatype at inOut[pos] are supported by the plugin. If support is conditional on other input/output formats/datatypes, the plugin can make its result conditional on the formats/datatypes in inOut[0..pos-1], which will be set to values that the plugin supports. The override must not inspect inOut[pos+1..nbInputs+nbOutputs-1], which will have invalid values. In other words, the decision for pos must be based on inOut[0..pos] only.

Some examples:

- A definition for a plugin that supports only FP16 NCHW:

  ```
  return inOut.format[pos] == TensorFormat::kLINEAR && inOut.type[pos] == DataType::kHALF;
  ```

- A definition for a plugin that supports only FP16 NCHW for its two inputs, and FP32 NCHW for its single output:

  ```
  return inOut.format[pos] == TensorFormat::kLINEAR &&
         (inOut.type[pos] == (pos < 2 ?  DataType::kHALF : DataType::kFLOAT));
  ```

- A definition for a ”polymorphic” plugin with two inputs and one output that supports any format or type, but the inputs and output must have the same format and type:

  ```
  return pos == 0 || (inOut.format[pos] == inOut.format[0] && inOut.type[pos] == inOut.type[0]);
  ```

  Warning: TensorRT will stop asking for formats once it finds kFORMAT_COMBINATION_LIMIT on combinations.

Usage considerations

- Allowed context for the API call
  - Thread-safe: Yes, this method is required to be thread-safe and may be called from multiple threads when building networks on multiple devices sharing the same plugin.

The documentation for this class was generated from the following file:

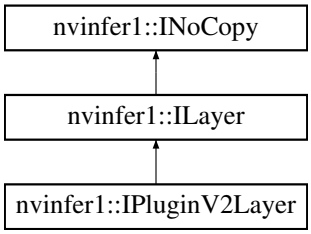
- NvInferRuntimePlugin.h

9.128 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

9.128.1 Detailed Description

Layer type for pluginV2.

See also

IPluginV2

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

Deprecated Deprecated in TensorRT 10.8. Superseded by IPluginV3Layer.

9.128.2 Constructor & Destructor Documentation

9.128.2.1 ~IPluginV2Layer()

```
virtual nvinfer1::IPluginV2Layer::~IPluginV2Layer ( )  [protected], [virtual], [default], [noexcept]
```

9.128.3 Member Function Documentation

9.128.3.1 getPlugin()

```
IPluginV2 & nvinfer1::IPluginV2Layer::getPlugin ( )  [inline], [noexcept]
```

Get the plugin for the layer.

See also

IPluginV2

9.128.4 Member Data Documentation

9.128.4.1 mImpl

```
apiv::VPluginV2Layer* nvinfer1::IPluginV2Layer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.129 IPluginV3 Class Reference

Plugin class for the V3 generation of user-implemented layers.

```
#include <NvInferPluginBase.h>
```

9.129.1 Detailed Description

Plugin class for the V3 generation of user-implemented layers.

IPluginV3 acts as a wrapper around the plugin capability interfaces that define the actual behavior of the plugin.

See also

IPluginCapability

IPluginCreatorV3One

IPluginRegistry

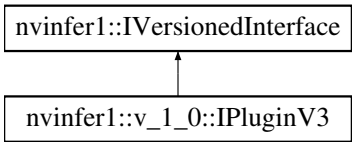
The documentation for this class was generated from the following file:

- NvInferPluginBase.h

9.130 nvinfer1::v_1_0::IPluginV3 Class Reference

`#include <NvInferPluginBase.h>`

Inheritance diagram for nvinfer1::v_1_0::IPluginV3:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual IPluginCapability * getCapabilityInterface (PluginCapabilityType type) noexcept=0

  Return a pointer to plugin object implementing the specified PluginCapabilityType.
- virtual IPluginV3 * clone () noexcept=0

  Clone the plugin object. This copies over internal plugin parameters and returns a new plugin object with these parameters. The cloned object must be in a fully initialized state.

Additional Inherited Members

9.130.1 Member Function Documentation

9.130.1.1 clone()

```
virtual IPluginV3 * nvinfer1::v_1_0::IPluginV3::clone ( )  [pure virtual], [noexcept]
```

Clone the plugin object. This copies over internal plugin parameters and returns a new plugin object with these parameters. The cloned object must be in a fully initialized state.

Note

The cloned object must return valid objects through getCapabilityInterface() for at least the same PluginCapabilityTypes as the original object.

Returns

A cloned plugin object in an initialized state with the same parameters as the current object. nullptr must be returned if the cloning fails.

9.130.1.2 getCapabilityInterface()

```
virtual IPluginCapability * nvinfer1::v_1_0::IPluginV3::getCapabilityInterface (
            PluginCapabilityType type )  [pure virtual], [noexcept]
```

Return a pointer to plugin object implementing the specified PluginCapabilityType.

Note

IPluginV3 objects added for the build phase (through addPluginV3()) must return valid objects for PluginCapabilityType::kCORE, PluginCapabilityType::kBUILD and PluginCapabilityType::kRUNTIME.

IPluginV3 objects added for the runtime phase must return valid objects for PluginCapabilityType::kCORE and PluginCapabilityType::kRUNTIME.

See also

TensorRTPhase

IPluginCreatorV3One::createPlugin()

9.130.1.3 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IPluginV3::getInterfaceInfo ( ) const  [inline], [override], [virtual],
[noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

The documentation for this class was generated from the following file:

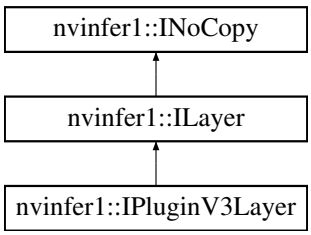
- NvInferPluginBase.h

9.131 nvinfer1::IPluginV3Layer Class Reference

Layer type for V3 plugins.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IPluginV3Layer:



Public Member Functions

- IPluginV3 & getPlugin () noexcept
  Get the plugin for the layer.

Protected Member Functions

- virtual ~IPluginV3Layer () noexcept=default

Protected Attributes

- apiv::VPluginV3Layer * mImpl

9.131.1 Detailed Description

Layer type for V3 plugins.

See also

IPluginV3

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.131.2 Constructor & Destructor Documentation

9.131.2.1 ~IPluginV3Layer()

```
virtual nvinfer1::IPluginV3Layer::~IPluginV3Layer ( )  [protected], [virtual], [default], [noexcept]
```

9.131.3 Member Function Documentation

9.131.3.1 getPlugin()

```
IPluginV3 & nvinfer1::IPluginV3Layer::getPlugin ( )  [inline], [noexcept]
```

Get the plugin for the layer.

See also

IPluginV3

9.131.4 Member Data Documentation

9.131.4.1 mImpl

```
apiv::VPluginV3Layer* nvinfer1::IPluginV3Layer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.132 IPluginV3OneBuild Class Reference

A plugin capability interface that enables the build capability (PluginCapabilityType::kBUILD). Exposes methods that allow the expression of the build time properties and behavior of a plugin.

```
#include <NvInferRuntime.h>
```

9.132.1 Detailed Description

A plugin capability interface that enables the build capability (PluginCapabilityType::kBUILD). Exposes methods that allow the expression of the build time properties and behavior of a plugin.

See also

IPluginCapability

PluginCapabilityType

IPluginV3::getCapabilityInterface()

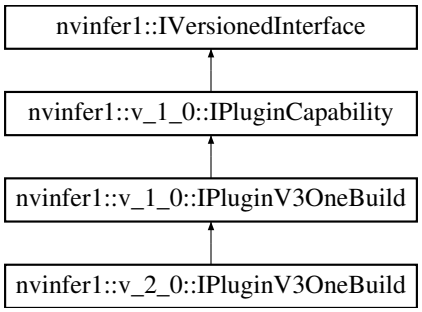
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.133 nvinfer1::v_1_0::IPluginV3OneBuild Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IPluginV3OneBuild:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual int32_t configurePlugin (DynamicPluginTensorDesc const ∗in, int32_t nbInputs, DynamicPluginTensorDesc const ∗out, int32_t nbOutputs) noexcept=0

  Configure the plugin.
- virtual int32_t getOutputDataTypes (DataType ∗outputTypes, int32_t nbOutputs, const DataType ∗inputTypes, int32_t nbInputs) const noexcept=0

  Provide the data types of the plugin outputs if the input tensors have the data types provided.
- virtual int32_t getOutputShapes (DimsExprs const ∗inputs, int32_t nbInputs, DimsExprs const ∗shapeInputs, int32_t nbShapeInputs, DimsExprs ∗outputs, int32_t nbOutputs, IExprBuilder &exprBuilder) noexcept=0

  Provide expressions for computing dimensions of the output tensors from dimensions of the input tensors.

- virtual bool supportsFormatCombination (int32_t pos, DynamicPluginTensorDesc const *inOut, int32_t nbInputs, int32_t nbOutputs) noexcept=0

  Return true if plugin supports the format and datatype for the input/output indexed by pos.
- virtual int32_t getNbOutputs () const noexcept=0

  Get the number of outputs from the plugin.
- virtual size_t getWorkspaceSize (DynamicPluginTensorDesc const *inputs, int32_t nbInputs, DynamicPluginTensorDesc const *outputs, int32_t nbOutputs) const noexcept

  Find the workspace size required by the layer.
- virtual int32_t getValidTactics (int32_t *tactics, int32_t nbTactics) noexcept

  Query for any custom tactics that the plugin intends to use.
- virtual int32_t getNbTactics () noexcept

  Query for the number of custom tactics the plugin intends to use.
- virtual char const * getTimingCacheID () noexcept

  Called to query the suffix to use for the timing cache ID. May be called anytime after plugin creation.
- virtual int32_t getFormatCombinationLimit () noexcept

  Return the maximum number of format combinations that will be timed by TensorRT during the build phase.
- virtual char const * getMetadataString () noexcept

  Query for a string representing the configuration of the plugin. May be called anytime after plugin creation.

Static Public Attributes

- static constexpr int32_t kDEFAULT_FORMAT_COMBINATION_LIMIT = 100

  The default maximum number of format combinations that will be timed by TensorRT during the build phase.

Additional Inherited Members

9.133.1 Member Function Documentation

9.133.1.1 configurePlugin()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneBuild::configurePlugin (
            DynamicPluginTensorDesc const * in,
            int32_t nbInputs,
            DynamicPluginTensorDesc const * out,
            int32_t nbOutputs )  [pure virtual], [noexcept]
```

Configure the plugin.

configurePlugin() can be called multiple times in the build phase during creation of an engine by IBuilder.

configurePlugin() is called when a plugin is being prepared for profiling but not for any specific input size. This provides an opportunity for the plugin to make algorithmic choices on the basis of input and output formats, along with the bound of possible dimensions. The min, opt and max value of the DynamicPluginTensorDesc correspond to the kMIN, kOPT and kMAX value of the current profile that the plugin is being profiled for, with the desc.dims field corresponding to the dimensions of plugin specified at network creation. Wildcard dimensions may exist during this phase in the desc.dims field.

Parameters

in	The input tensors attributes that are used for configuration.
nbInputs	Number of input tensors.
out	The output tensors attributes that are used for configuration.
nbOutputs	Number of output tensors.

Returns

0 for success, else non-zero (which will cause engine termination, if invoked by TensorRT).

9.133.1.2 getFormatCombinationLimit()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneBuild::getFormatCombinationLimit ( )  [inline], [virtual],
[noexcept]
```

Return the maximum number of format combinations that will be timed by TensorRT during the build phase.

9.133.1.3 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IPluginV3OneBuild::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

Reimplemented in nvinfer1::v_2_0::IPluginV3OneBuild.

9.133.1.4 getMetadataString()

```
virtual char const * nvinfer1::v_1_0::IPluginV3OneBuild::getMetadataString ( )  [inline], [virtual],
[noexcept]
```

Query for a string representing the configuration of the plugin. May be called anytime after plugin creation.

Returns

A string representing the plugin's creation state, especially with regard to its attribute values.

9.133.1.5 getNbOutputs()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneBuild::getNbOutputs ( ) const  [pure virtual], [noexcept]
```

Get the number of outputs from the plugin.

Returns

The number of outputs, which must be a positive integer.

9.133.1.6 getNbTactics()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneBuild::getNbTactics ( )  [inline], [virtual], [noexcept]
```

Query for the number of custom tactics the plugin intends to use.

9.133.1.7 getOutputDataTypes()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneBuild::getOutputDataTypes (
            DataType * outputTypes,
            int32_t nbOutputs,
            const DataType * inputTypes,
            int32_t nbInputs ) const  [pure virtual], [noexcept]
```

Provide the data types of the plugin outputs if the input tensors have the data types provided.

Parameters

outputTypes	Pre-allocated array to which the output data types should be written.
nbOutputs	The number of output tensors. This matches the value returned from getNbOutputs().
inputTypes	The input data types.
nbInputs	The number of input tensors.

Returns

0 for success, else non-zero (which will cause engine termination). The returned code will be reported through the error recorder.

Note

Provide DataType::kFLOATs if the layer has no inputs. The data type for any size tensor outputs must be DataType::kINT32. The returned data types must each have a format that is supported by the plugin.

Warning

DataType:kBOOL and DataType::kUINT8 are not supported.

9.133.1.8 getOutputShapes()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneBuild::getOutputShapes (
            DimsExprs const * inputs,
            int32_t nbInputs,
            DimsExprs const * shapeInputs,
            int32_t nbShapeInputs,
            DimsExprs * outputs,
            int32_t nbOutputs,
            IExprBuilder & exprBuilder ) [pure virtual], [noexcept]
```

Provide expressions for computing dimensions of the output tensors from dimensions of the input tensors.

Parameters

inputs	Expressions for dimensions of the input tensors
nbInputs	The number of input tensors
shapeInputs	Expressions for values of the shape tensor inputs
nbShapeInputs	The number of shape tensor inputs
outputs	Pre-allocated array to which the output dimensions must be written
nbOutputs	Number of outputs.
exprBuilder	Object for generating new dimension expressions

Note

Any size tensor outputs must be declared to be 0D.

The declaration of shapeInputs as DimsExprs is slightly abusive, because the ”dimensions” are actually the values of the shape tensor. For example, if the input shape tensor is a 2x3 matrix, the DimsExprs will have six ”dimensions”: the three values from the first row of the matrix followed by the three values from the second row of the matrix.

Returns

0 for success, else non-zero (which will cause engine termination). Returned code will be reported through the error recorder.

9.133.1.9 getTimingCacheID()

```
virtual char const * nvinfer1::v_1_0::IPluginV3OneBuild::getTimingCacheID ( )  [inline], [virtual],
[noexcept]
```

Called to query the suffix to use for the timing cache ID. May be called anytime after plugin creation.

Returns

Suffix to use for timing cache ID, considering only the creation state of the plugin. Returning nullptr will disable timing caching for the plugin altogether.

Note

If timing caching is enabled for the plugin (by returning non-null), the I/O shape and format information will be automatically considered to form the prefix of the timing cache ID. Therefore, only other factors determining the creation state of the plugin, such as its attribute values, should be considered to compose the return value.

9.133.1.10 getValidTactics()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneBuild::getValidTactics (
            int32_t * tactics,
            int32_t nbTactics )  [inline], [virtual], [noexcept]
```

Query for any custom tactics that the plugin intends to use.

This method queries for the set of tactics T(f) supported by the plugin for the format combination f indicated by the immediately preceding call to configurePlugin(). It is guaranteed to be called after configurePlugin().

For each format combination provided through configurePlugin(), up to a maximum of getFormatCombinationLimit(), the plugin will be timed for each tactic advertised through this method for that format combination. i.e. The plugin will be timed $N = \sum_{i=0}^{i<getFormatCombinationLimit()}(T(f[i]))$ times. If $N = 1$, the plugin may not be timed. In peudocode, the timing protocol appears as the following:

counter = 0 for each supported format combination ++counter if counter $>$ getFormatCombinationLimit() goto done configurePlugin(...) for each tactic in getValidTactics(...) time tactic done:

Parameters

tactics	Pre-allocated buffer to which the tactic values should be written
nbTactics	The number of tactics advertised through getNbTactics()

Note

The provided tactic values must be unique and non-zero. The tactic value 0 is reserved for the default tactic attached to each format combination.

Returns

0 for success, else non-zero (which will cause engine termination). The returned code will be reported through the error recorder.

9.133.1.11 getWorkspaceSize()

```
virtual size_t nvinfer1::v_1_0::IPluginV3OneBuild::getWorkspaceSize (
            DynamicPluginTensorDesc const * inputs,
            int32_t nbInputs,
            DynamicPluginTensorDesc const * outputs,
            int32_t nbOutputs ) const  [inline], [virtual], [noexcept]
```

Find the workspace size required by the layer.

This function is called after the plugin is configured, and possibly during execution. The result should be a sufficient workspace size to deal with inputs and outputs of the given size or any smaller problem.

Returns

The workspace size.

9.133.1.12 supportsFormatCombination()

```
virtual bool nvinfer1::v_1_0::IPluginV3OneBuild::supportsFormatCombination (
            int32_t pos,
            DynamicPluginTensorDesc const * inOut,
            int32_t nbInputs,
            int32_t nbOutputs )  [pure virtual], [noexcept]
```

Return true if plugin supports the format and datatype for the input/output indexed by pos.

For this method inputs are numbered 0.. (nbInputs - 1) and outputs are numbered nbInputs.. (nbInputs + nbOutputs

- 1). Using this numbering, pos is an index into InOut, where 0 <= pos < nbInputs + nbOutputs - 1.

TensorRT invokes this method to ask if the input/output indexed by pos supports the format/datatype specified by inOut[pos].format and inOut[pos].type. The override should return true if that format/datatype at inOut[pos] are supported by the plugin. If support is conditional on other input/output formats/datatypes, the plugin can make its result conditional on the formats/datatypes in inOut[0.. pos - 1], which will be set to values that the plugin supports. The override should not inspect inOut[pos1.. nbInputs + nbOutputs - 1], which will have invalid values. In other words, the decision for pos must be based on inOut[0..pos] only.

Some examples:

- A definition for a plugin that supports only FP16 NCHW:

```
return inOut.format[pos] == TensorFormat::kLINEAR && inOut.type[pos] == DataType::kHALF;
```

- A definition for a plugin that supports only FP16 NCHW for its two inputs, and FP32 NCHW for its single output:

```
return inOut.format[pos] == TensorFormat::kLINEAR && (inOut.type[pos] == pos < 2 ?  DataType::kHALF :
DataType::kFLOAT);
```

- A definition for a "polymorphic" plugin with two inputs and one output that supports any format or type, but the inputs and output must have the same format and type:

```
return pos == 0 || (inOut.format[pos] == inOut.format[0] && inOut.type[pos] == inOut.type[0]);
```

> Warning
>
> TensorRT will stop querying once it finds getFormatCombinationLimit() of combinations.

See also

getFormatCombinationLimit

9.133.2 Member Data Documentation

9.133.2.1 kDEFAULT_FORMAT_COMBINATION_LIMIT

```
constexpr int32_t nvinfer1::v_1_0::IPluginV3OneBuild::kDEFAULT_FORMAT_COMBINATION_LIMIT = 100  [static],
[constexpr]
```

The default maximum number of format combinations that will be timed by TensorRT during the build phase.

See also

getFormatCombinationLimit

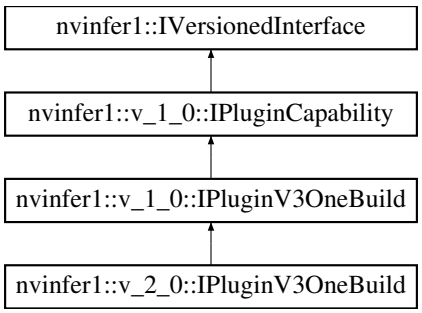
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.134 nvinfer1::v_2_0::IPluginV3OneBuild Class Reference

```
#include <NvInferRuntime.h>
```

Inheritance diagram for nvinfer1::v_2_0::IPluginV3OneBuild:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual int32_t getAliasedInput (int32_t outputIndex) noexcept

  Communicates to TensorRT that the output at the specified output index is aliased to the input at the returned index.

Additional Inherited Members

9.134.1 Member Function Documentation

9.134.1.1 getAliasedInput()

```
virtual int32_t nvinfer1::v_2_0::IPluginV3OneBuild::getAliasedInput (
            int32_t outputIndex )  [inline], [virtual], [noexcept]
```

Communicates to TensorRT that the output at the specified output index is aliased to the input at the returned index.

Enables read-modify-write behavior in plugins. TensorRT may insert copies to facilitate this capability.

Returns

An integer denoting the index of the input which is aliased to the output at outputIndex. Returning -1 indicates that the output is not aliased to any input. Otherwise, the valid range for return value is [0, nbInputs - 1].

Note

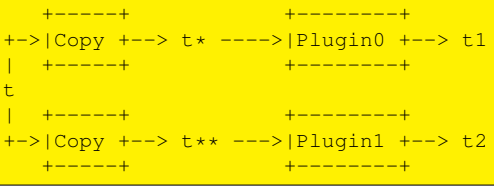
A given plugin input can only be aliased to a single plugin output.

This API will only be called and have an effect when PreviewFeature::kALIASED_PLUGIN_IO_10_03 is turned on.

Warning

If an input is not shallow copyable, a copy inserted by TensorRT may not work as intended. Therefore, using this feature with tensors requiring deep copies is not supported.

If a given tensor is requested to be aliased by two different plugins, this may result in divergent copies of the tensor after writes from each plugin. e.g. In the below example, t1 and t2 could be divergent.



9.134.1.2 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_2_0::IPluginV3OneBuild::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Reimplemented from nvinfer1::v_1_0::IPluginV3OneBuild.

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.135 IPluginV3OneBuildV2 Class Reference

A plugin capability interface that extends IPluginV3OneBuild by providing I/O aliasing functionality.

```
#include <NvInferRuntime.h>
```

9.135.1 Detailed Description

A plugin capability interface that extends IPluginV3OneBuild by providing I/O aliasing functionality.

See also

IPluginV3OneBuild

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.136 IPluginV3OneCore Class Reference

A plugin capability interface that enables the core capability (PluginCapabilityType::kCORE).

```
#include <NvInferRuntime.h>
```

9.136.1 Detailed Description

A plugin capability interface that enables the core capability (PluginCapabilityType::kCORE).

See also

IPluginCapability

PluginCapabilityType

IPluginV3::getCapabilityInterface()

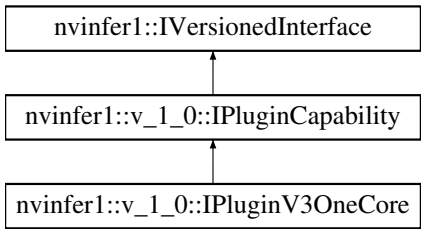
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.137 nvinfer1::v_1_0::IPluginV3OneCore Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IPluginV3OneCore:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual AsciiChar const * getPluginName () const noexcept=0

  Return the plugin name. Should match the plugin name returned by the corresponding plugin creator.
- virtual AsciiChar const * getPluginVersion () const noexcept=0

  Return the plugin version. Should match the plugin version returned by the corresponding plugin creator.
- virtual AsciiChar const * getPluginNamespace () const noexcept=0

  Return the namespace of the plugin object. Should match the plugin namespace returned by the corresponding plugin creator.

Additional Inherited Members

9.137.1 Member Function Documentation

9.137.1.1 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IPluginV3OneCore::getInterfaceInfo ( ) const  [inline], [override], [virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.137.1.2 getPluginName()

```
virtual AsciiChar const * nvinfer1::v_1_0::IPluginV3OneCore::getPluginName ( ) const  [pure virtual], [noexcept]
```

Return the plugin name. Should match the plugin name returned by the corresponding plugin creator.

See also

IPluginCreatorV3One::getPluginName()

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

9.137.1.3 getPluginNamespace()

```
virtual AsciiChar const * nvinfer1::v_1_0::IPluginV3OneCore::getPluginNamespace ( ) const  [pure virtual], [noexcept]
```

Return the namespace of the plugin object. Should match the plugin namespace returned by the corresponding plugin creator.

See also

IPluginCreatorV3One::getPluginNamespace()

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

9.137.1.4 getPluginVersion()

```
virtual AsciiChar const * nvinfer1::v_1_0::IPluginV3OneCore::getPluginVersion ( ) const  [pure virtual], [noexcept]
```

Return the plugin version. Should match the plugin version returned by the corresponding plugin creator.

See also

IPluginCreatorV3One::getPluginVersion()

Warning

The string returned must be NULL-terminated and have a length of 1024 bytes or less including the NULL terminator.

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.138 IPluginV3OneRuntime Class Reference

A plugin capability interface that enables the runtime capability (PluginCapabilityType::kRUNTIME). Exposes methods that allow the expression of the runtime properties and behavior of a plugin.

`#include <NvInferRuntime.h>`

9.138.1 Detailed Description

A plugin capability interface that enables the runtime capability (PluginCapabilityType::kRUNTIME). Exposes methods that allow the expression of the runtime properties and behavior of a plugin.

See also

IPluginCapability

PluginCapabilityType

IPluginV3::getCapabilityInterface()

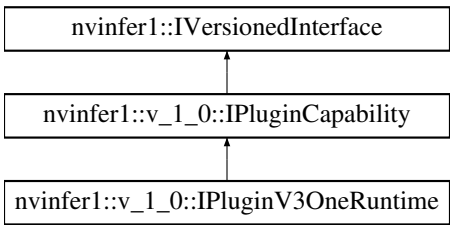
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.139 nvinfer1::v_1_0::IPluginV3OneRuntime Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IPluginV3OneRuntime:



Public Member Functions

- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual int32_t setTactic (int32_t tactic) noexcept

  Set the tactic to be used in the subsequent call to enqueue(). If no custom tactics were advertised, this will have a value of 0, which is designated as the default tactic.
- virtual int32_t onShapeChange (PluginTensorDesc const ∗in, int32_t nbInputs, PluginTensorDesc const ∗out, int32_t nbOutputs) noexcept=0

  Called when a plugin is being prepared for execution for specific dimensions. This could happen multiple times in the execution phase, both during creation of an engine by IBuilder and execution of an engine by IExecutionContext.
- virtual int32_t enqueue (PluginTensorDesc const ∗inputDesc, PluginTensorDesc const ∗outputDesc, void const ∗const ∗inputs, void ∗const ∗outputs, void ∗workspace, cudaStream_t stream) noexcept=0

  Execute the layer.
- virtual IPluginV3 ∗ attachToContext (IPluginResourceContext ∗context) noexcept=0

  Clone the plugin, attach the cloned plugin object to a execution context and grant the cloned plugin access to some context resources.
- virtual PluginFieldCollection const ∗ getFieldsToSerialize () noexcept=0

  Get the plugin fields which should be serialized.

Additional Inherited Members

9.139.1 Member Function Documentation

9.139.1.1 attachToContext()

```
virtual IPluginV3 * nvinfer1::v_1_0::IPluginV3OneRuntime::attachToContext (
            IPluginResourceContext * context )  [pure virtual], [noexcept]
```

Clone the plugin, attach the cloned plugin object to a execution context and grant the cloned plugin access to some context resources.

This function is called automatically for each plugin when a new execution context is created. The plugin may use resources provided by the IPluginResourceContext until the plugin is deleted by TensorRT.

If the plugin needs per-context resources, it can be allocated here.

Parameters

context	A resource context that exposes methods to get access to execution context specific resources. A different resource context is guaranteed for each different execution context to which the plugin is attached.

See also

IPluginResourceContext

Note

This method should clone the entire IPluginV3 object, not just the runtime interface

Returns

A clone of the IPluginV3 object whose runtime interface on which this method is invoked, which has attached to the provided resource context.

9.139.1.2 enqueue()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneRuntime::enqueue (
            PluginTensorDesc const * inputDesc,
            PluginTensorDesc const * outputDesc,
            void const *const * inputs,
            void *const * outputs,
            void * workspace,
            cudaStream_t stream )  [pure virtual], [noexcept]
```

Execute the layer.

Parameters

inputDesc	how to interpret the memory for the input tensors.
outputDesc	how to interpret the memory for the output tensors.
inputs	The memory for the input tensors.
outputs	The memory for the output tensors.
workspace	Workspace for execution.
stream	The stream in which to execute the kernels.

Returns

0 for success, else non-zero (which will cause engine termination). The returned code will be reported through the error recorder.

9.139.1.3 getFieldsToSerialize()

```
virtual PluginFieldCollection const * nvinfer1::v_1_0::IPluginV3OneRuntime::getFieldsToSerialize (
)  [pure virtual], [noexcept]
```

Get the plugin fields which should be serialized.

Note

The set of plugin fields returned does not necessarily need to match that advertised through getFieldNames() of the corresponding plugin creator.

To serialize arbitrary plugin data, use a PluginField of PluginFieldType::kUNKNOWN, with the length of the PluginField set to the correct number of bytes.

9.139.1.4 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IPluginV3OneRuntime::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.139.1.5 onShapeChange()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneRuntime::onShapeChange (
            PluginTensorDesc const * in,
            int32_t nbInputs,
            PluginTensorDesc const * out,
            int32_t nbOutputs )  [pure virtual], [noexcept]
```

Called when a plugin is being prepared for execution for specific dimensions. This could happen multiple times in the execution phase, both during creation of an engine by IBuilder and execution of an engine by IExecutionContext.

- IBuilder will call this function once per profile, with `in` resolved to the values specified by the kOPT field of the current profile.
- IExecutionContext will call this during the next subsequent instance of enqueueV3() or executeV2() if:
  - The optimization profile is changed via setOptimizationProfile() or setOptimizationProfileAsync().
  - An input binding is changed via setInputTensorAddress() or setTensorAddress() or setInputShape().

Warning

The execution phase is timing critical during IExecutionContext but is not part of the timing loop when called from IBuilder. Performance bottlenecks of onShapeChange() will not show up during engine building but will be visible during execution if any triggering functions are called.

Parameters

in	The input tensors attributes that are used for configuration.
nbInputs	Number of input tensors.
out	The output tensors attributes that are used for configuration.
nbOutputs	Number of output tensors.

9.139.1.6 setTactic()

```
virtual int32_t nvinfer1::v_1_0::IPluginV3OneRuntime::setTactic (
            int32_t tactic )  [inline], [virtual], [noexcept]
```

Set the tactic to be used in the subsequent call to enqueue(). If no custom tactics were advertised, this will have a value of 0, which is designated as the default tactic.

Returns

0 for success, else non-zero (which will cause engine termination). The returned code will be reported through the error recorder.

The documentation for this class was generated from the following file:

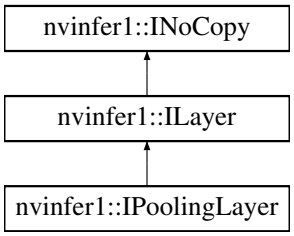
- NvInferRuntime.h

9.140 nvinfer1::IPoolingLayer Class Reference

A Pooling layer in a network definition.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IPoolingLayer:



Public Member Functions

- void setPoolingType (PoolingType type) noexcept

  Set the type of activation to be performed.
- PoolingType getPoolingType () const noexcept

  Get the type of activation to be performed.
- void setBlendFactor (float blendFactor) noexcept

  Set the blending factor for the max_average_blend mode: max_average_blendPool = (1-blendFactor)∗maxPool + blend↩ Factor∗avgPool blendFactor is a user value in [0,1] with the default value of 0.0 This value only applies for the kMAX↩ _AVERAGE_BLEND mode.

- float getBlendFactor () const noexcept

  Get the blending factor for the max_average_blend mode: max_average_blendPool = (1-blendFactor)∗maxPool + blendFactor∗avgPool blendFactor is a user value in [0,1] with the default value of 0.0 In modes other than kMAX↩ _AVERAGE_BLEND, blendFactor is ignored.
- void setAverageCountExcludesPadding (bool exclusive) noexcept

  Set whether average pooling uses as a denominator the overlap area between the window and the unpadded input. If this is not set, the denominator is the overlap between the pooling window and the padded input.
- bool getAverageCountExcludesPadding () const noexcept

  Get whether average pooling uses as a denominator the overlap area between the window and the unpadded input.
- void setPrePadding (Dims const &padding) noexcept

  Set the multi-dimension pre-padding for pooling.
- Dims getPrePadding () const noexcept

  Get the pre-padding.
- void setPostPadding (Dims const &padding) noexcept

  Set the multi-dimension post-padding for pooling.
- Dims getPostPadding () const noexcept

  Get the padding.
- void setPaddingMode (PaddingMode paddingMode) noexcept

  Set the padding mode.
- PaddingMode getPaddingMode () const noexcept

  Get the padding mode.
- void setWindowSizeNd (Dims const &windowSize) noexcept

  Set the multi-dimension window size for pooling.
- Dims getWindowSizeNd () const noexcept

  Get the multi-dimension window size for pooling.
- void setStrideNd (Dims const &stride) noexcept

  Set the multi-dimension stride for pooling.
- Dims getStrideNd () const noexcept

  Get the multi-dimension stride for pooling.
- void setPaddingNd (Dims const &padding) noexcept

  Set the multi-dimension padding for pooling.
- Dims getPaddingNd () const noexcept

  Get the multi-dimension padding for pooling.

Protected Member Functions

- virtual ∼IPoolingLayer () noexcept=default

Protected Attributes

- apiv::VPoolingLayer ∗ mImpl

9.140.1 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.

9.140.2 Constructor & Destructor Documentation

9.140.2.1 ~IPoolingLayer()

```
virtual nvinfer1::IPoolingLayer::~IPoolingLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.140.3 Member Function Documentation

9.140.3.1 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()

9.140.3.2 getBlendFactor()

```
float nvinfer1::IPoolingLayer::getBlendFactor ( ) const  [inline], [noexcept]
```

Get the blending factor for the max_average_blend mode: max_average_blendPool = (1-blendFactor)∗maxPool + blendFactor∗avgPool blendFactor is a user value in [0,1] with the default value of 0.0 In modes other than kMAX_↩ AVERAGE_BLEND, blendFactor is ignored.

See also

setBlendFactor()

9.140.3.3 getPaddingMode()

```
PaddingMode nvinfer1::IPoolingLayer::getPaddingMode ( ) const  [inline], [noexcept]
```

Get the padding mode.

Default: kEXPLICIT_ROUND_DOWN

See also

setPaddingMode()

9.140.3.4 getPaddingNd()

```
Dims nvinfer1::IPoolingLayer::getPaddingNd ( ) const  [inline], [noexcept]
```

Get the multi-dimension padding for pooling.

If the padding is asymmetric, the pre-padding is returned.

See also

setPaddingNd()

9.140.3.5 getPoolingType()

```
PoolingType nvinfer1::IPoolingLayer::getPoolingType ( ) const  [inline], [noexcept]
```

Get the type of activation to be performed.

See also

setPoolingType(), PoolingType

9.140.3.6 getPostPadding()

```
Dims nvinfer1::IPoolingLayer::getPostPadding ( ) const  [inline], [noexcept]
```

Get the padding.

See also

setPostPadding()

9.140.3.7 getPrePadding()

```
Dims nvinfer1::IPoolingLayer::getPrePadding ( ) const  [inline], [noexcept]
```

Get the pre-padding.

See also

setPrePadding()

9.140.3.8 getStrideNd()

```
Dims nvinfer1::IPoolingLayer::getStrideNd ( ) const  [inline], [noexcept]
```

Get the multi-dimension stride for pooling.

See also

setStrideNd()

9.140.3.9 getWindowSizeNd()

```
Dims nvinfer1::IPoolingLayer::getWindowSizeNd ( ) const  [inline], [noexcept]
```

Get the multi-dimension window size for pooling.

See also

setWindowSizeNd()

9.140.3.10 setAverageCountExcludesPadding()

```
void nvinfer1::IPoolingLayer::setAverageCountExcludesPadding (
            bool exclusive )  [inline], [noexcept]
```

Set whether average pooling uses as a denominator the overlap area between the window and the unpadded input. If this is not set, the denominator is the overlap between the pooling window and the padded input.

Default: true

See also

getAverageCountExcludesPadding()

9.140.3.11 setBlendFactor()

```
void nvinfer1::IPoolingLayer::setBlendFactor (
            float blendFactor )  [inline], [noexcept]
```

Set the blending factor for the max_average_blend mode: max_average_blendPool = (1-blendFactor)∗maxPool + blendFactor∗avgPool blendFactor is a user value in [0,1] with the default value of 0.0 This value only applies for the kMAX_AVERAGE_BLEND mode.

Since DLA does not support kMAX_AVERAGE_BLEND, blendFactor is ignored on the DLA.

See also

getBlendFactor()

9.140.3.12 setPaddingMode()

```
void nvinfer1::IPoolingLayer::setPaddingMode (
            PaddingMode paddingMode )  [inline], [noexcept]
```

Set the padding mode.

Padding mode takes precedence if both setPaddingMode and setPre/PostPadding are used.

Default: kEXPLICIT_ROUND_DOWN

See also

getPaddingMode()

9.140.3.13 setPaddingNd()

```
void nvinfer1::IPoolingLayer::setPaddingNd (
            Dims const & padding )  [inline], [noexcept]
```

Set the multi-dimension padding for pooling.

The input will be padded by this number of elements in each dimension. Padding is symmetric. Padding value depends on pooling type, -inf is used for max pooling and zero padding for average pooling.

Default: (0, 0, ..., 0)

If executing this layer on DLA, only support 2D padding, both height and width of padding must be in the range [0,7].

See also

getPaddingNd() setPadding() getPadding()

9.140.3.14 setPoolingType()

```
void nvinfer1::IPoolingLayer::setPoolingType (
            PoolingType type )  [inline], [noexcept]
```

Set the type of activation to be performed.

DLA only supports kMAX and kAVERAGE pooling types.

See also

getPoolingType(), PoolingType

9.140.3.15 setPostPadding()

```
void nvinfer1::IPoolingLayer::setPostPadding (
            Dims const & padding )  [inline], [noexcept]
```

Set the multi-dimension post-padding for pooling.

The end of the input will be padded by this number of elements in each dimension. Padding value depends on pooling type, -inf is used for max pooling and zero padding for average pooling.

Default: (0, 0, ..., 0)

If executing this layer on DLA, only support 2D padding, both height and width of padding must be in the range [0,7].

See also

getPostPadding()

9.140.3.16 setPrePadding()

```
void nvinfer1::IPoolingLayer::setPrePadding (
            Dims const & padding )  [inline], [noexcept]
```

Set the multi-dimension pre-padding for pooling.

The start of the input will be padded by this number of elements in each dimension. Padding value depends on pooling type, -inf is used for max pooling and zero padding for average pooling.

Default: (0, 0, ..., 0)

If executing this layer on DLA, only support 2D padding, both height and width of padding must be in the range [0,7].

See also

getPrePadding()

9.140.3.17 setStrideNd()

```
void nvinfer1::IPoolingLayer::setStrideNd (
            Dims const & stride )  [inline], [noexcept]
```

Set the multi-dimension stride for pooling.

Default: (1, 1, ..., 1)

If executing this layer on DLA, only support 2D stride, both height and width of stride must be in the range [1,16].

See also

getStrideNd()

9.140.3.18 setWindowSizeNd()

```
void nvinfer1::IPoolingLayer::setWindowSizeNd (
            Dims const & windowSize )  [inline], [noexcept]
```

Set the multi-dimension window size for pooling.

If executing this layer on DLA, only support 2D window size, both height and width of window size must be in the range [1,8].

See also

getWindowSizeNd() setWindowSize() getWindowSize()

9.140.4 Member Data Documentation

9.140.4.1 mImpl

```
apiv::VPoolingLayer* nvinfer1::IPoolingLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.141 IProfiler Class Reference

Application-implemented interface for profiling.

`#include <NvInferRuntime.h>`

9.141.1 Detailed Description

Application-implemented interface for profiling.

When this class is added to an execution context, the profiler will be called once per layer for each invocation of executeV2()/enqueueV3().

It is not recommended to run inference with profiler enabled when the inference execution time is critical since the profiler may affect execution time negatively.

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.142 nvinfer1::v_1_0::IProfiler Class Reference

`#include <NvInferRuntime.h>`

Public Member Functions

- virtual void reportLayerTime (char const *layerName, float ms) noexcept=0
  Layer time reporting callback.
- virtual ~IProfiler () noexcept

9.142.1 Constructor & Destructor Documentation

9.142.1.1 ~IProfiler()

```
virtual nvinfer1::v_1_0::IProfiler::~IProfiler ( )  [inline], [virtual], [noexcept]
```

9.142.2 Member Function Documentation

9.142.2.1 reportLayerTime()

```
virtual void nvinfer1::v_1_0::IProfiler::reportLayerTime (
            char const * layerName,
            float ms )  [pure virtual], [noexcept]
```

Layer time reporting callback.

Parameters

layerName	The name of the layer, set when constructing the network definition. If the engine is built with profiling verbosity set to kNONE, the layerName is the decimal index of the layer.
ms	The time in milliseconds to execute the layer.

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.143 IProgressMonitor Class Reference

Application-implemented progress reporting interface for TensorRT.

`#include <NvInfer.h>`

9.143.1 Detailed Description

Application-implemented progress reporting interface for TensorRT.

The IProgressMonitor is a user-defined object that TensorRT uses to report back when an internal algorithm has started or finished a phase to help provide feedback on the progress of the optimizer.

The IProgressMonitor will trigger its start function when a phase is entered and will trigger its finish function when that phase is exited. Each phase consists of one or more steps. When each step is completed, the stepComplete function is triggered. This will allow an application using the builder to communicate progress relative to when the optimization step is expected to complete.

The implementation of IProgressMonitor must be thread-safe so that it can be called from multiple internal threads. The lifetime of the IProgressMonitor must exceed the lifetime of all TensorRT objects that use it.

Note

To ensure compatibility of source code with future versions of TensorRT, use IProgressMonitor, not v_1_0::↩ IProgressMonitor

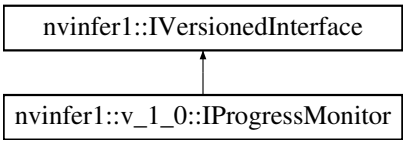
The documentation for this class was generated from the following file:

- NvInfer.h

9.144 nvinfer1::v_1_0::IProgressMonitor Class Reference

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::v_1_0::IProgressMonitor:



Public Member Functions

- IProgressMonitor ()=default
- virtual ~IProgressMonitor () noexcept=default
- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual void phaseStart (char const *phaseName, char const *parentPhase, int32_t nbSteps) noexcept=0

  Signal that a phase of the optimizer has started.
- virtual bool stepComplete (char const *phaseName, int32_t step) noexcept=0

  Signal that a step of an optimizer phase has finished.
- virtual void phaseFinish (char const *phaseName) noexcept=0

  Signal that a phase of the optimizer has finished.

Additional Inherited Members

9.144.1 Constructor & Destructor Documentation

9.144.1.1 IProgressMonitor()

`nvinfer1::v_1_0::IProgressMonitor::IProgressMonitor ( ) [default]`

9.144.1.2 ~IProgressMonitor()

`virtual nvinfer1::v_1_0::IProgressMonitor::~IProgressMonitor ( ) [virtual], [default], [noexcept]`

9.144.2 Member Function Documentation

9.144.2.1 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IProgressMonitor::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.144.2.2 phaseFinish()

```
virtual void nvinfer1::v_1_0::IProgressMonitor::phaseFinish (
            char const * phaseName )  [pure virtual], [noexcept]
```

Signal that a phase of the optimizer has finished.

Parameters

phaseName	The name of the phase that has finished.

The phaseFinish function signals to the application that the phase is complete. This function may be called before all steps in the range [0, nbSteps) have been reported to stepComplete. This scenario can be triggered by error handling, internal optimizations, or when stepComplete returns false to request cancellation of the build.

See also

phaseStart

9.144.2.3 phaseStart()

```
virtual void nvinfer1::v_1_0::IProgressMonitor::phaseStart (
            char const * phaseName,
            char const * parentPhase,
            int32_t nbSteps )  [pure virtual], [noexcept]
```

Signal that a phase of the optimizer has started.

Parameters

phaseName	The name of this phase for tracking purposes.
parentPhase	The parent phase that this phase belongs to, or nullptr if there is no parent.
nbSteps	The number of steps that are involved in this phase.

The phaseStart function signals to the application that the current phase is beginning, and that it has a certain number of steps to perform. If `phaseParent` is nullptr, then the phaseStart is beginning an independent phase, and if `phaseParent` is specified, then the current phase, specified by `phaseName`, is within the scope of the parent phase. `nbSteps` will always be a positive number. The phaseStart function implies that the first step is being executed. TensorRT will signal when each step is complete.

Phase names are human readable English strings which are unique within a single phase hierarchy but which can be reused once the previous instance has completed. Phase names and their hierarchies may change between versions of TensorRT.

See also

phaseFinish

9.144.2.4 stepComplete()

```
virtual bool nvinfer1::v_1_0::IProgressMonitor::stepComplete (
            char const * phaseName,
            int32_t step )  [pure virtual], [noexcept]
```

Signal that a step of an optimizer phase has finished.

Parameters

phaseName	The name of the innermost phase being executed.
step	The step number that was completed.

The stepComplete function signals to the application that TensorRT has finished the current `step` for the phase `phaseName`, and will move onto the next step if there is one. The application can return false for TensorRT to exit the build early. The step value will increase on subsequent calls in the range [0, nbSteps).

Returns

true to continue to the next step or false to stop the build.

The documentation for this class was generated from the following file:

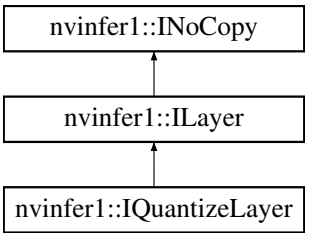
- NvInfer.h

9.145 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.
- void setToType (DataType toType) noexcept

  Set the Quantize layer output type.
- DataType getToType () const noexcept

  Return the Quantize layer output type.

Protected Member Functions

- virtual ~IQuantizeLayer () noexcept=default

Protected Attributes

- apiv::VQuantizeLayer * mImpl

9.145.1 Detailed Description

A Quantize layer in a network definition.

This layer accepts a floating-point data input tensor, and uses the scale and zeroPt inputs to quantize the data according to: `output` = clamp(round(`input` / `scale`) + `zeroPt`)

Rounding type is rounding-to-nearest ties-to-even (https://en.wikipedia.org/wiki/Rounding#Round_half_to_even). Clamping range according to data type:

- FP8: [-448, 448]
- INT4: [-8, 7]
- INT8: [-128, 127]

The first input (index 0) is the tensor to be quantized. The second (index 1) and third (index 2) are the scale and zero point respectively. `scale` and `zeroPt` should have identical dimensions, and rank lower or equal to 2.

The `zeroPt` tensor is optional, and if not set, will be assumed to be zero. Its data type must match the output data type. `zeroPt` must only contain zero-valued coefficients, because only symmetric quantization is supported. The `scale` value must be a scalar for per-tensor quantization, a 1D tensor for per-channel quantization, or the same rank as the input tensor for block quantization (supported for DataType::kINT4 only). All `scale` coefficients must have positive values. The size of the 1D `scale` tensor must match the size of the quantization axis. For block quantization, the shape of `scale` tensor must match the shape of the input, except for one dimension (the last or second to last dimension) in which blocking occurs. The size of `zeroPt` must match the size of `scale`.

The subgraph which terminates with the `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, DataType::kFP8 or DataType::kINT4. The input type, if constrained, must be constrained to DataType::kFLOAT, DataType::kHALF, or DataType::kBF16. The output size is the same as the input size. The quantization axis is in reference to the input tensor's dimensions.

IQuantizeLayer supports DataType::kFLOAT, DataType::kHALF, or DataType::kBF16 precision and will default to DataType::kFLOAT precision during instantiation. For strongly typed networks, `input` data type must match the `scale` data type.

IQuantizeLayer supports DataType::kINT8, DataType::kFP8, or DataType::kINT4 output.

As an example of the operation of this layer, imagine a 4D NCHW activation input which can be quantized using a single scale coefficient (referred to as per-tensor quantization): For each n in N: For each c in C: For each h in H: For each w in W: output[n,c,h,w] = clamp(round(`input`[n,c,h,w] / `scale`) + `zeroPt`)

Per-channel quantization is supported only for weight inputs. Thus, Activations cannot be quantized per-channel. As an example of per-channel operation, imagine a 4D KCRS weights input and K (dimension 0) as the quantization axis. The scale is an array of coefficients, and must have the same size as the quantization axis. For each k in K: For each c in C: For each r in R: For each s in S: output[k,c,r,s] = clamp(round(`input`[k,c,r,s] / `scale`[k]) + `zeroPt`[k])

Block quantization is supported only for weight inputs of DataType::kINT4. As an example of blocked operation, imagine a 2D RS weights input, R (dimension 0) as the blocking axis and B as the block size. The scale is a 2D array of coefficients, with dimensions (R//B, S). For each r in R: For each s in S: output[r,s] = clamp(round(`input`[r,s] / `scale`[r//B, s]) + `zeroPt`[r//B, s])

Note

Only symmetric quantization is supported.

Currently the only allowed build-time constant `scale` and `zeroPt` subgraphs are:

1. Constant -> Quantize
2. Constant -> Cast -> Quantize

The input tensor for this layer must not be a scalar.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.145.2 Constructor & Destructor Documentation

9.145.2.1 ~IQuantizeLayer()

```
virtual nvinfer1::IQuantizeLayer::~IQuantizeLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.145.3 Member Function Documentation

9.145.3.1 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.

9.145.3.2 getToType()

```
DataType nvinfer1::IQuantizeLayer::getToType ( ) const  [inline], [noexcept]
```

Return the Quantize layer output type.

Returns

toType parameter set during layer creation or by setToType(). The return value is the output type of the quantize layer. The default value is DataType::kINT8.

9.145.3.3 setAxis()

```
void nvinfer1::IQuantizeLayer::setAxis (
            int32_t axis )  [inline], [noexcept]
```

Set the quantization axis.

Set the index of the quantization axis (with reference to the input tensor's dimensions). The axis must be a valid axis if the scale tensor has more than one coefficient. The axis value is used only for per-axis (per-channel) quantization.

9.145.3.4 setToType()

```
void nvinfer1::IQuantizeLayer::setToType (
            DataType toType )  [inline], [noexcept]
```

Set the Quantize layer output type.

Parameters

toType	The DataType of the output tensor.

Set the output type of the quantize layer. Valid values are DataType::kINT8 and DataType::kFP8. If the network is strongly typed, setToType must be used to set the output type, and use of setOutputType is an error. Otherwise, types passed to setOutputType and setToType must be the same.

See also

NetworkDefinitionCreationFlag::kSTRONGLY_TYPED

9.145.4 Member Data Documentation

9.145.4.1 mImpl

```
apiv::VQuantizeLayer* nvinfer1::IQuantizeLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

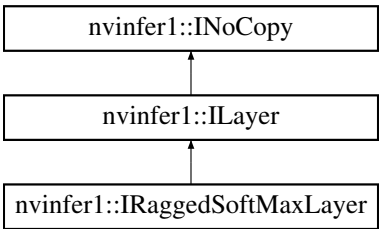
- NvInfer.h

9.146 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

9.146.1 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.

9.146.2 Constructor & Destructor Documentation

9.146.2.1 ~IRaggedSoftMaxLayer()

```
virtual nvinfer1::IRaggedSoftMaxLayer::~IRaggedSoftMaxLayer ( )  [protected], [virtual], [default],
[noexcept]
```

9.146.3 Member Data Documentation

9.146.3.1 mImpl

```
apiv::VRaggedSoftMaxLayer* nvinfer1::IRaggedSoftMaxLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

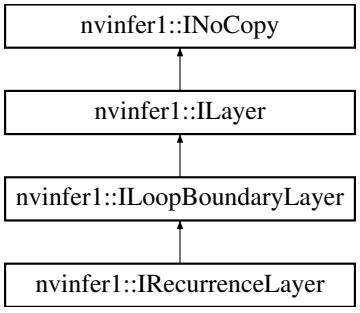
- NvInfer.h

9.147 nvinfer1::IRecurrenceLayer Class Reference

A recurrence layer in a network definition.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IRecurrenceLayer:



Public Member Functions

- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~IRecurrenceLayer () noexcept=default

Protected Attributes

- apiv::VRecurrenceLayer * mImpl

9.147.1 Detailed Description

A recurrence layer in a network definition.

The recurrence layer allows a loop iteration to compute a result from a value computed in the previous iteration.

9.147.2 Constructor & Destructor Documentation

9.147.2.1 ~IRecurrenceLayer()

```
virtual nvinfer1::IRecurrenceLayer::~IRecurrenceLayer ( )  [protected], [virtual], [default],
[noexcept]
```

9.147.3 Member Function Documentation

9.147.3.1 setInput()

```
void nvinfer1::ILayer::setInput (
            int32_t index,
            ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor Sets the input tensor for the given index.

For a recurrence layer, the values 0 and 1 are valid. The indices are as follows:

- 0: The initial value of the output tensor. The value must come from outside the loop.
- 1: The next value of the output tensor. The value usually comes from inside the loop, and must have the same dimensions as input 0.

If this function is called with the value 1, then the function getNbInputs() changes from returning 1 to 2.

9.147.4 Member Data Documentation

9.147.4.1 mImpl

```
apiv::VRecurrenceLayer* nvinfer1::IRecurrenceLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

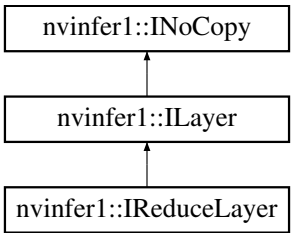
- NvInfer.h

9.148 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

9.148.1 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.

9.148.2 Constructor & Destructor Documentation

9.148.2.1 ~IReduceLayer()

```
virtual nvinfer1::IReduceLayer::~IReduceLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.148.3 Member Function Documentation

9.148.3.1 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

9.148.3.2 getOperation()

```
ReduceOperation nvinfer1::IReduceLayer::getOperation ( ) const  [inline], [noexcept]
```

Get the reduce operation for the layer.

See also

setOperation(), ReduceOperation

9.148.3.3 getReduceAxes()

```
uint32_t nvinfer1::IReduceLayer::getReduceAxes ( ) const  [inline], [noexcept]
```

Get the axes over which to reduce for the layer.

See also

setReduceAxes

9.148.3.4 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

9.148.3.5 setOperation()

```
void nvinfer1::IReduceLayer::setOperation (
            ReduceOperation op )  [inline], [noexcept]
```

Set the reduce operation for the layer.

See also

getOperation(), ReduceOperation

9.148.3.6 setReduceAxes()

```
void nvinfer1::IReduceLayer::setReduceAxes (
            uint32_t reduceAxes )  [inline], [noexcept]
```

Set the axes over which to reduce.

See also

getReduceAxes

9.148.4 Member Data Documentation

9.148.4.1 mImpl

```
apiv::VReduceLayer* nvinfer1::IReduceLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

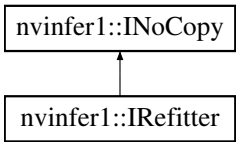
- NvInfer.h

9.149 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

  Refits associated engine.
- int32_t getMissing (int32_t size, char const **layerNames, WeightsRole *roles) noexcept

  Get description of missing weights.
- int32_t getAll (int32_t size, char const **layerNames, WeightsRole *roles) noexcept

  Get description of all weights that could be refit.
- TRT_DEPRECATED bool setDynamicRange (char const *tensorName, float min, float max) noexcept
- TRT_DEPRECATED float getDynamicRangeMin (char const *tensorName) const noexcept

  Get minimum of dynamic range.
- TRT_DEPRECATED float getDynamicRangeMax (char const *tensorName) const noexcept

  Get maximum of dynamic range.
- TRT_DEPRECATED 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.
- bool setNamedWeights (char const *name, Weights weights, TensorLocation location) noexcept

  Specify new weights on a specified device of given name.
- Weights getNamedWeights (char const *weightsName) const noexcept

  Get weights associated with the given name.
- TensorLocation getWeightsLocation (char const *weightsName) const noexcept

  Get location for the weights associated with the given name.
- bool unsetNamedWeights (char const *weightsName) noexcept

  Unset weights associated with the given name.
- void setWeightsValidation (bool weightsValidation) noexcept

  Set whether to validate weights during refitting.
- bool getWeightsValidation () const noexcept

  Get whether to validate weights values during refitting.

- bool refitCudaEngineAsync (cudaStream_t stream) noexcept

  Enqueue weights refitting of the associated engine on the given stream.
- Weights getWeightsPrototype (char const *weightsName) const noexcept

  Get the Weights prototype associated with the given name.

Protected Attributes

- apiv::VRefitter * mImpl

Additional Inherited Members

9.149.1 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.

9.149.2 Constructor & Destructor Documentation

9.149.2.1 ~IRefitter()

```
virtual nvinfer1::IRefitter::~IRefitter ( )  [virtual], [default], [noexcept]
```

9.149.3 Member Function Documentation

9.149.3.1 getAll()

```
int32_t nvinfer1::IRefitter::getAll (
            int32_t size,
            char const ** layerNames,
            WeightsRole * roles )  [inline], [noexcept]
```

Get description of all weights that could be refit.

Parameters

size	The number of items that can be safely written to a non-null layerNames or roles.
layerNames	Where to write the layer names.
roles	Where to write the weights roles.

Returns

The number of Weights that could be refit.

If layerNames!=nullptr, each written pointer points to a string owned by the engine being refit, and becomes invalid when the engine is destroyed.

9.149.3.2 getAllWeights()

```
int32_t nvinfer1::IRefitter::getAllWeights (
            int32_t size,
            char const ** weightsNames )  [inline], [noexcept]
```

Get names of all weights that could be refit.

Parameters

size	The number of weights names that can be safely written to.
weightsNames	The names of the weights to be updated, or nullptr for unnamed weights.

Returns

The number of Weights that could be refit.

If layerNames!=nullptr, each written pointer points to a string owned by the engine being refit, and becomes invalid when the engine is destroyed.

9.149.3.3 getDynamicRangeMax()

```
TRT_DEPRECATED float nvinfer1::IRefitter::getDynamicRangeMax (
            char const * tensorName ) const  [inline], [noexcept]
```

Get maximum of dynamic range.

Returns

Maximum of dynamic range.

If the dynamic range was never set, returns the maximum computed during calibration.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

Deprecated Deprecated in TensorRT 10.1. Superseded by explicit quantization.

9.149.3.4 getDynamicRangeMin()

```
TRT_DEPRECATED float nvinfer1::IRefitter::getDynamicRangeMin (
            char const * tensorName ) const  [inline], [noexcept]
```

Get minimum of dynamic range.

Returns

Minimum of dynamic range.

If the dynamic range was never set, returns the minimum computed during calibration.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

Deprecated Deprecated in TensorRT 10.1. Superseded by explicit quantization.

9.149.3.5 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()

9.149.3.6 getLogger()

```
ILogger * nvinfer1::IRefitter::getLogger ( ) const  [inline], [noexcept]
```

get the logger with which the refitter was created

Returns

the logger

9.149.3.7 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()

9.149.3.8 getMissing()

```
int32_t nvinfer1::IRefitter::getMissing (
            int32_t size,
            char const ** layerNames,
            WeightsRole * roles )  [inline], [noexcept]
```

Get description of missing weights.

For example, if some Weights have been set, but the engine was optimized in a way that combines weights, any unsupplied Weights in the combination are considered missing.

Parameters

size	The number of items that can be safely written to a non-null layerNames or roles.
layerNames	Where to write the layer names.
roles	Where to write the weights roles.

Returns

The number of missing Weights.

If layerNames!=nullptr, each written pointer points to a string owned by the engine being refit, and becomes invalid when the engine is destroyed.

9.149.3.9 getMissingWeights()

```
int32_t nvinfer1::IRefitter::getMissingWeights (
            int32_t size,
            char const ** weightsNames )  [inline], [noexcept]
```

Get names of missing weights.

For example, if some Weights have been set, but the engine was optimized in a way that combines weights, any unsupplied Weights in the combination are considered missing.

Parameters

size	The number of weights names that can be safely written to.
weightsNames	The names of the weights to be updated, or nullptr for unnamed weights.

Returns

The number of missing Weights.

If layerNames!=nullptr, each written pointer points to a string owned by the engine being refit, and becomes invalid when the engine is destroyed.

9.149.3.10 getNamedWeights()

```
Weights nvinfer1::IRefitter::getNamedWeights (
            char const * weightsName ) const  [inline], [noexcept]
```

Get weights associated with the given name.

Parameters

weightsName	The name of the weights to be refitted.

Returns

Weights associated with the given name.

If the weights were never set, returns null weights and reports an error to the refitter errorRecorder.

Warning

The string weightsName must be null-terminated, and be at most 4096 bytes including the terminator.

9.149.3.11 getTensorsWithDynamicRange()

```
TRT_DEPRECATED 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

size	The number of items that can be safely written to a non-null tensorNames.
tensorNames	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.

Deprecated Deprecated in TensorRT 10.1. Superseded by explicit quantization.

9.149.3.12 getWeightsLocation()

```
TensorLocation nvinfer1::IRefitter::getWeightsLocation (
            char const * weightsName ) const  [inline], [noexcept]
```

Get location for the weights associated with the given name.

Parameters

weightsName	The name of the weights to be refitted.

Returns

Location for the weights associated with the given name.

If the weights were never set, returns TensorLocation::kHOST and reports an error to the refitter errorRecorder.

Warning

The string weightsName must be null-terminated, and be at most 4096 bytes including the terminator.

9.149.3.13 getWeightsPrototype()

```
Weights nvinfer1::IRefitter::getWeightsPrototype (
            char const * weightsName ) const  [inline], [noexcept]
```

Get the Weights prototype associated with the given name.

Parameters

weightsName	The name of the weights to be refitted.

Returns

Weights prototype associated with the given name.

The type and count of weights prototype is the same as weights used for engine building. The values property is nullptr for weights prototypes. The count of the weights prototype is -1 when the name of the weights is nullptr or does not correspond to any refittable weights.

Warning

The string weightsName must be null-terminated, and be at most 4096 bytes including the terminator.

9.149.3.14 getWeightsValidation()

```
bool nvinfer1::IRefitter::getWeightsValidation ( ) const  [inline], [noexcept]
```

Get whether to validate weights values during refitting.

9.149.3.15 refitCudaEngine()

```
bool nvinfer1::IRefitter::refitCudaEngine ( )  [inline], [noexcept]
```

Refits associated engine.

Returns

True on success, or false if new weights validation fails or getMissingWeights() != 0 before the call. If false is returned, a subset of weights may have been refitted.

The behavior is undefined if the engine has pending enqueued work. Provided weights on CPU or GPU can be unset and released, or updated after refitCudaEngine returns.

IExecutionContexts associated with the engine remain valid for use afterwards. There is no need to set the same weights repeatedly for multiple refit calls as the weights memory can be updated directly instead.

9.149.3.16 refitCudaEngineAsync()

```
bool nvinfer1::IRefitter::refitCudaEngineAsync (
            cudaStream_t stream )  [inline], [noexcept]
```

Enqueue weights refitting of the associated engine on the given stream.

Parameters

stream	The stream to enqueue the weights updating task.

Returns

True on success, or false if new weights validation fails or getMissingWeights() != 0 before the call. If false is returned, a subset of weights may have been refitted.

The behavior is undefined if the engine has pending enqueued work on a different stream from the provided one. Provided weights on CPU can be unset and released, or updated after refitCudaEngineAsync returns. Freeing or updating of the provided weights on GPU can be enqueued on the same stream after refitCudaEngineAsync returns.

IExecutionContexts associated with the engine remain valid for use afterwards. There is no need to set the same weights repeatedly for multiple refit calls as the weights memory can be updated directly instead. The weights updating task should use the same stream as the one used for the refit call.

9.149.3.17 setDynamicRange()

```
TRT_DEPRECATED bool nvinfer1::IRefitter::setDynamicRange (
            char const * tensorName,
            float min,
            float max )  [inline], [noexcept]
```

Update dynamic range for a tensor.

Parameters

tensorName	The name of an ITensor in the network.
min	The minimum of the dynamic range for the tensor.
max	The maximum of the dynamic range for the tensor.

Returns

True if successful; false otherwise.

Returns false if there is no Int8 engine tensor derived from a network tensor of that name. If successful, then getMissing may report that some weights need to be supplied.

Warning

The string tensorName must be null-terminated, and be at most 4096 bytes including the terminator.

Deprecated Deprecated in TensorRT 10.1. Superseded by explicit quantization.

9.149.3.18 setErrorRecorder()

```
void nvinfer1::IRefitter::setErrorRecorder (
            IErrorRecorder * recorder )  [inline], [noexcept]
```

Set the ErrorRecorder for this interface.

Assigns the ErrorRecorder to this interface. The ErrorRecorder will track all errors during execution. This function will call incRefCount of the registered ErrorRecorder at least once. Setting recorder to nullptr unregisters the recorder with the interface, resulting in a call to decRefCount if a recorder has been registered.

If an error recorder is not set, messages will be sent to the global log stream.

Parameters

recorder	The error recorder to register with this interface.

See also

getErrorRecorder()

9.149.3.19 setMaxThreads()

```
bool nvinfer1::IRefitter::setMaxThreads (
            int32_t maxThreads )  [inline], [noexcept]
```

Set the maximum number of threads.

Parameters

maxThreads	The maximum number of threads that can be used by the refitter.

Returns

True if successful, false otherwise.

The default value is 1 and includes the current thread. A value greater than 1 permits TensorRT to use multi-threaded algorithms. A value less than 1 triggers a kINVALID_ARGUMENT error.

9.149.3.20 setNamedWeights() [1/2]

```
bool nvinfer1::IRefitter::setNamedWeights (
            char const * name,
            Weights weights )  [inline], [noexcept]
```

Specify new weights of given name.

Parameters

name	The name of the weights to be refit.
weights	The new weights to associate with the name.

Returns true on success, or false if new weights are rejected. Possible reasons for rejection are:

- The name of weights is nullptr or does not correspond to any refittable weights.
- The count of the weights is inconsistent with the count returned from calling getWeightsPrototype() with the same name.
- The type of the weights is inconsistent with the type returned from calling getWeightsPrototype() with the same name.

Modifying the weights before method refitCudaEngine or refitCudaEngineAsync returns will result in undefined behavior.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

9.149.3.21 setNamedWeights() [2/2]

```
bool nvinfer1::IRefitter::setNamedWeights (
            char const * name,
            Weights weights,
            TensorLocation location )  [inline], [noexcept]
```

Specify new weights on a specified device of given name.

Parameters

name	The name of the weights to be refitted.
weights	The new weights on the specified device.
location	The location (host vs. device) of the new weights.

Returns

True on success, or false if new weights are rejected. Possible reasons for rejection are:

- The name of the weights is nullptr or does not correspond to any refittable weights.
- The count of the weights is inconsistent with the count returned from calling getWeightsPrototype() with the same name.
- The type of the weights is inconsistent with the type returned from calling getWeightsPrototype() with the same name.

It is allowed to provide some weights on CPU and others on GPU. Modifying the weights before the method refitCudaEngine() or refitCudaEngineAsync() completes will result in undefined behavior.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

9.149.3.22 setWeights()

```
bool nvinfer1::IRefitter::setWeights (
            char const * layerName,
            WeightsRole role,
            Weights weights )  [inline], [noexcept]
```

Specify new weights for a layer of given name. Returns true on success, or false if new weights are rejected. Possible reasons for rejection are:

- There is no such layer by that name.
- The layer does not have weights with the specified role.
- The count of weights is inconsistent with the layer's original specification.
- The type of weights is inconsistent with the layer's original specification.

Modifying the weights before method refitCudaEngine or refitCudaEngineAsync returns will result in undefined behavior.

Warning

The string layerName must be null-terminated, and be at most 4096 bytes including the terminator.

9.149.3.23 setWeightsValidation()

```
void nvinfer1::IRefitter::setWeightsValidation (
            bool weightsValidation )  [inline], [noexcept]
```

Set whether to validate weights during refitting.

Parameters

weightsValidation	Indicate whether to validate weights during refitting.

When set to true, TensorRT will validate weights during FP32 to FP16/BF16 weights conversions or sparsifying weights in the refit call. If provided weights are not proper for some weights transformations, TensorRT will issue a warning and continue the transformation for minor issues (such as overflow during narrowing conversion), or issue an error and stop the refitting process for severe issues (such as sparsifying dense weights). By default the flag is true. Set the flag to false for faster refitting performance.

9.149.3.24 unsetNamedWeights()

```
bool nvinfer1::IRefitter::unsetNamedWeights (
            char const * weightsName )  [inline], [noexcept]
```

Unset weights associated with the given name.

Parameters

weightsName	The name of the weights to be refitted.

Returns

False if the weights were never set, returns true otherwise.

Unset weights before releasing them.

Warning

The string weightsName must be null-terminated, and be at most 4096 bytes including the terminator.

9.149.4 Member Data Documentation

9.149.4.1 mImpl

```
apiv::VRefitter* nvinfer1::IRefitter::mImpl  [protected]
```

The documentation for this class was generated from the following file:

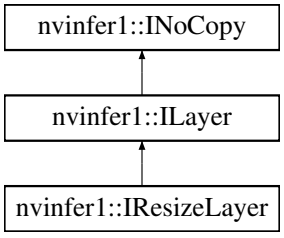
- NvInferRuntime.h

9.150 nvinfer1::IResizeLayer Class Reference

A resize layer in a network definition.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IResizeLayer:



Public Member Functions

- void setOutputDimensions (Dims const &dimensions) noexcept

  Set the output dimensions.
- Dims getOutputDimensions () const noexcept

  Get the output dimensions.
- void setScales (float const *scales, int32_t nbScales) noexcept

  Set the resize scales.
- int32_t getScales (int32_t size, float *scales) const noexcept

  Copies resize scales to scales[0, ..., nbScales-1], where nbScales is the number of scales that were set.
- void setResizeMode (InterpolationMode interpolationMode) noexcept

  Set resize mode for an input tensor.
- InterpolationMode getResizeMode () const noexcept

  Get resize mode for an input tensor.
- void setCoordinateTransformation (ResizeCoordinateTransformation coordTransform) noexcept

  Set coordinate transformation function.
- ResizeCoordinateTransformation getCoordinateTransformation () const noexcept

  Get coordinate transformation function.
- void setSelectorForSinglePixel (ResizeSelector selector) noexcept

  Set coordinate selector function when resized to single pixel.
- ResizeSelector getSelectorForSinglePixel () const noexcept

  Get the coordinate selector function when resized to single pixel.
- void setNearestRounding (ResizeRoundMode value) noexcept

  Set rounding mode for nearest neighbor resize.
- ResizeRoundMode getNearestRounding () const noexcept

  Get rounding mode for nearest neighbor resize.
- void setCubicCoeff (float A) noexcept

  Set the coefficient 'A' used in cubic interpolation.
- float getCubicCoeff () const noexcept

  Get the coefficient 'A' used in cubic interpolation.
- void setExcludeOutside (bool excludeFlag) noexcept

  Set the state for excluding outside pixels.
- bool getExcludeOutside () const noexcept

  Get the state for excluding outside pixels.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~IResizeLayer () noexcept=default

Protected Attributes

- apiv::VResizeLayer * mImpl

9.150.1 Detailed Description

A resize layer in a network definition.

Resize layer can be used for resizing a N-D tensor.

Resize layer currently supports the following configurations:

- InterpolationMode::kNEAREST - resizes last `m` dimensions of N-D, where 0 < m <= min(8, N) and N > 0
- InterpolationMode::kLINEAR - resizes last `m` dimensions of N-D, where 0 < m <= min(3, N) and N > 0

Default resize mode is InterpolationMode::kNEAREST.

The coordinates in the output tensor are mapped to coordinates in the input tensor using a function set by calling setCoordinateTransformation(). The default for all InterpolationMode settings (nearest, linear, bilinear, etc.) is ResizeCoordinateTransformation::kASYMMETRIC.

The resize layer provides two ways to resize tensor dimensions.

- Set output dimensions directly. It can be done for static as well as dynamic resize layer. Static resize layer requires output dimensions to be known at build-time. Dynamic resize layer requires output dimensions to be set as one of the input tensors.
- Set scales for resize. Each output dimension is calculated as floor(input dimension ∗ scale). Only static resize layer allows setting scales where the scales are known at build-time.

If executing this layer on DLA, the following combinations of parameters are supported:

- In kNEAREST mode:
  - (ResizeCoordinateTransformation::kASYMMETRIC, ResizeSelector::kFORMULA, ResizeRoundMode::kFLOOR)
  - (ResizeCoordinateTransformation::kHALF_PIXEL, ResizeSelector::kFORMULA, ResizeRoundMode::kHALF_DOWN)
  - (ResizeCoordinateTransformation::kHALF_PIXEL, ResizeSelector::kFORMULA, ResizeRoundMode::kHALF_UP)
- In kLINEAR mode:
  - (ResizeCoordinateTransformation::kHALF_PIXEL, ResizeSelector::kFORMULA)
  - (ResizeCoordinateTransformation::kHALF_PIXEL, ResizeSelector::kUPPER)

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.150.2 Constructor & Destructor Documentation

9.150.2.1 ~IResizeLayer()

```
virtual nvinfer1::IResizeLayer::~IResizeLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.150.3 Member Function Documentation

9.150.3.1 getCoordinateTransformation()

```
ResizeCoordinateTransformation nvinfer1::IResizeLayer::getCoordinateTransformation ( ) const  [inline],
[noexcept]
```

Get coordinate transformation function.

Returns

The coordinate transformation function.

9.150.3.2 getCubicCoeff()

```
float nvinfer1::IResizeLayer::getCubicCoeff ( ) const  [inline], [noexcept]
```

Get the coefficient 'A' used in cubic interpolation.

See also

setCubicCoeff()

9.150.3.3 getExcludeOutside()

```
bool nvinfer1::IResizeLayer::getExcludeOutside ( ) const  [inline], [noexcept]
```

Get the state for excluding outside pixels.

See also

setExcludeOutside()

9.150.3.4 getNearestRounding()

```
ResizeRoundMode nvinfer1::IResizeLayer::getNearestRounding ( ) const  [inline], [noexcept]
```

Get rounding mode for nearest neighbor resize.

Returns

The rounding mode.

9.150.3.5 getOutputDimensions()

```
Dims nvinfer1::IResizeLayer::getOutputDimensions ( ) const  [inline], [noexcept]
```

Get the output dimensions.

Returns

The output dimensions.

9.150.3.6 getResizeMode()

```
InterpolationMode nvinfer1::IResizeLayer::getResizeMode ( ) const  [inline], [noexcept]
```

Get resize mode for an input tensor.

Returns

The resize mode.

9.150.3.7 getScales()

```
int32_t nvinfer1::IResizeLayer::getScales (
            int32_t size,
            float * scales ) const  [inline], [noexcept]
```

Copies resize scales to scales[0, ..., nbScales-1], where nbScales is the number of scales that were set.

Parameters

size	The number of scales to get. If size != nbScales, no scales will be copied.
scales	Pointer to where to copy the scales. Scales will be copied only if size == nbScales and scales != nullptr.

In case the size is not known consider using size = 0 and scales = nullptr. This method will return the number of resize scales.

Returns

The number of resize scales i.e. nbScales if scales were set. Return -1 in case no scales were set or resize layer is used in dynamic mode.

9.150.3.8 getSelectorForSinglePixel()

```
ResizeSelector nvinfer1::IResizeLayer::getSelectorForSinglePixel ( ) const  [inline], [noexcept]
```

Get the coordinate selector function when resized to single pixel.

Returns

The selector function.

9.150.3.9 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

9.150.3.10 setCubicCoeff()

```
void nvinfer1::IResizeLayer::setCubicCoeff (
            float A )  [inline], [noexcept]
```

Set the coefficient 'A' used in cubic interpolation.

Cubic uses the coefficient 'A' to calculate the weight of input pixels:

```
x := The relative distance between the sampled pixels and the input coordinates.

weight(x) := for |x| <= 1, ((A + 2) * x - (A + 3)) * x * x + 1,
             for 1 < |x| < 2, ((A * x - 5 * A) * x + 8 * A) * x - 4 * A,
             others 0;
```

This attribute is valid only if ”resize mode” is ”cubic”.

The default value is -0.75.

9.150.3.11 setExcludeOutside()

```
void nvinfer1::IResizeLayer::setExcludeOutside (
            bool excludeFlag )  [inline], [noexcept]
```

Set the state for excluding outside pixels.

If set to true, the weight of sampling locations outside the input tensor will be set to false, and the weight will be renormalized so that their sum is 1.0.

The default value is false.

9.150.3.12 setInput()

```
void nvinfer1::ILayer::setInput (
            int32_t index,
            ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor.

Sets the input tensor for the given index. The index must be 0 for a static resize layer. A static resize layer is converted to a dynamic resize layer by calling setInput with an index 1. A dynamic resize layer cannot be converted back to a static resize layer.

For a dynamic resize layer, the values 0 and 1 are valid. The indices in the dynamic case are as follows:

- 0: Execution tensor to be resized.
- 1: The output dimensions, as a 1D tensor of type Int32 or Int64.

If this function is called with the value 1, then the function getNbInputs() changes from returning 1 to 2.

9.150.3.13 setNearestRounding()

```
void nvinfer1::IResizeLayer::setNearestRounding (
            ResizeRoundMode value )  [inline], [noexcept]
```

Set rounding mode for nearest neighbor resize.

This value is used for nearest neighbor interpolation rounding. It is applied after coordinate transformation.

Default is kFLOOR.

See also

ResizeRoundMode

9.150.3.14 setOutputDimensions()

```
void nvinfer1::IResizeLayer::setOutputDimensions (
            Dims const & dimensions )  [inline], [noexcept]
```

Set the output dimensions.

Parameters

dimensions	The output dimensions. Number of output dimensions must be the same as the number of input dimensions.

If executing this layer on DLA, setOutputDimensions() is not supported.

If there is a second input, i.e. resize layer is dynamic, calling setOutputDimensions() is an error and does not update the dimensions.

Output dimensions can be specified directly, or via scale factors relative to input dimensions. Scales for resize can be provided using setScales().

See also

setScales

getOutputDimensions

9.150.3.15 setResizeMode()

```
void nvinfer1::IResizeLayer::setResizeMode (
            InterpolationMode interpolationMode )  [inline], [noexcept]
```

Set resize mode for an input tensor.

Supported resize modes are Nearest Neighbor and Linear.

See also

InterpolationMode

9.150.3.16 setScales()

```
void nvinfer1::IResizeLayer::setScales (
            float const * scales,
            int32_t nbScales )  [inline], [noexcept]
```

Set the resize scales.

Parameters

scales	An array of resize scales.
nbScales	Number of scales. Number of scales must be equal to the number of input dimensions.

If executing this layer on DLA, there are three restrictions: 1) nbScales has to be exactly 4. 2) the first two elements in scales need to be exactly 1 (for unchanged batch and channel dimensions). 3) The last two elements in scales, representing the scale values along height and width dimensions, respectively, need to be integer values in the range of [1, 32] for kNEAREST mode and [1, 4] for kLINEAR. Example of DLA-supported scales: {1, 1, 2, 2}.

If there is a second input, i.e. resize layer is dynamic, calling setScales() is an error and does not update the scales.

Output dimensions are calculated as follows: outputDims[i] = floor(inputDims[i] $*$ scales[i])

Output dimensions can be specified directly, or via scale factors relative to input dimensions. Output dimensions can be provided directly using setOutputDimensions().

See also

setOutputDimensions

getScales

9.150.3.17 setSelectorForSinglePixel()

```
void nvinfer1::IResizeLayer::setSelectorForSinglePixel (
            ResizeSelector selector )  [inline], [noexcept]
```

Set coordinate selector function when resized to single pixel.

When resize to single pixel image, use this function to decide how to map the coordinate in the original image.

Default is ResizeSelector::kFORMULA.

See also

ResizeSelector

9.150.4 Member Data Documentation

9.150.4.1 mImpl

```
apiv::VResizeLayer* nvinfer1::IResizeLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

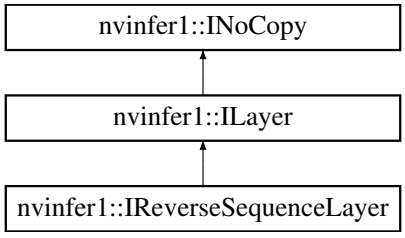
- NvInfer.h

9.151 nvinfer1::IReverseSequenceLayer Class Reference

A ReverseSequence layer in a network definition.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IReverseSequenceLayer:



Public Member Functions

- void setBatchAxis (int32_t batchAxis) noexcept

  Set the batch axis. Default is 1.
- int32_t getBatchAxis () const noexcept

  Return the batch axis. Return 1 if no batch axis was set.
- void setSequenceAxis (int32_t sequenceAxis) noexcept

  Set the sequence axis. Default is 0.
- int32_t getSequenceAxis () const noexcept

  Return the sequence axis. Return 0 if no sequence axis was set.

Protected Member Functions

- virtual ~IReverseSequenceLayer () noexcept=default

Protected Attributes

- apiv::VReverseSequenceLayer * mImpl

9.151.1 Detailed Description

A ReverseSequence layer in a network definition.

This layer performs batch-wise reversal, which slices the input tensor along the axis batchAxis. For the i-th slice, the operation reverses the first N elements, specified by the corresponding i-th value in sequenceLens, along sequenceAxis and keeps the remaining elements unchanged. The output tensor will have the same shape as the input tensor.

> **Warning**
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.151.2 Constructor & Destructor Documentation

9.151.2.1 ~IReverseSequenceLayer()

```
virtual nvinfer1::IReverseSequenceLayer::~IReverseSequenceLayer ( )  [protected], [virtual], [default],
[noexcept]
```

9.151.3 Member Function Documentation

9.151.3.1 getBatchAxis()

```
int32_t nvinfer1::IReverseSequenceLayer::getBatchAxis ( ) const  [inline], [noexcept]
```

Return the batch axis. Return 1 if no batch axis was set.

See also

getBatchAxis()

9.151.3.2 getSequenceAxis()

```
int32_t nvinfer1::IReverseSequenceLayer::getSequenceAxis ( ) const  [inline], [noexcept]
```

Return the sequence axis. Return 0 if no sequence axis was set.

See also

getSequenceAxis()

9.151.3.3 setBatchAxis()

```
void nvinfer1::IReverseSequenceLayer::setBatchAxis (
            int32_t batchAxis )  [inline], [noexcept]
```

Set the batch axis. Default is 1.

batchAxis should be between zero (inclusive) and the rank of input (exclusive), and different from sequenceAxis. Otherwise, ErrorCode::kINVALID_ARGUMENT will be triggered.

See also

setBatchAxis()

9.151.3.4 setSequenceAxis()

```
void nvinfer1::IReverseSequenceLayer::setSequenceAxis (
            int32_t sequenceAxis )  [inline], [noexcept]
```

Set the sequence axis. Default is 0.

sequenceAxis should be between zero (inclusive) and the rank of input (exclusive), and different from batchAxis. Otherwise, ErrorCode::kINVALID_ARGUMENT will be triggered.

See also

setSequenceAxis()

9.151.4 Member Data Documentation

9.151.4.1 mImpl

```
apiv::VReverseSequenceLayer* nvinfer1::IReverseSequenceLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

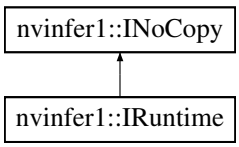
- NvInfer.h

9.152 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
- void setDLACore (int32_t dlaCore) noexcept

  Sets the DLA core used by the network. Defaults to -1.
- int32_t getDLACore () const noexcept

  Get the DLA core that the engine executes on.
- int32_t getNbDLACores () const noexcept

  Returns number of DLA hardware cores accessible or 0 if DLA is unavailable.
- void setGpuAllocator (IGpuAllocator *allocator) noexcept

  Set the GPU allocator.
- void setErrorRecorder (IErrorRecorder *recorder) noexcept

  Set the ErrorRecorder for this interface.
- IErrorRecorder * getErrorRecorder () const noexcept

  get the ErrorRecorder assigned to this interface.
- ICudaEngine * deserializeCudaEngine (void const *blob, std::size_t size) noexcept

  Deserialize an engine from host memory.
- TRT_DEPRECATED ICudaEngine * deserializeCudaEngine (IStreamReader &streamReader)

Deserialize an engine from a stream.

- ICudaEngine * deserializeCudaEngine (IStreamReaderV2 &streamReader)

  Deserialize an engine from a stream. IStreamReaderV2 is expected to support reading to both host and device pointers.
- ILogger * getLogger () const noexcept

  get the logger with which the runtime was created
- bool setMaxThreads (int32_t maxThreads) noexcept

  Set the maximum number of threads.
- int32_t getMaxThreads () const noexcept

  Get the maximum number of threads that can be used by the runtime.
- void setTemporaryDirectory (char const *path) noexcept

  Set the directory that will be used by this runtime for temporary files.
- char const * getTemporaryDirectory () const noexcept

  Get the directory that will be used by this runtime for temporary files.
- void setTempfileControlFlags (TempfileControlFlags flags) noexcept

  Set the tempfile control flags for this runtime.
- TempfileControlFlags getTempfileControlFlags () const noexcept

  Get the tempfile control flags for this runtime.
- IPluginRegistry & getPluginRegistry () noexcept

  Get the local plugin registry that can be used by the runtime.
- IRuntime * loadRuntime (char const *path) noexcept

  Load IRuntime from the file.
- void setEngineHostCodeAllowed (bool allowed) noexcept

  Set whether the runtime is allowed to deserialize engines with host executable code.
- bool getEngineHostCodeAllowed () const noexcept

  Get whether the runtime is allowed to deserialize engines with host executable code.

Protected Attributes

- apiv::VRuntime * mImpl

Additional Inherited Members

9.152.1 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.

9.152.2 Constructor & Destructor Documentation

9.152.2.1 ~IRuntime()

```
virtual nvinfer1::IRuntime::~IRuntime ( )  [virtual], [default], [noexcept]
```

9.152.3 Member Function Documentation

9.152.3.1 deserializeCudaEngine() [1/3]

```
TRT_DEPRECATED ICudaEngine * nvinfer1::IRuntime::deserializeCudaEngine (
            IStreamReader & streamReader )  [inline]
```

Deserialize an engine from a stream.

If an error recorder has been set for the runtime, it will also be passed to the engine.

This deserialization path will reduce host memory usage when weight streaming is enabled.

Parameters

streamReader	a read-only stream from which TensorRT will deserialize a previously serialized engine.

Returns

The engine, or nullptr if it could not be deserialized.

Deprecated Deprecated in TensorRT 10.7. Superseded by deserializeCudaEngine that takes an IStreamReaderV2 instead of IStreamReader.

9.152.3.2 deserializeCudaEngine() [2/3]

```
ICudaEngine * nvinfer1::IRuntime::deserializeCudaEngine (
            IStreamReaderV2 & streamReader )  [inline]
```

Deserialize an engine from a stream. IStreamReaderV2 is expected to support reading to both host and device pointers.

If an error recorder has been set for the runtime, it will also be passed to the engine.

This deserialization path will reduce engine load time when applied with GDS (GPU Direct storage), or when weight streaming is enabled.

Parameters

streamReader	a read-only stream from which TensorRT will deserialize a previously serialized engine.

Returns

The engine, or nullptr if it could not be deserialized. The pointer may not be valid immediately after the function returns.

9.152.3.3 deserializeCudaEngine() [3/3]

```
ICudaEngine * nvinfer1::IRuntime::deserializeCudaEngine (
            void const * blob,
            std::size_t size )  [inline], [noexcept]
```

Deserialize an engine from host memory.

If an error recorder has been set for the runtime, it will also be passed to the engine.

Parameters

blob	The memory that holds the serialized engine.
size	The size of the memory.

Returns

The engine, or nullptr if it could not be deserialized.

9.152.3.4 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.

9.152.3.5 getEngineHostCodeAllowed()

```
bool nvinfer1::IRuntime::getEngineHostCodeAllowed ( ) const  [inline], [noexcept]
```

Get whether the runtime is allowed to deserialize engines with host executable code.

Returns

Whether the runtime is allowed to deserialize engines with host executable code.

9.152.3.6 getErrorRecorder()

```
IErrorRecorder * nvinfer1::IRuntime::getErrorRecorder ( ) const  [inline], [noexcept]
```

get the ErrorRecorder assigned to this interface.

Retrieves the assigned error recorder object for the given class. A nullptr will be returned if an error handler has not been set.

Returns

A pointer to the IErrorRecorder object that has been registered.

See also

setErrorRecorder()

9.152.3.7 getLogger()

```
ILogger * nvinfer1::IRuntime::getLogger ( ) const  [inline], [noexcept]
```

get the logger with which the runtime was created

Returns

the logger

9.152.3.8 getMaxThreads()

```
int32_t nvinfer1::IRuntime::getMaxThreads ( ) const  [inline], [noexcept]
```

Get the maximum number of threads that can be used by the runtime.

Retrieves the maximum number of threads that can be used by the runtime.

Returns

The maximum number of threads that can be used by the runtime.

See also

setMaxThreads()

9.152.3.9 getNbDLACores()

```
int32_t nvinfer1::IRuntime::getNbDLACores ( ) const  [inline], [noexcept]
```

Returns number of DLA hardware cores accessible or 0 if DLA is unavailable.

9.152.3.10 getPluginRegistry()

```
IPluginRegistry & nvinfer1::IRuntime::getPluginRegistry ( )  [inline], [noexcept]
```

Get the local plugin registry that can be used by the runtime.

Returns

The local plugin registry that can be used by the runtime.

9.152.3.11 getTempfileControlFlags()

```
TempfileControlFlags nvinfer1::IRuntime::getTempfileControlFlags ( ) const  [inline], [noexcept]
```

Get the tempfile control flags for this runtime.

Returns

The flags currently set.

See also

TempfileControlFlag, TempfileControlFlags, setTempfileControlFlags()

9.152.3.12 getTemporaryDirectory()

```
char const * nvinfer1::IRuntime::getTemporaryDirectory ( ) const  [inline], [noexcept]
```

Get the directory that will be used by this runtime for temporary files.

Returns

A path to the temporary directory in use, or nullptr if no path is specified.

See also

setTemporaryDirectory()

9.152.3.13 loadRuntime()

```
IRuntime * nvinfer1::IRuntime::loadRuntime (
            char const * path )  [inline], [noexcept]
```

Load IRuntime from the file.

This method loads a runtime library from a shared library file. The runtime can then be used to execute a plan file built with BuilderFlag::kVERSION_COMPATIBLE and BuilderFlag::kEXCLUDE_LEAN_RUNTIME both set and built with the same version of TensorRT as the loaded runtime library.

Parameters

path	Path to the runtime lean library.

Returns

the runtime library, or nullptr if it could not be loaded

Warning

The path string must be null-terminated, and be at most 4096 bytes including the terminator.

9.152.3.14 setDLACore()

```
void nvinfer1::IRuntime::setDLACore (
            int32_t dlaCore )  [inline], [noexcept]
```

Sets the DLA core used by the network. Defaults to -1.

Parameters

dlaCore	The DLA core to execute the engine on, in the range [0,getNbDlaCores()).

This function is used to specify which DLA core to use via indexing, if multiple DLA cores are available.

Warning

if getNbDLACores() returns 0, then this function does nothing.

See also

getDLACore()

9.152.3.15 setEngineHostCodeAllowed()

```
void nvinfer1::IRuntime::setEngineHostCodeAllowed (
            bool allowed )  [inline], [noexcept]
```

Set whether the runtime is allowed to deserialize engines with host executable code.

Parameters

allowed	Whether the runtime is allowed to deserialize engines with host executable code.

The default value is false.

9.152.3.16 setErrorRecorder()

```
void nvinfer1::IRuntime::setErrorRecorder (
            IErrorRecorder * recorder )  [inline], [noexcept]
```

Set the ErrorRecorder for this interface.

Assigns the ErrorRecorder to this interface. The ErrorRecorder will track all errors during execution. This function will call incRefCount of the registered ErrorRecorder at least once. Setting recorder to nullptr unregisters the recorder with the interface, resulting in a call to decRefCount if a recorder has been registered.

If an error recorder is not set, messages will be sent to the global log stream.

Parameters

recorder	The error recorder to register with this interface.

See also

getErrorRecorder()

9.152.3.17 setGpuAllocator()

```
void nvinfer1::IRuntime::setGpuAllocator (
            IGpuAllocator * allocator )  [inline], [noexcept]
```

Set the GPU allocator.

Parameters

allocator	Set the GPU allocator to be used by the runtime. All GPU memory acquired will use this allocator. If NULL is passed, the default allocator will be used.

Default: allocateAsync uses cudaMallocAsync if cudaDevAttrMemoryPoolsSupported returns true, otherwise falls back to cudaMalloc. allocate always uses cudaMalloc.

If nullptr is passed, the default allocator will be used.

9.152.3.18 setMaxThreads()

```
bool nvinfer1::IRuntime::setMaxThreads (
            int32_t maxThreads )  [inline], [noexcept]
```

Set the maximum number of threads.

Parameters

maxThreads	The maximum number of threads that can be used by the runtime.

Returns

True if successful, false otherwise.

The default value is 1 and includes the current thread. A value greater than 1 permits TensorRT to use multi-threaded algorithms. A value less than 1 triggers a kINVALID_ARGUMENT error.

9.152.3.19 setTempfileControlFlags()

```
void nvinfer1::IRuntime::setTempfileControlFlags (
            TempfileControlFlags flags )  [inline], [noexcept]
```

Set the tempfile control flags for this runtime.

Parameters

flags	The flags to set.

The default value is all flags set, i.e.

(1U << static_cast<uint32_t>(kALLOW_IN_MEMORY_FILES)) | (1U << static_cast<uint32_t>(kALLOW_↩ TEMPORARY_FILES))

See also

TempfileControlFlag, TempfileControlFlags, getTempfileControlFlags()

9.152.3.20 setTemporaryDirectory()

```
void nvinfer1::IRuntime::setTemporaryDirectory (
             char const * path )  [inline], [noexcept]
```

Set the directory that will be used by this runtime for temporary files.

On some platforms the TensorRT runtime may need to create and use temporary files with read/write/execute permissions to implement runtime functionality.

Parameters

path	Path to the temporary directory for use, or nullptr.

If path is nullptr, then TensorRT will use platform-specific heuristics to pick a default temporary directory if required:

- On UNIX/Linux platforms, TensorRT will first try the TMPDIR environment variable, then fall back to /tmp
- On Windows, TensorRT will try the TEMP environment variable.

See the TensorRT Developer Guide for more information.

The default value is nullptr.

Warning

If path is not nullptr, it must be a non-empty string representing a relative or absolute path in the format expected by the host operating system.

The string path must be null-terminated, and be at most 4096 bytes including the terminator. Note that the operating system may have stricter path length requirements.

The process using TensorRT must have rwx permissions for the temporary directory, and the directory shall be configured to disallow other users from modifying created files (e.g. on Linux, if the directory is shared

with other users, the sticky bit must be set).

See also

getTemporaryDirectory()

9.152.4 Member Data Documentation

9.152.4.1 mImpl

```
apiv::VRuntime* nvinfer1::IRuntime::mImpl  [protected]
```

The documentation for this class was generated from the following file:

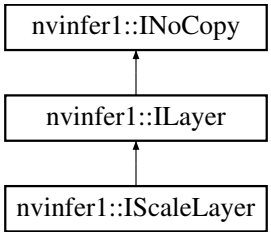
- NvInferRuntime.h

9.153 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

9.153.1 Detailed Description

A Scale layer in a network definition.

This layer applies a per-element computation to its input:

`output = (input* scale + shift)^ power`

The coefficients can be applied on a per-tensor, per-channel, or per-element basis.

Note

If the number of weights is 0, then a default value is used for shift, power, and scale. The default shift is 0, the default power is 1, and the default scale is 1.

The output size is the same as the input size.

Note

The input tensor is required to have at least 4 dimensions.

A scale layer may be used as an INT8 quantization node in a graph, if the output is constrained to INT8 and the input to FP32. Quantization rounds ties to even, and clamps to [-128, 127].

See also

ScaleMode

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.153.2 Constructor & Destructor Documentation

9.153.2.1 ~IScaleLayer()

```
virtual nvinfer1::IScaleLayer::~IScaleLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.153.3 Member Function Documentation

9.153.3.1 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()

9.153.3.2 getMode()

```
ScaleMode nvinfer1::IScaleLayer::getMode ( ) const  [inline], [noexcept]
```

Get the scale mode.

See also

setMode()

9.153.3.3 getPower()

```
Weights nvinfer1::IScaleLayer::getPower ( ) const  [inline], [noexcept]
```

Get the power value.

See also

setPower()

9.153.3.4 getScale()

```
Weights nvinfer1::IScaleLayer::getScale ( ) const  [inline], [noexcept]
```

Get the scale value.

See also

setScale()

9.153.3.5 getShift()

```
Weights nvinfer1::IScaleLayer::getShift ( ) const  [inline], [noexcept]
```

Get the shift value.

See also

setShift()

9.153.3.6 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()

9.153.3.7 setMode()

```
void nvinfer1::IScaleLayer::setMode (
            ScaleMode mode )  [inline], [noexcept]
```

Set the scale mode.

See also

getMode()

9.153.3.8 setPower()

```
void nvinfer1::IScaleLayer::setPower (
            Weights power )  [inline], [noexcept]
```

Set the power value.

See also

getPower()

9.153.3.9 setScale()

```
void nvinfer1::IScaleLayer::setScale (
             Weights scale )  [inline], [noexcept]
```

Set the scale value.

See also

getScale()

9.153.3.10 setShift()

```
void nvinfer1::IScaleLayer::setShift (
             Weights shift )  [inline], [noexcept]
```

Set the shift value.

See also

getShift()

9.153.4 Member Data Documentation

9.153.4.1 mImpl

```
apiv::VScaleLayer* nvinfer1::IScaleLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

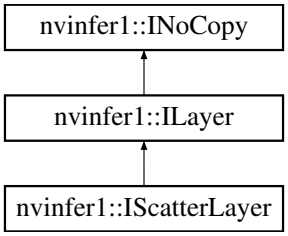
- NvInfer.h

9.154 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

9.154.1 Detailed Description

A scatter layer in a network definition. Supports several kinds of scattering.

The Scatter layer has three input tensors: Data, Indices, and Updates, one output tensor Output, and a scatter mode. When kELEMENT mode is used an optional axis parameter is available.

- Data is a tensor of rank r >= 1 that stores the values to be duplicated in Output.
- Indices is a tensor of rank q that determines which locations in Output to write new values to. Constraints on the rank q depend on the mode: ScatterMode::kND: q >= 1 ScatterMode::kELEMENT: q must be the same as r
- Updates is a tensor of rank s >= 1 that provides the data to write to Output specified by its corresponding location in Indices. Constraints on the rank of Updates depend on the mode: ScatterMode::kND: s = r + q - shape(Indices)[-1] - 1 Scattermode::kELEMENT: s = q = r
- Output is a tensor with the same dimensions as Data that stores the resulting values of the transformation. It must not be a shape tensor. The types of Data, Update, and Output shall be the same, and Indices shall be of type DataType::kINT32 or DataType::kINT64.

The output is computed by copying the data, and then updating elements of it based on indices. How Indices are interpreted depends upon the ScatterMode.

ScatterMode::kND

```
The indices are interpreted as a tensor of rank q-1 of indexing tuples.
The axis parameter is ignored.

Given that data dims are {d_0,...,d_{r-1}} and indices dims are {i_0,...,i_{q-1}},
define k = indices[q-1], it follows that updates dims are {i_0,...,i_{q-2},d_k,...,d_{r-1}}
The updating can be computed by:
    foreach slice in indices[i_0,...,i_{q-2}]
        output[indices[slice]] = updates[slice]
```

ScatterMode::kELEMENT

```
Here "axis" denotes the result of getAxis().

For each element X of indices:
    Let J denote a sequence for the subscripts of X
    Let K = sequence J with element [axis] replaced by X
    output[K] = updates[J]

For example, if indices has dimensions [N,C,H,W] and axis is 2, then the updates happen as:

    for n in [0,n)
        for c in [0,n)
            for h in [0,n)
                for w in [0,n)
                    output[n,c,indices[n,c,h,w],w] = updates[n,c,h,w]
```

Writes to the same output element cause undefined behavior.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.154.2 Constructor & Destructor Documentation

9.154.2.1 ~IScatterLayer()

```
virtual nvinfer1::IScatterLayer::~IScatterLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.154.3 Member Function Documentation

9.154.3.1 getAxis()

```
int32_t nvinfer1::IScatterLayer::getAxis ( ) const  [inline], [noexcept]
```

Get the axis.

9.154.3.2 getMode()

```
ScatterMode nvinfer1::IScatterLayer::getMode ( ) const  [inline], [noexcept]
```

Get the scatter mode.

See also

setMode()

9.154.3.3 setAxis()

```
void nvinfer1::IScatterLayer::setAxis (
            int32_t axis )  [inline], [noexcept]
```

Set the axis used by ScatterMode::kELEMENTS.

The axis defaults to 0.

9.154.3.4 setMode()

```
void nvinfer1::IScatterLayer::setMode (
            ScatterMode mode )  [inline], [noexcept]
```

Set the scatter mode.

See also

getMode()

9.154.4 Member Data Documentation

9.154.4.1 mImpl

```
apiv::VScatterLayer* nvinfer1::IScatterLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

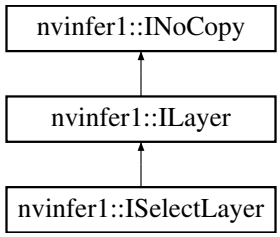
- NvInfer.h

9.155 nvinfer1::ISelectLayer Class Reference

Select elements from two data tensors based on a condition tensor.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::ISelectLayer:



Protected Member Functions

- virtual ~ISelectLayer () noexcept=default

Protected Attributes

- apiv::VSelectLayer * mImpl

Additional Inherited Members

9.155.1 Detailed Description

Select elements from two data tensors based on a condition tensor.

The select layer makes elementwise selections from two data tensors based on a condition tensor, behaving similarly to the numpy.where function with three parameters. The three input tensors must share the same rank. Multidirectional broadcasting is supported. The output tensor has the dimensions of the inputs AFTER applying the broadcast rule.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.155.2 Constructor & Destructor Documentation

9.155.2.1 ~ISelectLayer()

`virtual nvinfer1::ISelectLayer::~ISelectLayer ( ) [protected], [virtual], [default], [noexcept]`

9.155.3 Member Data Documentation

9.155.3.1 mImpl

```
apiv::VSelectLayer* nvinfer1::ISelectLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

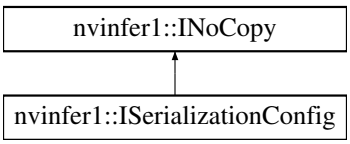
- NvInfer.h

9.156 nvinfer1::ISerializationConfig Class Reference

Holds properties for configuring an engine to serialize the binary.

```
#include <NvInferRuntime.h>
```

Inheritance diagram for nvinfer1::ISerializationConfig:



Public Member Functions

- virtual ~ISerializationConfig () noexcept=default
- bool setFlags (SerializationFlags serializationFlags) noexcept

  Set the serialization flags to turn on for this config.
- SerializationFlags getFlags () const noexcept

  Get the serialization flags for this config.
- bool clearFlag (SerializationFlag serializationFlag) noexcept

  clear a serialization flag.
- bool setFlag (SerializationFlag serializationFlag) noexcept

  Set a serialization flag.
- bool getFlag (SerializationFlag serializationFlag) const noexcept

  Returns true if the serialization flag is set.

Protected Attributes

- apiv::VSerializationConfig * mImpl

Additional Inherited Members

9.156.1 Detailed Description

Holds properties for configuring an engine to serialize the binary.

See also

SerializationFlag

9.156.2 Constructor & Destructor Documentation

9.156.2.1 ~ISerializationConfig()

```
virtual nvinfer1::ISerializationConfig::~ISerializationConfig ( )  [virtual], [default], [noexcept]
```

9.156.3 Member Function Documentation

9.156.3.1 clearFlag()

```
bool nvinfer1::ISerializationConfig::clearFlag (
            SerializationFlag serializationFlag )  [inline], [noexcept]
```

clear a serialization flag.

clears the serialization flag from the config.

See also

setFlags()

9.156.3.2 getFlag()

```
bool nvinfer1::ISerializationConfig::getFlag (
            SerializationFlag serializationFlag ) const  [inline], [noexcept]
```

Returns true if the serialization flag is set.

See also

getFlags()

Returns

True if flag is set, false if unset.

9.156.3.3 getFlags()

```
SerializationFlags nvinfer1::ISerializationConfig::getFlags ( ) const  [inline], [noexcept]
```

Get the serialization flags for this config.

Returns

The serialization flags as a bitmask.

See also

setFlags()

9.156.3.4 setFlag()

```
bool nvinfer1::ISerializationConfig::setFlag (
            SerializationFlag serializationFlag )  [inline], [noexcept]
```

Set a serialization flag.

Add the input serialization flag to the already enabled flags.

See also

setFlags()

9.156.3.5 setFlags()

```
bool nvinfer1::ISerializationConfig::setFlags (
            SerializationFlags serializationFlags )  [inline], [noexcept]
```

Set the serialization flags to turn on for this config.

The flags are listed in the SerializationFlag enum.

Parameters

serializationFlags	The serialization flags for an engine.

Note

This function will override the previous set flags, rather than bitwise ORing the new flag.

See also

getFlags()

9.156.4 Member Data Documentation

9.156.4.1 mImpl

```
apiv::VSerializationConfig* nvinfer1::ISerializationConfig::mImpl  [protected]
```

The documentation for this class was generated from the following file:

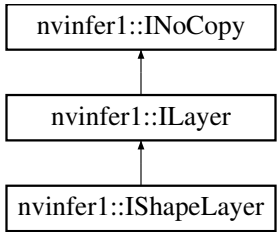
- NvInferRuntime.h

9.157 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

9.157.1 Detailed Description

Layer type for getting shape of a tensor.

This layer sets the output to a 1D tensor of type Int64 with the dimensions of the input tensor.

For example, if the input is a four-dimensional tensor (of any type) with dimensions [2,3,5,7], the output tensor is a one-dimensional Int64 tensor of length 4 containing the sequence 2, 3, 5, 7.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.157.2 Constructor & Destructor Documentation

9.157.2.1 ~IShapeLayer()

```
virtual nvinfer1::IShapeLayer::~IShapeLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.157.3 Member Data Documentation

9.157.3.1 mImpl

```
apiv::VShapeLayer* nvinfer1::IShapeLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

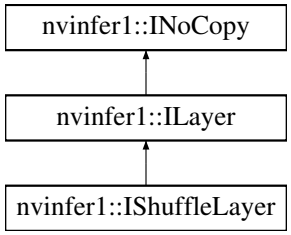
- NvInfer.h

9.158 nvinfer1::IShuffleLayer Class Reference

Layer type for shuffling data.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IShuffleLayer:



Public Member Functions

- void setFirstTranspose (Permutation permutation) noexcept

  Set the permutation applied by the first transpose operation.
- Permutation getFirstTranspose () const noexcept

  Get the permutation applied by the first transpose operation.
- void setReshapeDimensions (Dims const &dimensions) noexcept

  Set the reshaped dimensions.
- Dims getReshapeDimensions () const noexcept

  Get the reshaped dimensions.
- void setSecondTranspose (Permutation permutation) noexcept

  Set the permutation applied by the second transpose operation.
- Permutation getSecondTranspose () const noexcept

  Get the permutation applied by the second transpose operation.
- void setZeroIsPlaceholder (bool zeroIsPlaceholder) noexcept

  Set meaning of 0 in reshape dimensions.
- bool getZeroIsPlaceholder () const noexcept

  Get meaning of 0 in reshape dimensions.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~IShuffleLayer () noexcept=default

Protected Attributes

- apiv::VShuffleLayer * mImpl

9.158.1 Detailed Description

Layer type for shuffling data.

This layer shuffles data by applying in sequence: a transpose operation, a reshape operation and a second transpose operation. The dimension types of the output are those of the reshape dimension.

The layer has an optional second input. If present, it must be a 1D tensor of type Int32 or Int64, and the reshape dimensions are taken from it.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.158.2 Constructor & Destructor Documentation

9.158.2.1 ~IShuffleLayer()

```
virtual nvinfer1::IShuffleLayer::~IShuffleLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.158.3 Member Function Documentation

9.158.3.1 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

9.158.3.2 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.

9.158.3.3 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

9.158.3.4 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

9.158.3.5 setFirstTranspose()

```
void nvinfer1::IShuffleLayer::setFirstTranspose (
            Permutation permutation )  [inline], [noexcept]
```

Set the permutation applied by the first transpose operation.

Parameters

permutation	The dimension permutation applied before the reshape.

The default is the identity permutation.

See also

getFirstTranspose

9.158.3.6 setInput()

```
void nvinfer1::ILayer::setInput (
             int32_t index,
             ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor Sets the input tensor for the given index. The index must be 0 for a static shuffle layer. A static shuffle layer is converted to a dynamic shuffle layer by calling setInput with an index 1. A dynamic shuffle layer cannot be converted back to a static shuffle layer.

For a dynamic shuffle layer, the values 0 and 1 are valid. The indices in the dynamic case are as follows:

- 0: Data or Shape tensor to be shuffled.
- 1: The dimensions for the reshape operation, as a 1D tensor of type Int32 or Int64.

If this function is called with the value 1, then the function getNbInputs() changes from returning 1 to 2.

The reshape dimensions are treated identically to how they are treated if set statically via setReshapeDimensions. In particular, a -1 is treated as a wildcard even if dynamically supplied at runtime, and a 0 is treated as a placeholder if getZeroIsPlaceholder() = true, which is the default. If the placeholder interpretation of 0 is unwanted because the runtime dimension should be 0 when the reshape dimension is 0, be sure to call setZeroIsPlacholder(false) on the IShuffleLayer.

See also

setReshapeDimensions.

9.158.3.7 setReshapeDimensions()

```
void nvinfer1::IShuffleLayer::setReshapeDimensions (
            Dims const & dimensions )  [inline], [noexcept]
```

Set the reshaped dimensions.

Parameters

dimensions	The reshaped dimensions.

Two special values can be used as dimensions.

Value 0 copies the corresponding dimension from input. This special value can be used more than once in the dimensions. If number of reshape dimensions is less than input, 0s are resolved by aligning the most significant dimensions of input.

Value -1 infers that particular dimension by looking at input and rest of the reshape dimensions. Note that only a maximum of one dimension is permitted to be specified as -1. Avoid using -1 if the input can have zero volume and any of the other reshape dimensions can be zero (after resolving special treatment of 0), because the solution for the -1 becomes indeterminate and TensorRT will report an error.

The product of the new dimensions must be equal to the product of the old.

If a second input had been used to create this layer, that input is reset to null by this method.

9.158.3.8 setSecondTranspose()

```
void nvinfer1::IShuffleLayer::setSecondTranspose (
            Permutation permutation )  [inline], [noexcept]
```

Set the permutation applied by the second transpose operation.

Parameters

permutation	The dimension permutation applied after the reshape.

The default is the identity permutation.

The permutation is applied as outputDimensionIndex = permutation.order[inputDimensionIndex], so to permute from CHW order to HWC order, the required permutation is [1, 2, 0].

See also

getSecondTranspose

9.158.3.9 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();

9.158.4 Member Data Documentation

9.158.4.1 mImpl

```
apiv::VShuffleLayer* nvinfer1::IShuffleLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

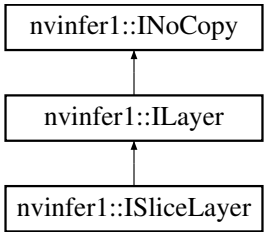
- NvInfer.h

9.159 nvinfer1::ISliceLayer Class Reference

Slices an input tensor into an output tensor based on the offset and strides.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::ISliceLayer:



Public Member Functions

- void setStart (Dims const &start) noexcept

  Set the start offset that the slice layer uses to create the output slice.
- Dims getStart () const noexcept

  Get the start offset for the slice layer.
- void setSize (Dims const &size) noexcept

  Set the dimensions of the output slice.
- Dims getSize () const noexcept

  Get dimensions of the output slice.
- void setStride (Dims const &stride) noexcept

  Set the stride for computing the output slice data.
- Dims getStride () const noexcept

  Get the stride for the output slice.
- void setMode (SampleMode mode) noexcept

  Set the slice mode.
- SampleMode getMode () const noexcept

  Get the slice mode.
- void setAxes (Dims const &axes) noexcept

  Set the axes for this ISliceLayer.
- Dims getAxes () const noexcept

  Get the axes for this ISliceLayer.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~ISliceLayer () noexcept=default

Protected Attributes

- apiv::VSliceLayer * mImpl

9.159.1 Detailed Description

Slices an input tensor into an output tensor based on the offset and strides.

The slice layer has two variants, static and dynamic. Static slice specifies the start, size, and stride dimensions at layer creation time via Dims and can use the get/set accessor functions of the ISliceLayer. Static slice layers can also optionally specify axes through the get/set accessor functions of the ISliceLayer. Dynamic slice specifies one or more of start, size, stride, or axes as ITensors, by using ILayer::setInput to add a second, third, fourth, or sixth input respectively. The corresponding Dims are used if an input is missing or null.

An application can determine if the ISliceLayer has a dynamic output shape based on whether the size or axes input is present and non-null.

The slice layer selects for each dimension a start location from within the input tensor, and copies elements to the output tensor using the specified stride across the input tensor. Start, size, and stride tensors must be 1D tensors of type Int32 or Int64 if not specified via Dims.

An example of using slice on a tensor: input = {{0, 2, 4}, {1, 3, 5}} start = {1, 0} size = {1, 2} stride = {1, 2} output = {{1, 5}}

If axes are provided then starts, ends, and strides must have the same length as axes and specifies a subset of dimensions to slice. If axes are not provided, starts, ends, and strides must be of the same length as the rank of the input tensor.

An example of using slice on a tensor with axes specified: input = {{0, 2, 4}, {1, 3, 5}} start = {1} size = {2} stride = {1} axes = {1} output = {{2, 4}, {3, 5}}

When the sampleMode is kCLAMP or kREFLECT, for each input dimension, if its size is 0 then the corresponding output dimension must be 0 too.

When the sampleMode is kFILL, the fifth input to the slice layer is used to determine the value to fill in out-of-bound indices. It is an error to specify the fifth input in any other sampleMode.

A slice layer can produce a shape tensor if the following conditions are met:

- start, size, and stride are build time constants, either as static Dims or as constant input tensors.
- axes, if provided, are build time constants, either as static Dims or as a constant input tensor.
- The number of elements in the output tensor does not exceed 2 ∗ Dims::MAX_DIMS.

The input tensor is a shape tensor if the output is a shape tensor.

The following constraints must be satisfied to execute this layer on DLA:

- start, size, and stride are build time constants, either as static Dims or as constant input tensors.
- axes, if provided, are build time constants, either as static Dims or as a constant input tensor.
- sampleMode is kDEFAULT, kWRAP, or kFILL.
- Strides are 1 for all dimensions.
- Slicing is not performed on the first dimension.
- The input tensor has four dimensions.
- For kFILL sliceMode, the fill value input is a scalar output of an IConstantLayer with value 0 that is not consumed by any other layer.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.159.2 Constructor & Destructor Documentation

9.159.2.1 ~ISliceLayer()

```
virtual nvinfer1::ISliceLayer::~ISliceLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.159.3 Member Function Documentation

9.159.3.1 getAxes()

```
Dims nvinfer1::ISliceLayer::getAxes ( ) const  [inline], [noexcept]
```

Get the axes for this ISliceLayer.

Returns

The axes on which the starts, ends, and strides parameters of this slice apply to.

If the sixth input is present and non-null, this function returns a Dims with nbDims = -1.

See also

setAxes

9.159.3.2 getMode()

```
SampleMode nvinfer1::ISliceLayer::getMode ( ) const  [inline], [noexcept]
```

Get the slice mode.

See also

setMode()

9.159.3.3 getSize()

```
Dims nvinfer1::ISliceLayer::getSize ( ) const  [inline], [noexcept]
```

Get dimensions of the output slice.

Returns

The output dimension, or an invalid Dims structure.

If the third input is present and non-null, this function returns a Dims with nbDims = -1.

See also

setSize

9.159.3.4 getStart()

```
Dims nvinfer1::ISliceLayer::getStart ( ) const  [inline], [noexcept]
```

Get the start offset for the slice layer.

Returns

The start offset, or an invalid Dims structure.

If the second input is present and non-null, this function returns a Dims with nbDims = -1.

See also

setStart

9.159.3.5 getStride()

```
Dims nvinfer1::ISliceLayer::getStride ( ) const  [inline], [noexcept]
```

Get the stride for the output slice.

Returns

The slicing stride, or an invalid Dims structure.

If the fourth input is present and non-null, this function returns a Dims with nbDims = -1.

See also

setStride

9.159.3.6 setAxes()

```
void nvinfer1::ISliceLayer::setAxes (
            Dims const & axes )  [inline], [noexcept]
```

Set the axes for this ISliceLayer.

Parameters

axes	The axes on which the starts, ends, and strides parameters of the slice apply to.

If a sixth input had been used to create this layer, that input is reset to null by this method.

See also

getAxes

9.159.3.7 setInput()

```
void nvinfer1::ILayer::setInput (
             int32_t index,
             ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	the index of the input to modify.
tensor	the new input tensor

For a slice layer, the values 0-5 are valid. The indices are as follows:

- 0: Tensor to be sliced.
- 1: The start tensor to begin slicing, as a 1D tensor of type Int32 or Int64.
- 2: The size tensor of the resulting slice, as a 1D tensor of type Int32 or Int64.
- 3: The stride of the slicing operation, as a 1D tensor of type Int32 or Int64.
- 4: Value for the kFILL slice mode. The fill value data type should either be the same or be implicitly convertible to the input data type. Implicit data type conversion is supported among kFLOAT, kHALF, kINT8, and kFP8 data types. This input is disallowed for other modes.
- 5: The axes tensor indicating the corresponding axes that start, size, and stride should apply to, as a 1D tensor or type Int32 or Int64. Negative values for axes indicate indexing from the back of the input tensor. Values must be unique and be within the interval of [-rank(input), rank(input)-1].

Using the corresponding setter resets the input to null.

If this function is called with a value greater than 0, then the function getNbInputs() changes from returning 1 to index + 1.

9.159.3.8 setMode()

```
void nvinfer1::ISliceLayer::setMode (
            SampleMode mode )  [inline], [noexcept]
```

Set the slice mode.

See also

getMode()

9.159.3.9 setSize()

```
void nvinfer1::ISliceLayer::setSize (
            Dims const & size )  [inline], [noexcept]
```

Set the dimensions of the output slice.

Parameters

size	The dimensions of the output slice.

If a third input had been used to create this layer, that input is reset to null by this method.

See also

getSize

9.159.3.10 setStart()

```
void nvinfer1::ISliceLayer::setStart (
            Dims const & start )  [inline], [noexcept]
```

Set the start offset that the slice layer uses to create the output slice.

Parameters

start	The start offset to read data from the input tensor.

If a second input had been used to create this layer, that input is reset to null by this method.

See also

getStart

9.159.3.11 setStride()

```
void nvinfer1::ISliceLayer::setStride (
            Dims const & stride )  [inline], [noexcept]
```

Set the stride for computing the output slice data.

Parameters

stride	The dimensions of the stride to compute the values to store in the output slice.

If a fourth input had been used to create this layer, that input is reset to null by this method.

See also

getStride

9.159.4 Member Data Documentation

9.159.4.1 mImpl

```
apiv::VSliceLayer* nvinfer1::ISliceLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

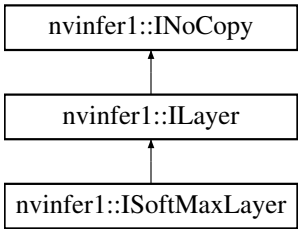
- NvInfer.h

9.160 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

9.160.1 Detailed Description

A Softmax layer in a network definition.

This layer applies a per-channel softmax to its input.

The output size is the same as the input size.

The following constraints must be satisfied to execute this layer on DLA:

- Axis must be one of the channel or spatial dimensions.
- There are two classes of supported input sizes:
  1. Non-axis, non-batch dimensions are all 1 and the axis dimension is at most 8192. This is the recommended case for using softmax since it is the most accurate.
  2. At least one non-axis, non-batch dimension greater than 1 and the axis dimension is at most 1024. Note that in this case, there may be some approximation error as the axis dimension size approaches the upper bound. See the TensorRT Developer Guide for more details on the approximation error.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.160.2 Constructor & Destructor Documentation

9.160.2.1 ~ISoftMaxLayer()

```
virtual nvinfer1::ISoftMaxLayer::~ISoftMaxLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.160.3 Member Function Documentation

9.160.3.1 getAxes()

```
uint32_t nvinfer1::ISoftMaxLayer::getAxes ( ) const  [inline], [noexcept]
```

Get the axis along which softmax occurs.

See also

setAxes()

9.160.3.2 setAxes()

```
void nvinfer1::ISoftMaxLayer::setAxes (
             uint32_t axes )  [inline], [noexcept]
```

Set the axis along which softmax is computed. Currently, only one axis can be set.

The axis is specified by setting the bit corresponding to the axis to 1. For example, consider an NCHW tensor as input.

Bit 0 corresponds to the N dimension boolean. Bit 1 corresponds to the C dimension boolean. Bit 2 corresponds to the H dimension boolean. Bit 3 corresponds to the W dimension boolean. By default, softmax is performed on the axis which is the number of axes minus three. It is 0 if there are fewer than 3 axes. For example, if the input is NCHW, the default axis is C. If the input is NHW, then the default axis is N.

For example, to perform softmax on axis R of a NPQRCHW input, set bit 3.

Parameters

axes	The axis along which softmax is computed. Here axes is a bitmap. For example, when doing softmax along axis 0, bit 0 is set to 1, axes = 1 << axis = 1.

9.160.4 Member Data Documentation

9.160.4.1 mImpl

```
apiv::VSoftMaxLayer* nvinfer1::ISoftMaxLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

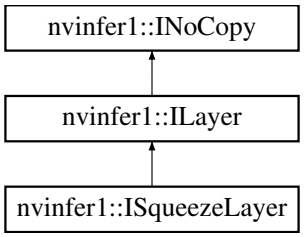
- NvInfer.h

9.161 nvinfer1::ISqueezeLayer Class Reference

Layer that represents a squeeze operation, removing unit dimensions of the input tensor on a set of axes.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::ISqueezeLayer:



Public Member Functions

- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~ISqueezeLayer () noexcept=default

Protected Attributes

- apiv::VSqueezeLayer * mImpl

9.161.1 Detailed Description

Layer that represents a squeeze operation, removing unit dimensions of the input tensor on a set of axes.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.161.2 Constructor & Destructor Documentation

9.161.2.1 ~ISqueezeLayer()

```
virtual nvinfer1::ISqueezeLayer::~ISqueezeLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.161.3 Member Function Documentation

9.161.3.1 setInput()

```
void nvinfer1::ILayer::setInput (
             int32_t index,
             ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	The index of the input to modify.
tensor	The new input tensor.

For a Squeeze layer, the values 0-1 are valid for index. The indices are as follows:

- 0: Input data tensor.
- 1: The axes to remove. Must resolvable to a constant Int32 or Int64 1D shape tensor.

9.161.4 Member Data Documentation

9.161.4.1 mImpl

```
apiv::VSqueezeLayer* nvinfer1::ISqueezeLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

- NvInfer.h

9.162 IStreamReader Class Reference

Application-implemented class for reading data in a stream-based manner.

```
#include <NvInferRuntime.h>
```

9.162.1 Detailed Description

Application-implemented class for reading data in a stream-based manner.

Note

To ensure compatibility of source code with future versions of TensorRT, use IStreamReader, not v_1_0::IStreamReader

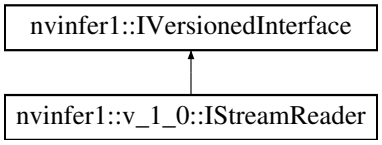
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.163 nvinfer1::v_1_0::IStreamReader Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IStreamReader:



Public Member Functions

- ~IStreamReader () override=default
- IStreamReader ()=default
- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual int64_t read (void *destination, int64_t nbBytes)=0

  Read the next number of bytes in the stream.

Protected Member Functions

- IStreamReader (IStreamReader const &)=default
- IStreamReader (IStreamReader &&)=default
- IStreamReader & operator= (IStreamReader const &) &=default
- IStreamReader & operator= (IStreamReader &&) &=default

9.163.1 Constructor & Destructor Documentation

9.163.1.1 ~IStreamReader()

```
nvinfer1::v_1_0::IStreamReader::~IStreamReader ( )  [override], [default]
```

TensorRT never calls the destructor for an IStreamReader defined by the application.

9.163.1.2 IStreamReader() [1/3]

```
nvinfer1::v_1_0::IStreamReader::IStreamReader ( )  [default]
```

9.163.1.3 IStreamReader() [2/3]

```
nvinfer1::v_1_0::IStreamReader::IStreamReader (
            IStreamReader const &  )  [protected], [default]
```

9.163.1.4 IStreamReader() [3/3]

```
nvinfer1::v_1_0::IStreamReader::IStreamReader (
            IStreamReader &&  )  [protected], [default]
```

9.163.2 Member Function Documentation

9.163.2.1 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IStreamReader::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.163.2.2 operator=() [1/2]

```
IStreamReader & nvinfer1::v_1_0::IStreamReader::operator= (
            IStreamReader &&  ) &  [protected], [default]
```

9.163.2.3 operator=() [2/2]

```
IStreamReader & nvinfer1::v_1_0::IStreamReader::operator= (
            IStreamReader const &  ) &  [protected], [default]
```

9.163.2.4 read()

```
virtual int64_t nvinfer1::v_1_0::IStreamReader::read (
            void * destination,
            int64_t nbBytes )  [pure virtual]
```

Read the next number of bytes in the stream.

Parameters

destination	The memory to write to
nbBytes	The number of bytes to read

Returns

The number of bytes read. Negative values will be considered an automatic error.

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.164 IStreamReaderV2 Class Reference

Application-implemented class for reading data in a stream-based manner asynchronously. Intended for use with the GDS API for optimizing load times.

`#include <NvInferRuntime.h>`

9.164.1 Detailed Description

Application-implemented class for reading data in a stream-based manner asynchronously. Intended for use with the GDS API for optimizing load times.

Note

To ensure compatibility of source code with future versions of TensorRT, use IStreamReaderV2, not v_1_0::↩ IStreamReaderV2

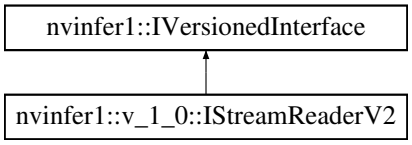
The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.165 nvinfer1::v_1_0::IStreamReaderV2 Class Reference

`#include <NvInferRuntime.h>`

Inheritance diagram for nvinfer1::v_1_0::IStreamReaderV2:



Public Member Functions

- ~IStreamReaderV2 () override=default
- IStreamReaderV2 ()=default
- InterfaceInfo getInterfaceInfo () const noexcept override

  Return version information associated with this interface. Applications must not override this method.
- virtual int64_t read (void *destination, int64_t nbBytes, cudaStream_t stream) noexcept=0

  Read the next number of bytes in the stream asynchronously.
- virtual bool seek (int64_t offset, SeekPosition where) noexcept=0

  Sets the position of the stream to the given offset.

Protected Member Functions

- IStreamReaderV2 (IStreamReaderV2 const &)=default
- IStreamReaderV2 (IStreamReaderV2 &&)=default
- IStreamReaderV2 & operator= (IStreamReaderV2 const &) &=default
- IStreamReaderV2 & operator= (IStreamReaderV2 &&) &=default

9.165.1 Constructor & Destructor Documentation

9.165.1.1 ~IStreamReaderV2()

`nvinfer1::v_1_0::IStreamReaderV2::~IStreamReaderV2 ( ) [override], [default]`

TensorRT never calls the destructor for an IStreamReaderV2 defined by the application.

9.165.1.2 IStreamReaderV2() [1/3]

`nvinfer1::v_1_0::IStreamReaderV2::IStreamReaderV2 ( ) [default]`

9.165.1.3 IStreamReaderV2() [2/3]

```
nvinfer1::v_1_0::IStreamReaderV2::IStreamReaderV2 (
            IStreamReaderV2 const &  )  [protected], [default]
```

9.165.1.4 IStreamReaderV2() [3/3]

```
nvinfer1::v_1_0::IStreamReaderV2::IStreamReaderV2 (
            IStreamReaderV2 &&  )  [protected], [default]
```

9.165.2 Member Function Documentation

9.165.2.1 getInterfaceInfo()

```
InterfaceInfo nvinfer1::v_1_0::IStreamReaderV2::getInterfaceInfo ( ) const  [inline], [override],
[virtual], [noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implements nvinfer1::IVersionedInterface.

9.165.2.2 operator=() [1/2]

```
IStreamReaderV2 & nvinfer1::v_1_0::IStreamReaderV2::operator= (
            IStreamReaderV2 &&  ) &  [protected], [default]
```

9.165.2.3 operator=() [2/2]

```
IStreamReaderV2 & nvinfer1::v_1_0::IStreamReaderV2::operator= (
            IStreamReaderV2 const &  ) &  [protected], [default]
```

9.165.2.4 read()

```
virtual int64_t nvinfer1::v_1_0::IStreamReaderV2::read (
            void * destination,
            int64_t nbBytes,
            cudaStream_t stream )  [pure virtual], [noexcept]
```

Read the next number of bytes in the stream asynchronously.

Parameters

destination	The memory to write to, call cudaPointerGetAttributes to get the memory location
nbBytes	The number of bytes to read
stream	The CUDA stream used to do the copy

Returns

The number of bytes read. Negative values indicate an unrecoverable error. A zero indicates that the end of the stream has been reached.

9.165.2.5 seek()

```
virtual bool nvinfer1::v_1_0::IStreamReaderV2::seek (
            int64_t offset,
            SeekPosition where )  [pure virtual], [noexcept]
```

Sets the position of the stream to the given offset.

Parameters

offset	The number of bytes to offset from where.
where	The position from where the offset is added.

See also

SeekPosition

Returns

True if the position is updated successfully.

The documentation for this class was generated from the following file:

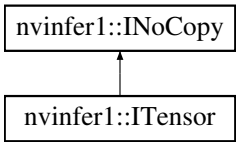
- NvInferRuntime.h

9.166 nvinfer1::ITensor Class Reference

A tensor in a network definition.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::ITensor:



Public Member Functions

- void setName (char const *name) noexcept

  Set the tensor name.
- char const * getName () const noexcept

  Get the tensor name.
- void setDimensions (Dims const &dimensions) noexcept

  Set the dimensions of a tensor.
- Dims getDimensions () const noexcept

  Get the dimensions of a tensor.
- void setType (DataType type) noexcept

  Set the data type of a tensor.
- DataType getType () const noexcept

  Get the data type of a tensor.
- TRT_DEPRECATED 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.
- TRT_DEPRECATED void setBroadcastAcrossBatch (bool broadcastAcrossBatch) noexcept

  Set whether to enable broadcast of tensor across the implicit batch dimension.
- TRT_DEPRECATED bool getBroadcastAcrossBatch () const noexcept

  Check if tensor is broadcast across the implicit batch dimension.
- TensorLocation getLocation () const noexcept

  Get the storage location of a tensor.
- TRT_DEPRECATED void setLocation (TensorLocation location) noexcept

  Set the storage location of a tensor.
- TRT_DEPRECATED 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 an input or output tensor. By default all formats are allowed. Shape tensors (for which isShapeTensor() returns true) may only have row-major linear format.

- TensorFormats getAllowedFormats () const noexcept

  Get a bitmask of TensorFormat values that the tensor supports. For a shape tensor, only row-major linear format is allowed.
- bool isShapeTensor () const noexcept

  Whether the tensor is a shape tensor.
- bool isExecutionTensor () const noexcept

  Whether the tensor is an execution tensor.
- void setDimensionName (int32_t index, char const *name) noexcept

  Name a dimension of an input tensor.
- char const * getDimensionName (int32_t index) const noexcept

  Get the name of an input dimension.

Protected Member Functions

- virtual ~ITensor () noexcept=default

Protected Attributes

- apiv::VTensor * mImpl

9.166.1 Detailed Description

A tensor in a network definition.

To remove a tensor from a network definition, use INetworkDefinition::removeTensor().

When using the DLA, the cumulative size of all Tensors that are not marked as Network Input or Output tensors, must be less than 1GB in size to fit into a single subgraph. If the build option kGPU_FALLBACK is specified, then multiple subgraphs can be created, with each subgraph limited to less than 1GB of internal tensors data.

Warning

The volume of the tensor must be less than 2^31 elements. If the tensor is a shape tensor, its volume must not exceed 64.

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.166.2 Constructor & Destructor Documentation

9.166.2.1 ~ITensor()

```
virtual nvinfer1::ITensor::~ITensor ( )  [protected], [virtual], [default], [noexcept]
```

9.166.3 Member Function Documentation

9.166.3.1 dynamicRangeIsSet()

```
TRT_DEPRECATED bool nvinfer1::ITensor::dynamicRangeIsSet ( ) const  [inline], [noexcept]
```

Query whether dynamic range is set.

Returns

True if dynamic range is set, false otherwise.

Deprecated Deprecated in TensorRT 10.1. Superseded by explicit quantization.

9.166.3.2 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()

9.166.3.3 getBroadcastAcrossBatch()

```
TRT_DEPRECATED bool nvinfer1::ITensor::getBroadcastAcrossBatch ( ) const  [inline], [noexcept]
```

Check if tensor is broadcast across the implicit batch dimension.

Returns

Always false since TensorRT 10.0 does not support an implicit batch dimension.

See also

setBroadcastAcrossBatch()

Deprecated Deprecated in TensorRT 10.0. Implicit batch is not supported since TensorRT 10.0.

9.166.3.4 getDimensionName()

```
char const * nvinfer1::ITensor::getDimensionName (
            int32_t index ) const  [inline], [noexcept]
```

Get the name of an input dimension.

Parameters

index	index of the dimension

Returns

The name of the input dimension, or nullptr if the dimension has no name. The name is a pointer to a null-terminated character sequence.

See also

setDimensionName()

9.166.3.5 getDimensions()

```
Dims nvinfer1::ITensor::getDimensions ( ) const  [inline], [noexcept]
```

Get the dimensions of a tensor.

Returns

The dimensions of the tensor.

Warning

getDimensions() returns a -1 for dimensions that are derived from a wildcard dimension.

See also

setDimensions()

9.166.3.6 getDynamicRangeMax()

```
float nvinfer1::ITensor::getDynamicRangeMax ( ) const  [inline], [noexcept]
```

Get maximum of dynamic range.

Returns

Maximum of dynamic range, or quiet NaN if range was not set.

9.166.3.7 getDynamicRangeMin()

```
float nvinfer1::ITensor::getDynamicRangeMin ( ) const  [inline], [noexcept]
```

Get minimum of dynamic range.

Returns

Minimum of dynamic range, or quiet NaN if range was not set.

9.166.3.8 getLocation()

```
TensorLocation nvinfer1::ITensor::getLocation ( ) const  [inline], [noexcept]
```

Get the storage location of a tensor.

Returns

The location of tensor data.

See also

setLocation()

9.166.3.9 getName()

```
char const * nvinfer1::ITensor::getName ( ) const  [inline], [noexcept]
```

Get the tensor name.

Returns

The name as a null-terminated C-style string.

See also

setName()

9.166.3.10 getType()

```
DataType nvinfer1::ITensor::getType ( ) const  [inline], [noexcept]
```

Get the data type of a tensor.

Returns

The data type of the tensor.

See also

setType()

9.166.3.11 isExecutionTensor()

```
bool nvinfer1::ITensor::isExecutionTensor ( ) const  [inline], [noexcept]
```

Whether the tensor is an execution tensor.

Tensors are usually execution tensors. The exceptions are tensors used solely for shape calculations or whose contents not needed to compute the outputs.

The result of isExecutionTensor() is reliable only when network construction is complete. For example, if a partially built network has no path from a tensor to a network output, isExecutionTensor() returns false. Completing the path would cause it to become true.

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.

9.166.3.12 isNetworkInput()

```
bool nvinfer1::ITensor::isNetworkInput ( ) const  [inline], [noexcept]
```

Whether the tensor is a network input.

9.166.3.13 isNetworkOutput()

```
bool nvinfer1::ITensor::isNetworkOutput ( ) const  [inline], [noexcept]
```

Whether the tensor is a network output.

9.166.3.14 isShapeTensor()

```
bool nvinfer1::ITensor::isShapeTensor ( ) const  [inline], [noexcept]
```

Whether the tensor is a shape tensor.

A shape tensor is a tensor that is related to shape calculations. It must have type Int32, Int64, Bool, or Float, and its shape must be determinable at build time. Furthermore, it must be needed as a shape tensor, either marked as a network shape output via markOutputForShapes(), or as a layer input that is required to be a shape tensor, such as the second input to IShuffleLayer. Some layers are "polymorphic" in this respect. For example, the inputs to IElementWiseLayer must be shape tensors if the output is a shape tensor.

The TensorRT Developer Guide 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".

It is possible for a tensor to be both a shape tensor and an execution tensor.

Returns

True if tensor is a shape tensor, false otherwise.

See also

INetworkDefinition::markOutputForShapes()

9.166.3.15 resetDynamicRange()

```
void nvinfer1::ITensor::resetDynamicRange ( )  [inline], [noexcept]
```

Undo effect of setDynamicRange.

9.166.3.16 setAllowedFormats()

```
void nvinfer1::ITensor::setAllowedFormats (
            TensorFormats formats )  [inline], [noexcept]
```

Set allowed formats for an input or output tensor. By default all formats are allowed. Shape tensors (for which isShapeTensor() returns true) may only have row-major linear format.

When running network on DLA and the build option kGPU_FALLBACK is not specified, if DLA format(kCHW4 with Int8, kCHW4 with FP16, kCHW16 with FP16, kCHW32 with Int8) is set, the input format is treated as native DLA format with line stride requirement. Input/output binding with these format should have correct layout during inference.

Tensor formats are determined at build time by TensorRT for tensors not marked as input or output.

Parameters

formats	A bitmask of TensorFormat values that are supported for this tensor.

See also

ITensor::getAllowedFormats()

TensorFormats

9.166.3.17 setBroadcastAcrossBatch()

```
TRT_DEPRECATED void nvinfer1::ITensor::setBroadcastAcrossBatch (
            bool broadcastAcrossBatch )  [inline], [noexcept]
```

Set whether to enable broadcast of tensor across the implicit batch dimension.

Warning

This method has no effect other than issuing a warning.

Parameters

broadcastAcrossBatch	Whether to broadcast the tensor across the implicit batch dimension that was a feature of TensorRT 9.x and prior.

See also

getBroadcastAcrossBatch()

Deprecated Deprecated in TensorRT 10.0. Implicit batch is not supported since TensorRT 10.0.

9.166.3.18 setDimensionName()

```
void nvinfer1::ITensor::setDimensionName (
            int32_t index,
            char const * name )  [inline], [noexcept]
```

Name a dimension of an input tensor.

Associate a runtime dimension of an input tensor with a symbolic name. Dimensions with the same non-empty name must be equal at runtime. Knowing this equality for runtime dimensions may help the TensorRT optimizer. Both runtime and build-time dimensions can be named.

For example, setDimensionName(0, ”n”) associates the symbolic name ”n” with the leading dimension.

This method copies the name string. If the function is called again, with the same index, it will overwrite the previous name. If nullptr is passed as name, it will clear the name of the dimension.

Parameters

index	index of the dimension
name	of the dimension, as a pointer to a null-terminated character sequence.

Warning

The string name must be null-terminated, and be at most 4096 bytes including the terminator.

See also

getDimensionName()

9.166.3.19 setDimensions()

```
void nvinfer1::ITensor::setDimensions (
            Dims const & dimensions )  [inline], [noexcept]
```

Set the dimensions of a tensor.

For a network input, the dimensions are assigned by the application. For a network output, the dimensions are computed based on the layer parameters and the inputs to the layer. If a tensor size or a parameter is modified in the network, the dimensions of all dependent tensors will be recomputed.

This call is only legal for network input tensors, since the dimensions of layer output tensors are inferred based on layer inputs and parameters. The volume must be less than 2^31 elements.

Parameters

dimensions	The dimensions of the tensor.

See also

getDimensions()

9.166.3.20 setDynamicRange()

```
TRT_DEPRECATED bool nvinfer1::ITensor::setDynamicRange (
            float min,
            float max )  [inline], [noexcept]
```

Set dynamic range for the tensor.

Currently, only symmetric ranges are supported. Therefore, the larger of the absolute values of the provided bounds is used.

Returns

Whether the dynamic range was set successfully.

Requires that min and max be finite, and min <= max.

Deprecated Deprecated in TensorRT 10.1. Superseded by explicit quantization.

9.166.3.21 setLocation()

```
TRT_DEPRECATED void nvinfer1::ITensor::setLocation (
            TensorLocation location )  [inline], [noexcept]
```

Set the storage location of a tensor.

Parameters

location	the location of tensor data

Only network input tensors for storing sequence lengths for RNNv2 are supported. Using host storage for layers that do not support it will generate errors at build time.

See also

getLocation()

Deprecated Deprecated in TensorRT 10.0. RNNv2 is not supported and the location must always be TensorLocation::kDEVICE since TensorRT 10.0.

9.166.3.22 setName()

```
void nvinfer1::ITensor::setName (
            char const * name )  [inline], [noexcept]
```

Set the tensor name.

For a network input, the name is assigned by the application. For tensors which are layer outputs, a default name is assigned consisting of the layer name followed by the index of the output in brackets. Each input and output tensor must have a unique name.

This method copies the name string.

Parameters

name	The name.

> Warning
>
> The string name must be null-terminated, and be at most 4096 bytes including the terminator.

See also

getName()

9.166.3.23 setType()

```
void nvinfer1::ITensor::setType (
            DataType type )  [inline], [noexcept]
```

Set the data type of a tensor.

Parameters

type	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()

9.166.4 Member Data Documentation

9.166.4.1 mImpl

```
apiv::VTensor* nvinfer1::ITensor::mImpl  [protected]
```

The documentation for this class was generated from the following file:

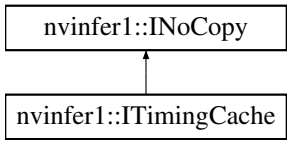
- NvInfer.h

9.167 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.
- int64_t queryKeys (TimingCacheKey *keyBuffer, int64_t capacity) const noexcept

  Query cache keys from Timing Cache.
- TimingCacheValue query (TimingCacheKey const &key) const noexcept

  Query value in a cache entry.
- bool update (TimingCacheKey const &key, TimingCacheValue const &value) noexcept

  Update values in a cache entry.

Protected Attributes

- apiv::VTimingCache * mImpl

Additional Inherited Members

9.167.1 Detailed Description

Class to handle tactic timing info collected from builder.

The timing cache is created or initialized by IBuilderConfig. It can be shared across builder instances to reduce the builder wallclock time.

Warning

It is a known issue that the same timing cache may not guarantee stable engine build reproducibility in all cases.

See also

IBuilderConfig

9.167.2 Constructor & Destructor Documentation

9.167.2.1 ~ITimingCache()

```
virtual nvinfer1::ITimingCache::~ITimingCache ( )  [virtual], [default], [noexcept]
```

9.167.3 Member Function Documentation

9.167.3.1 combine()

```
bool nvinfer1::ITimingCache::combine (
            ITimingCache const & inputCache,
            bool ignoreMismatch )  [inline], [noexcept]
```

Combine input timing cache into local instance.

This function allows combining entries in the input timing cache to local cache object.

Parameters

inputCache	The input timing cache.
ignoreMismatch	Whether or not to allow cache verification header mismatch.

Returns

True if combined successfully, false otherwise.

Append entries in input cache to local cache. Conflicting entries will be skipped The input cache must be generated by a TensorRT build of exact same version, otherwise combine will be skipped and return false. ignoreMismatch must be set to true if combining a timing cache created from a different device.

Warning

Combining caches generated from devices with different device properties may lead to functional/performance bugs!

9.167.3.2 query()

```
TimingCacheValue nvinfer1::ITimingCache::query (
            TimingCacheKey const & key ) const  [inline], [noexcept]
```

Query value in a cache entry.

The function queries the value in a specific cache entry.

Parameters

key	The query key.

Returns

Cache value if the key exists, otherwise an invalid value.

Query the value of the given cache key. If the key exists, write the value out, otherwise return an invalid value.

9.167.3.3 queryKeys()

```
int64_t nvinfer1::ITimingCache::queryKeys (
            TimingCacheKey * keyBuffer,
            int64_t capacity ) const  [inline], [noexcept]
```

Query cache keys from Timing Cache.

This function queries the entry count and writes the keys out.

Parameters

keyBuffer	The buffer to store keys.
capacity	The capacity of the buffer.

Returns

The count of entries in the cache and fill keys if keyBuffer is non-null. If an error occurrs, -1 will be returned.

Query the count of entries in the cache and write out cache keys if keyBuffer is provided. Any key entries exceeding the capacity of the keyBuffer will not be copied.

9.167.3.4 reset()

```
bool nvinfer1::ITimingCache::reset ( )  [inline], [noexcept]
```

Empty the timing cache.

Returns

True if reset successfully, false otherwise.

9.167.3.5 serialize()

```
nvinfer1::IHostMemory * nvinfer1::ITimingCache::serialize ( ) const  [inline], [noexcept]
```

Serialize a timing cache to IHostMemory object.

This function allows serialization of current timing cache.

Returns

A pointer to a IHostMemory object that contains a serialized timing cache.

See also

IHostMemory

9.167.3.6 update()

```
bool nvinfer1::ITimingCache::update (
            TimingCacheKey const & key,
            TimingCacheValue const & value )  [inline], [noexcept]
```

Update values in a cache entry.

The function updates the value in a specific cache entry.

Parameters

key	The key to the entry to be updated.
value	New cache value.

Returns

True if update succeeds, otherwise false.

Update the value of the given cache key. If the key does not exist, return false. If the key exists and the new tactic timing is NaN, delete the cache entry and return true. If tactic timing is not NaN and the new value is valid, override the cache value and return true. False is returned when the new value is invalid. If this layer cannot use the new tactic, build errors will be reported when building the next engine.

9.167.4 Member Data Documentation

9.167.4.1 mImpl

```
apiv::VTimingCache* nvinfer1::ITimingCache::mImpl  [protected]
```

The documentation for this class was generated from the following file:

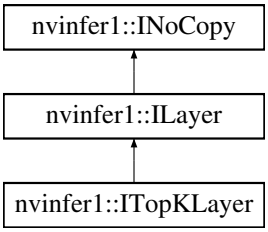
- NvInfer.h

9.168 nvinfer1::ITopKLayer Class Reference

Layer that represents a TopK reduction.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::ITopKLayer:



Public Member Functions

- void setOperation (TopKOperation op) noexcept

  Set the operation for the layer.
- TopKOperation getOperation () const noexcept

  Get the operation for the layer.
- void setK (int32_t k) noexcept

  Set the static k value for the layer.
- int32_t getK () const noexcept

  Get the k value for the layer.
- void setReduceAxes (uint32_t reduceAxes) noexcept

  Set which axes to reduce for the layer.
- uint32_t getReduceAxes () const noexcept

  Get the axes to reduce for the layer.
- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~ITopKLayer () noexcept=default

Protected Attributes

- apiv::VTopKLayer * mImpl

9.168.1 Detailed Description

Layer that represents a TopK reduction.

This layer can accept both static and dynamic k. Static k can be set through the addTopK() API function, or accessed using the getK() and setK() functions after layer creation. For dynamic k, use the setInput() method to pass in k as a tensor with index 1, which overrides the static k value in calculations.

> Warning
>
> Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.168.2 Constructor & Destructor Documentation

9.168.2.1 ~ITopKLayer()

```
virtual nvinfer1::ITopKLayer::~ITopKLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.168.3 Member Function Documentation

9.168.3.1 getK()

```
int32_t nvinfer1::ITopKLayer::getK ( ) const  [inline], [noexcept]
```

Get the k value for the layer.

This function will return the static k value passed into addTopK(), or the value passed into setK().

If a second layer input is present and non-null, this function returns -1.

See also

setK()

9.168.3.2 getOperation()

```
TopKOperation nvinfer1::ITopKLayer::getOperation ( ) const  [inline], [noexcept]
```

Get the operation for the layer.

See also

setOperation(), TopKOperation

9.168.3.3 getReduceAxes()

```
uint32_t nvinfer1::ITopKLayer::getReduceAxes ( ) const  [inline], [noexcept]
```

Get the axes to reduce for the layer.

See also

setReduceAxes()

9.168.3.4 setInput()

```
void nvinfer1::ILayer::setInput (
            int32_t index,
            ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	The index of the input to modify.
tensor	The new input tensor.

For a TopK layer, the values 0-1 are valid. The indices are as follows:

- 0: Input data tensor.
- 1: A scalar Int32 tensor containing a positive value corresponding to the number of top elements to retrieve. Values larger than 3840 will result in a runtime error. If provided, this will override the static k value in calculations.

9.168.3.5 setK()

```
void nvinfer1::ITopKLayer::setK (
             int32_t k )  [inline], [noexcept]
```

Set the static k value for the layer.

Currently only values up to 3840 are supported.

If a second input to this layer has been set, it will be reset to null by this method.

See also

getK()

9.168.3.6 setOperation()

```
void nvinfer1::ITopKLayer::setOperation (
             TopKOperation op )  [inline], [noexcept]
```

Set the operation for the layer.

See also

getOperation(), TopKOperation

9.168.3.7 setReduceAxes()

```
void nvinfer1::ITopKLayer::setReduceAxes (
            uint32_t reduceAxes )  [inline], [noexcept]
```

Set which axes to reduce for the layer.

See also

getReduceAxes()

9.168.4 Member Data Documentation

9.168.4.1 mImpl

```
apiv::VTopKLayer* nvinfer1::ITopKLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

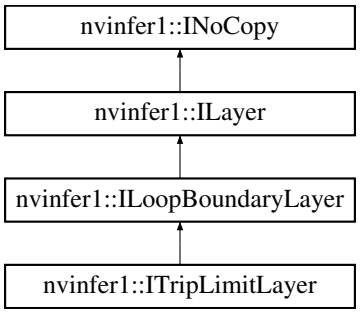
- NvInfer.h

9.169 nvinfer1::ITripLimitLayer Class Reference

A layer that represents a trip-count limiter.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::ITripLimitLayer:



Public Member Functions

- TripLimit getTripLimit () const noexcept
  Get a trip limiter type.

Protected Member Functions

- virtual ~ITripLimitLayer () noexcept=default

Protected Attributes

- apiv::VTripLimitLayer * mImpl

9.169.1 Detailed Description

A layer that represents a trip-count limiter.

The trip limit layer sets the execution condition for loops, using kCOUNT to define the number of iterations or kWHILE for a conditional loop. A loop can have one of each kind of limit, in which case the loop exits when the trip count is reached or the condition becomes false.

See INetworkDefinition::addTripLimit().

9.169.2 Constructor & Destructor Documentation

9.169.2.1 ~ITripLimitLayer()

```
virtual nvinfer1::ITripLimitLayer::~ITripLimitLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.169.3 Member Function Documentation

9.169.3.1 getTripLimit()

```
TripLimit nvinfer1::ITripLimitLayer::getTripLimit ( ) const  [inline], [noexcept]
```

Get a trip limiter type.

9.169.4 Member Data Documentation

9.169.4.1 mImpl

```
apiv::VTripLimitLayer* nvinfer1::ITripLimitLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

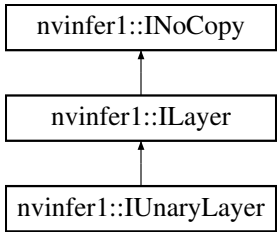
- NvInfer.h

9.170 nvinfer1::IUnaryLayer Class Reference

Layer that represents an unary operation.

`#include <NvInfer.h>`

Inheritance diagram for nvinfer1::IUnaryLayer:



Public Member Functions

- void setOperation (UnaryOperation op) noexcept
  Set the unary operation for the layer.
- UnaryOperation getOperation () const noexcept
  Get the unary operation for the layer.

Protected Member Functions

- virtual ~IUnaryLayer () noexcept=default

Protected Attributes

- apiv::VUnaryLayer * mImpl

9.170.1 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.

9.170.2 Constructor & Destructor Documentation

9.170.2.1 ~IUnaryLayer()

```
virtual nvinfer1::IUnaryLayer::~IUnaryLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.170.3 Member Function Documentation

9.170.3.1 getOperation()

```
UnaryOperation nvinfer1::IUnaryLayer::getOperation ( ) const  [inline], [noexcept]
```

Get the unary operation for the layer.

See also

setOperation(), UnaryOperation

9.170.3.2 setOperation()

```
void nvinfer1::IUnaryLayer::setOperation (
            UnaryOperation op )  [inline], [noexcept]
```

Set the unary operation for the layer.

When running this layer on DLA, only UnaryOperation::kABS is supported.

See also

getOperation(), UnaryOperation

9.170.4 Member Data Documentation

9.170.4.1 mImpl

```
apiv::VUnaryLayer* nvinfer1::IUnaryLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

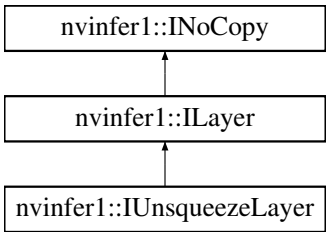
- NvInfer.h

9.171 nvinfer1::IUnsqueezeLayer Class Reference

Layer that represents an unsqueeze operation, which reshapes the input tensor by inserting unit-length dimensions at specified axes of the output.

```
#include <NvInfer.h>
```

Inheritance diagram for nvinfer1::IUnsqueezeLayer:



Public Member Functions

- void setInput (int32_t index, ITensor &tensor) noexcept

  Append or replace an input of this layer with a specific tensor.

Protected Member Functions

- virtual ~IUnsqueezeLayer () noexcept=default

Protected Attributes

- apiv::VUnsqueezeLayer * mImpl

9.171.1 Detailed Description

Layer that represents an unsqueeze operation, which reshapes the input tensor by inserting unit-length dimensions at specified axes of the output.

Warning

Do not inherit from this class, as doing so will break forward-compatibility of the API and ABI.

9.171.2 Constructor & Destructor Documentation

9.171.2.1 ~IUnsqueezeLayer()

```
virtual nvinfer1::IUnsqueezeLayer::~IUnsqueezeLayer ( )  [protected], [virtual], [default], [noexcept]
```

9.171.3 Member Function Documentation

9.171.3.1 setInput()

```
void nvinfer1::ILayer::setInput (
             int32_t index,
             ITensor & tensor )  [inline], [noexcept]
```

Append or replace an input of this layer with a specific tensor.

Parameters

index	The index of the input to modify.
tensor	The new input tensor.

For an Unsqueeze layer, the values 0-1 are valid for index. The indices are as follows:

- 0: Input data tensor.
- 1: The output axes at which unit-length dimensions are inserted. Must resolvable to a constant Int32 or Int64 1D shape tensor.

9.171.4 Member Data Documentation

9.171.4.1 mImpl

```
apiv::VUnsqueezeLayer* nvinfer1::IUnsqueezeLayer::mImpl  [protected]
```

The documentation for this class was generated from the following file:

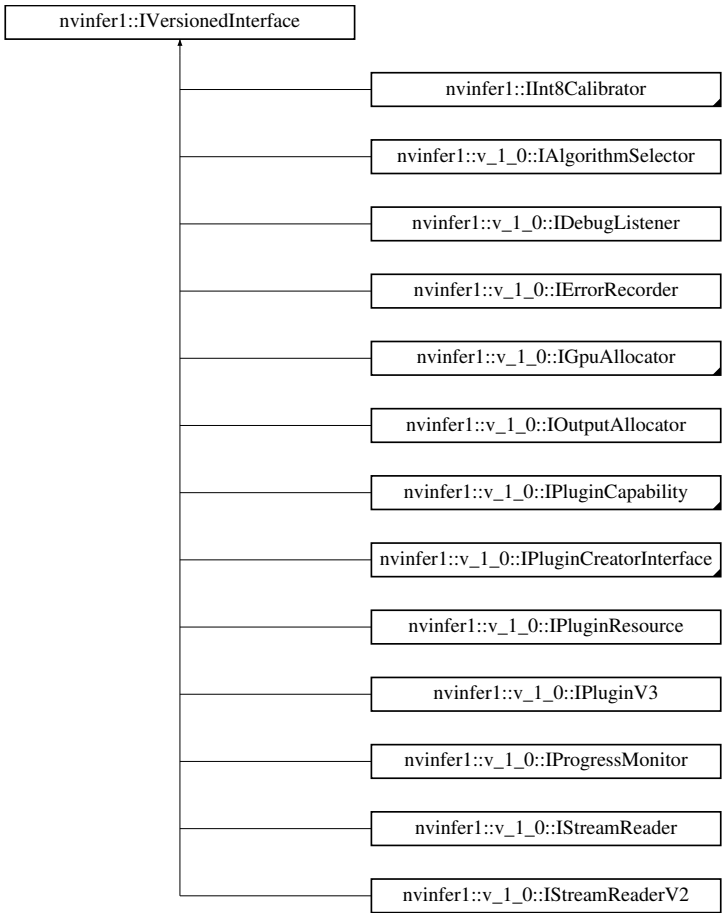
- NvInfer.h

9.172 nvinfer1::IVersionedInterface Class Reference

An Interface class for version control.

`#include <NvInferRuntimeBase.h>`

Inheritance diagram for nvinfer1::IVersionedInterface:



Public Member Functions

- virtual APILanguage getAPILanguage () const noexcept

  The language used to build the implementation of this Interface.
- virtual InterfaceInfo getInterfaceInfo () const noexcept=0

  Return version information associated with this interface. Applications must not override this method.
- virtual ~IVersionedInterface () noexcept=default

Protected Member Functions

- IVersionedInterface ()=default
- IVersionedInterface (IVersionedInterface const &)=default
- IVersionedInterface (IVersionedInterface &&)=default
- IVersionedInterface & operator= (IVersionedInterface const &) &=default
- IVersionedInterface & operator= (IVersionedInterface &&) &=default

9.172.1 Detailed Description

An Interface class for version control.

9.172.2 Constructor & Destructor Documentation

9.172.2.1 ~IVersionedInterface()

```
virtual nvinfer1::IVersionedInterface::~IVersionedInterface ( )  [virtual], [default], [noexcept]
```

9.172.2.2 IVersionedInterface() [1/3]

```
nvinfer1::IVersionedInterface::IVersionedInterface ( )  [protected], [default]
```

9.172.2.3 IVersionedInterface() [2/3]

```
nvinfer1::IVersionedInterface::IVersionedInterface (
            IVersionedInterface const &  )  [protected], [default]
```

9.172.2.4 IVersionedInterface() [3/3]

```
nvinfer1::IVersionedInterface::IVersionedInterface (
            IVersionedInterface &&  )  [protected], [default]
```

9.172.3 Member Function Documentation

9.172.3.1 getAPILanguage()

```
virtual APILanguage nvinfer1::IVersionedInterface::getAPILanguage ( ) const  [inline], [virtual],
[noexcept]
```

The language used to build the implementation of this Interface.

Applications must not override this method.

9.172.3.2 getInterfaceInfo()

```
virtual InterfaceInfo nvinfer1::IVersionedInterface::getInterfaceInfo ( ) const  [pure virtual],
[noexcept]
```

Return version information associated with this interface. Applications must not override this method.

Implemented in nvinfer1::v_1_0::IInt8EntropyCalibrator, nvinfer1::v_1_0::IInt8EntropyCalibrator2, nvinfer1::v_1_0::IInt8MinMaxCalibr nvinfer1::v_1_0::IInt8LegacyCalibrator, nvinfer1::v_1_0::IAlgorithmSelector, nvinfer1::v_1_0::IProgressMonitor, nvinfer1::v_1_0::IPluginResource, nvinfer1::v_1_0::IPluginV3, nvinfer1::v_1_0::IStreamReader, nvinfer1::v_1_0::IStreamReaderV2, nvinfer1::v_1_0::IPluginV3OneCore, nvinfer1::v_1_0::IPluginV3OneBuild, nvinfer1::v_1_0::IPluginV3OneRuntime, nvinfer1::v_2_0::IPluginV3OneBuild, nvinfer1::v_1_0::IGpuAllocator, nvinfer1::v_1_0::IOutputAllocator, nvinfer1::v_1_0::IDebugListen nvinfer1::v_1_0::IGpuAsyncAllocator, nvinfer1::v_1_0::IPluginCreatorV3One, nvinfer1::v_1_0::IErrorRecorder, and nvinfer1::v_1_0::IPluginCreator.

9.172.3.3 operator=() [1/2]

```
IVersionedInterface & nvinfer1::IVersionedInterface::operator= (
            IVersionedInterface &&  ) &  [protected], [default]
```

9.172.3.4 operator=() [2/2]

```
IVersionedInterface & nvinfer1::IVersionedInterface::operator= (
            IVersionedInterface const &  ) &  [protected], [default]
```

The documentation for this class was generated from the following file:

- NvInferRuntimeBase.h

9.173 nvinfer1::plugin::NMSParameters Struct Reference

The NMSParameters are used by the BatchedNMSPlugin 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

  Number of classes in the network.
- int32_t topK

  Number of bounding boxes to be fed into the NMS step.
- int32_t keepTopK
- float scoreThreshold

  Scalar threshold for score (low scoring boxes are removed).
- float iouThreshold
- bool isNormalized

9.173.1 Detailed Description

The NMSParameters are used by the BatchedNMSPlugin for performing the non_max_suppression operation over boxes for object detection networks.

Deprecated Deprecated in TensorRT 10.0. BatchedNMSPlugin plugin is deprecated.

9.173.2 Member Data Documentation

9.173.2.1 backgroundLabelId

```
int32_t nvinfer1::plugin::NMSParameters::backgroundLabelId
```

Label ID for the background class. If there is no background class, set it as -1

9.173.2.2 iouThreshold

```
float nvinfer1::plugin::NMSParameters::iouThreshold
```

A scalar threshold for IOU (new boxes that have high IOU overlap with previously selected boxes are removed).

9.173.2.3 isNormalized

```
bool nvinfer1::plugin::NMSParameters::isNormalized
```

Set to false, if the box coordinates are not normalized, i.e. not in the range [0,1]. Defaults to false.

9.173.2.4 keepTopK

```
int32_t nvinfer1::plugin::NMSParameters::keepTopK
```

Number of total bounding boxes to be kept per image after NMS step. Should be less than or equal to the topK value.

9.173.2.5 numClasses

```
int32_t nvinfer1::plugin::NMSParameters::numClasses
```

Number of classes in the network.

9.173.2.6 scoreThreshold

```
float nvinfer1::plugin::NMSParameters::scoreThreshold
```

Scalar threshold for score (low scoring boxes are removed).

9.173.2.7 shareLocation

```
bool nvinfer1::plugin::NMSParameters::shareLocation
```

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.

9.173.2.8 topK

```
int32_t nvinfer1::plugin::NMSParameters::topK
```

Number of bounding boxes to be fed into the NMS step.

The documentation for this struct was generated from the following file:

- NvInferPluginUtils.h

9.174 nvinfer1::Permutation Struct Reference

Represents a permutation of dimensions.

```
#include <NvInfer.h>
```

Public Attributes

- int32_t order [Dims::MAX_DIMS]

9.174.1 Detailed Description

Represents a permutation of dimensions.

9.174.2 Member Data Documentation

9.174.2.1 order

```
int32_t nvinfer1::Permutation::order[Dims::MAX_DIMS]
```

The elements of the permutation. The permutation is applied as outputDimensionIndex = permutation.order[input↩ DimensionIndex], so to permute from CHW order to HWC order, the required permutation is [1, 2, 0], and to permute from HWC to CHW, the required permutation is [2, 0, 1].

The documentation for this struct was generated from the following file:

- NvInfer.h

9.175 nvinfer1::PluginField Class Reference

Structure containing plugin attribute field names and associated data This information can be parsed to decode necessary plugin metadata.

```
#include <NvInferPluginBase.h>
```

Public Member Functions

- PluginField (AsciiChar const *const name_=nullptr, void const *const data_=nullptr, PluginFieldType const type_=PluginFieldType::kUNKNOWN, int32_t const length_=0) noexcept

Public Attributes

- AsciiChar const * name

  Plugin field attribute name.
- void const * data

  Plugin field attribute data.
- PluginFieldType type

  Plugin field attribute type.
- int32_t length

  Number of data entries in the Plugin attribute.

9.175.1 Detailed Description

Structure containing plugin attribute field names and associated data This information can be parsed to decode necessary plugin metadata.

9.175.2 Constructor & Destructor Documentation

9.175.2.1 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]
```

9.175.3 Member Data Documentation

9.175.3.1 data

```
void const* nvinfer1::PluginField::data
```

Plugin field attribute data.

9.175.3.2 length

```
int32_t nvinfer1::PluginField::length
```

Number of data entries in the Plugin attribute.

9.175.3.3 name

```
AsciiChar const* nvinfer1::PluginField::name
```

Plugin field attribute name.

9.175.3.4 type

```
PluginFieldType nvinfer1::PluginField::type
```

Plugin field attribute type.

The documentation for this class was generated from the following file:

- NvInferPluginBase.h

9.176 nvinfer1::PluginFieldCollection Struct Reference

Plugin field collection struct.

```
#include <NvInferPluginBase.h>
```

Public Attributes

- int32_t nbFields {}
  Number of PluginField entries.
- PluginField const * fields {}
  Pointer to PluginField entries.

9.176.1 Detailed Description

Plugin field collection struct.

9.176.2 Member Data Documentation

9.176.2.1 fields

```
PluginField const* nvinfer1::PluginFieldCollection::fields {}
```

Pointer to PluginField entries.

9.176.2.2 nbFields

```
int32_t nvinfer1::PluginFieldCollection::nbFields {}
```

Number of PluginField entries.

The documentation for this struct was generated from the following file:

- NvInferPluginBase.h

9.177 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 ()

9.177.1 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.

9.177.2 Constructor & Destructor Documentation

9.177.2.1 PluginRegistrar()

```
template<typename T >
nvinfer1::PluginRegistrar< T >::PluginRegistrar ( )  [inline]
```

The documentation for this class was generated from the following file:

- NvInferRuntime.h

9.178 nvinfer1::PluginTensorDesc Struct Reference

Fields that a plugin might see for an input or output.

`#include <NvInferRuntimePlugin.h>`

Public Attributes

- Dims dims
  Dimensions.
- DataType type
- TensorFormat format
  Tensor format.
- float scale
  Scale for INT8 data type.

9.178.1 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

9.178.2 Member Data Documentation

9.178.2.1 dims

`Dims nvinfer1::PluginTensorDesc::dims`

Dimensions.

9.178.2.2 format

`TensorFormat nvinfer1::PluginTensorDesc::format`

Tensor format.

9.178.2.3 scale

```
float nvinfer1::PluginTensorDesc::scale
```

Scale for INT8 data type.

9.178.2.4 type

```
DataType nvinfer1::PluginTensorDesc::type
```

> **Warning**
>
> DataType:kBOOL and DataType::kUINT8 are not supported.

The documentation for this struct was generated from the following file:

- NvInferRuntimePlugin.h

9.179 PluginVersion Struct Reference

Definition of plugin versions.

```
#include <NvInferRuntimePlugin.h>
```

9.179.1 Detailed Description

Definition of plugin versions.

Tag for plug-in versions. Used in upper byte of getTensorRTVersion().

Deprecated Deprecated in TensorRT 10.10. PluginVersion is used only in relation to IPluginV2-descendent plugin interfaces, which are all deprecated.

The documentation for this struct was generated from the following file:

- NvInferRuntimePlugin.h

9.180 nvinfer1::plugin::PriorBoxParameters Struct Reference

The PriorBox plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions (H x W).

```
#include <NvInferPluginUtils.h>
```

Public Attributes

- float * minSize

  Minimum box size in pixels. Can not be nullptr.
- float * maxSize

  Maximum box size in pixels. Can be nullptr.
- float * aspectRatios

  Aspect ratios of the boxes. Can be nullptr.
- int32_t numMinSize

  Number of elements in minSize. Must be larger than 0.
- int32_t numMaxSize

  Number of elements in maxSize. Can be 0 or same as numMinSize.
- int32_t numAspectRatios

  Number of elements in aspectRatios. Can be 0.
- bool flip
- bool clip

  If true, will clip the prior so that it is within [0,1].
- float variance [4]

  Variance for adjusting the prior boxes.
- int32_t imgH

  Image height. If 0, then the H dimension of the data tensor will be used.
- int32_t imgW

  Image width. If 0, then the W dimension of the data tensor will be used.
- float stepH

  Step in H. If 0, then (float)imgH/h will be used where h is the H dimension of the 1st input tensor.
- float stepW

  Step in W. If 0, then (float)imgW/w will be used where w is the W dimension of the 1st input tensor.
- float offset

  Offset to the top left corner of each cell.

9.180.1 Detailed Description

The PriorBox plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions (H x W).

PriorBoxParameters defines a set of parameters for creating the PriorBox plugin layer.

9.180.2 Member Data Documentation

9.180.2.1 aspectRatios

```
float* nvinfer1::plugin::PriorBoxParameters::aspectRatios
```

Aspect ratios of the boxes. Can be nullptr.

9.180.2.2 clip

```
bool nvinfer1::plugin::PriorBoxParameters::clip
```

If true, will clip the prior so that it is within [0,1].

9.180.2.3 flip

```
bool nvinfer1::plugin::PriorBoxParameters::flip
```

If true, will flip each aspect ratio. For example, if there is an aspect ratio "r", the aspect ratio "1.0/r" will be generated as well.

9.180.2.4 imgH

```
int32_t nvinfer1::plugin::PriorBoxParameters::imgH
```

Image height. If 0, then the H dimension of the data tensor will be used.

9.180.2.5 imgW

```
int32_t nvinfer1::plugin::PriorBoxParameters::imgW
```

Image width. If 0, then the W dimension of the data tensor will be used.

9.180.2.6 maxSize

```
float* nvinfer1::plugin::PriorBoxParameters::maxSize
```

Maximum box size in pixels. Can be nullptr.

9.180.2.7 minSize

```
float* nvinfer1::plugin::PriorBoxParameters::minSize
```

Minimum box size in pixels. Can not be nullptr.

9.180.2.8 numAspectRatios

```
int32_t nvinfer1::plugin::PriorBoxParameters::numAspectRatios
```

Number of elements in aspectRatios. Can be 0.

9.180.2.9 numMaxSize

```
int32_t nvinfer1::plugin::PriorBoxParameters::numMaxSize
```

Number of elements in maxSize. Can be 0 or same as numMinSize.

9.180.2.10 numMinSize

```
int32_t nvinfer1::plugin::PriorBoxParameters::numMinSize
```

Number of elements in minSize. Must be larger than 0.

9.180.2.11 offset

```
float nvinfer1::plugin::PriorBoxParameters::offset
```

Offset to the top left corner of each cell.

9.180.2.12 stepH

```
float nvinfer1::plugin::PriorBoxParameters::stepH
```

Step in H. If 0, then (float)imgH/h will be used where h is the H dimension of the 1st input tensor.

9.180.2.13 stepW

```
float nvinfer1::plugin::PriorBoxParameters::stepW
```

Step in W. If 0, then (float)imgW/w will be used where w is the W dimension of the 1st input tensor.

9.180.2.14 variance

```
float nvinfer1::plugin::PriorBoxParameters::variance[4]
```

Variance for adjusting the prior boxes.

The documentation for this struct was generated from the following file:

- NvInferPluginUtils.h

9.181 nvinfer1::plugin::RegionParameters Struct Reference

The Region plugin layer performs region proposal calculation.

```
#include <NvInferPluginUtils.h>
```

Public Attributes

- int32_t num

  Number of predicted bounding box for each grid cell.
- int32_t coords

  Number of coordinates for a bounding box.
- int32_t classes

  Number of classifications to be predicted.
- softmaxTree * smTree

  Helping structure to do softmax on confidence scores.

9.181.1 Detailed Description

The Region plugin layer performs region proposal calculation.

Generate 5 bounding boxes per cell (for yolo9000, generate 3 bounding boxes per cell). For each box, calculating its probabilities of objects detections from 80 pre-defined classifications (yolo9000 has 9418 pre-defined classifications, and these 9418 items are organized as work-tree structure). RegionParameters defines a set of parameters for creating the Region plugin layer.

9.181.2 Member Data Documentation

9.181.2.1 classes

```
int32_t nvinfer1::plugin::RegionParameters::classes
```

Number of classifications to be predicted.

9.181.2.2 coords

```
int32_t nvinfer1::plugin::RegionParameters::coords
```

Number of coordinates for a bounding box.

9.181.2.3 num

```
int32_t nvinfer1::plugin::RegionParameters::num
```

Number of predicted bounding box for each grid cell.

9.181.2.4 smTree

```
softmaxTree* nvinfer1::plugin::RegionParameters::smTree
```

Helping structure to do softmax on confidence scores.

The documentation for this struct was generated from the following file:

- NvInferPluginUtils.h

9.182 nvinfer1::plugin::RPROIParams Struct Reference

RPROIParams is used to create the RPROIPlugin instance.

```
#include <NvInferPluginUtils.h>
```

Public Attributes

- int32_t poolingH

  Height of the output in pixels after ROI pooling on feature map.
- int32_t poolingW

  Width of the output in pixels after ROI pooling on feature map.
- int32_t featureStride
- int32_t preNmsTop

  Number of proposals to keep before applying NMS.
- int32_t nmsMaxOut

  Number of remaining proposals after applying NMS.
- int32_t anchorsRatioCount

  Number of anchor box ratios.
- int32_t anchorsScaleCount

  Number of anchor box scales.
- float iouThreshold

  IoU (Intersection over Union) threshold used for the NMS step.
- float minBoxSize

  Minimum allowed bounding box size before scaling, used for anchor box calculation.
- float spatialScale

  Spatial scale between the input image and the last feature map.

9.182.1 Detailed Description

RPROIParams is used to create the RPROIPlugin instance.

9.182.2 Member Data Documentation

9.182.2.1 anchorsRatioCount

```
int32_t nvinfer1::plugin::RPROIParams::anchorsRatioCount
```

Number of anchor box ratios.

9.182.2.2 anchorsScaleCount

```
int32_t nvinfer1::plugin::RPROIParams::anchorsScaleCount
```

Number of anchor box scales.

9.182.2.3 featureStride

```
int32_t nvinfer1::plugin::RPROIParams::featureStride
```

Feature stride; ratio of input image size to feature map size. Assuming that max pooling layers in the neural network use square filters.

9.182.2.4 iouThreshold

```
float nvinfer1::plugin::RPROIParams::iouThreshold
```

IoU (Intersection over Union) threshold used for the NMS step.

9.182.2.5 minBoxSize

```
float nvinfer1::plugin::RPROIParams::minBoxSize
```

Minimum allowed bounding box size before scaling, used for anchor box calculation.

9.182.2.6 nmsMaxOut

```
int32_t nvinfer1::plugin::RPROIParams::nmsMaxOut
```

Number of remaining proposals after applying NMS.

9.182.2.7 poolingH

```
int32_t nvinfer1::plugin::RPROIParams::poolingH
```

Height of the output in pixels after ROI pooling on feature map.

9.182.2.8 poolingW

```
int32_t nvinfer1::plugin::RPROIParams::poolingW
```

Width of the output in pixels after ROI pooling on feature map.

9.182.2.9 preNmsTop

```
int32_t nvinfer1::plugin::RPROIParams::preNmsTop
```

Number of proposals to keep before applying NMS.

9.182.2.10 spatialScale

```
float nvinfer1::plugin::RPROIParams::spatialScale
```

Spatial scale between the input image and the last feature map.

The documentation for this struct was generated from the following file:

- NvInferPluginUtils.h

9.183 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

9.183.1 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.

9.183.2 Member Data Documentation

9.183.2.1 child

```
int32_t* nvinfer1::plugin::softmaxTree::child
```

9.183.2.2 group

```
int32_t* nvinfer1::plugin::softmaxTree::group
```

9.183.2.3 groupOffset

```
int32_t* nvinfer1::plugin::softmaxTree::groupOffset
```

9.183.2.4 groups

```
int32_t nvinfer1::plugin::softmaxTree::groups
```

9.183.2.5 groupSize

```
int32_t* nvinfer1::plugin::softmaxTree::groupSize
```

9.183.2.6 leaf

```
int32_t* nvinfer1::plugin::softmaxTree::leaf
```

9.183.2.7 n

```
int32_t nvinfer1::plugin::softmaxTree::n
```

9.183.2.8 name

```
char** nvinfer1::plugin::softmaxTree::name
```

9.183.2.9 parent

```
int32_t* nvinfer1::plugin::softmaxTree::parent
```

The documentation for this struct was generated from the following file:

- NvInferPluginUtils.h

9.184 nvinfer1::v_1_0::TimingCacheKey Struct Reference

The key to retrieve timing cache entries.

```
#include <NvInfer.h>
```

Public Attributes

- uint8_t data [16]

9.184.1 Detailed Description

The key to retrieve timing cache entries.

TimingCacheKey has two types of representation: binary and string. The conversion rule from binary to string is: 1) Convert each uint8_t element in binary key into two hexadecimal ascii chars, e.g. 0xab -> ”ab” 2) Concat the ascii chars of all elements in sequence. The result should have exact 32 chars 3) Add prefix ”0x” to the string produced in step 2.

See also

ITimingCache::query(), ITimingCache::update()

9.184.2 Member Data Documentation

9.184.2.1 data

```
uint8_t nvinfer1::v_1_0::TimingCacheKey::data[16]
```

The documentation for this struct was generated from the following file:

- NvInfer.h

9.185 nvinfer1::v_1_0::TimingCacheValue Struct Reference

```
#include <NvInfer.h>
```

Public Attributes

- uint64_t tacticHash

  Hash of the selected tactic.
- float timingMSec

  Timing of this tactic in milliseconds. Negative numbers and NaN are invalid values.

Static Public Attributes

- static constexpr uint64_t kINVALID_TACTIC_HASH = UINT64_MAX

  UINT64_MAX represents the invalid tactic hash.

9.185.1 Member Data Documentation

9.185.1.1 kINVALID_TACTIC_HASH

```
constexpr uint64_t nvinfer1::v_1_0::TimingCacheValue::kINVALID_TACTIC_HASH = UINT64_MAX  [static],
[constexpr]
```

UINT64_MAX represents the invalid tactic hash.

9.185.1.2 tacticHash

```
uint64_t nvinfer1::v_1_0::TimingCacheValue::tacticHash
```

Hash of the selected tactic.

9.185.1.3 timingMSec

`float nvinfer1::v_1_0::TimingCacheValue::timingMSec`

Timing of this tactic in milliseconds. Negative numbers and NaN are invalid values.

The documentation for this struct was generated from the following file:

- NvInfer.h

9.186 Value Struct Reference

The values in the cache entry.

`#include <NvInfer.h>`

9.186.1 Detailed Description

The values in the cache entry.

See also

ITimingCache::query(), ITimingCache::update()

The documentation for this struct was generated from the following file:

- NvInfer.h

9.187 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.

9.187.1 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).

9.187.2 Member Data Documentation

9.187.2.1 count

```
int64_t nvinfer1::Weights::count
```

The number of weights in the array.

9.187.2.2 type

```
DataType nvinfer1::Weights::type
```

The type of the weights.

9.187.2.3 values

```
void const* nvinfer1::Weights::values
```

The weight values, in a contiguous array.

The documentation for this class was generated from the following file:

- NvInferRuntime.h

Chapter 10

File Documentation

10.1 NvInfer.h File Reference

```
#include "NvInferLegacyDims.h"
#include "NvInferRuntime.h"
```

Classes

- struct nvinfer1::impl::EnumMaxImpl< ActivationType >
- class nvinfer1::ITensor

  A tensor in a network definition.
- class nvinfer1::ILayer

  Base class for all layer classes in a network definition.
- struct nvinfer1::impl::EnumMaxImpl< PaddingMode >
- class nvinfer1::IConvolutionLayer

  A convolution layer in a network definition.
- class nvinfer1::IActivationLayer

  An Activation layer in a network definition.
- struct nvinfer1::impl::EnumMaxImpl< PoolingType >
- class nvinfer1::IPoolingLayer

  A Pooling layer in a network definition.
- class nvinfer1::ILRNLayer

  A LRN layer in a network definition.
- 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::IPluginV2Layer

  Layer type for pluginV2.
- class nvinfer1::IPluginV3Layer

  Layer type for V3 plugins.
- 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

  Represents a permutation of dimensions.
- class nvinfer1::IShuffleLayer

  Layer type for shuffling data.
- class nvinfer1::ISliceLayer

  Slices an input tensor into an output tensor based on the offset and strides.
- class nvinfer1::IShapeLayer

  Layer type for getting shape of a tensor.
- class nvinfer1::ITopKLayer

  Layer that represents a TopK reduction.
- class nvinfer1::IMatrixMultiplyLayer

  Layer that represents a Matrix Multiplication.
- class nvinfer1::INonZeroLayer
- class nvinfer1::IRaggedSoftMaxLayer

  A RaggedSoftmax layer in a network definition.
- class nvinfer1::IIdentityLayer

  A layer that represents the identity function.
- class nvinfer1::ICastLayer

  A cast layer in a network.
- class nvinfer1::IConstantLayer

  Layer that represents a constant value.
- class nvinfer1::IParametricReLULayer

  Layer that represents a parametric ReLU operation.
- struct nvinfer1::impl::EnumMaxImpl< InterpolationMode >
- struct nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation >
- struct nvinfer1::impl::EnumMaxImpl< ResizeSelector >
- struct nvinfer1::impl::EnumMaxImpl< ResizeRoundMode >
- class nvinfer1::IResizeLayer

  A resize layer in a network definition.
- class nvinfer1::ILoopBoundaryLayer

  This is a base class for Loop boundary layers.
- class nvinfer1::IIfConditionalBoundaryLayer

This is a base class for Conditional boundary layers.

- class nvinfer1::IConditionLayer

  This layer represents a condition input to an IIfConditional.
- class nvinfer1::IIfConditionalOutputLayer

  This layer represents an output of an IIfConditional.
- class nvinfer1::IIfConditionalInputLayer

  This layer represents an input to an IIfConditional.
- class nvinfer1::IIfConditional

  Helper for constructing conditionally-executed subgraphs.
- class nvinfer1::IRecurrenceLayer

  A recurrence layer in a network definition.
- class nvinfer1::ILoopOutputLayer

  An ILoopOutputLayer is the sole way to get output from a loop.
- class nvinfer1::ITripLimitLayer

  A layer that represents a trip-count limiter.
- class nvinfer1::IIteratorLayer

  A layer to do iterations.
- class nvinfer1::ILoop

  Helper for creating a recurrent subgraph.
- class nvinfer1::ISelectLayer

  Select elements from two data tensors based on a condition tensor.
- class nvinfer1::IAssertionLayer

  An assertion layer in a network.
- class nvinfer1::IFillLayer

  Generate a tensor according to a specified mode.
- class nvinfer1::IQuantizeLayer

  A Quantize layer in a network definition.
- class nvinfer1::IDequantizeLayer

  A Dequantize layer in a network definition.
- class nvinfer1::IDynamicQuantizeLayer

  A network layer to perform dynamic quantization.
- class nvinfer1::IEinsumLayer

  An Einsum layer in a network.
- class nvinfer1::IScatterLayer

  A scatter layer in a network definition. Supports several kinds of scattering.
- class nvinfer1::IOneHotLayer

  A OneHot layer in a network definition.
- class nvinfer1::IGridSampleLayer

  A GridSample layer in a network definition.
- class nvinfer1::INMSLayer

  A non-maximum suppression layer in a network definition.
- class nvinfer1::IReverseSequenceLayer

  A ReverseSequence layer in a network definition.
- class nvinfer1::INormalizationLayer

  A normalization layer in a network definition.
- class nvinfer1::ISqueezeLayer

  Layer that represents a squeeze operation, removing unit dimensions of the input tensor on a set of axes.

- class nvinfer1::IUnsqueezeLayer

  Layer that represents an unsqueeze operation, which reshapes the input tensor by inserting unit-length dimensions at specified axes of the output.

- struct nvinfer1::impl::EnumMaxImpl< CumulativeOperation >

  Maximum number of elements in CumulativeOperation enum.

- class nvinfer1::ICumulativeLayer

  Layer that represents a cumulative operation across a tensor.

- class nvinfer1::INetworkDefinition

  A network definition for input to the builder.

- class nvinfer1::IInt8Calibrator

  Application-implemented interface for calibration.

- class nvinfer1::v_1_0::IInt8EntropyCalibrator
- class nvinfer1::v_1_0::IInt8EntropyCalibrator2
- class nvinfer1::v_1_0::IInt8MinMaxCalibrator
- class nvinfer1::v_1_0::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::select↩ Algorithms().

- class nvinfer1::v_1_0::IAlgorithmSelector
- struct nvinfer1::impl::EnumMaxImpl< RuntimePlatform >
- struct nvinfer1::v_1_0::TimingCacheKey

  The key to retrieve timing cache entries.

- struct nvinfer1::v_1_0::TimingCacheValue
- class nvinfer1::ITimingCache

  Class to handle tactic timing info collected from builder.

- struct nvinfer1::impl::EnumMaxImpl< PreviewFeature >
- struct nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel >
- struct nvinfer1::impl::EnumMaxImpl< TilingOptimizationLevel >
- class nvinfer1::v_1_0::IProgressMonitor
- class nvinfer1::IBuilderConfig

  Holds properties for configuring a builder to produce an engine.

- class nvinfer1::IBuilder

  Builds an engine from a network definition.

Namespaces

- namespace nvinfer1

  The TensorRT API version 1 namespace.
- namespace nvinfer1::impl
- namespace nvinfer1::v_1_0

  Forward declare IErrorRecorder for use in other interfaces.
- namespace nvinfer1::anonymous_namespace{NvInfer.h}
- namespace nvinfer1::safe

Typedefs

- using nvinfer1::TensorFormats = uint32_t

  It is capable of representing one or more TensorFormat by binary OR operations, e.g., 1U << TensorFormat::kCHW4 | 1U << TensorFormat::kCHW32.
- using nvinfer1::IInt8EntropyCalibrator = v_1_0::IInt8EntropyCalibrator
- using nvinfer1::IInt8EntropyCalibrator2 = v_1_0::IInt8EntropyCalibrator2
- using nvinfer1::IInt8MinMaxCalibrator = v_1_0::IInt8MinMaxCalibrator
- using nvinfer1::IInt8LegacyCalibrator = v_1_0::IInt8LegacyCalibrator
- using nvinfer1::IAlgorithmSelector = v_1_0::IAlgorithmSelector
- using nvinfer1::QuantizationFlags = uint32_t

  Represents one or more QuantizationFlag values using binary OR operations.
- using nvinfer1::BuilderFlags = uint32_t

  Represents one or more BuilderFlag values using binary OR operations, e.g., 1U << BuilderFlag::kFP16 | 1U << BuilderFlag::kDEBUG.
- using nvinfer1::IProgressMonitor = v_1_0::IProgressMonitor
- using nvinfer1::NetworkDefinitionCreationFlags = uint32_t

  Represents one or more NetworkDefinitionCreationFlag flags using binary OR operations. e.g., 1U << NetworkDefinitionCreationFlag::kSTRONGLY_TYPED.

Enumerations

- enum class nvinfer1::LayerType : int32_t {
  nvinfer1::kCONVOLUTION = 0 , nvinfer1::kCAST = 1 , nvinfer1::kACTIVATION = 2 , nvinfer1::kPOOLING = 3 ,
  nvinfer1::kLRN = 4 , nvinfer1::kSCALE = 5 , nvinfer1::kSOFTMAX = 6 , nvinfer1::kDECONVOLUTION = 7 ,
  nvinfer1::kCONCATENATION = 8 , nvinfer1::kELEMENTWISE = 9 , nvinfer1::kPLUGIN = 10 , nvinfer1::kUNARY = 11 ,
  nvinfer1::kPADDING = 12 , nvinfer1::kSHUFFLE = 13 , nvinfer1::kREDUCE = 14 , nvinfer1::kTOPK = 15 ,
  nvinfer1::kGATHER = 16 , nvinfer1::kMATRIX_MULTIPLY = 17 , nvinfer1::kRAGGED_SOFTMAX = 18 ,
  nvinfer1::kCONSTANT = 19 ,
  nvinfer1::kIDENTITY = 20 , nvinfer1::kPLUGIN_V2 = 21 , nvinfer1::kSLICE = 22 , nvinfer1::kSHAPE = 23 ,
  nvinfer1::kPARAMETRIC_RELU = 24 , nvinfer1::kRESIZE = 25 , nvinfer1::kTRIP_LIMIT = 26 , nvinfer1::kRECURRENCE = 27 ,
  nvinfer1::kITERATOR = 28 , nvinfer1::kLOOP_OUTPUT = 29 , nvinfer1::kSELECT = 30 , nvinfer1::kFILL = 31 ,
  nvinfer1::kQUANTIZE = 32 , nvinfer1::kDEQUANTIZE = 33 , nvinfer1::kCONDITION = 34 , nvinfer1::kCONDITIONAL_INPU
  = 35 ,

nvinfer1::kCONDITIONAL_OUTPUT = 36 , nvinfer1::kSCATTER = 37 , nvinfer1::kEINSUM = 38 , nvinfer1::kASSERTION = 39 ,
nvinfer1::kONE_HOT = 40 , nvinfer1::kNON_ZERO = 41 , nvinfer1::kGRID_SAMPLE = 42 , nvinfer1::kNMS = 43 ,
nvinfer1::kREVERSE_SEQUENCE = 44 , nvinfer1::kNORMALIZATION = 45 , nvinfer1::kPLUGIN_V3 = 46 , nvinfer1::kSQUEEZE = 47 ,
nvinfer1::kUNSQUEEZE = 48 , nvinfer1::kCUMULATIVE = 49 , nvinfer1::kDYNAMIC_QUANTIZE = 50 }

The type values of layer classes.

- enum class nvinfer1::ActivationType : int32_t {
nvinfer1::kRELU = 0 , nvinfer1::kSIGMOID = 1 , nvinfer1::kTANH = 2 , nvinfer1::kLEAKY_RELU = 3 ,
nvinfer1::kELU = 4 , nvinfer1::kSELU = 5 , nvinfer1::kSOFTSIGN = 6 , nvinfer1::kSOFTPLUS = 7 ,
nvinfer1::kCLIP = 8 , nvinfer1::kHARD_SIGMOID = 9 , nvinfer1::kSCALED_TANH = 10 , nvinfer1::kTHRESHOLDED_RELU = 11 ,
nvinfer1::kGELU_ERF = 12 , nvinfer1::kGELU_TANH = 13 }

  Enumerates the types of activation to perform in an activation layer.
- enum class nvinfer1::PaddingMode : int32_t { nvinfer1::kEXPLICIT_ROUND_DOWN = 0 , nvinfer1::kEXPLICIT_ROUND_UP = 1 , nvinfer1::kSAME_UPPER = 2 , nvinfer1::kSAME_LOWER = 3 }

  Enumerates the modes of padding to perform in convolution, deconvolution and pooling layer, padding mode takes precedence if setPaddingMode() and setPrePadding() are also used.
- enum class nvinfer1::PoolingType : int32_t { nvinfer1::kMAX = 0 , nvinfer1::kAVERAGE = 1 , nvinfer1::kMAX_AVERAGE_BLEND = 2 }

  The type of pooling to perform in a pooling layer.
- enum class nvinfer1::ScaleMode : int32_t { nvinfer1::kUNIFORM = 0 , nvinfer1::kCHANNEL = 1 , nvinfer1::kELEMENTWISE = 2 }

  Controls how shift, scale and power are applied in a Scale layer.
- enum class nvinfer1::ElementWiseOperation : int32_t {
nvinfer1::kSUM = 0 , nvinfer1::kPROD = 1 , nvinfer1::kMAX = 2 , nvinfer1::kMIN = 3 ,
nvinfer1::kSUB = 4 , nvinfer1::kDIV = 5 , nvinfer1::kPOW = 6 , nvinfer1::kFLOOR_DIV = 7 ,
nvinfer1::kAND = 8 , nvinfer1::kOR = 9 , nvinfer1::kXOR = 10 , nvinfer1::kEQUAL = 11 ,
nvinfer1::kGREATER = 12 , nvinfer1::kLESS = 13 }

  Enumerates the binary operations that may be performed by an ElementWise layer.
- enum class nvinfer1::GatherMode : int32_t { nvinfer1::kDEFAULT = 0 , nvinfer1::kELEMENT = 1 , nvinfer1::kND = 2 }

  Control form of IGatherLayer.
- enum class nvinfer1::UnaryOperation : int32_t {
nvinfer1::kEXP = 0 , nvinfer1::kLOG = 1 , nvinfer1::kSQRT = 2 , nvinfer1::kRECIP = 3 ,
nvinfer1::kABS = 4 , nvinfer1::kNEG = 5 , nvinfer1::kSIN = 6 , nvinfer1::kCOS = 7 ,
nvinfer1::kTAN = 8 , nvinfer1::kSINH = 9 , nvinfer1::kCOSH = 10 , nvinfer1::kASIN = 11 ,
nvinfer1::kACOS = 12 , nvinfer1::kATAN = 13 , nvinfer1::kASINH = 14 , nvinfer1::kACOSH = 15 ,
nvinfer1::kATANH = 16 , nvinfer1::kCEIL = 17 , nvinfer1::kFLOOR = 18 , nvinfer1::kERF = 19 ,
nvinfer1::kNOT = 20 , nvinfer1::kSIGN = 21 , nvinfer1::kROUND = 22 , nvinfer1::kISINF = 23 ,
nvinfer1::kISNAN = 24 }

  Enumerates the unary operations that may be performed by a Unary layer.
- enum class nvinfer1::ReduceOperation : int32_t {
nvinfer1::kSUM = 0 , nvinfer1::kPROD = 1 , nvinfer1::kMAX = 2 , nvinfer1::kMIN = 3 ,
nvinfer1::kAVG = 4 }

  Enumerates the reduce operations that may be performed by a Reduce layer.
- enum class nvinfer1::SampleMode : int32_t {
nvinfer1::kSTRICT_BOUNDS = 0 , nvinfer1::kWRAP = 1 , nvinfer1::kCLAMP = 2 , nvinfer1::kFILL = 3 ,
nvinfer1::kREFLECT = 4 }

  Controls how ISliceLayer and IGridSample handle out-of-bounds coordinates.

- enum class nvinfer1::TopKOperation : int32_t { nvinfer1::kMAX = 0 , nvinfer1::kMIN = 1 }

  Enumerates the operations that may be performed by a TopK layer.
- enum class nvinfer1::MatrixOperation : int32_t { nvinfer1::kNONE = 0 , nvinfer1::kTRANSPOSE = 1 , nvinfer1::kVECTOR = 2 }

  Enumerates the operations that may be performed on a tensor by IMatrixMultiplyLayer before multiplication.
- enum class nvinfer1::InterpolationMode : int32_t { nvinfer1::kNEAREST = 0 , nvinfer1::kLINEAR = 1 , nvinfer1::kCUBIC = 2 }

  Enumerates various modes of interpolation.
- enum class nvinfer1::ResizeCoordinateTransformation : int32_t { nvinfer1::kALIGN_CORNERS = 0 , nvinfer1::kASYMMETRIC = 1 , nvinfer1::kHALF_PIXEL = 2 }

  The resize coordinate transformation function.
- enum class nvinfer1::ResizeSelector : int32_t { nvinfer1::kFORMULA = 0 , nvinfer1::kUPPER = 1 }

  The coordinate selector when resize to single pixel output.
- enum class nvinfer1::ResizeRoundMode : int32_t { nvinfer1::kHALF_UP = 0 , nvinfer1::kHALF_DOWN = 1 , nvinfer1::kFLOOR = 2 , nvinfer1::kCEIL = 3 }

  The rounding mode for nearest neighbor resize.
- enum class nvinfer1::LoopOutput : int32_t { nvinfer1::kLAST_VALUE = 0 , nvinfer1::kCONCATENATE = 1 , nvinfer1::kREVERSE = 2 }

  Enum that describes kinds of loop outputs.
- enum class nvinfer1::TripLimit : int32_t { nvinfer1::kCOUNT = 0 , nvinfer1::kWHILE = 1 }

  Enum that describes kinds of trip limits.
- enum class nvinfer1::FillOperation : int32_t { nvinfer1::kLINSPACE = 0 , nvinfer1::kRANDOM_UNIFORM = 1 , nvinfer1::kRANDOM_NORMAL = 2 }

  Enumerates the tensor fill operations that may performed by a fill layer.
- enum class nvinfer1::ScatterMode : int32_t { nvinfer1::kELEMENT = 0 , nvinfer1::kND = 1 }

  Control form of IScatterLayer.
- enum class nvinfer1::BoundingBoxFormat : int32_t { nvinfer1::kCORNER_PAIRS = 0 , nvinfer1::kCENTER_SIZES = 1 }

  Representation of bounding box data used for the Boxes input tensor in INMSLayer.
- enum class nvinfer1::CumulativeOperation : int32_t { nvinfer1::kSUM = 0 }

  Enumerates the cumulative operations that may be performed by a Cumulative layer.
- enum class nvinfer1::CalibrationAlgoType : int32_t { nvinfer1::kLEGACY_CALIBRATION = 0 , nvinfer1::kENTROPY_CALIBRATION = 1 , nvinfer1::kENTROPY_CALIBRATION_2 = 2 , nvinfer1::kMINMAX_CALIBRATIC = 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::RuntimePlatform : int32_t { nvinfer1::kSAME_AS_BUILD = 0 , nvinfer1::kWINDOWS_AMD64 = 1 }

  Describes the intended runtime platform (operating system and CPU architecture) for the execution of the TensorRT engine. TensorRT provides support for cross-platform engine compatibility when the target runtime platform is different from the build platform.
- enum class nvinfer1::BuilderFlag : int32_t { nvinfer1::kFP16 = 0 , nvinfer1::kINT8 = 1 , nvinfer1::kDEBUG = 2 , nvinfer1::kGPU_FALLBACK = 3 , nvinfer1::kREFIT = 4 , nvinfer1::kDISABLE_TIMING_CACHE = 5 , nvinfer1::kTF32 = 6 , nvinfer1::kSPARSE_WEIGHTS = 7 , nvinfer1::kSAFETY_SCOPE = 8 , nvinfer1::kOBEY_PRECISION_CONSTRAINTS = 9 , nvinfer1::kPREFER_PRECISION_CON = 10 , nvinfer1::kDIRECT_IO = 11 , nvinfer1::kREJECT_EMPTY_ALGORITHMS = 12 , nvinfer1::kVERSION_COMPATIBLE = 13 , nvinfer1::kEXCLUDE_LEAN_F = 14 , nvinfer1::kFP8 = 15 ,

nvinfer1::kERROR_ON_TIMING_CACHE_MISS = 16 , nvinfer1::kBF16 = 17 , nvinfer1::kDISABLE_COMPILATION_CACHE = 18 , nvinfer1::kSTRIP_PLAN = 19 , nvinfer1::kWEIGHTLESS = kSTRIP_PLAN , nvinfer1::kREFIT_IDENTICAL = 20 , nvinfer1::kWEIGHT_STREAMING = 21 , nvinfer1::kINT4 = 22 , nvinfer1::kREFIT_INDIVIDUAL = 23 , nvinfer1::kSTRICT_NANS = 24 , nvinfer1::kMONITOR_MEMORY = 25 , nvinfer1::kFP4 = 26 , nvinfer1::kEDITABLE_TIMING_CACHE = 27 }

List of valid modes that the builder can enable when creating an engine from a network definition.

- enum class nvinfer1::MemoryPoolType : int32_t { nvinfer1::kWORKSPACE = 0 , nvinfer1::kDLA_MANAGED_SRAM = 1 , nvinfer1::kDLA_LOCAL_DRAM = 2 , nvinfer1::kDLA_GLOBAL_DRAM = 3 , nvinfer1::kTACTIC_DRAM = 4 , nvinfer1::kTACTIC_SHARED_MEMORY = 5 }

  The type for memory pools used by TensorRT.
- enum class nvinfer1::PreviewFeature : int32_t { nvinfer1::kPROFILE_SHARING_0806 = 0 , nvinfer1::kALIASED_PLUGIN_IO_1(= 1 , nvinfer1::kRUNTIME_ACTIVATION_RESIZE_10_10 = 2 }

  Define preview features.
- enum class nvinfer1::HardwareCompatibilityLevel : int32_t { nvinfer1::kNONE = 0 , nvinfer1::kAMPERE_PLUS = 1 , nvinfer1::kSAME_COMPUTE_CAPABILITY = 2 }

  Describes requirements of compatibility with GPU architectures other than that of the GPU on which the engine was built.
- enum class nvinfer1::TilingOptimizationLevel : int32_t { nvinfer1::kNONE = 0 , nvinfer1::kFAST = 1 , nvinfer1::kMODERATE = 2 , nvinfer1::kFULL = 3 }

  Define the optimization levels for Tiling.
- enum class nvinfer1::NetworkDefinitionCreationFlag : int32_t { nvinfer1::kEXPLICIT_BATCH = 0 , nvinfer1::kSTRONGLY_TYPED = 1 }

  List of immutable network properties expressed at network creation time. NetworkDefinitionCreationFlag is used with createNetworkV2() to specify immutable properties of the network.

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< UnaryOperation > () noexcept
- template<> constexpr int32_t nvinfer1::EnumMax< ReduceOperation > () noexcept
- template<> constexpr int32_t nvinfer1::EnumMax< SampleMode > () noexcept
- template<> constexpr int32_t nvinfer1::EnumMax< TopKOperation > () noexcept
- template<> constexpr int32_t nvinfer1::EnumMax< MatrixOperation > () noexcept
- template<> constexpr int32_t nvinfer1::EnumMax< LoopOutput > () noexcept
- template<> constexpr int32_t nvinfer1::EnumMax< TripLimit > () noexcept
- template<> constexpr int32_t nvinfer1::EnumMax< FillOperation > () noexcept
- template<> constexpr int32_t nvinfer1::EnumMax< ScatterMode > () noexcept
- template<> constexpr int32_t nvinfer1::EnumMax< BoundingBoxFormat > () noexcept
- template<> constexpr int32_t nvinfer1::EnumMax< CalibrationAlgoType > () noexcept
- template<> constexpr int32_t nvinfer1::EnumMax< QuantizationFlag > () noexcept
- template<> constexpr int32_t nvinfer1::EnumMax< BuilderFlag > () noexcept
- template<> constexpr int32_t nvinfer1::EnumMax< MemoryPoolType > () noexcept
- template<> constexpr int32_t nvinfer1::EnumMax< NetworkDefinitionCreationFlag > () noexcept
- IBuilder * nvinfer1::anonymous_namespace{NvInfer.h}::createInferBuilder (ILogger &logger) noexcept

  Create an instance of an IBuilder class.

- nvinfer1::IPluginRegistry * nvinfer1::getBuilderPluginRegistry (nvinfer1::EngineCapability capability) noexcept

  Return the plugin registry for building a Standard engine, or nullptr if no registry exists.
- nvinfer1::safe::IPluginRegistry * nvinfer1::getBuilderSafePluginRegistry (nvinfer1::EngineCapability capability) noexcept

  Return the plugin registry for building a Safety engine, or nullptr if no registry exists.

10.1.1 Detailed Description

This is the top-level API file for TensorRT.

10.2 NvInfer.h

Go to the documentation of this file.

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2025 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: Apache-2.0
4  *
5  * Licensed under the Apache License, Version 2.0 (the "License");
6  * you may not use this file except in compliance with the License.
7  * You may obtain a copy of the License at
8  *
9  * http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in writing, software
12  * distributed under the License is distributed on an "AS IS" BASIS,
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14  * See the License for the specific language governing permissions and
15  * limitations under the License.
16  */
17
18 #ifndef NV_INFER_H
19 #define NV_INFER_H
20
21 #include "NvInferLegacyDims.h"
22 #include "NvInferRuntime.h"
23
35
41
47 namespace nvinfer1
48 {
49
57 enum class LayerType : int32_t
58 {
59     kCONVOLUTION = 0,
60     kCAST = 1,
61     kACTIVATION = 2,
62     kPOOLING = 3,
63     kLRN = 4,
64     kSCALE = 5,
65     kSOFTMAX = 6,
66     kDECONVOLUTION = 7,
67     kCONCATENATION = 8,
68     kELEMENTWISE = 9,
69     kPLUGIN = 10,
70     kUNARY = 11,
71     kPADDING = 12,
72     kSHUFFLE = 13,
73     kREDUCE = 14,
74     kTOPK = 15,
75     kGATHER = 16,
76     kMATRIX_MULTIPLY = 17,
77     kRAGGED_SOFTMAX = 18,
78     kCONSTANT = 19,
79     kIDENTITY = 20,
80     kPLUGIN_V2 = 21,
81     kSLICE = 22,
```

```
82      kSHAPE = 23,
83      kPARAMETRIC_RELU = 24,
84      kRESIZE = 25,
85      kTRIP_LIMIT = 26,
86      kRECURRENCE = 27,
87      kITERATOR = 28,
88      kLOOP_OUTPUT = 29,
89      kSELECT = 30,
90      kFILL = 31,
91      kQUANTIZE = 32,
92      kDEQUANTIZE = 33,
93      kCONDITION = 34,
94      kCONDITIONAL_INPUT = 35,
95      kCONDITIONAL_OUTPUT = 36,
96      kSCATTER = 37,
97      kEINSUM = 38,
98      kASSERTION = 39,
99      kONE_HOT = 40,
100      kNON_ZERO = 41,
101      kGRID_SAMPLE = 42,
102      kNMS = 43,
103      kREVERSE_SEQUENCE = 44,
104      kNORMALIZATION = 45,
105      kPLUGIN_V3 = 46,
106      kSQUEEZE = 47,
107      kUNSQUEEZE = 48,
108      kCUMULATIVE = 49,
109      kDYNAMIC_QUANTIZE = 50,
110 };
111
117 template <>
118 constexpr inline int32_t EnumMax<LayerType>() noexcept
119 {
120     return 51;
121 }
122
129 using TensorFormats = uint32_t;
130
136 enum class ActivationType : int32_t
137 {
138     kRELU = 0,
139     kSIGMOID = 1,
140     kTANH = 2,
141     kLEAKY_RELU = 3,
142     kELU = 4,
143     kSELU = 5,
144     kSOFTSIGN = 6,
145     kSOFTPLUS = 7,
146     kCLIP = 8,
147     kHARD_SIGMOID = 9,
148     kSCALED_TANH = 10,
149     kTHRESHOLDED_RELU = 11,
150     kGELU_ERF = 12,
151     kGELU_TANH = 13
152 };
153
154 namespace impl
155 {
161 template <>
162 struct EnumMaxImpl<ActivationType>
163 {
164     static constexpr int32_t kVALUE = 14;
165 };
166 } // namespace impl
167
184 class ITensor : public INoCopy
185 {
186 public:
202     void setName(char const* name) noexcept
203     {
204         mImpl->setName(name);
205     }
206
214     char const* getName() const noexcept
215     {
216         return mImpl->getName();
217     }
218
233     void setDimensions(Dims const& dimensions) noexcept
234     {
235         mImpl->setDimensions(dimensions);
```

```
236     }
237
247     Dims getDimensions() const noexcept
248     {
249         return mImpl->getDimensions();
250     }
251
262     void setType(DataType type) noexcept
263     {
264         mImpl->setType(type);
265     }
266
274     DataType getType() const noexcept
275     {
276         return mImpl->getType();
277     }
278
291     TRT_DEPRECATED bool setDynamicRange(float min, float max) noexcept
292     {
293         return mImpl->setDynamicRange(min, max);
294     }
295
299     bool isNetworkInput() const noexcept
300     {
301         return mImpl->isNetworkInput();
302     }
303
307     bool isNetworkOutput() const noexcept
308     {
309         return mImpl->isNetworkOutput();
310     }
311
324     TRT_DEPRECATED void setBroadcastAcrossBatch(bool broadcastAcrossBatch) noexcept
325     {
326         mImpl->setBroadcastAcrossBatch(broadcastAcrossBatch);
327     }
328
338     TRT_DEPRECATED bool getBroadcastAcrossBatch() const noexcept
339     {
340         return mImpl->getBroadcastAcrossBatch();
341     }
342
350     TensorLocation getLocation() const noexcept
351     {
352         return mImpl->getLocation();
353     }
354
369     TRT_DEPRECATED void setLocation(TensorLocation location) noexcept
370     {
371         mImpl->setLocation(location);
372     }
373
381     TRT_DEPRECATED bool dynamicRangeIsSet() const noexcept
382     {
383         return mImpl->dynamicRangeIsSet();
384     }
385
389     void resetDynamicRange() noexcept
390     {
391         mImpl->resetDynamicRange();
392     }
393
399     float getDynamicRangeMin() const noexcept
400     {
401         return mImpl->getDynamicRangeMin();
402     }
403
409     float getDynamicRangeMax() const noexcept
410     {
411         return mImpl->getDynamicRangeMax();
412     }
413
431     void setAllowedFormats(TensorFormats formats) noexcept
432     {
433         mImpl->setAllowedFormats(formats);
434     }
435
444     TensorFormats getAllowedFormats() const noexcept
445     {
446         return mImpl->getAllowedFormats();
447     }
```

```
448
475     bool isShapeTensor() const noexcept
476     {
477         return mImpl->isShapeTensor();
478     }
479
496     bool isExecutionTensor() const noexcept
497     {
498         return mImpl->isExecutionTensor();
499     }
500
522     void setDimensionName(int32_t index, char const* name) noexcept
523     {
524         mImpl->setDimensionName(index, name);
525     }
526
537     char const* getDimensionName(int32_t index) const noexcept
538     {
539         return mImpl->getDimensionName(index);
540     }
541
542 protected:
543     apiv::VTensor* mImpl;
544     virtual ~ITensor() noexcept = default;
545 };
546
554 class ILayer : public INoCopy
555 {
556 public:
562     LayerType getType() const noexcept
563     {
564         return mLayer->getType();
565     }
566
576     void setName(char const* name) noexcept
577     {
578         mLayer->setName(name);
579     }
580
586     char const* getName() const noexcept
587     {
588         return mLayer->getName();
589     }
590
594     int32_t getNbInputs() const noexcept
595     {
596         return mLayer->getNbInputs();
597     }
598
607     ITensor* getInput(int32_t index) const noexcept
608     {
609         return mLayer->getInput(index);
610     }
611
615     int32_t getNbOutputs() const noexcept
616     {
617         return mLayer->getNbOutputs();
618     }
619
625     ITensor* getOutput(int32_t index) const noexcept
626     {
627         return mLayer->getOutput(index);
628     }
629
642     void setInput(int32_t index, ITensor& tensor) noexcept
643     {
644         return mLayer->setInput(index, tensor);
645     }
646
673     void setPrecision(DataType dataType) noexcept
674     {
675         mLayer->setPrecision(dataType);
676     }
677
685     DataType getPrecision() const noexcept
686     {
687         return mLayer->getPrecision();
688     }
689
697     bool precisionIsSet() const noexcept
698     {
```

```
699         return mLayer->precisionIsSet();
700     }
701
707     void resetPrecision() noexcept
708     {
709         mLayer->resetPrecision();
710     }
711
754     void setOutputType(int32_t index, DataType dataType) noexcept
755     {
756         mLayer->setOutputType(index, dataType);
757     }
758
769     DataType getOutputType(int32_t index) const noexcept
770     {
771         return mLayer->getOutputType(index);
772     }
773
783     bool outputTypeIsSet(int32_t index) const noexcept
784     {
785         return mLayer->outputTypeIsSet(index);
786     }
787
795     void resetOutputType(int32_t index) noexcept
796     {
797         return mLayer->resetOutputType(index);
798     }
799
813     void setMetadata(char const* metadata) noexcept
814     {
815         mLayer->setMetadata(metadata);
816     }
817
826     char const* getMetadata() const noexcept
827     {
828         return mLayer->getMetadata();
829     }
830
831 protected:
832     virtual ~ILayer() noexcept = default;
833     apiv::VLayer* mLayer;
834 };
835
992 enum class PaddingMode : int32_t
993 {
994     kEXPLICIT_ROUND_DOWN = 0,
995     kEXPLICIT_ROUND_UP = 1,
996     kSAME_UPPER = 2,
997     kSAME_LOWER = 3,
998 };
999
1000 namespace impl
1001 {
1007 template <>
1008 struct EnumMaxImpl<PaddingMode>
1009 {
1010     static constexpr int32_t kVALUE = 4;
1011 };
1012 } // namespace impl
1013
1026 class IConvolutionLayer : public ILayer
1027 {
1028 public:
1036     void setNbOutputMaps(int64_t nbOutputMaps) noexcept
1037     {
1038         mImpl->setNbOutputMaps(nbOutputMaps);
1039     }
1040
1046     int64_t getNbOutputMaps() const noexcept
1047     {
1048         return mImpl->getNbOutputMaps();
1049     }
1050
1066     void setNbGroups(int64_t nbGroups) noexcept
1067     {
1068         mImpl->setNbGroups(nbGroups);
1069     }
1070
1076     int64_t getNbGroups() const noexcept
1077     {
1078         return mImpl->getNbGroups();
```

```
1079     }
1080
1090     void setKernelWeights(Weights weights) noexcept
1091     {
1092         mImpl->setKernelWeights(weights);
1093     }
1094
1100     Weights getKernelWeights() const noexcept
1101     {
1102         return mImpl->getKernelWeights();
1103     }
1104
1115     void setBiasWeights(Weights weights) noexcept
1116     {
1117         mImpl->setBiasWeights(weights);
1118     }
1119
1125     Weights getBiasWeights() const noexcept
1126     {
1127         return mImpl->getBiasWeights();
1128     }
1129
1142     void setPrePadding(Dims const& padding) noexcept
1143     {
1144         mImpl->setPrePadding(padding);
1145     }
1146
1152     Dims getPrePadding() const noexcept
1153     {
1154         return mImpl->getPrePadding();
1155     }
1156
1169     void setPostPadding(Dims const& padding) noexcept
1170     {
1171         mImpl->setPostPadding(padding);
1172     }
1173
1179     Dims getPostPadding() const noexcept
1180     {
1181         return mImpl->getPostPadding();
1182     }
1183
1193     void setPaddingMode(PaddingMode paddingMode) noexcept
1194     {
1195         mImpl->setPaddingMode(paddingMode);
1196     }
1197
1205     PaddingMode getPaddingMode() const noexcept
1206     {
1207         return mImpl->getPaddingMode();
1208     }
1209
1218     void setKernelSizeNd(Dims const& kernelSize) noexcept
1219     {
1220         mImpl->setKernelSizeNd(kernelSize);
1221     }
1222
1228     Dims getKernelSizeNd() const noexcept
1229     {
1230         return mImpl->getKernelSizeNd();
1231     }
1232
1243     void setStrideNd(Dims const& stride) noexcept
1244     {
1245         mImpl->setStrideNd(stride);
1246     }
1247
1253     Dims getStrideNd() const noexcept
1254     {
1255         return mImpl->getStrideNd();
1256     }
1257
1271     void setPaddingNd(Dims const& padding) noexcept
1272     {
1273         mImpl->setPaddingNd(padding);
1274     }
1275
1283     Dims getPaddingNd() const noexcept
1284     {
1285         return mImpl->getPaddingNd();
1286     }
```

```
1287
1297     void setDilationNd(Dims const& dilation) noexcept
1298     {
1299         mImpl->setDilationNd(dilation);
1300     }
1301
1307     Dims getDilationNd() const noexcept
1308     {
1309         return mImpl->getDilationNd();
1310     }
1311
1326     using ILayer::setInput;
1327
1328 protected:
1329     virtual ~IConvolutionLayer() noexcept = default;
1330     apiv::VConvolutionLayer* mImpl;
1331 };
1332
1346 class IActivationLayer : public ILayer
1347 {
1348 public:
1356     void setActivationType(ActivationType type) noexcept
1357     {
1358         mImpl->setActivationType(type);
1359     }
1360
1366     ActivationType getActivationType() const noexcept
1367     {
1368         return mImpl->getActivationType();
1369     }
1370
1381     void setAlpha(float alpha) noexcept
1382     {
1383         mImpl->setAlpha(alpha);
1384     }
1385
1395     void setBeta(float beta) noexcept
1396     {
1397         mImpl->setBeta(beta);
1398     }
1399
1404     float getAlpha() const noexcept
1405     {
1406         return mImpl->getAlpha();
1407     }
1408
1413     float getBeta() const noexcept
1414     {
1415         return mImpl->getBeta();
1416     }
1417
1418 protected:
1419     virtual ~IActivationLayer() noexcept = default;
1420     apiv::VActivationLayer* mImpl;
1421 };
1422
1428 enum class PoolingType : int32_t
1429 {
1430     kMAX = 0,
1431     kAVERAGE = 1,
1432     kMAX_AVERAGE_BLEND = 2
1433 };
1434
1435 namespace impl
1436 {
1442 template <>
1443 struct EnumMaxImpl<PoolingType>
1444 {
1445     static constexpr int32_t kVALUE = 3;
1446 };
1447 } // namespace impl
1448
1460 class IPoolingLayer : public ILayer
1461 {
1462 public:
1470     void setPoolingType(PoolingType type) noexcept
1471     {
1472         mImpl->setPoolingType(type);
1473     }
1474
1480     PoolingType getPoolingType() const noexcept
```

```
1481     {
1482         return mImpl->getPoolingType();
1483     }
1484
1495     void setBlendFactor(float blendFactor) noexcept
1496     {
1497         mImpl->setBlendFactor(blendFactor);
1498     }
1499
1508     float getBlendFactor() const noexcept
1509     {
1510         return mImpl->getBlendFactor();
1511     }
1512
1522     void setAverageCountExcludesPadding(bool exclusive) noexcept
1523     {
1524         mImpl->setAverageCountExcludesPadding(exclusive);
1525     }
1526
1533     bool getAverageCountExcludesPadding() const noexcept
1534     {
1535         return mImpl->getAverageCountExcludesPadding();
1536     }
1537
1551     void setPrePadding(Dims const& padding) noexcept
1552     {
1553         mImpl->setPrePadding(padding);
1554     }
1555
1561     Dims getPrePadding() const noexcept
1562     {
1563         return mImpl->getPrePadding();
1564     }
1565
1579     void setPostPadding(Dims const& padding) noexcept
1580     {
1581         mImpl->setPostPadding(padding);
1582     }
1583
1589     Dims getPostPadding() const noexcept
1590     {
1591         return mImpl->getPostPadding();
1592     }
1593
1602     void setPaddingMode(PaddingMode paddingMode) noexcept
1603     {
1604         mImpl->setPaddingMode(paddingMode);
1605     }
1606
1613     PaddingMode getPaddingMode() const noexcept
1614     {
1615         return mImpl->getPaddingMode();
1616     }
1617
1626     void setWindowSizeNd(Dims const& windowSize) noexcept
1627     {
1628         mImpl->setWindowSizeNd(windowSize);
1629     }
1630
1636     Dims getWindowSizeNd() const noexcept
1637     {
1638         return mImpl->getWindowSizeNd();
1639     }
1640
1651     void setStrideNd(Dims const& stride) noexcept
1652     {
1653         mImpl->setStrideNd(stride);
1654     }
1655
1661     Dims getStrideNd() const noexcept
1662     {
1663         return mImpl->getStrideNd();
1664     }
1665
1680     void setPaddingNd(Dims const& padding) noexcept
1681     {
1682         mImpl->setPaddingNd(padding);
1683     }
1684
1692     Dims getPaddingNd() const noexcept
1693     {
```

```
1694         return mImpl->getPaddingNd();
1695     }
1696
1697 protected:
1698     virtual ~IPoolingLayer() noexcept = default;
1699     apiv::VPoolingLayer* mImpl;
1700 };
1701
1711 class ILRNLayer : public ILayer
1712 {
1713 public:
1723     void setWindowSize(int64_t windowSize) noexcept
1724     {
1725         mImpl->setWindowSize(windowSize);
1726     }
1727
1733     int64_t getWindowSize() const noexcept
1734     {
1735         return mImpl->getWindowSize();
1736     }
1737
1745     void setAlpha(float alpha) noexcept
1746     {
1747         mImpl->setAlpha(alpha);
1748     }
1749
1755     float getAlpha() const noexcept
1756     {
1757         return mImpl->getAlpha();
1758     }
1759
1767     void setBeta(float beta) noexcept
1768     {
1769         mImpl->setBeta(beta);
1770     }
1771
1777     float getBeta() const noexcept
1778     {
1779         return mImpl->getBeta();
1780     }
1781
1789     void setK(float k) noexcept
1790     {
1791         mImpl->setK(k);
1792     }
1793
1799     float getK() const noexcept
1800     {
1801         return mImpl->getK();
1802     }
1803
1804 protected:
1805     virtual ~ILRNLayer() noexcept = default;
1806     apiv::VLRNLayer* mImpl;
1807 };
1808
1814 enum class ScaleMode : int32_t
1815 {
1816     kUNIFORM = 0,
1817     kCHANNEL = 1,
1818     kELEMENTWISE = 2
1819 };
1820
1826 template <>
1827 constexpr inline int32_t EnumMax<ScaleMode>() noexcept
1828 {
1829     return 3;
1830 }
1831
1857 class IScaleLayer : public ILayer
1858 {
1859 public:
1865     void setMode(ScaleMode mode) noexcept
1866     {
1867         mImpl->setMode(mode);
1868     }
1869
1875     ScaleMode getMode() const noexcept
1876     {
1877         return mImpl->getMode();
1878     }
```

```
1879
1885     void setShift(Weights shift) noexcept
1886     {
1887         mImpl->setShift(shift);
1888     }
1889
1895     Weights getShift() const noexcept
1896     {
1897         return mImpl->getShift();
1898     }
1899
1905     void setScale(Weights scale) noexcept
1906     {
1907         mImpl->setScale(scale);
1908     }
1909
1915     Weights getScale() const noexcept
1916     {
1917         return mImpl->getScale();
1918     }
1919
1925     void setPower(Weights power) noexcept
1926     {
1927         mImpl->setPower(power);
1928     }
1929
1935     Weights getPower() const noexcept
1936     {
1937         return mImpl->getPower();
1938     }
1939
1950     int32_t getChannelAxis() const noexcept
1951     {
1952         return mImpl->getChannelAxis();
1953     }
1954
1971     void setChannelAxis(int32_t channelAxis) noexcept
1972     {
1973         mImpl->setChannelAxis(channelAxis);
1974     }
1975
1976 protected:
1977     virtual ~IScaleLayer() noexcept = default;
1978     apiv::VScaleLayer* mImpl;
1979 };
1980
2001 class ISoftMaxLayer : public ILayer
2002 {
2003 public:
2024     void setAxes(uint32_t axes) noexcept
2025     {
2026         mImpl->setAxes(axes);
2027     }
2028
2034     uint32_t getAxes() const noexcept
2035     {
2036         return mImpl->getAxes();
2037     }
2038
2039 protected:
2040     virtual ~ISoftMaxLayer() noexcept = default;
2041     apiv::VSoftMaxLayer* mImpl;
2042 };
2043
2056 class IConcatenationLayer : public ILayer
2057 {
2058 public:
2070     void setAxis(int32_t axis) noexcept
2071     {
2072         mImpl->setAxis(axis);
2073     }
2074
2080     int32_t getAxis() const noexcept
2081     {
2082         return mImpl->getAxis();
2083     }
2084
2085 protected:
2086     virtual ~IConcatenationLayer() noexcept = default;
2087     apiv::VConcatenationLayer* mImpl;
2088 };
```

```
2089
2097 class IDeconvolutionLayer : public ILayer
2098 {
2099 public:
2107     void setNbOutputMaps(int64_t nbOutputMaps) noexcept
2108     {
2109         mImpl->setNbOutputMaps(nbOutputMaps);
2110     }
2111
2117     int64_t getNbOutputMaps() const noexcept
2118     {
2119         return mImpl->getNbOutputMaps();
2120     }
2121
2137     void setNbGroups(int64_t nbGroups) noexcept
2138     {
2139         mImpl->setNbGroups(nbGroups);
2140     }
2141
2147     int64_t getNbGroups() const noexcept
2148     {
2149         return mImpl->getNbGroups();
2150     }
2151
2161     void setKernelWeights(Weights weights) noexcept
2162     {
2163         mImpl->setKernelWeights(weights);
2164     }
2165
2171     Weights getKernelWeights() const noexcept
2172     {
2173         return mImpl->getKernelWeights();
2174     }
2175
2186     void setBiasWeights(Weights weights) noexcept
2187     {
2188         mImpl->setBiasWeights(weights);
2189     }
2190
2196     Weights getBiasWeights() const noexcept
2197     {
2198         return mImpl->getBiasWeights();
2199     }
2200
2213     void setPrePadding(Dims const& padding) noexcept
2214     {
2215         mImpl->setPrePadding(padding);
2216     }
2217
2223     Dims getPrePadding() const noexcept
2224     {
2225         return mImpl->getPrePadding();
2226     }
2227
2240     void setPostPadding(Dims const& padding) noexcept
2241     {
2242         mImpl->setPostPadding(padding);
2243     }
2244
2250     Dims getPostPadding() const noexcept
2251     {
2252         return mImpl->getPostPadding();
2253     }
2254
2264     void setPaddingMode(PaddingMode paddingMode) noexcept
2265     {
2266         mImpl->setPaddingMode(paddingMode);
2267     }
2268
2276     PaddingMode getPaddingMode() const noexcept
2277     {
2278         return mImpl->getPaddingMode();
2279     }
2280
2291     void setKernelSizeNd(Dims const& kernelSize) noexcept
2292     {
2293         mImpl->setKernelSizeNd(kernelSize);
2294     }
2295
2301     Dims getKernelSizeNd() const noexcept
2302     {
```

```
2303         return mImpl->getKernelSizeNd();
2304     }
2305
2318     void setStrideNd(Dims const& stride) noexcept
2319     {
2320         mImpl->setStrideNd(stride);
2321     }
2322
2328     Dims getStrideNd() const noexcept
2329     {
2330         return mImpl->getStrideNd();
2331     }
2332
2346     void setPaddingNd(Dims const& padding) noexcept
2347     {
2348         mImpl->setPaddingNd(padding);
2349     }
2350
2358     Dims getPaddingNd() const noexcept
2359     {
2360         return mImpl->getPaddingNd();
2361     }
2362
2375     using ILayer::setInput;
2376
2384     void setDilationNd(Dims const& dilation) noexcept
2385     {
2386         mImpl->setDilationNd(dilation);
2387     }
2388
2394     Dims getDilationNd() const noexcept
2395     {
2396         return mImpl->getDilationNd();
2397     }
2398
2399 protected:
2400     virtual ~IDeconvolutionLayer() noexcept = default;
2401     apiv::VDeconvolutionLayer* mImpl;
2402 };
2403
2418 enum class ElementWiseOperation : int32_t
2419 {
2420     kSUM = 0,
2421     kPROD = 1,
2422     kMAX = 2,
2423     kMIN = 3,
2424     kSUB = 4,
2425     kDIV = 5,
2426     kPOW = 6,
2427     kFLOOR_DIV = 7,
2428     kAND = 8,
2429     kOR = 9,
2430     kXOR = 10,
2431     kEQUAL = 11,
2432     kGREATER = 12,
2433     kLESS = 13
2434 };
2435
2436 namespace impl
2437 {
2443 template <>
2444 struct EnumMaxImpl<ElementWiseOperation>
2445 {
2446     static constexpr int32_t kVALUE = 14;
2447 };
2448 } // namespace impl
2449
2469 class IElementWiseLayer : public ILayer
2470 {
2471 public:
2481     void setOperation(ElementWiseOperation op) noexcept
2482     {
2483         return mImpl->setOperation(op);
2484     }
2485
2493     ElementWiseOperation getOperation() const noexcept
2494     {
2495         return mImpl->getOperation();
2496     }
2497
2498 protected:
```

```
2499     apiv::VElementWiseLayer* mImpl;
2500     virtual ~IElementWiseLayer() noexcept = default;
2501 };
2502
2508 enum class GatherMode : int32_t
2509 {
2510     kDEFAULT = 0,
2511     kELEMENT = 1,
2512     kND = 2
2513 };
2514
2520 template <>
2521 constexpr inline int32_t EnumMax<GatherMode>() noexcept
2522 {
2523     return 3;
2524 }
2525
2602 class IGatherLayer : public ILayer
2603 {
2604 public:
2614     void setGatherAxis(int32_t axis) noexcept
2615     {
2616         mImpl->setGatherAxis(axis);
2617     }
2618
2626     int32_t getGatherAxis() const noexcept
2627     {
2628         return mImpl->getGatherAxis();
2629     }
2630
2649     void setNbElementWiseDims(int32_t elementWiseDims) noexcept
2650     {
2651         mImpl->setNbElementWiseDims(elementWiseDims);
2652     }
2653
2659     int32_t getNbElementWiseDims() const noexcept
2660     {
2661         return mImpl->getNbElementWiseDims();
2662     }
2663
2669     void setMode(GatherMode mode) noexcept
2670     {
2671         mImpl->setMode(mode);
2672     }
2673
2679     GatherMode getMode() const noexcept
2680     {
2681         return mImpl->getMode();
2682     }
2683
2684 protected:
2685     apiv::VGatherLayer* mImpl;
2686     virtual ~IGatherLayer() noexcept = default;
2687 };
2688
2700 class TRT_DEPRECATED IPluginV2Layer : public ILayer
2701 {
2702 public:
2708     IPluginV2& getPlugin() noexcept
2709     {
2710         return mImpl->getPlugin();
2711     }
2712
2713 protected:
2714     apiv::VPluginV2Layer* mImpl;
2715     virtual ~IPluginV2Layer() noexcept = default;
2716 };
2717
2727 class IPluginV3Layer : public ILayer
2728 {
2729 public:
2735     IPluginV3& getPlugin() noexcept
2736     {
2737         return mImpl->getPlugin();
2738     }
2739
2740 protected:
2741     apiv::VPluginV3Layer* mImpl;
2742     virtual ~IPluginV3Layer() noexcept = default;
2743 };
2744
```

```
2761 enum class UnaryOperation : int32_t
2762 {
2763     kEXP = 0,
2764     kLOG = 1,
2765     kSQRT = 2,
2766     kRECIP = 3,
2767     kABS = 4,
2768     kNEG = 5,
2769     kSIN = 6,
2770     kCOS = 7,
2771     kTAN = 8,
2772     kSINH = 9,
2773     kCOSH = 10,
2774     kASIN = 11,
2775     kACOS = 12,
2776     kATAN = 13,
2777     kASINH = 14,
2778     kACOSH = 15,
2779     kATANH = 16,
2780     kCEIL = 17,
2781     kFLOOR = 18,
2782     kERF = 19,
2783     kNOT = 20,
2784     kSIGN = 21,
2785     kROUND = 22,
2786     kISINF = 23,
2787     kISNAN = 24,
2788 };
2789
2795 template <>
2796 constexpr inline int32_t EnumMax<UnaryOperation>() noexcept
2797 {
2798     return 25;
2799 }
2800
2808 class IUnaryLayer : public ILayer
2809 {
2810 public:
2818     void setOperation(UnaryOperation op) noexcept
2819     {
2820         mImpl->setOperation(op);
2821     }
2822
2828     UnaryOperation getOperation() const noexcept
2829     {
2830         return mImpl->getOperation();
2831     }
2832
2833 protected:
2834     apiv::VUnaryLayer* mImpl;
2835     virtual ~IUnaryLayer() noexcept = default;
2836 };
2837
2856 enum class ReduceOperation : int32_t
2857 {
2858     kSUM = 0,
2859     kPROD = 1,
2860     kMAX = 2,
2861     kMIN = 3,
2862     kAVG = 4
2863 };
2864
2870 template <>
2871 constexpr inline int32_t EnumMax<ReduceOperation>() noexcept
2872 {
2873     return 5;
2874 }
2875
2883 class IReduceLayer : public ILayer
2884 {
2885 public:
2891     void setOperation(ReduceOperation op) noexcept
2892     {
2893         mImpl->setOperation(op);
2894     }
2895
2901     ReduceOperation getOperation() const noexcept
2902     {
2903         return mImpl->getOperation();
2904     }
2905
```

```
2911     void setReduceAxes(uint32_t reduceAxes) noexcept
2912     {
2913         mImpl->setReduceAxes(reduceAxes);
2914     }
2915
2921     uint32_t getReduceAxes() const noexcept
2922     {
2923         return mImpl->getReduceAxes();
2924     }
2925
2931     void setKeepDimensions(bool keepDimensions) noexcept
2932     {
2933         mImpl->setKeepDimensions(keepDimensions);
2934     }
2935
2941     bool getKeepDimensions() const noexcept
2942     {
2943         return mImpl->getKeepDimensions();
2944     }
2945
2946 protected:
2947     apiv::VReduceLayer* mImpl;
2948     virtual ~IReduceLayer() noexcept = default;
2949 };
2950
2963 class IPaddingLayer : public ILayer
2964 {
2965 public:
2975     void setPrePaddingNd(Dims const& padding) noexcept
2976     {
2977         mImpl->setPrePaddingNd(padding);
2978     }
2979
2987     Dims getPrePaddingNd() const noexcept
2988     {
2989         return mImpl->getPrePaddingNd();
2990     }
2991
3001     void setPostPaddingNd(Dims const& padding) noexcept
3002     {
3003         mImpl->setPostPaddingNd(padding);
3004     }
3005
3013     Dims getPostPaddingNd() const noexcept
3014     {
3015         return mImpl->getPostPaddingNd();
3016     }
3017
3018 protected:
3019     apiv::VPaddingLayer* mImpl;
3020     virtual ~IPaddingLayer() noexcept = default;
3021 };
3022
3028 struct Permutation
3029 {
3036     int32_t order[Dims::MAX_DIMS];
3037 };
3038
3051 class IShuffleLayer : public ILayer
3052 {
3053 public:
3063     void setFirstTranspose(Permutation permutation) noexcept
3064     {
3065         mImpl->setFirstTranspose(permutation);
3066     }
3067
3075     Permutation getFirstTranspose() const noexcept
3076     {
3077         return mImpl->getFirstTranspose();
3078     }
3079
3103     void setReshapeDimensions(Dims const& dimensions) noexcept
3104     {
3105         mImpl->setReshapeDimensions(dimensions);
3106     }
3107
3116     Dims getReshapeDimensions() const noexcept
3117     {
3118         return mImpl->getReshapeDimensions();
3119     }
3120
```

```
3126     //
3149     using ILayer::setInput;
3150
3163     void setSecondTranspose(Permutation permutation) noexcept
3164     {
3165         mImpl->setSecondTranspose(permutation);
3166     }
3167
3175     Permutation getSecondTranspose() const noexcept
3176     {
3177         return mImpl->getSecondTranspose();
3178     }
3179
3191     void setZeroIsPlaceholder(bool zeroIsPlaceholder) noexcept
3192     {
3193         return mImpl->setZeroIsPlaceholder(zeroIsPlaceholder);
3194     }
3195
3204     bool getZeroIsPlaceholder() const noexcept
3205     {
3206         return mImpl->getZeroIsPlaceholder();
3207     }
3208
3209 protected:
3210     apiv::VShuffleLayer* mImpl;
3211     virtual ~IShuffleLayer() noexcept = default;
3212 };
3213
3219 enum class SampleMode : int32_t
3220 {
3221     kSTRICT_BOUNDS = 0,
3222     kWRAP = 1,
3223     kCLAMP = 2,
3224     kFILL = 3,
3225     kREFLECT = 4,
3228 };
3229
3235 template <>
3236 constexpr inline int32_t EnumMax<SampleMode>() noexcept
3237 {
3238     return 5;
3239 }
3240
3303 class ISliceLayer : public ILayer
3304 {
3305 public:
3315     void setStart(Dims const& start) noexcept
3316     {
3317         mImpl->setStart(start);
3318     }
3319
3330     Dims getStart() const noexcept
3331     {
3332         return mImpl->getStart();
3333     }
3334
3344     void setSize(Dims const& size) noexcept
3345     {
3346         return mImpl->setSize(size);
3347     }
3348
3359     Dims getSize() const noexcept
3360     {
3361         return mImpl->getSize();
3362     }
3363
3373     void setStride(Dims const& stride) noexcept
3374     {
3375         mImpl->setStride(stride);
3376     }
3377
3388     Dims getStride() const noexcept
3389     {
3390         return mImpl->getStride();
3391     }
3392
3398     void setMode(SampleMode mode) noexcept
3399     {
3400         mImpl->setMode(mode);
3401     }
3402
```

```
3408     SampleMode getMode() const noexcept
3409     {
3410         return mImpl->getMode();
3411     }
3412
3440     using ILayer::setInput;
3441
3451     void setAxes(Dims const& axes) noexcept
3452     {
3453         mImpl->setAxes(axes);
3454     }
3455
3466     Dims getAxes() const noexcept
3467     {
3468         return mImpl->getAxes();
3469     }
3470
3471 protected:
3472     apiv::VSliceLayer* mImpl;
3473     virtual ~ISliceLayer() noexcept = default;
3474 };
3475
3488 class IShapeLayer : public ILayer
3489 {
3490 protected:
3491     apiv::VShapeLayer* mImpl;
3492     virtual ~IShapeLayer() noexcept = default;
3493 };
3494
3500 enum class TopKOperation : int32_t
3501 {
3502     kMAX = 0,
3503     kMIN = 1,
3504 };
3505
3511 template <>
3512 constexpr inline int32_t EnumMax<TopKOperation>() noexcept
3513 {
3514     return 2;
3515 }
3516
3528 class ITopKLayer : public ILayer
3529 {
3530 public:
3536     void setOperation(TopKOperation op) noexcept
3537     {
3538         mImpl->setOperation(op);
3539     }
3540
3546     TopKOperation getOperation() const noexcept
3547     {
3548         return mImpl->getOperation();
3549     }
3550
3560     void setK(int32_t k) noexcept
3561     {
3562         mImpl->setK(k);
3563     }
3564
3574     int32_t getK() const noexcept
3575     {
3576         return mImpl->getK();
3577     }
3578
3584     void setReduceAxes(uint32_t reduceAxes) noexcept
3585     {
3586         mImpl->setReduceAxes(reduceAxes);
3587     }
3588
3594     uint32_t getReduceAxes() const noexcept
3595     {
3596         return mImpl->getReduceAxes();
3597     }
3598
3613     using ILayer::setInput;
3614
3615 protected:
3616     apiv::VTopKLayer* mImpl;
3617     virtual ~ITopKLayer() noexcept = default;
3618 };
3619
```

```
3626 enum class MatrixOperation : int32_t
3627 {
3631     kNONE = 0,
3632
3634     kTRANSPOSE = 1,
3635
3646     kVECTOR = 2,
3647 };
3648
3654 template <>
3655 constexpr inline int32_t EnumMax<MatrixOperation>() noexcept
3656 {
3657     return 3;
3658 }
3659
3685 class IMatrixMultiplyLayer : public ILayer
3686 {
3687 public:
3696     void setOperation(int32_t index, MatrixOperation op) noexcept
3697     {
3698         mImpl->setOperation(index, op);
3699     }
3700
3708     MatrixOperation getOperation(int32_t index) const noexcept
3709     {
3710         return mImpl->getOperation(index);
3711     }
3712
3713 protected:
3714     apiv::VMatrixMultiplyLayer* mImpl;
3715     virtual ~IMatrixMultiplyLayer() noexcept = default;
3716 };
3717
3739 class INonZeroLayer : public ILayer
3740 {
3741 protected:
3742     virtual ~INonZeroLayer() noexcept = default;
3743     apiv::VNonZeroLayer* mImpl;
3744 };
3745
3760 class IRaggedSoftMaxLayer : public ILayer
3761 {
3762 protected:
3763     apiv::VRaggedSoftMaxLayer* mImpl;
3764     virtual ~IRaggedSoftMaxLayer() noexcept = default;
3765 };
3766
3793 class IIdentityLayer : public ILayer
3794 {
3795 protected:
3796     apiv::VIdentityLayer* mImpl;
3797     virtual ~IIdentityLayer() noexcept = default;
3798 };
3799
3806 class ICastLayer : public ILayer
3807 {
3808 public:
3816     void setToType(DataType toType) noexcept
3817     {
3818         mImpl->setToType(toType);
3819     }
3820
3827     DataType getToType() const noexcept
3828     {
3829         return mImpl->getToType();
3830     }
3831
3832 protected:
3833     apiv::VCastLayer* mImpl;
3834     virtual ~ICastLayer() noexcept = default;
3835 };
3836
3845 class IConstantLayer : public ILayer
3846 {
3847 public:
3856     void setWeights(Weights weights) noexcept
3857     {
3858         mImpl->setWeights(weights);
3859     }
3860
3866     Weights getWeights() const noexcept
```

```
3867     {
3868         return mImpl->getWeights();
3869     }
3870
3878     void setDimensions(Dims const& dimensions) noexcept
3879     {
3880         mImpl->setDimensions(dimensions);
3881     }
3882
3890     Dims getDimensions() const noexcept
3891     {
3892         return mImpl->getDimensions();
3893     }
3894
3895 protected:
3896     apiv::VConstantLayer* mImpl;
3897     virtual ~IConstantLayer() noexcept = default;
3898 };
3899
3909 class IParametricReLULayer : public ILayer
3910 {
3911 protected:
3912     apiv::VParametricReLULayer* mImpl;
3913     virtual ~IParametricReLULayer() noexcept = default;
3914 };
3915
3921 enum class InterpolationMode : int32_t
3922 {
3923     kNEAREST = 0,
3924     kLINEAR = 1,
3925     kCUBIC = 2
3926 };
3927
3928 namespace impl
3929 {
3935 template <>
3936 struct EnumMaxImpl<InterpolationMode>
3937 {
3938     static constexpr int32_t kVALUE = 3;
3939 };
3940 } // namespace impl
3941
3949 enum class ResizeCoordinateTransformation : int32_t
3950 {
3963     kALIGN_CORNERS = 0,
3964
3971     kASYMMETRIC = 1,
3972
3979     kHALF_PIXEL = 2,
3980 };
3981
3982 namespace impl
3983 {
3989 template <>
3990 struct EnumMaxImpl<ResizeCoordinateTransformation>
3991 {
3992     static constexpr int32_t kVALUE = 3;
3993 };
3994 } // namespace impl
3995
4003 enum class ResizeSelector : int32_t
4004 {
4006     kFORMULA = 0,
4007
4009     kUPPER = 1,
4010 };
4011
4012 namespace impl
4013 {
4019 template <>
4020 struct EnumMaxImpl<ResizeSelector>
4021 {
4022     static constexpr int32_t kVALUE = 2;
4023 };
4024 } // namespace impl
4025
4033 enum class ResizeRoundMode : int32_t
4034 {
4036     kHALF_UP = 0,
4037
4039     kHALF_DOWN = 1,
```

```
4040
4042     kFLOOR = 2,
4043
4045     kCEIL = 3,
4046 };
4047
4048 namespace impl
4049 {
4055 template <>
4056 struct EnumMaxImpl<ResizeRoundMode>
4057 {
4058     static constexpr int32_t kVALUE = 4;
4059 };
4060 } // namespace impl
4061
4098 class IResizeLayer : public ILayer
4099 {
4100 public:
4119     void setOutputDimensions(Dims const& dimensions) noexcept
4120     {
4121         return mImpl->setOutputDimensions(dimensions);
4122     }
4123
4129     Dims getOutputDimensions() const noexcept
4130     {
4131         return mImpl->getOutputDimensions();
4132     }
4133
4159     void setScales(float const* scales, int32_t nbScales) noexcept
4160     {
4161         mImpl->setScales(scales, nbScales);
4162     }
4163
4178     int32_t getScales(int32_t size, float* scales) const noexcept
4179     {
4180         return mImpl->getScales(size, scales);
4181     }
4182
4190     void setResizeMode(InterpolationMode interpolationMode) noexcept
4191     {
4192         mImpl->setResizeMode(interpolationMode);
4193     }
4194
4200     InterpolationMode getResizeMode() const noexcept
4201     {
4202         return mImpl->getResizeMode();
4203     }
4204
4224     using ILayer::setInput;
4225
4235     void setCoordinateTransformation(ResizeCoordinateTransformation coordTransform) noexcept
4236     {
4237         mImpl->setCoordinateTransformation(coordTransform);
4238     }
4239
4245     ResizeCoordinateTransformation getCoordinateTransformation() const noexcept
4246     {
4247         return mImpl->getCoordinateTransformation();
4248     }
4249
4260     void setSelectorForSinglePixel(ResizeSelector selector) noexcept
4261     {
4262         mImpl->setSelectorForSinglePixel(selector);
4263     }
4264
4270     ResizeSelector getSelectorForSinglePixel() const noexcept
4271     {
4272         return mImpl->getSelectorForSinglePixel();
4273     }
4274
4284     void setNearestRounding(ResizeRoundMode value) noexcept
4285     {
4286         mImpl->setNearestRounding(value);
4287     }
4288
4294     ResizeRoundMode getNearestRounding() const noexcept
4295     {
4296         return mImpl->getNearestRounding();
4297     }
4298
4316     void setCubicCoeff(float A) noexcept
```

```
4317     {
4318         mImpl->setCubicCoeff(A);
4319     }
4320
4326     float getCubicCoeff() const noexcept
4327     {
4328         return mImpl->getCubicCoeff();
4329     }
4330
4339     void setExcludeOutside(bool excludeFlag) noexcept
4340     {
4341         mImpl->setExcludeOutside(excludeFlag);
4342     }
4343
4349     bool getExcludeOutside() const noexcept
4350     {
4351         return mImpl->getExcludeOutside();
4352     }
4353
4354 protected:
4355     virtual ~IResizeLayer() noexcept = default;
4356     apiv::VResizeLayer* mImpl;
4357 };
4358
4364 enum class LoopOutput : int32_t
4365 {
4367     kLAST_VALUE = 0,
4368
4370     kCONCATENATE = 1,
4371
4373     kREVERSE = 2
4374 };
4375
4381 template <>
4382 constexpr inline int32_t EnumMax<LoopOutput>() noexcept
4383 {
4384     return 3;
4385 }
4386
4392 enum class TripLimit : int32_t
4393 {
4394
4395     kCOUNT = 0,
4396     kWHILE = 1
4397 };
4398
4404 template <>
4405 constexpr inline int32_t EnumMax<TripLimit>() noexcept
4406 {
4407     return 2;
4408 }
4409
4410 class ILoop;
4411
4425 class ILoopBoundaryLayer : public ILayer
4426 {
4427 public:
4431     ILoop* getLoop() const noexcept
4432     {
4433         return mBoundary->getLoop();
4434     }
4435
4436 protected:
4437     virtual ~ILoopBoundaryLayer() noexcept = default;
4438     apiv::VLoopBoundaryLayer* mBoundary;
4439 };
4440
4448 class IIfConditionalBoundaryLayer : public ILayer
4449 {
4450 public:
4454     IIfConditional* getConditional() const noexcept
4455     {
4456         return mBoundary->getConditional();
4457     }
4458
4459 protected:
4460     virtual ~IIfConditionalBoundaryLayer() noexcept = default;
4461     apiv::VConditionalBoundaryLayer* mBoundary;
4462 };
4463
4469 class IConditionLayer : public IIfConditionalBoundaryLayer
```

```
4470 {
4471 public:
4472 protected:
4473     virtual ~IConditionLayer() noexcept = default;
4474     apiv::VConditionLayer* mImpl;
4475 };
4476
4486 class IIfConditionalOutputLayer : public IIfConditionalBoundaryLayer
4487 {
4488 public:
4489 protected:
4490     virtual ~IIfConditionalOutputLayer() noexcept = default;
4491     apiv::VConditionalOutputLayer* mImpl;
4492 };
4493
4499 class IIfConditionalInputLayer : public IIfConditionalBoundaryLayer
4500 {
4501 public:
4502 protected:
4503     virtual ~IIfConditionalInputLayer() noexcept = default;
4504     apiv::VConditionalInputLayer* mImpl;
4505 };
4506
4530 class IIfConditional : public INoCopy
4531 {
4532 public:
4542     IConditionLayer* setCondition(ITensor& condition) noexcept
4543     {
4544         return mImpl->setCondition(condition);
4545     }
4546
4560     IIfConditionalOutputLayer* addOutput(ITensor& trueSubgraphOutput, ITensor& falseSubgraphOutput)
      noexcept
4561     {
4562         return mImpl->addOutput(trueSubgraphOutput, falseSubgraphOutput);
4563     }
4564
4572     IIfConditionalInputLayer* addInput(ITensor& input) noexcept
4573     {
4574         return mImpl->addInput(input);
4575     }
4576
4587     void setName(char const* name) noexcept
4588     {
4589         mImpl->setName(name);
4590     }
4591
4597     char const* getName() const noexcept
4598     {
4599         return mImpl->getName();
4600     }
4601
4602 protected:
4603     virtual ~IIfConditional() noexcept = default;
4604     apiv::VIfConditional* mImpl;
4605 };
4606
4614 class IRecurrenceLayer : public ILoopBoundaryLayer
4615 {
4616 public:
4622     //
4635     using ILayer::setInput;
4636
4637 protected:
4638     virtual ~IRecurrenceLayer() noexcept = default;
4639     apiv::VRecurrenceLayer* mImpl;
4640 };
4641
4661 class ILoopOutputLayer : public ILoopBoundaryLayer
4662 {
4663 public:
4667     LoopOutput getLoopOutput() const noexcept
4668     {
4669         return mImpl->getLoopOutput();
4670     }
4671
4684     void setAxis(int32_t axis) noexcept
4685     {
4686         mImpl->setAxis(axis);
4687     }
4688
```

```
4692     int32_t getAxis() const noexcept
4693     {
4694         return mImpl->getAxis();
4695     }
4696 
4702     //
4717     using ILayer::setInput;
4718 
4719 protected:
4720     virtual ~ILoopOutputLayer() noexcept = default;
4721     apiv::VLoopOutputLayer* mImpl;
4722 };
4723 
4735 class ITripLimitLayer : public ILoopBoundaryLayer
4736 {
4737 public:
4741     TripLimit getTripLimit() const noexcept
4742     {
4743         return mImpl->getTripLimit();
4744     }
4745 
4746 protected:
4747     virtual ~ITripLimitLayer() noexcept = default;
4748     apiv::VTripLimitLayer* mImpl;
4749 };
4750 
4761 class IIteratorLayer : public ILoopBoundaryLayer
4762 {
4763 public:
4767     void setAxis(int32_t axis) noexcept
4768     {
4769         mImpl->setAxis(axis);
4770     }
4771 
4775     int32_t getAxis() const noexcept
4776     {
4777         return mImpl->getAxis();
4778     }
4779 
4789     void setReverse(bool reverse) noexcept
4790     {
4791         mImpl->setReverse(reverse);
4792     }
4793 
4799     bool getReverse() const noexcept
4800     {
4801         return mImpl->getReverse();
4802     }
4803 
4804 protected:
4805     virtual ~IIteratorLayer() noexcept = default;
4806     apiv::VIteratorLayer* mImpl;
4807 };
4808 
4818 class ILoop : public INoCopy
4819 {
4820 public:
4827     IRecurrenceLayer* addRecurrence(ITensor& initialValue) noexcept
4828     {
4829         return mImpl->addRecurrence(initialValue);
4830     }
4831 
4848     ITripLimitLayer* addTripLimit(ITensor& tensor, TripLimit limit) noexcept
4849     {
4850         return mImpl->addTripLimit(tensor, limit);
4851     }
4852 
4861     IIteratorLayer* addIterator(ITensor& tensor, int32_t axis = 0, bool reverse = false) noexcept
4862     {
4863         return mImpl->addIterator(tensor, axis, reverse);
4864     }
4865 
4874     ILoopOutputLayer* addLoopOutput(ITensor& tensor, LoopOutput outputKind, int32_t axis = 0) noexcept
4875     {
4876         return mImpl->addLoopOutput(tensor, outputKind, axis);
4877     }
4878 
4889     void setName(char const* name) noexcept
4890     {
4891         mImpl->setName(name);
4892     }
```

```
4893
4899     char const* getName() const noexcept
4900     {
4901         return mImpl->getName();
4902     }
4903
4904 protected:
4905     virtual ~ILoop() noexcept = default;
4906     apiv::VLoop* mImpl;
4907 };
4908
4921 class ISelectLayer : public ILayer
4922 {
4923 protected:
4924     virtual ~ISelectLayer() noexcept = default;
4925     apiv::VSelectLayer* mImpl;
4926 };
4927
4943 class IAssertionLayer : public ILayer
4944 {
4945 public:
4954     void setMessage(char const* message) noexcept
4955     {
4956         mImpl->setMessage(message);
4957     }
4958
4964     char const* getMessage() const noexcept
4965     {
4966         return mImpl->getMessage();
4967     }
4968
4969 protected:
4970     virtual ~IAssertionLayer() noexcept = default;
4971
4972     apiv::VAssertionLayer* mImpl;
4973 };
4974
4982 enum class FillOperation : int32_t
4983 {
4999     kLINSPACE = 0,
5000
5002     kRANDOM_UNIFORM = 1,
5003
5005     kRANDOM_NORMAL = 2
5006 };
5007
5013 template <>
5014 constexpr inline int32_t EnumMax<FillOperation>() noexcept
5015 {
5016     return 3;
5017 }
5018
5054 class IFillLayer : public ILayer
5055 {
5056 public:
5065     //
5066     void setDimensions(Dims const& dimensions) noexcept
5067     {
5068         mImpl->setDimensions(dimensions);
5069     }
5070
5081     Dims getDimensions() const noexcept
5082     {
5083         return mImpl->getDimensions();
5084     }
5085
5091     void setOperation(FillOperation op) noexcept
5092     {
5093         mImpl->setOperation(op);
5094     }
5095
5101     FillOperation getOperation() const noexcept
5102     {
5103         return mImpl->getOperation();
5104     }
5105
5119     //
5120     void setAlpha(double alpha) noexcept
5121     {
5122         mImpl->setAlpha(alpha);
5123     }
```

```
5124
5135     double getAlpha() const noexcept
5136     {
5137         return mImpl->getAlpha();
5138     }
5139
5154     void setBeta(double beta) noexcept
5155     {
5156         mImpl->setBeta(beta);
5157     }
5158
5169     double getBeta() const noexcept
5170     {
5171         return mImpl->getBeta();
5172     }
5173
5214     using ILayer::setInput;
5215
5229     //
5230     void setAlphaInt64(int64_t alpha) noexcept
5231     {
5232         mImpl->setAlphaInt64(alpha);
5233     }
5234
5245     int64_t getAlphaInt64() const noexcept
5246     {
5247         return mImpl->getAlphaInt64();
5248     }
5249
5264     void setBetaInt64(int64_t beta) noexcept
5265     {
5266         mImpl->setBetaInt64(beta);
5267     }
5268
5279     int64_t getBetaInt64() const noexcept
5280     {
5281         return mImpl->getBetaInt64();
5282     }
5283
5287     bool isAlphaBetaInt64() const noexcept
5288     {
5289         return mImpl->isAlphaBetaInt64();
5290     }
5291
5304     void setToType(DataType toType) noexcept
5305     {
5306         mImpl->setToType(toType);
5307     }
5308
5316     DataType getToType() const noexcept
5317     {
5318         return mImpl->getToType();
5319     }
5320
5321 protected:
5322     virtual ~IFillLayer() noexcept = default;
5323     apiv::VFillLayer* mImpl;
5324 };
5325
5400 class IQuantizeLayer : public ILayer
5401 {
5402 public:
5411     int32_t getAxis() const noexcept
5412     {
5413         return mImpl->getAxis();
5414     }
5422     void setAxis(int32_t axis) noexcept
5423     {
5424         mImpl->setAxis(axis);
5425     }
5426
5438     void setToType(DataType toType) noexcept
5439     {
5440         mImpl->setToType(toType);
5441     }
5442
5450     DataType getToType() const noexcept
5451     {
5452         return mImpl->getToType();
5453     }
5454
```

```
5455 protected:
5456     virtual ~IQuantizeLayer() noexcept = default;
5457     apiv::VQuantizeLayer* mImpl;
5458 };
5459
5531 class IDequantizeLayer : public ILayer
5532 {
5533 public:
5542     int32_t getAxis() const noexcept
5543     {
5544         return mImpl->getAxis();
5545     }
5553     void setAxis(int32_t axis) noexcept
5554     {
5555         mImpl->setAxis(axis);
5556     }
5557
5569     void setToType(DataType toType) noexcept
5570     {
5571         mImpl->setToType(toType);
5572     }
5573
5581     DataType getToType() const noexcept
5582     {
5583         return mImpl->getToType();
5584     }
5585
5586 protected:
5587     virtual ~IDequantizeLayer() noexcept = default;
5588     apiv::VDequantizeLayer* mImpl;
5589 };
5590
5608 class IDynamicQuantizeLayer : public ILayer
5609 {
5610 public:
5622     using ILayer::setInput;
5623
5636     void setToType(DataType toType) noexcept
5637     {
5638         mImpl->setToType(toType);
5639     }
5640
5649     DataType getToType() const noexcept
5650     {
5651         return mImpl->getToType();
5652     }
5653
5661     void setScaleType(DataType scaleType) noexcept
5662     {
5663         mImpl->setScaleType(scaleType);
5664     }
5665
5674     DataType getScaleType() const noexcept
5675     {
5676         return mImpl->getScaleType();
5677     }
5678
5687     void setAxis(int32_t axis) noexcept
5688     {
5689         mImpl->setAxis(axis);
5690     }
5691
5697     int32_t getAxis() const noexcept
5698     {
5699         return mImpl->getAxis();
5700     }
5701
5710     void setBlockSize(int32_t size) noexcept
5711     {
5712         mImpl->setBlockSize(size);
5713     }
5714
5720     int32_t getBlockSize() const noexcept
5721     {
5722         return mImpl->getBlockSize();
5723     }
5724
5725 protected:
5726     virtual ~IDynamicQuantizeLayer() noexcept = default;
5727     apiv::VDynamicQuantizeLayer* mImpl;
5728 };
```

```
5729
5766 class IEinsumLayer : public ILayer
5767 {
5768 public:
5778     bool setEquation(char const* equation) noexcept
5779     {
5780         return mImpl->setEquation(equation);
5781     }
5782
5788     char const* getEquation() const noexcept
5789     {
5790         return mImpl->getEquation();
5791     }
5792
5793 protected:
5794     virtual ~IEinsumLayer() noexcept = default;
5795     apiv::VEinsumLayer* mImpl;
5796 };
5797
5805 enum class ScatterMode : int32_t
5806 {
5807     kELEMENT = 0,
5808     kND = 1,
5809 };
5810
5816 template <>
5817 constexpr inline int32_t EnumMax<ScatterMode>() noexcept
5818 {
5819     return 2;
5820 }
5821
5879 class IScatterLayer : public ILayer
5880 {
5881 public:
5887     void setMode(ScatterMode mode) noexcept
5888     {
5889         mImpl->setMode(mode);
5890     }
5891
5897     ScatterMode getMode() const noexcept
5898     {
5899         return mImpl->getMode();
5900     }
5901
5907     void setAxis(int32_t axis) noexcept
5908     {
5909         mImpl->setAxis(axis);
5910     }
5911
5915     int32_t getAxis() const noexcept
5916     {
5917         return mImpl->getAxis();
5918     }
5919
5920 protected:
5921     apiv::VScatterLayer* mImpl;
5922     virtual ~IScatterLayer() noexcept = default;
5923 }; // class IScatterLayer
5924
5951 class IOneHotLayer : public ILayer
5952 {
5953 public:
5959     void setAxis(int32_t axis) noexcept
5960     {
5961         mImpl->setAxis(axis);
5962     }
5963
5967     int32_t getAxis() const noexcept
5968     {
5969         return mImpl->getAxis();
5970     }
5971
5972 protected:
5973     apiv::VOneHotLayer* mImpl;
5974     virtual ~IOneHotLayer() noexcept = default;
5975 };
5976
5988 class IGridSampleLayer : public ILayer
5989 {
5990 public:
5996     void setInterpolationMode(InterpolationMode mode) noexcept
```

```
5997     {
5998         mImpl->setInterpolationMode(mode);
5999     }
6000
6008     InterpolationMode getInterpolationMode() const noexcept
6009     {
6010         return mImpl->getInterpolationMode();
6011     }
6012
6018     void setAlignCorners(bool alignCorners) noexcept
6019     {
6020         mImpl->setAlignCorners(alignCorners);
6021     }
6022
6030     bool getAlignCorners() const noexcept
6031     {
6032         return mImpl->getAlignCorners();
6033     }
6034
6042     bool setSampleMode(SampleMode mode) noexcept
6043     {
6044         return mImpl->setSampleMode(mode);
6045     }
6046
6054     SampleMode getSampleMode() const noexcept
6055     {
6056         return mImpl->getSampleMode();
6057     }
6058
6059 protected:
6060     apiv::VGridSampleLayer* mImpl;
6061     virtual ~IGridSampleLayer() noexcept = default;
6062 }; // class IGridSampleLayer
6063
6071 enum class BoundingBoxFormat : int32_t
6072 {
6074     kCORNER_PAIRS = 0,
6076     kCENTER_SIZES = 1
6077 };
6078
6084 template <>
6085 constexpr inline int32_t EnumMax<BoundingBoxFormat>() noexcept
6086 {
6087     return 2;
6088 }
6089
6136 class INMSLayer : public ILayer
6137 {
6138 public:
6148     void setBoundingBoxFormat(BoundingBoxFormat fmt) noexcept
6149     {
6150         mImpl->setBoundingBoxFormat(fmt);
6151     }
6152
6160     BoundingBoxFormat getBoundingBoxFormat() const noexcept
6161     {
6162         return mImpl->getBoundingBoxFormat();
6163     }
6164
6174     void setTopKBoxLimit(int32_t limit) noexcept
6175     {
6176         mImpl->setTopKBoxLimit(limit);
6177     }
6178
6184     int32_t getTopKBoxLimit() const noexcept
6185     {
6186         return mImpl->getTopKBoxLimit();
6187     }
6188
6207     using ILayer::setInput;
6208
6209 protected:
6210     apiv::VNMSLayer* mImpl;
6211     virtual ~INMSLayer() noexcept = default;
6212 }; // class INMSLayer
6213
6226 class IReverseSequenceLayer : public ILayer
6227 {
6228 public:
6237     void setBatchAxis(int32_t batchAxis) noexcept
6238     {
```

```
6239         mImpl->setBatchAxis(batchAxis);
6240     }
6241
6247     int32_t getBatchAxis() const noexcept
6248     {
6249         return mImpl->getBatchAxis();
6250     }
6251
6260     void setSequenceAxis(int32_t sequenceAxis) noexcept
6261     {
6262         mImpl->setSequenceAxis(sequenceAxis);
6263     }
6264
6270     int32_t getSequenceAxis() const noexcept
6271     {
6272         return mImpl->getSequenceAxis();
6273     }
6274
6275 protected:
6276     apiv::VReverseSequenceLayer* mImpl;
6277     virtual ~IReverseSequenceLayer() noexcept = default;
6278 }; // class IReverseSequenceLayer
6279
6298 class INormalizationLayer : public ILayer
6299 {
6300 public:
6308     void setEpsilon(float eps) noexcept
6309     {
6310         return mImpl->setEpsilon(eps);
6311     }
6312
6318     float getEpsilon() const noexcept
6319     {
6320         return mImpl->getEpsilon();
6321     }
6322
6328     void setAxes(uint32_t axesMask) noexcept
6329     {
6330         return mImpl->setAxes(axesMask);
6331     }
6332
6338     uint32_t getAxes() const noexcept
6339     {
6340         return mImpl->getAxes();
6341     }
6342
6359     void setNbGroups(int64_t nbGroups) noexcept
6360     {
6361         return mImpl->setNbGroups(nbGroups);
6362     }
6363
6369     int64_t getNbGroups() const noexcept
6370     {
6371         return mImpl->getNbGroups();
6372     }
6373
6395     void setComputePrecision(DataType type) noexcept
6396     {
6397         return mImpl->setComputePrecision(type);
6398     }
6399
6405     DataType getComputePrecision() const noexcept
6406     {
6407         return mImpl->getComputePrecision();
6408     }
6409
6410 protected:
6411     apiv::VNormalizationLayer* mImpl;
6412     virtual ~INormalizationLayer() noexcept = default;
6413 };
6414
6415
6424 class ISqueezeLayer : public ILayer
6425 {
6426 public:
6439     using ILayer::setInput;
6440
6441 protected:
6442     apiv::VSqueezeLayer* mImpl;
6443     virtual ~ISqueezeLayer() noexcept = default;
6444 };
```

```
6445
6453 class IUnsqueezeLayer : public ILayer
6454 {
6455 public:
6468     using ILayer::setInput;
6469
6470 protected:
6471     apiv::VUnsqueezeLayer* mImpl;
6472     virtual ~IUnsqueezeLayer() noexcept = default;
6473 };
6474
6486 enum class CumulativeOperation : int32_t
6487 {
6488     kSUM = 0,
6489 };
6490
6491 namespace impl
6492 {
6493
6499 template <>
6500 struct EnumMaxImpl<CumulativeOperation>
6501 {
6502     static constexpr int32_t kVALUE = 1;
6503 };
6504
6505 } // namespace impl
6506
6534 class ICumulativeLayer : public ILayer
6535 {
6536 public:
6546     bool setOperation(CumulativeOperation op) noexcept
6547     {
6548         return mImpl->setOperation(op);
6549     }
6550
6558     CumulativeOperation getOperation() const noexcept
6559     {
6560         return mImpl->getOperation();
6561     }
6562
6570     void setExclusive(bool exclusive) noexcept
6571     {
6572         mImpl->setExclusive(exclusive);
6573     }
6574
6582     bool getExclusive() const noexcept
6583     {
6584         return mImpl->getExclusive();
6585     }
6586
6594     void setReverse(bool reverse) noexcept
6595     {
6596         mImpl->setReverse(reverse);
6597     }
6598
6606     bool getReverse() const noexcept
6607     {
6608         return mImpl->getReverse();
6609     }
6610
6611 protected:
6612     apiv::VCumulativeLayer* mImpl;
6613     virtual ~ICumulativeLayer() noexcept = default;
6614 };
6615
6633 class INetworkDefinition : public INoCopy
6634 {
6635 public:
6636     virtual ~INetworkDefinition() noexcept = default;
6637
6674     ITensor* addInput(char const* name, DataType type, Dims const& dimensions) noexcept
6675     {
6676         return mImpl->addInput(name, type, dimensions);
6677     }
6678
6688     void markOutput(ITensor& tensor) noexcept
6689     {
6690         mImpl->markOutput(tensor);
6691     }
6692
6706     bool markDebug(ITensor& tensor) noexcept
```

```
6707     {
6708         return mImpl->markDebug(tensor);
6709     }
6710
6722     bool unmarkDebug(ITensor& tensor) noexcept
6723     {
6724         return mImpl->unmarkDebug(tensor);
6725     }
6726
6732     bool isDebugTensor(nvinfer1::ITensor const& tensor) const noexcept
6733     {
6734         return mImpl->isDebugTensor(tensor);
6735     }
6736
6752     IActivationLayer* addActivation(ITensor& input, ActivationType type) noexcept
6753     {
6754         return mImpl->addActivation(input, type);
6755     }
6756
6771     ILRNLayer* addLRN(ITensor& input, int64_t window, float alpha, float beta, float k) noexcept
6772     {
6773         return mImpl->addLRN(input, window, alpha, beta, k);
6774     }
6775
6797     IScaleLayer* addScale(ITensor& input, ScaleMode mode, Weights shift, Weights scale, Weights power)
    noexcept
6798     {
6799         return mImpl->addScale(input, mode, shift, scale, power);
6800     }
6801
6810     ISoftMaxLayer* addSoftMax(ITensor& input) noexcept
6811     {
6812         return mImpl->addSoftMax(input);
6813     }
6814
6827     IConcatenationLayer* addConcatenation(ITensor* const* inputs, int32_t nbInputs) noexcept
6828     {
6829         return mImpl->addConcatenation(inputs, nbInputs);
6830     }
6831
6854     IElementWiseLayer* addElementWise(ITensor& input1, ITensor& input2, ElementWiseOperation op) noexcept
6855     {
6856         return mImpl->addElementWise(input1, input2, op);
6857     }
6858
6876     IUnaryLayer* addUnary(ITensor& input, UnaryOperation operation) noexcept
6877     {
6878         return mImpl->addUnary(input, operation);
6879     }
6880
6890     IShuffleLayer* addShuffle(ITensor& input) noexcept
6891     {
6892         return mImpl->addShuffle(input);
6893     }
6894
6907     IOneHotLayer* addOneHot(ITensor& indices, ITensor& values, ITensor& depth, int32_t axis) noexcept
6908     {
6909         return mImpl->addOneHot(indices, values, depth, axis);
6910     }
6911
6919     int32_t getNbLayers() const noexcept
6920     {
6921         return mImpl->getNbLayers();
6922     }
6923
6933     ILayer* getLayer(int32_t index) const noexcept
6934     {
6935         return mImpl->getLayer(index);
6936     }
6937
6945     int32_t getNbInputs() const noexcept
6946     {
6947         return mImpl->getNbInputs();
6948     }
6949
6961     ITensor* getInput(int32_t index) const noexcept
6962     {
6963         return mImpl->getInput(index);
6964     }
6965
6975     int32_t getNbOutputs() const noexcept
```

```
6976     {
6977         return mImpl->getNbOutputs();
6978     }
6979
6991     ITensor* getOutput(int32_t index) const noexcept
6992     {
6993         return mImpl->getOutput(index);
6994     }
6995
7017     IReduceLayer* addReduce(
7018         ITensor& input, ReduceOperation operation, uint32_t reduceAxes, bool keepDimensions) noexcept
7019     {
7020         return mImpl->addReduce(input, operation, reduceAxes, keepDimensions);
7021     }
7022
7050     ITopKLayer* addTopK(ITensor& input, TopKOperation op, int32_t k, uint32_t reduceAxes) noexcept
7051     {
7052         return mImpl->addTopK(input, op, k, reduceAxes);
7053     }
7054
7066     IGatherLayer* addGather(ITensor& data, ITensor& indices, int32_t axis) noexcept
7067     {
7068         return mImpl->addGather(data, indices, axis);
7069     }
7070
7082     IGatherLayer* addGatherV2(ITensor& data, ITensor& indices, GatherMode mode) noexcept
7083     {
7084         return mImpl->addGatherV2(data, indices, mode);
7085     }
7086
7101     IRaggedSoftMaxLayer* addRaggedSoftMax(ITensor& input, ITensor& bounds) noexcept
7102     {
7103         return mImpl->addRaggedSoftMax(input, bounds);
7104     }
7105
7122     IMatrixMultiplyLayer* addMatrixMultiply(
7123         ITensor& input0, MatrixOperation op0, ITensor& input1, MatrixOperation op1) noexcept
7124     {
7125         return mImpl->addMatrixMultiply(input0, op0, input1, op1);
7126     }
7127
7137     INonZeroLayer* addNonZero(ITensor& input) noexcept
7138     {
7139         return mImpl->addNonZero(input);
7140     }
7141
7161     IConstantLayer* addConstant(Dims const& dimensions, Weights weights) noexcept
7162     {
7163         return mImpl->addConstant(dimensions, weights);
7164     }
7165
7175     IIdentityLayer* addIdentity(ITensor& input) noexcept
7176     {
7177         return mImpl->addIdentity(input);
7178     }
7179
7190     ICastLayer* addCast(ITensor& input, DataType toType) noexcept
7191     {
7192         return mImpl->addCast(input, toType);
7193     }
7194
7205     void removeTensor(ITensor& tensor) noexcept
7206     {
7207         mImpl->removeTensor(tensor);
7208     }
7209
7217     void unmarkOutput(ITensor& tensor) noexcept
7218     {
7219         mImpl->unmarkOutput(tensor);
7220     }
7221
7238     TRT_DEPRECATED IPluginV2Layer* addPluginV2(ITensor* const* inputs, int32_t nbInputs, IPluginV2& plugin)
    noexcept
7239     {
7240         return mImpl->addPluginV2(inputs, nbInputs, plugin);
7241     }
7242
7256     IPluginV3Layer* addPluginV3(ITensor* const* inputs, int32_t nbInputs, ITensor* const* shapeInputs,
7257         int32_t nbShapeInputs, IPluginV3& plugin) noexcept
7258     {
7259         return mImpl->addPluginV3(inputs, nbInputs, shapeInputs, nbShapeInputs, plugin);
```

```
7260     }
7261
7276     ISliceLayer* addSlice(ITensor& input, Dims const& start, Dims const& size, Dims const& stride) noexcept
7277     {
7278         return mImpl->addSlice(input, start, size, stride);
7279     }
7280
7300     void setName(char const* name) noexcept
7301     {
7302         mImpl->setName(name);
7303     }
7304
7314     char const* getName() const noexcept
7315     {
7316         return mImpl->getName();
7317     }
7318
7330     IShapeLayer* addShape(ITensor& input) noexcept
7331     {
7332         return mImpl->addShape(input);
7333     }
7334
7344     TRT_DEPRECATED bool hasImplicitBatchDimension() const noexcept
7345     {
7346         return mImpl->hasImplicitBatchDimension();
7347     }
7348
7354     NetworkDefinitionCreationFlags getFlags() const noexcept
7355     {
7356         return mImpl->getFlags();
7357     }
7358
7366     bool getFlag(NetworkDefinitionCreationFlag networkDefinitionCreationFlag) const noexcept
7367     {
7368         return mImpl->getFlag(networkDefinitionCreationFlag);
7369     }
7370
7383     bool markOutputForShapes(ITensor& tensor) noexcept
7384     {
7385         return mImpl->markOutputForShapes(tensor);
7386     }
7387
7395     bool unmarkOutputForShapes(ITensor& tensor) noexcept
7396     {
7397         return mImpl->unmarkOutputForShapes(tensor);
7398     }
7399
7413     IParametricReLULayer* addParametricReLU(ITensor& input, ITensor& slope) noexcept
7414     {
7415         return mImpl->addParametricReLU(input, slope);
7416     }
7417
7435     IConvolutionLayer* addConvolutionNd(
7436         ITensor& input, int64_t nbOutputMaps, Dims const& kernelSize, Weights kernelWeights, Weights
    biasWeights) noexcept
7437     {
7438         return mImpl->addConvolutionNd(input, nbOutputMaps, kernelSize, kernelWeights, biasWeights);
7439     }
7440
7455     IPoolingLayer* addPoolingNd(ITensor& input, PoolingType type, Dims const& windowSize) noexcept
7456     {
7457         return mImpl->addPoolingNd(input, type, windowSize);
7458     }
7459
7474     //
7477     IDeconvolutionLayer* addDeconvolutionNd(
7478         ITensor& input, int64_t nbOutputMaps, Dims kernelSize, Weights kernelWeights, Weights biasWeights)
    noexcept
7479     {
7480         return mImpl->addDeconvolutionNd(input, nbOutputMaps, kernelSize, kernelWeights, biasWeights);
7481     }
7482
7514     IScaleLayer* addScaleNd(
7515         ITensor& input, ScaleMode mode, Weights shift, Weights scale, Weights power, int32_t channelAxis)
    noexcept
7516     {
7517         return mImpl->addScaleNd(input, mode, shift, scale, power, channelAxis);
7518     }
7519
7531     IResizeLayer* addResize(ITensor& input) noexcept
7532     {
```

```
7533        return mImpl->addResize(input);
7534    }
7535
7545    ILoop* addLoop() noexcept
7546    {
7547        return mImpl->addLoop();
7548    }
7549
7560    IIfConditional* addIfConditional() noexcept
7561    {
7562        return mImpl->addIfConditional();
7563    }
7564
7599    ISelectLayer* addSelect(ITensor& condition, ITensor& thenInput, ITensor& elseInput) noexcept
7600    {
7601        return mImpl->addSelect(condition, thenInput, elseInput);
7602    }
7603
7616    IAssertionLayer* addAssertion(ITensor& condition, char const* message) noexcept
7617    {
7618        return mImpl->addAssertion(condition, message);
7619    }
7620
7641    TRT_DEPRECATED IFillLayer* addFill(Dims const& dimensions, FillOperation op) noexcept
7642    {
7643        return mImpl->addFill(dimensions, op);
7644    }
7645
7667    IFillLayer* addFill(Dims const& dimensions, FillOperation op, DataType outputType) noexcept
7668    {
7669        return mImpl->addFillV2(dimensions, op, outputType);
7670    }
7671
7683    IPaddingLayer* addPaddingNd(ITensor& input, Dims const& prePadding, Dims const& postPadding) noexcept
7684    {
7685        return mImpl->addPaddingNd(input, prePadding, postPadding);
7686    }
7687
7707    bool setWeightsName(Weights weights, char const* name) noexcept
7708    {
7709        return mImpl->setWeightsName(weights, name);
7710    }
7711
7723    //
7726    void setErrorRecorder(IErrorRecorder* recorder) noexcept
7727    {
7728        mImpl->setErrorRecorder(recorder);
7729    }
7730
7741    IErrorRecorder* getErrorRecorder() const noexcept
7742    {
7743        return mImpl->getErrorRecorder();
7744    }
7745
7762    TRT_DEPRECATED IDequantizeLayer* addDequantize(ITensor& input, ITensor& scale) noexcept
7763    {
7764        return mImpl->addDequantize(input, scale);
7765    }
7766
7784    IDequantizeLayer* addDequantize(ITensor& input, ITensor& scale, DataType outputType) noexcept
7785    {
7786        return mImpl->addDequantizeV2(input, scale, outputType);
7787    }
7788
7804    IScatterLayer* addScatter(ITensor& data, ITensor& indices, ITensor& updates, ScatterMode mode) noexcept
7805    {
7806        return mImpl->addScatter(data, indices, updates, mode);
7807    }
7808
7825    TRT_DEPRECATED IQuantizeLayer* addQuantize(ITensor& input, ITensor& scale) noexcept
7826    {
7827        return mImpl->addQuantize(input, scale);
7828    }
7829
7847    IQuantizeLayer* addQuantize(ITensor& input, ITensor& scale, DataType outputType) noexcept
7848    {
7849        return mImpl->addQuantizeV2(input, scale, outputType);
7850    }
7851
7872    IDynamicQuantizeLayer* addDynamicQuantize(
7873        ITensor& input, int32_t axis, int32_t blockSize, DataType outputType, DataType scaleType) noexcept
```

```
7874     {
7875         return mImpl->addDynamicQuantize(input, axis, blockSize, outputType, scaleType);
7876     }
7877
7888     IEinsumLayer* addEinsum(ITensor* const* inputs, int32_t nbInputs, char const* equation) noexcept
7889     {
7890         return mImpl->addEinsum(inputs, nbInputs, equation);
7891     }
7892
7906     IGridSampleLayer* addGridSample(ITensor& input, ITensor& grid) noexcept
7907     {
7908         return mImpl->addGridSample(input, grid);
7909     }
7910
7924     INMSLayer* addNMS(ITensor& boxes, ITensor& scores, ITensor& maxOutputBoxesPerClass) noexcept
7925     {
7926         return mImpl->addNMS(boxes, scores, maxOutputBoxesPerClass);
7927     }
7928
7941     IReverseSequenceLayer* addReverseSequence(ITensor& input, ITensor& sequenceLens) noexcept
7942     {
7943         return mImpl->addReverseSequence(input, sequenceLens);
7944     }
7945
7967     INormalizationLayer* addNormalization(ITensor& input, ITensor& scale, ITensor& bias, uint32_t axesMask)
    noexcept
7968     {
7969         return mImpl->addNormalization(input, scale, bias, axesMask);
7970     }
7971
7989     ICumulativeLayer* addCumulative(ITensor& input, ITensor& axis, CumulativeOperation operation, bool
    exclusive, bool reverse) noexcept
7990     {
7991         return mImpl->addCumulative(input, axis, operation, exclusive, reverse);
7992     }
7993
8000     virtual IBuilder& getBuilder() const noexcept
8001     {
8002         return mImpl->getBuilder();
8003     }
8004
8013     bool markWeightsRefittable(char const* name) noexcept
8014     {
8015         return mImpl->markWeightsRefittable(name);
8016     }
8017
8025     bool unmarkWeightsRefittable(char const* name) noexcept
8026     {
8027         return mImpl->unmarkWeightsRefittable(name);
8028     }
8029
8038     bool areWeightsMarkedRefittable(char const* name) const noexcept
8039     {
8040         return mImpl->areWeightsMarkedRefittable(name);
8041     }
8042
8057     ISqueezeLayer* addSqueeze(ITensor& input, ITensor& axes) noexcept
8058     {
8059         return mImpl->addSqueeze(input, axes);
8060     }
8061
8078     IUnsqueezeLayer* addUnsqueeze(ITensor& input, ITensor& axes) noexcept
8079     {
8080         return mImpl->addUnsqueeze(input, axes);
8081     }
8082
8083 protected:
8084     apiv::VNetworkDefinition* mImpl;
8085 };
8086
8094 enum class CalibrationAlgoType : int32_t
8095 {
8096     kLEGACY_CALIBRATION TRT_DEPRECATED_ENUM = 0,
8097     kENTROPY_CALIBRATION TRT_DEPRECATED_ENUM = 1,
8098     kENTROPY_CALIBRATION_2 TRT_DEPRECATED_ENUM = 2,
8099     kMINMAX_CALIBRATION TRT_DEPRECATED_ENUM = 3,
8100 };
8101
8107 template <>
8108 constexpr inline int32_t EnumMax<CalibrationAlgoType>() noexcept
8109 {
```

```
8110     return 4;
8111 }
8112
8126 class TRT_DEPRECATED IInt8Calibrator : public IVersionedInterface
8127 {
8128 public:
8136     TRT_DEPRECATED virtual int32_t getBatchSize() const noexcept = 0;
8137
8152     virtual bool getBatch(void* bindings[], char const* names[], int32_t nbBindings) noexcept = 0;
8153
8168     virtual void const* readCalibrationCache(std::size_t& length) noexcept = 0;
8169
8178     virtual void writeCalibrationCache(void const* ptr, std::size_t length) noexcept = 0;
8179
8185     virtual CalibrationAlgoType getAlgorithm() noexcept = 0;
8186
8187     ~IInt8Calibrator() noexcept override = default;
8188 };
8189
8190 namespace v_1_0
8191 {
8192 class TRT_DEPRECATED IInt8EntropyCalibrator : public IInt8Calibrator
8193 {
8194 public:
8198     InterfaceInfo getInterfaceInfo() const noexcept override
8199     {
8200         return InterfaceInfo{"IInt8EntropyCalibrator", 1, 0};
8201     }
8202
8206     CalibrationAlgoType getAlgorithm() noexcept override
8207     {
8208         return CalibrationAlgoType::kENTROPY_CALIBRATION;
8209     }
8210
8211     ~IInt8EntropyCalibrator() noexcept override = default;
8212 };
8213 } // namespace v_1_0
8214
8228 using IInt8EntropyCalibrator = v_1_0::IInt8EntropyCalibrator;
8229
8230 namespace v_1_0
8231 {
8232 class TRT_DEPRECATED IInt8EntropyCalibrator2 : public IInt8Calibrator
8233 {
8234 public:
8238     InterfaceInfo getInterfaceInfo() const noexcept override
8239     {
8240         return InterfaceInfo{"IInt8EntropyCalibrator2", 1, 0};
8241     }
8242
8246     CalibrationAlgoType getAlgorithm() noexcept override
8247     {
8248         return CalibrationAlgoType::kENTROPY_CALIBRATION_2;
8249     }
8250
8251     ~IInt8EntropyCalibrator2() noexcept override = default;
8252 };
8253 } // namespace v_1_0
8254
8268 using IInt8EntropyCalibrator2 = v_1_0::IInt8EntropyCalibrator2;
8269
8270 namespace v_1_0
8271 {
8272 class TRT_DEPRECATED IInt8MinMaxCalibrator : public IInt8Calibrator
8273 {
8274 public:
8278     InterfaceInfo getInterfaceInfo() const noexcept override
8279     {
8280         return InterfaceInfo{"IInt8MinMaxCalibrator", 1, 0};
8281     }
8282
8286     CalibrationAlgoType getAlgorithm() noexcept override
8287     {
8288         return CalibrationAlgoType::kMINMAX_CALIBRATION;
8289     }
8290
8291     ~IInt8MinMaxCalibrator() noexcept override = default;
8292 };
8293 } // namespace v_1_0
8294
8307 using IInt8MinMaxCalibrator = v_1_0::IInt8MinMaxCalibrator;
```

```
8308
8309 namespace v_1_0
8310 {
8311 class TRT_DEPRECATED IInt8LegacyCalibrator : public IInt8Calibrator
8312 {
8313 public:
8317     InterfaceInfo getInterfaceInfo() const noexcept override
8318     {
8319         return InterfaceInfo{"IInt8Calibrator", 1, 0};
8320     }
8321
8325     CalibrationAlgoType getAlgorithm() noexcept override
8326     {
8327         return CalibrationAlgoType::kLEGACY_CALIBRATION;
8328     }
8329
8336     virtual double getQuantile() const noexcept = 0;
8337
8344     virtual double getRegressionCutoff() const noexcept = 0;
8345
8358     virtual void const* readHistogramCache(std::size_t& length) noexcept = 0;
8359
8368     virtual void writeHistogramCache(void const* ptr, std::size_t length) noexcept = 0;
8369
8370     ~IInt8LegacyCalibrator() noexcept override = default;
8371 };
8372 } // namespace v_1_0
8373
8387 using IInt8LegacyCalibrator = v_1_0::IInt8LegacyCalibrator;
8388
8401 class TRT_DEPRECATED IAlgorithmIOInfo : public INoCopy
8402 {
8403 public:
8409     DataType getDataType() const noexcept
8410     {
8411         return mImpl->getDataType();
8412     }
8413
8420     Dims getStrides() const noexcept
8421     {
8422         return mImpl->getStrides();
8423     }
8424
8430     int64_t getVectorizedDim() const noexcept
8431     {
8432         return mImpl->getVectorizedDim();
8433     }
8434
8441     int64_t getComponentsPerElement() const noexcept
8442     {
8443         return mImpl->getComponentsPerElement();
8444     }
8445
8446 protected:
8447     virtual ~IAlgorithmIOInfo() noexcept = default;
8448     apiv::VAlgorithmIOInfo* mImpl;
8449 };
8450
8464 class TRT_DEPRECATED IAlgorithmVariant : public INoCopy
8465 {
8466 public:
8470     int64_t getImplementation() const noexcept
8471     {
8472         return mImpl->getImplementation();
8473     }
8474
8478     int64_t getTactic() const noexcept
8479     {
8480         return mImpl->getTactic();
8481     }
8482
8483 protected:
8484     virtual ~IAlgorithmVariant() noexcept = default;
8485     apiv::VAlgorithmVariant* mImpl;
8486 };
8487
8498 class TRT_DEPRECATED IAlgorithmContext : public INoCopy
8499 {
8500 public:
8506     char const* getName() const noexcept
8507     {
```

```
8508         return mImpl->getName();
8509     }
8510
8518     Dims getDimensions(int32_t index, OptProfileSelector select) const noexcept
8519     {
8520         return mImpl->getDimensions(index, select);
8521     }
8522
8526     int32_t getNbInputs() const noexcept
8527     {
8528         return mImpl->getNbInputs();
8529     }
8530
8534     int32_t getNbOutputs() const noexcept
8535     {
8536         return mImpl->getNbOutputs();
8537     }
8538
8539 protected:
8540     virtual ~IAlgorithmContext() noexcept = default;
8541     apiv::VAlgorithmContext* mImpl;
8542 };
8543
8557 class TRT_DEPRECATED IAlgorithm : public INoCopy
8558 {
8559 public:
8563     IAlgorithmVariant const& getAlgorithmVariant() const noexcept
8564     {
8565         return mImpl->getAlgorithmVariant();
8566     }
8567
8571     float getTimingMSec() const noexcept
8572     {
8573         return mImpl->getTimingMSec();
8574     }
8575
8579     std::size_t getWorkspaceSize() const noexcept
8580     {
8581         return mImpl->getWorkspaceSize();
8582     }
8583
8593     IAlgorithmIOInfo const* getAlgorithmIOInfoByIndex(int32_t index) const noexcept
8594     {
8595         return mImpl->getAlgorithmIOInfoByIndex(index);
8596     }
8597
8598 protected:
8599     virtual ~IAlgorithm() noexcept = default;
8600     apiv::VAlgorithm* mImpl;
8601 }; // IAlgorithm
8602
8603 namespace v_1_0
8604 {
8605 class TRT_DEPRECATED IAlgorithmSelector : public IVersionedInterface
8606 {
8607 public:
8611     InterfaceInfo getInterfaceInfo() const noexcept override
8612     {
8613         return InterfaceInfo{"IAlgorithmSelector", 1, 0};
8614     }
8629     virtual int32_t selectAlgorithms(IAlgorithmContext const& context, IAlgorithm const* const* choices,
8630         int32_t nbChoices, int32_t* selection) noexcept = 0;
8631
8642     virtual void reportAlgorithms(IAlgorithmContext const* const* algoContexts, IAlgorithm const* const*
      algoChoices,
8643         int32_t nbAlgorithms) noexcept = 0;
8644
8645     virtual ~IAlgorithmSelector() noexcept = default;
8646 };
8647 } // namespace v_1_0
8648
8661 using IAlgorithmSelector = v_1_0::IAlgorithmSelector;
8662
8669 using QuantizationFlags = uint32_t;
8670
8680 enum class QuantizationFlag : int32_t
8681 {
8685     kCALIBRATE_BEFORE_FUSION TRT_DEPRECATED_ENUM = 0
8686 };
8687
8693 template <>
```

```
8694 constexpr inline int32_t EnumMax<QuantizationFlag>() noexcept
8695 {
8696     return 1;
8697 }
8698
8716 enum class RuntimePlatform : int32_t
8717 {
8720     kSAME_AS_BUILD = 0,
8721
8724     kWINDOWS_AMD64 = 1,
8725 };
8726
8727 namespace impl
8728 {
8734 template <>
8735 struct EnumMaxImpl<RuntimePlatform>
8736 {
8737     static constexpr int32_t kVALUE = 2;
8738 };
8739 } // namespace impl
8740
8747 using BuilderFlags = uint32_t;
8748
8756 enum class BuilderFlag : int32_t
8757 {
8759     kFP16 = 0,
8760
8762     kINT8 = 1,
8763
8765     kDEBUG = 2,
8766
8768     kGPU_FALLBACK = 3,
8769
8771     kREFIT = 4,
8772
8774     kDISABLE_TIMING_CACHE = 5,
8775
8779     kTF32 = 6,
8780
8782     kSPARSE_WEIGHTS = 7,
8783
8790     kSAFETY_SCOPE = 8,
8791
8793     kOBEY_PRECISION_CONSTRAINTS = 9,
8794
8797     kPREFER_PRECISION_CONSTRAINTS = 10,
8798
8803     kDIRECT_IO TRT_DEPRECATED_ENUM = 11,
8804
8807     kREJECT_EMPTY_ALGORITHMS TRT_DEPRECATED_ENUM = 12,
8808
8814     kVERSION_COMPATIBLE = 13,
8815
8820     kEXCLUDE_LEAN_RUNTIME = 14,
8821
8827     kFP8 = 15,
8828
8831     kERROR_ON_TIMING_CACHE_MISS = 16,
8832
8835     kBF16 = 17,
8836
8841     kDISABLE_COMPILATION_CACHE = 18,
8842
8844     kSTRIP_PLAN = 19,
8845
8847     kWEIGHTLESS TRT_DEPRECATED_ENUM = kSTRIP_PLAN,
8848
8855     kREFIT_IDENTICAL = 20,
8856
8882     kWEIGHT_STREAMING = 21,
8883
8885     kINT4 = 22,
8886
8891     kREFIT_INDIVIDUAL = 23,
8892
8901     kSTRICT_NANS = 24,
8902
8904     kMONITOR_MEMORY = 25,
8905
8907     kFP4 = 26,
8908
```

```
8910     kEDITABLE_TIMING_CACHE = 27,
8911 };
8912
8918 template <>
8919 constexpr inline int32_t EnumMax<BuilderFlag>() noexcept
8920 {
8921     return 28;
8922 }
8923
8924 namespace v_1_0
8925 {
8938 struct TimingCacheKey
8939 {
8940     uint8_t data[16];
8941 };
8942
8950 struct TimingCacheValue
8951 {
8953     uint64_t tacticHash;
8955     float timingMSec;
8957     static constexpr uint64_t kINVALID_TACTIC_HASH = UINT64_MAX;
8958 };
8959 } // namespace v_1_0
8960
8974 class ITimingCache : public INoCopy
8975 {
8976 public:
8977     virtual ~ITimingCache() noexcept = default;
8978
8988     nvinfer1::IHostMemory* serialize() const noexcept
8989     {
8990         return mImpl->serialize();
8991     }
8992
9012     bool combine(ITimingCache const& inputCache, bool ignoreMismatch) noexcept
9013     {
9014         return mImpl->combine(inputCache, ignoreMismatch);
9015     }
9016
9022     bool reset() noexcept
9023     {
9024         return mImpl->reset();
9025     }
9026
9041     int64_t queryKeys(TimingCacheKey* keyBuffer, int64_t capacity) const noexcept
9042     {
9043         return mImpl->queryKeys(keyBuffer, capacity);
9044     }
9045
9058     TimingCacheValue query(TimingCacheKey const& key) const noexcept
9059     {
9060         return mImpl->query(key);
9061     }
9062
9080     bool update(TimingCacheKey const& key, TimingCacheValue const& value) noexcept
9081     {
9082         return mImpl->update(key, value);
9083     }
9084
9085 protected:
9086     apiv::VTimingCache* mImpl;
9087 };
9088
9096 enum class MemoryPoolType : int32_t
9097 {
9104     kWORKSPACE = 0,
9105
9112     kDLA_MANAGED_SRAM = 1,
9113
9119     kDLA_LOCAL_DRAM = 2,
9120
9126     kDLA_GLOBAL_DRAM = 3,
9127
9135     kTACTIC_DRAM = 4,
9136
9149     kTACTIC_SHARED_MEMORY = 5,
9150 };
9151
9157 template <>
9158 constexpr inline int32_t EnumMax<MemoryPoolType>() noexcept
9159 {
```

```
9160     return 6;
9161 }
9162
9171 enum class PreviewFeature : int32_t
9172 {
9178     kPROFILE_SHARING_0806 TRT_DEPRECATED_ENUM = 0,
9179
9183     kALIASED_PLUGIN_IO_10_03 = 1,
9184
9190     kRUNTIME_ACTIVATION_RESIZE_10_10 = 2
9191 };
9192
9193 namespace impl
9194 {
9200 template <>
9201 struct EnumMaxImpl<PreviewFeature>
9202 {
9203     static constexpr int32_t kVALUE = 3;
9204 };
9205 } // namespace impl
9206
9215 enum class HardwareCompatibilityLevel : int32_t
9216 {
9219     kNONE = 0,
9220
9232     kAMPERE_PLUS = 1,
9233
9242     kSAME_COMPUTE_CAPABILITY = 2,
9243 };
9244
9245 namespace impl
9246 {
9252 template <>
9253 struct EnumMaxImpl<HardwareCompatibilityLevel>
9254 {
9255     static constexpr int32_t kVALUE = 3;
9256 };
9257 } // namespace impl
9258
9267 enum class TilingOptimizationLevel : int32_t
9268 {
9270     kNONE = 0,
9271
9273     kFAST = 1,
9274
9277     kMODERATE = 2,
9278
9280     kFULL = 3
9281
9282 };
9283
9284 namespace impl
9285 {
9291 template <>
9292 struct EnumMaxImpl<TilingOptimizationLevel>
9293 {
9294     static constexpr int32_t kVALUE = 4;
9295 };
9296 } // namespace impl
9297
9298 namespace v_1_0
9299 {
9300 class IProgressMonitor : public IVersionedInterface
9301 {
9302 public:
9303     IProgressMonitor() = default;
9304     virtual ~IProgressMonitor() noexcept = default;
9305
9309     InterfaceInfo getInterfaceInfo() const noexcept override
9310     {
9311         return InterfaceInfo{"IProgressMonitor", 1, 0};
9312     }
9313
9333     virtual void phaseStart(char const* phaseName, char const* parentPhase, int32_t nbSteps) noexcept = 0;
9334
9347     virtual bool stepComplete(char const* phaseName, int32_t step) noexcept = 0;
9348
9360     virtual void phaseFinish(char const* phaseName) noexcept = 0;
9361
9362 }; // class IProgressMonitor
9363 } // namespace v_1_0
```

```
9364
9384 using IProgressMonitor = v_1_0::IProgressMonitor;
9385
9393 class IBuilderConfig : public INoCopy
9394 {
9395 public:
9396     virtual ~IBuilderConfig() noexcept = default;
9397
9406     virtual void setAvgTimingIterations(int32_t avgTiming) noexcept
9407     {
9408         mImpl->setAvgTimingIterations(avgTiming);
9409     }
9410
9418     int32_t getAvgTimingIterations() const noexcept
9419     {
9420         return mImpl->getAvgTimingIterations();
9421     }
9422
9431     void setEngineCapability(EngineCapability capability) noexcept
9432     {
9433         mImpl->setEngineCapability(capability);
9434     }
9435
9443     EngineCapability getEngineCapability() const noexcept
9444     {
9445         return mImpl->getEngineCapability();
9446     }
9447
9455     TRT_DEPRECATED void setInt8Calibrator(IInt8Calibrator* calibrator) noexcept
9456     {
9457         mImpl->setInt8Calibrator(calibrator);
9458     }
9459
9465     TRT_DEPRECATED IInt8Calibrator* getInt8Calibrator() const noexcept
9466     {
9467         return mImpl->getInt8Calibrator();
9468     }
9469
9482     void setFlags(BuilderFlags builderFlags) noexcept
9483     {
9484         mImpl->setFlags(builderFlags);
9485     }
9486
9494     BuilderFlags getFlags() const noexcept
9495     {
9496         return mImpl->getFlags();
9497     }
9498
9506     void clearFlag(BuilderFlag builderFlag) noexcept
9507     {
9508         mImpl->clearFlag(builderFlag);
9509     }
9510
9518     void setFlag(BuilderFlag builderFlag) noexcept
9519     {
9520         mImpl->setFlag(builderFlag);
9521     }
9522
9530     bool getFlag(BuilderFlag builderFlag) const noexcept
9531     {
9532         return mImpl->getFlag(builderFlag);
9533     }
9534
9547     void setDeviceType(ILayer const* layer, DeviceType deviceType) noexcept
9548     {
9549         mImpl->setDeviceType(layer, deviceType);
9550     }
9551
9557     DeviceType getDeviceType(ILayer const* layer) const noexcept
9558     {
9559         return mImpl->getDeviceType(layer);
9560     }
9561
9569     bool isDeviceTypeSet(ILayer const* layer) const noexcept
9570     {
9571         return mImpl->isDeviceTypeSet(layer);
9572     }
9573
9579     void resetDeviceType(ILayer const* layer) noexcept
9580     {
9581         mImpl->resetDeviceType(layer);
```

```
9582     }
9583
9589     bool canRunOnDLA(ILayer const* layer) const noexcept
9590     {
9591         return mImpl->canRunOnDLA(layer);
9592     }
9593
9605     void setDLACore(int32_t dlaCore) noexcept
9606     {
9607         mImpl->setDLACore(dlaCore);
9608     }
9609
9615     int32_t getDLACore() const noexcept
9616     {
9617         return mImpl->getDLACore();
9618     }
9619
9626     void setDefaultDeviceType(DeviceType deviceType) noexcept
9627     {
9628         mImpl->setDefaultDeviceType(deviceType);
9629     }
9630
9636     DeviceType getDefaultDeviceType() const noexcept
9637     {
9638         return mImpl->getDefaultDeviceType();
9639     }
9640
9646     void reset() noexcept
9647     {
9648         mImpl->reset();
9649     }
9650
9658     void setProfileStream(const cudaStream_t stream) noexcept
9659     {
9660         return mImpl->setProfileStream(stream);
9661     }
9662
9670     cudaStream_t getProfileStream() const noexcept
9671     {
9672         return mImpl->getProfileStream();
9673     }
9674
9687     int32_t addOptimizationProfile(IOptimizationProfile const* profile) noexcept
9688     {
9689         return mImpl->addOptimizationProfile(profile);
9690     }
9691
9700     int32_t getNbOptimizationProfiles() const noexcept
9701     {
9702         return mImpl->getNbOptimizationProfiles();
9703     }
9704
9712     void setProfilingVerbosity(ProfilingVerbosity verbosity) noexcept
9713     {
9714         mImpl->setProfilingVerbosity(verbosity);
9715     }
9716
9725     ProfilingVerbosity getProfilingVerbosity() const noexcept
9726     {
9727         return mImpl->getProfilingVerbosity();
9728     }
9729
9737     TRT_DEPRECATED void setAlgorithmSelector(IAlgorithmSelector* selector) noexcept
9738     {
9739         mImpl->setAlgorithmSelector(selector);
9740     }
9741
9747     TRT_DEPRECATED IAlgorithmSelector* getAlgorithmSelector() const noexcept
9748     {
9749         return mImpl->getAlgorithmSelector();
9750     }
9751
9765     TRT_DEPRECATED bool setCalibrationProfile(IOptimizationProfile const* profile) noexcept
9766     {
9767         return mImpl->setCalibrationProfile(profile);
9768     }
9769
9777     TRT_DEPRECATED IOptimizationProfile const* getCalibrationProfile() noexcept
9778     {
9779         return mImpl->getCalibrationProfile();
9780     }
```

```
9781
9796     TRT_DEPRECATED void setQuantizationFlags(QuantizationFlags flags) noexcept
9797     {
9798         mImpl->setQuantizationFlags(flags);
9799     }
9800
9810     TRT_DEPRECATED QuantizationFlags getQuantizationFlags() const noexcept
9811     {
9812         return mImpl->getQuantizationFlags();
9813     }
9814
9824     TRT_DEPRECATED void clearQuantizationFlag(QuantizationFlag flag) noexcept
9825     {
9826         mImpl->clearQuantizationFlag(flag);
9827     }
9828
9838     TRT_DEPRECATED void setQuantizationFlag(QuantizationFlag flag) noexcept
9839     {
9840         mImpl->setQuantizationFlag(flag);
9841     }
9842
9852     TRT_DEPRECATED bool getQuantizationFlag(QuantizationFlag flag) const noexcept
9853     {
9854         return mImpl->getQuantizationFlag(flag);
9855     }
9856
9874     bool setTacticSources(TacticSources tacticSources) noexcept
9875     {
9876         return mImpl->setTacticSources(tacticSources);
9877     }
9878
9889     TacticSources getTacticSources() const noexcept
9890     {
9891         return mImpl->getTacticSources();
9892     }
9893
9909     nvinfer1::ITimingCache* createTimingCache(void const* blob, std::size_t size) const noexcept
9910     {
9911         return mImpl->createTimingCache(blob, size);
9912     }
9913
9932     bool setTimingCache(ITimingCache const& cache, bool ignoreMismatch) noexcept
9933     {
9934         return mImpl->setTimingCache(cache, ignoreMismatch);
9935     }
9936
9942     nvinfer1::ITimingCache const* getTimingCache() const noexcept
9943     {
9944         return mImpl->getTimingCache();
9945     }
9946
9974     void setMemoryPoolLimit(MemoryPoolType pool, std::size_t poolSize) noexcept
9975     {
9976         mImpl->setMemoryPoolLimit(pool, poolSize);
9977     }
9978
9993     std::size_t getMemoryPoolLimit(MemoryPoolType pool) const noexcept
9994     {
9995         return mImpl->getMemoryPoolLimit(pool);
9996     }
9997
10011     void setPreviewFeature(PreviewFeature feature, bool enable) noexcept
10012     {
10013         mImpl->setPreviewFeature(feature, enable);
10014     }
10015
10025     bool getPreviewFeature(PreviewFeature feature) const noexcept
10026     {
10027         return mImpl->getPreviewFeature(feature);
10028     }
10029
10058     void setBuilderOptimizationLevel(int32_t level) noexcept
10059     {
10060         mImpl->setBuilderOptimizationLevel(level);
10061     }
10062
10070     int32_t getBuilderOptimizationLevel() noexcept
10071     {
10072         return mImpl->getBuilderOptimizationLevel();
10073     }
10074
```

```
10087     void setHardwareCompatibilityLevel(HardwareCompatibilityLevel hardwareCompatibilityLevel) noexcept
10088     {
10089         mImpl->setHardwareCompatibilityLevel(hardwareCompatibilityLevel);
10090     }
10091
10100     HardwareCompatibilityLevel getHardwareCompatibilityLevel() const noexcept
10101     {
10102         return mImpl->getHardwareCompatibilityLevel();
10103     }
10104
10113     void setPluginsToSerialize(char const* const* paths, int32_t nbPaths) noexcept
10114     {
10115         mImpl->setPluginsToSerialize(paths, nbPaths);
10116     }
10117
10126     char const* getPluginToSerialize(int32_t index) const noexcept
10127     {
10128         return mImpl->getPluginToSerialize(index);
10129     }
10130
10136     int32_t getNbPluginsToSerialize() const noexcept
10137     {
10138         return mImpl->getNbPluginsToSerialize();
10139     }
10140
10165     void setMaxAuxStreams(int32_t nbStreams) noexcept
10166     {
10167         mImpl->setMaxAuxStreams(nbStreams);
10168     }
10169
10175     int32_t getMaxAuxStreams() const noexcept
10176     {
10177         return mImpl->getMaxAuxStreams();
10178     }
10179
10191     void setProgressMonitor(IProgressMonitor* monitor) noexcept
10192     {
10193         return mImpl->setProgressMonitor(monitor);
10194     }
10195
10201     IProgressMonitor* getProgressMonitor() const noexcept
10202     {
10203         return mImpl->getProgressMonitor();
10204     }
10205
10217     void setRuntimePlatform(RuntimePlatform runtimePlatform) noexcept
10218     {
10219         mImpl->setRuntimePlatform(runtimePlatform);
10220     }
10221
10229     RuntimePlatform getRuntimePlatform() const noexcept
10230     {
10231         return mImpl->getRuntimePlatform();
10232     }
10233
10241     void setMaxNbTactics(int32_t maxNbTactics) noexcept
10242     {
10243         mImpl->setMaxNbTactics(maxNbTactics);
10244     }
10245
10253     int32_t getMaxNbTactics() const noexcept
10254     {
10255         return mImpl->getMaxNbTactics();
10256     }
10257
10269     bool setTilingOptimizationLevel(TilingOptimizationLevel level) noexcept
10270     {
10271         return mImpl->setTilingOptimizationLevel(level);
10272     }
10273
10281     TilingOptimizationLevel getTilingOptimizationLevel() const noexcept
10282     {
10283         return mImpl->getTilingOptimizationLevel();
10284     }
10285
10297     bool setL2LimitForTiling(int64_t size) noexcept
10298     {
10299         return mImpl->setL2LimitForTiling(size);
10300     }
10301
10309     int64_t getL2LimitForTiling() const noexcept
```

```
10310     {
10311         return mImpl->getL2LimitForTiling();
10312     }
10313
10314 protected:
10315     apiv::VBuilderConfig* mImpl;
10316 };
10317
10325 using NetworkDefinitionCreationFlags = uint32_t;
10326
10335 enum class NetworkDefinitionCreationFlag : int32_t
10336 {
10340     kEXPLICIT_BATCH TRT_DEPRECATED_ENUM = 0,
10341
10346     kSTRONGLY_TYPED = 1,
10347 };
10348
10354 template <>
10355 constexpr inline int32_t EnumMax<NetworkDefinitionCreationFlag>() noexcept
10356 {
10357     return 2;
10358 }
10359
10367 class IBuilder : public INoCopy
10368 {
10369 public:
10370     virtual ~IBuilder() noexcept = default;
10371
10377     TRT_DEPRECATED bool platformHasFastFp16() const noexcept
10378     {
10379         return mImpl->platformHasFastFp16();
10380     }
10381
10387     TRT_DEPRECATED bool platformHasFastInt8() const noexcept
10388     {
10389         return mImpl->platformHasFastInt8();
10390     }
10391
10399     int32_t getMaxDLABatchSize() const noexcept
10400     {
10401         return mImpl->getMaxDLABatchSize();
10402     }
10403
10407     int32_t getNbDLACores() const noexcept
10408     {
10409         return mImpl->getNbDLACores();
10410     }
10411
10425     void setGpuAllocator(IGpuAllocator* allocator) noexcept
10426     {
10427         mImpl->setGpuAllocator(allocator);
10428     }
10429
10439     nvinfer1::IBuilderConfig* createBuilderConfig() noexcept
10440     {
10441         return mImpl->createBuilderConfig();
10442     }
10443
10465     nvinfer1::INetworkDefinition* createNetworkV2(NetworkDefinitionCreationFlags flags) noexcept
10466     {
10467         return mImpl->createNetworkV2(flags);
10468     }
10469
10480     nvinfer1::IOptimizationProfile* createOptimizationProfile() noexcept
10481     {
10482         return mImpl->createOptimizationProfile();
10483     }
10484
10499     void setErrorRecorder(IErrorRecorder* recorder) noexcept
10500     {
10501         mImpl->setErrorRecorder(recorder);
10502     }
10503
10514     IErrorRecorder* getErrorRecorder() const noexcept
10515     {
10516         return mImpl->getErrorRecorder();
10517     }
10518
10522     void reset() noexcept
10523     {
10524         mImpl->reset();
```

```
10525     }
10526
10532     TRT_DEPRECATED bool platformHasTf32() const noexcept
10533     {
10534         return mImpl->platformHasTf32();
10535     }
10536
10551     nvinfer1::IHostMemory* buildSerializedNetwork(INetworkDefinition& network, IBuilderConfig& config)
      noexcept
10552     {
10553         return mImpl->buildSerializedNetwork(network, config);
10554     }
10555
10571     nvinfer1::ICudaEngine* buildEngineWithConfig(INetworkDefinition& network, IBuilderConfig& config)
      noexcept
10572     {
10573         return mImpl->buildEngineWithConfig(network, config);
10574     }
10575
10593     bool isNetworkSupported(INetworkDefinition const& network, IBuilderConfig const& config) const
      noexcept
10594     {
10595         return mImpl->isNetworkSupported(network, config);
10596     }
10597
10603     ILogger* getLogger() const noexcept
10604     {
10605         return mImpl->getLogger();
10606     }
10607
10619     bool setMaxThreads(int32_t maxThreads) noexcept
10620     {
10621         return mImpl->setMaxThreads(maxThreads);
10622     }
10623
10633     int32_t getMaxThreads() const noexcept
10634     {
10635         return mImpl->getMaxThreads();
10636     }
10637
10643     IPluginRegistry& getPluginRegistry() noexcept
10644     {
10645         return mImpl->getPluginRegistry();
10646     }
10647
10648 protected:
10649     apiv::VBuilder* mImpl;
10650 };
10651
10652 } // namespace nvinfer1
10653
10658 extern "C" TENSORRTAPI void* createInferBuilder_INTERNAL(void* logger, int32_t version) noexcept;
10659
10660 namespace nvinfer1
10661 {
10662 namespace
10663 {
10664
10672 inline IBuilder* createInferBuilder(ILogger& logger) noexcept
10673 {
10674     return static_cast<IBuilder*>(createInferBuilder_INTERNAL(&logger, NV_TENSORRT_VERSION));
10675 }
10676
10677 } // namespace
10678
10691 extern "C" TENSORRTAPI nvinfer1::IPluginRegistry* getBuilderPluginRegistry(
10692     nvinfer1::EngineCapability capability) noexcept;
10693
10694 namespace safe
10695 {
10697 class IPluginRegistry;
10698 } // namespace safe
10699
10707 extern "C" TENSORRTAPI nvinfer1::safe::IPluginRegistry* getBuilderSafePluginRegistry(
10708     nvinfer1::EngineCapability capability) noexcept;
10709
10710 } // namespace nvinfer1
10711
10712 #endif // NV_INFER_H
```

10.3 NvInferImpl.h File Reference

```
#include "NvInferLegacyDims.h"
#include "NvInferRuntimeCommon.h"
```

10.4 NvInferImpl.h

Go to the documentation of this file.

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2025 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: Apache-2.0
4  *
5  * Licensed under the Apache License, Version 2.0 (the "License");
6  * you may not use this file except in compliance with the License.
7  * You may obtain a copy of the License at
8  *
9  * http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in writing, software
12  * distributed under the License is distributed on an "AS IS" BASIS,
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14  * See the License for the specific language governing permissions and
15  * limitations under the License.
16  */
17
18 #ifndef NV_INFER_IMPL_H
19 #define NV_INFER_IMPL_H
20
21 #include "NvInferLegacyDims.h"
22 #include "NvInferRuntimeCommon.h"
23
24 // @cond SuppressDoxyWarnings
25
26 namespace nvinfer1
27 {
28
29 class ILogger;
30
31 namespace v_1_0
32 {
33 class IProgressMonitor;
34 }
35 using IProgressMonitor = v_1_0::IProgressMonitor;
36
37 namespace v_1_0
38 {
39 class IAlgorithmSelector;
40 }
41 using IAlgorithmSelector = v_1_0::IAlgorithmSelector;
42
43 namespace v_1_0
44 {
45 class IProfiler;
46 }
47 using IProfiler = v_1_0::IProfiler;
48
49 namespace v_1_0
50 {
51 class IOutputAllocator;
52 }
53 using IOutputAllocator = v_1_0::IOutputAllocator;
54
55 namespace v_1_0
56 {
57 class IDebugListener;
58 }
59 using IDebugListener = v_1_0::IDebugListener;
60
61 class IActivationLayer;
62 class IAlgorithm;
63 class IAlgorithmContext;
```

```
64 class IAlgorithmIOInfo;
65 class IAlgorithmVariant;
66 class IAssertionLayer;
67 class IBuilder;
68 class IBuilderConfig;
69 class IConcatenationLayer;
70 class IConditionLayer;
71 class IConstantLayer;
72 class IConvolutionLayer;
73 class ICudaEngine;
74 class ICumulativeLayer;
75 class IDeconvolutionLayer;
76 class IDequantizeLayer;
77 class IDimensionExpr;
78 class IDynamicQuantizeLayer;
79 class IEinsumLayer;
80 class IElementWiseLayer;
81 class IEngineInspector;
82 class IExecutionContext;
83 class IFillLayer;
84 class IGatherLayer;
85 class IGridSampleLayer;
86 class IHostMemory;
87 class IIdentityLayer;
88 class ICastLayer;
89 class IIfConditional;
90 class IIfConditionalInputLayer;
91 class IIfConditionalOutputLayer;
92 class IInt8Calibrator;
93 class IIteratorLayer;
94 class ILayer;
95 class ILoop;
96 class ILoopOutputLayer;
97 class ILRNLayer;
98 class IMatrixMultiplyLayer;
99 class INetworkDefinition;
100 class INormalizationLayer;
101 class INMSLayer;
102 class INonZeroLayer;
103 class IOneHotLayer;
104 class IOptimizationProfile;
105 class IPaddingLayer;
106 class IParametricReLULayer;
107 class IPlugin;
108 class IPluginExt;
109 class IPluginFactory;
110 class IPluginLayer;
111 class IPluginRegistry;
112 class IPluginV2Layer;
113
114 namespace v_1_0
115 {
116 class IPluginV3;
117 } // namespace v_1_0
118 using IPluginV3 = v_1_0::IPluginV3;
119
120 namespace v_1_0
121 {
122 class IStreamReader;
123 } // namespace v_1_0
124 using IStreamReader = v_1_0::IStreamReader;
125 namespace v_1_0
126 {
127 class IStreamReaderV2;
128 } // namespace v_1_0
129 using IStreamReaderV2 = v_1_0::IStreamReaderV2;
130
131 class IPluginV3Layer;
132 class IPoolingLayer;
133 class IQuantizeLayer;
134 class IRaggedSoftMaxLayer;
135 class IRecurrenceLayer;
136 class IReduceLayer;
137 class IRefitter;
138 class IResizeLayer;
139 class IReverseSequenceLayer;
140 class IRuntime;
141 class IScaleLayer;
142 class IScatterLayer;
143 class ISelectLayer;
144 class ISerializationConfig;
```

```
145 class IShapeLayer;
146 class IShuffleLayer;
147 class ISliceLayer;
148 class ISoftMaxLayer;
149 class ISqueezeLayer;
150 class ITensor;
151
152 namespace v_1_0
153 {
154 struct TimingCacheKey;
155 struct TimingCacheValue;
156 } // namespace v_1_0
157 using TimingCacheKey = v_1_0::TimingCacheKey;
158 using TimingCacheValue = v_1_0::TimingCacheValue;
159
160 class ITimingCache;
161 class ITopKLayer;
162 class ITripLimitLayer;
163 class IUnaryLayer;
164 class IUnsqueezeLayer;
165 struct Permutation;
166 class Weights;
167
168 enum class ActivationType : int32_t;
169 enum class BoundingBoxFormat : int32_t;
170 enum class BuilderFlag : int32_t;
171 enum class CalibrationAlgoType : int32_t;
172 enum class CumulativeOperation : int32_t;
173 enum class DeviceType : int32_t;
174 enum class DimensionOperation : int32_t;
175 enum class ElementWiseOperation : int32_t;
176 enum class EngineCapability : int32_t;
177 enum class FillOperation : int32_t;
178 enum class GatherMode : int32_t;
179 enum class LayerInformationFormat : int32_t;
180 enum class LayerType : int32_t;
181 enum class LoopOutput : int32_t;
182 enum class MatrixOperation : int32_t;
183 enum class MemoryPoolType : int32_t;
184 enum class NetworkDefinitionCreationFlag : int32_t;
185 enum class OptProfileSelector : int32_t;
186 enum class PaddingMode : int32_t;
187 enum class PoolingType : int32_t;
188 enum class ProfilingVerbosity : int32_t;
189 enum class QuantizationFlag : int32_t;
190 enum class ReduceOperation : int32_t;
191 enum class ResizeCoordinateTransformation : int32_t;
192 enum class InterpolationMode : int32_t;
193 enum class ResizeRoundMode : int32_t;
194 enum class ResizeSelector : int32_t;
195 enum class ScaleMode : int32_t;
196 enum class ScatterMode : int32_t;
197 enum class SampleMode : int32_t;
198 enum class SerializationFlag : int32_t;
199 enum class TensorIOMode : int32_t;
200 enum class TensorLocation : int32_t;
201 enum class TopKOperation : int32_t;
202 enum class TripLimit : int32_t;
203 enum class UnaryOperation : int32_t;
204 enum class WeightsRole : int32_t;
205 enum class PreviewFeature : int32_t;
206 enum class HardwareCompatibilityLevel : int32_t;
207 enum class ExecutionContextAllocationStrategy : int32_t;
208 enum class RuntimePlatform : int32_t;
209 enum class TilingOptimizationLevel : int32_t;
210
211 using TacticSources = uint32_t;
212 using TensorFormats = uint32_t;
213 using BuilderFlags = uint32_t;
214 using NetworkDefinitionCreationFlags = uint32_t;
215 using QuantizationFlags = uint32_t;
216 using TempfileControlFlags = uint32_t;
217 using SerializationFlags = uint32_t;
218
226
227 namespace apiv
228 {
229
230 class VRoot
231 {
232 public:
```

```
233     virtual ~VRoot() noexcept = default;
234 };
235
236 class VHostMemory : public VRoot
237 {
238 public:
239     virtual void* data() const noexcept = 0;
240     virtual std::size_t size() const noexcept = 0;
241     virtual DataType type() const noexcept = 0;
242 };
243
244 class VDimensionExpr : public VRoot
245 {
246 public:
247     virtual bool isConstant() const = 0;
248     virtual int64_t getConstantValue() const = 0;
249     virtual bool isSizeTensor() const = 0;
250 };
251
252 class VExprBuilder : public VRoot
253 {
254 public:
255     virtual IDimensionExpr const* constant(int64_t value) = 0;
256     virtual IDimensionExpr const* operation(
257         DimensionOperation op, IDimensionExpr const& first, IDimensionExpr const& second)
258         = 0;
259     virtual IDimensionExpr const* declareSizeTensor(
260         int32_t outputIndex, IDimensionExpr const& opt, IDimensionExpr const& upper)
261         = 0;
262 };
263
264 class VRuntime : public VRoot
265 {
266 public:
267     virtual IRuntime* getPImpl() noexcept = 0;
268     virtual nvinfer1::ICudaEngine* deserializeCudaEngine(void const* blob, std::size_t size) noexcept = 0;
269     virtual nvinfer1::ICudaEngine* deserializeCudaEngine(IStreamReader& streamReader) noexcept = 0;
270     virtual void setDLACore(int32_t dlaCore) noexcept = 0;
271     virtual int32_t getDLACore() const noexcept = 0;
272     virtual int32_t getNbDLACores() const noexcept = 0;
273     virtual void setGpuAllocator(IGpuAllocator* allocator) noexcept = 0;
274     virtual void setErrorRecorder(IErrorRecorder* recorder) noexcept = 0;
275     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
276     virtual ILogger* getLogger() const noexcept = 0;
277     virtual bool setMaxThreads(int32_t maxThreads) noexcept = 0;
278     virtual int32_t getMaxThreads() const noexcept = 0;
279     virtual void setTemporaryDirectory(char const*) noexcept = 0;
280     virtual char const* getTemporaryDirectory() const noexcept = 0;
281     virtual void setTempfileControlFlags(TempfileControlFlags) noexcept = 0;
282     virtual TempfileControlFlags getTempfileControlFlags() const noexcept = 0;
283     virtual IPluginRegistry& getPluginRegistry() noexcept = 0;
284     virtual void setPluginRegistryParent(IPluginRegistry* parent) noexcept = 0;
285     virtual IRuntime* loadRuntime(char const* path) noexcept = 0;
286     virtual void setEngineHostCodeAllowed(bool allowed) noexcept = 0;
287     virtual bool getEngineHostCodeAllowed() const noexcept = 0;
288     // Added in TensorRT version 10.7
289     virtual nvinfer1::ICudaEngine* deserializeCudaEngineV2(IStreamReaderV2& streamReader) noexcept = 0;
290 };
291
292 class VRefitter : public VRoot
293 {
294 public:
295     virtual IRefitter* getPImpl() noexcept = 0;
296     virtual bool setWeights(char const* layerName, WeightsRole role, const Weights weights) noexcept = 0;
297     virtual bool refitCudaEngine() noexcept = 0;
298     virtual int32_t getMissing(int32_t size, char const** layerNames, WeightsRole* roles) noexcept = 0;
299     virtual int32_t getAll(int32_t size, char const** layerNames, WeightsRole* roles) noexcept = 0;
300     virtual bool setDynamicRange(char const* tensorName, float min, float max) noexcept = 0;
301     virtual float getDynamicRangeMin(char const* tensorName) const noexcept = 0;
302     virtual float getDynamicRangeMax(char const* tensorName) const noexcept = 0;
303     virtual int32_t getTensorsWithDynamicRange(int32_t size, char const** tensorNames) const noexcept = 0;
304     virtual void setErrorRecorder(IErrorRecorder* recorder) noexcept = 0;
305     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
306     virtual bool setNamedWeights(char const* name, Weights weights) noexcept = 0;
307     virtual int32_t getMissingWeights(int32_t size, char const** weightsNames) noexcept = 0;
308     virtual int32_t getAllWeights(int32_t size, char const** weightsNames) noexcept = 0;
309     virtual ILogger* getLogger() const noexcept = 0;
310     virtual bool setMaxThreads(int32_t maxThreads) noexcept = 0;
311     virtual int32_t getMaxThreads() const noexcept = 0;
312     virtual bool setNamedWeightsWithLocation(char const* name, Weights weights, TensorLocation location)
    noexcept = 0;
```

```
313     virtual Weights getNamedWeights(char const* weightsName) const noexcept = 0;
314     virtual TensorLocation getWeightsLocation(char const* weightsName) const noexcept = 0;
315     virtual bool unsetNamedWeights(char const* weightsName) noexcept = 0;
316     virtual void setWeightsValidation(bool weightsValidation) noexcept = 0;
317     virtual bool getWeightsValidation() const noexcept = 0;
318     virtual bool refitCudaEngineAsync(cudaStream_t stream) noexcept = 0;
319     virtual Weights getWeightsPrototype(char const* weightsName) const noexcept = 0;
320 };
321
322 class VOptimizationProfile : public VRoot
323 {
324 public:
325     virtual bool setDimensions(char const* inputName, OptProfileSelector select, Dims const& dims) noexcept
    = 0;
326     virtual Dims getDimensions(char const* inputName, OptProfileSelector select) const noexcept = 0;
327     virtual bool setShapeValues(
328         char const* inputName, OptProfileSelector select, int32_t const* values, int32_t nbValues) noexcept =
    0;
329     virtual int32_t getNbShapeValues(char const* inputName) const noexcept = 0;
330     virtual int32_t const* getShapeValues(char const* inputName, OptProfileSelector select) const noexcept =
    0;
331     virtual bool setExtraMemoryTarget(float target) noexcept = 0;
332     virtual float getExtraMemoryTarget() const noexcept = 0;
333     virtual bool isValid() const noexcept = 0;
334 };
335
336 class VCudaEngine : public VRoot
337 {
338 public:
339     virtual ICudaEngine* getPImpl() noexcept = 0;
340     virtual int32_t getNbLayers() const noexcept = 0;
341     virtual IHostMemory* serialize() const noexcept = 0;
342     virtual IExecutionContext* createExecutionContext(ExecutionContextAllocationStrategy strategy) noexcept
    = 0;
343     virtual IExecutionContext* createExecutionContextWithoutDeviceMemory() noexcept = 0;
344     virtual size_t getDeviceMemorySize() const noexcept = 0;
345     virtual bool isRefittable() const noexcept = 0;
346     virtual char const* getName() const noexcept = 0;
347     virtual int32_t getNbOptimizationProfiles() const noexcept = 0;
348     virtual int32_t const* getProfileTensorValues(
349         char const* tensorName, int32_t profileIndex, OptProfileSelector select) const noexcept = 0;
350     virtual EngineCapability getEngineCapability() const noexcept = 0;
351     virtual void setErrorRecorder(IErrorRecorder* recorder) noexcept = 0;
352     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
353     virtual bool hasImplicitBatchDimension() const noexcept = 0;
354     virtual TacticSources getTacticSources() const noexcept = 0;
355     virtual ProfilingVerbosity getProfilingVerbosity() const noexcept = 0;
356     virtual IEngineInspector* createEngineInspector() const noexcept = 0;
357     virtual Dims getTensorShape(char const* tensorName) const noexcept = 0;
358     virtual DataType getTensorDataType(char const* tensorName) const noexcept = 0;
359     virtual TensorLocation getTensorLocation(char const* tensorName) const noexcept = 0;
360     virtual bool isShapeInferenceIO(char const* tensorName) const noexcept = 0;
361     virtual TensorIOMode getTensorIOMode(char const* tensorName) const noexcept = 0;
362     virtual int32_t getTensorBytesPerComponent(char const* tensorName) const noexcept = 0;
363     virtual int32_t getTensorComponentsPerElement(char const* tensorName) const noexcept = 0;
364     virtual TensorFormat getTensorFormat(char const* tensorName) const noexcept = 0;
365     virtual char const* getTensorFormatDesc(char const* tensorName) const noexcept = 0;
366     virtual int32_t getTensorVectorizedDim(char const* tensorName) const noexcept = 0;
367     virtual Dims getProfileShape(
368         char const* tensorName, int32_t profileIndex, OptProfileSelector select) const noexcept = 0;
369     virtual int32_t getNbIOTensors() const noexcept = 0;
370     virtual char const* getIOTensorName(int32_t index) const noexcept = 0;
371     virtual HardwareCompatibilityLevel getHardwareCompatibilityLevel() const noexcept = 0;
372     virtual int32_t getNbAuxStreams() const noexcept = 0;
373
374     virtual int32_t getTensorBytesPerComponentV2(char const* tensorName, int32_t profileIndex) const noexcept
    = 0;
375     virtual int32_t getTensorComponentsPerElementV2(char const* tensorName, int32_t profileIndex) const
    noexcept = 0;
376     virtual TensorFormat getTensorFormatV2(char const* tensorName, int32_t profileIndex) const noexcept = 0;
377     virtual char const* getTensorFormatDescV2(char const* tensorName, int32_t profileIndex) const noexcept =
    0;
378     virtual int32_t getTensorVectorizedDimV2(char const* tensorName, int32_t profileIndex) const noexcept =
    0;
379
380     virtual ISerializationConfig* createSerializationConfig() noexcept = 0;
381     virtual IHostMemory* serializeWithConfig(ISerializationConfig& config) const noexcept = 0;
382
383     virtual size_t getDeviceMemorySizeForProfile(int32_t profileIndex) const noexcept = 0;
384     virtual IRefitter* createRefitter(ILogger& logger) noexcept = 0;
385
```

```
386     virtual bool setWeightStreamingBudget(int64_t gpuMemoryBudget) noexcept = 0;
387     virtual int64_t getWeightStreamingBudget() const noexcept = 0;
388     virtual int64_t getMinimumWeightStreamingBudget() const noexcept = 0;
389     virtual int64_t getStreamableWeightsSize() const noexcept = 0;
390
391     virtual bool isDebugTensor(char const* name) const noexcept = 0;
392
393     // Added in TensorRT 10.1
394     virtual bool setWeightStreamingBudgetV2(int64_t gpuMemoryBudget) noexcept = 0;
395     virtual int64_t getWeightStreamingBudgetV2() const noexcept = 0;
396     virtual int64_t getWeightStreamingAutomaticBudget() const noexcept = 0;
397     virtual int64_t getWeightStreamingScratchMemorySize() const noexcept = 0;
398     virtual int64_t getDeviceMemorySizeV2() const noexcept = 0;
399     virtual int64_t getDeviceMemorySizeForProfileV2(int32_t profileIndex) const noexcept = 0;
400 };
401
402 class VExecutionContext : public VRoot
403 {
404 public:
405     virtual IExecutionContext* getPImpl() noexcept = 0;
406     virtual void setDebugSync(bool sync) noexcept = 0;
407     virtual bool getDebugSync() const noexcept = 0;
408     virtual void setProfiler(IProfiler*) noexcept = 0;
409     virtual IProfiler* getProfiler() const noexcept = 0;
410     virtual ICudaEngine const& getEngine() const noexcept = 0;
411     virtual void setName(char const* name) noexcept = 0;
412     virtual char const* getName() const noexcept = 0;
413     virtual void setDeviceMemory(void* memory) noexcept = 0;
414     virtual int32_t getOptimizationProfile() const noexcept = 0;
415     virtual bool allInputDimensionsSpecified() const noexcept = 0;
416     virtual bool allInputShapesSpecified() const noexcept = 0;
417     virtual void setErrorRecorder(IErrorRecorder* recorder) noexcept = 0;
418     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
419     virtual bool executeV2(void* const* bindings) noexcept = 0;
420     virtual bool setOptimizationProfileAsync(int32_t profileIndex, cudaStream_t stream) noexcept = 0;
421     virtual void setEnqueueEmitsProfile(bool enqueueEmitsProfile) noexcept = 0;
422     virtual bool getEnqueueEmitsProfile() const noexcept = 0;
423     virtual bool reportToProfiler() const noexcept = 0;
424     virtual bool setInputShape(char const* tensorName, Dims const& dims) noexcept = 0;
425     virtual Dims getTensorShape(char const* tensorName) const noexcept = 0;
426     virtual Dims getTensorStrides(char const* tensorName) const noexcept = 0;
427     virtual bool setTensorAddress(char const* tensorName, void* data) noexcept = 0;
428     virtual void const* getTensorAddress(char const* tensorName) const noexcept = 0;
429     virtual bool setInputTensorAddress(char const* tensorName, void const* data) noexcept = 0;
430     virtual bool setOutputTensorAddress(char const* tensorName, void* data) noexcept = 0;
431     virtual int32_t inferShapes(int32_t nbMaxNames, char const** tensorNames) noexcept = 0;
432     virtual bool setInputConsumedEvent(cudaEvent_t event) noexcept = 0;
433     virtual cudaEvent_t getInputConsumedEvent() const noexcept = 0;
434     virtual void* getOutputTensorAddress(char const* tensorName) const noexcept = 0;
435     virtual bool setOutputAllocator(char const* tensorName, IOutputAllocator* outputAllocator) noexcept = 0;
436     virtual IOutputAllocator* getOutputAllocator(char const* name) noexcept = 0;
437     virtual int64_t getMaxOutputSize(char const* tensorName) const noexcept = 0;
438     virtual bool setTemporaryStorageAllocator(IGpuAllocator* allocator) noexcept = 0;
439     virtual IGpuAllocator* getTemporaryStorageAllocator() const noexcept = 0;
440     virtual bool enqueueV3(cudaStream_t stream) noexcept = 0;
441     virtual void setPersistentCacheLimit(size_t size) noexcept = 0;
442     virtual size_t getPersistentCacheLimit() const noexcept = 0;
443     virtual bool setNvtxVerbosity(ProfilingVerbosity verbosity) noexcept = 0;
444     virtual ProfilingVerbosity getNvtxVerbosity() const noexcept = 0;
445     virtual void setAuxStreams(cudaStream_t* auxStreams, int32_t nbStreams) noexcept = 0;
446     virtual bool setDebugListener(IDebugListener* listener) noexcept = 0;
447     virtual IDebugListener* getDebugListener() noexcept = 0;
448     virtual bool setTensorDebugState(char const* name, bool flag) noexcept = 0;
449     virtual bool getDebugState(char const* name) const noexcept = 0;
450     virtual bool setAllTensorsDebugState(bool flag) noexcept = 0;
451     virtual size_t updateDeviceMemorySizeForShapes() noexcept = 0;
452
453     // Added in TensorRT 10.1
454     virtual void setDeviceMemoryV2(void* memory, int64_t size) noexcept = 0;
455 };
456
457 class VEngineInspector : public VRoot
458 {
459 public:
460     virtual IEngineInspector* getPImpl() noexcept = 0;
461     virtual bool setExecutionContext(IExecutionContext const* context) noexcept = 0;
462     virtual IExecutionContext const* getExecutionContext() const noexcept = 0;
463     virtual char const* getLayerInformation(int32_t layerIndex, LayerInformationFormat format) const
        noexcept = 0;
464     virtual char const* getEngineInformation(LayerInformationFormat format) const noexcept = 0;
465     virtual void setErrorRecorder(IErrorRecorder* recorder) noexcept = 0;
```

```
466     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
467 };
468
469 class VTensor : public VRoot
470 {
471 public:
472     virtual void setName(char const* name) noexcept = 0;
473     virtual char const* getName() const noexcept = 0;
474     virtual void setDimensions(Dims const& dimensions) noexcept = 0;
475     virtual Dims getDimensions() const noexcept = 0;
476     virtual void setType(DataType type) noexcept = 0;
477     virtual DataType getType() const noexcept = 0;
478     virtual bool setDynamicRange(float min, float max) noexcept = 0;
479     virtual bool isNetworkInput() const noexcept = 0;
480     virtual bool isNetworkOutput() const noexcept = 0;
481     virtual void setBroadcastAcrossBatch(bool broadcastAcrossBatch) noexcept = 0;
482     virtual bool getBroadcastAcrossBatch() const noexcept = 0;
483     virtual TensorLocation getLocation() const noexcept = 0;
484     virtual void setLocation(TensorLocation location) noexcept = 0;
485     virtual bool dynamicRangeIsSet() const noexcept = 0;
486     virtual void resetDynamicRange() noexcept = 0;
487     virtual float getDynamicRangeMin() const noexcept = 0;
488     virtual float getDynamicRangeMax() const noexcept = 0;
489     virtual void setAllowedFormats(TensorFormats formats) noexcept = 0;
490     virtual TensorFormats getAllowedFormats() const noexcept = 0;
491     virtual bool isShapeTensor() const noexcept = 0;
492     virtual bool isExecutionTensor() const noexcept = 0;
493     virtual void setDimensionName(int32_t index, char const* name) noexcept = 0;
494     virtual char const* getDimensionName(int32_t index) const noexcept = 0;
495 };
496 class VLayer : public VRoot
497 {
498 public:
499     virtual LayerType getType() const noexcept = 0;
500     virtual void setName(char const* name) noexcept = 0;
501     virtual char const* getName() const noexcept = 0;
502     virtual int32_t getNbInputs() const noexcept = 0;
503     virtual ITensor* getInput(int32_t index) const noexcept = 0;
504     virtual int32_t getNbOutputs() const noexcept = 0;
505     virtual ITensor* getOutput(int32_t index) const noexcept = 0;
506     virtual void setInput(int32_t index, ITensor& tensor) noexcept = 0;
507     virtual void setPrecision(DataType dataType) noexcept = 0;
508     virtual DataType getPrecision() const noexcept = 0;
509     virtual bool precisionIsSet() const noexcept = 0;
510     virtual void resetPrecision() noexcept = 0;
511     virtual void setOutputType(int32_t index, DataType dataType) noexcept = 0;
512     virtual DataType getOutputType(int32_t index) const noexcept = 0;
513     virtual bool outputTypeIsSet(int32_t index) const noexcept = 0;
514     virtual void resetOutputType(int32_t index) noexcept = 0;
515     virtual void setMetadata(char const* docString) noexcept = 0;
516     virtual char const* getMetadata() const noexcept = 0;
517 };
518
519 class VConvolutionLayer : public VRoot
520 {
521 public:
522     virtual void setNbOutputMaps(int64_t nbOutputMaps) noexcept = 0;
523     virtual int64_t getNbOutputMaps() const noexcept = 0;
524     virtual void setNbGroups(int64_t nbGroups) noexcept = 0;
525     virtual int64_t getNbGroups() const noexcept = 0;
526     virtual void setKernelWeights(Weights weights) noexcept = 0;
527     virtual Weights getKernelWeights() const noexcept = 0;
528     virtual void setBiasWeights(Weights weights) noexcept = 0;
529     virtual Weights getBiasWeights() const noexcept = 0;
530     virtual void setPrePadding(Dims const&  padding) noexcept = 0;
531     virtual Dims getPrePadding() const noexcept = 0;
532     virtual void setPostPadding(Dims const& padding) noexcept = 0;
533     virtual Dims getPostPadding() const noexcept = 0;
534     virtual void setPaddingMode(PaddingMode paddingMode) noexcept = 0;
535     virtual PaddingMode getPaddingMode() const noexcept = 0;
536     virtual void setKernelSizeNd(Dims const& kernelSize) noexcept = 0;
537     virtual Dims getKernelSizeNd() const noexcept = 0;
538     virtual void setStrideNd(Dims const& stride) noexcept = 0;
539     virtual Dims getStrideNd() const noexcept = 0;
540     virtual void setPaddingNd(Dims const& padding) noexcept = 0;
541     virtual Dims getPaddingNd() const noexcept = 0;
542     virtual void setDilationNd(Dims const& dilation) noexcept = 0;
543     virtual Dims getDilationNd() const noexcept = 0;
544 };
545
546 class VActivationLayer : public VRoot
```

```
547 {
548 public:
549     virtual void setActivationType(ActivationType type) noexcept = 0;
550     virtual ActivationType getActivationType() const noexcept = 0;
551     virtual void setAlpha(float alpha) noexcept = 0;
552     virtual void setBeta(float beta) noexcept = 0;
553     virtual float getAlpha() const noexcept = 0;
554     virtual float getBeta() const noexcept = 0;
555 };
556
557 class VPoolingLayer : public VRoot
558 {
559 public:
560     virtual void setPoolingType(PoolingType type) noexcept = 0;
561     virtual PoolingType getPoolingType() const noexcept = 0;
562     virtual void setBlendFactor(float blendFactor) noexcept = 0;
563     virtual float getBlendFactor() const noexcept = 0;
564     virtual void setAverageCountExcludesPadding(bool exclusive) noexcept = 0;
565     virtual bool getAverageCountExcludesPadding() const noexcept = 0;
566     virtual void setPrePadding(Dims const& padding) noexcept = 0;
567     virtual Dims getPrePadding() const noexcept = 0;
568     virtual void setPostPadding(Dims const& padding) noexcept = 0;
569     virtual Dims getPostPadding() const noexcept = 0;
570     virtual void setPaddingMode(PaddingMode paddingMode) noexcept = 0;
571     virtual PaddingMode getPaddingMode() const noexcept = 0;
572     virtual void setWindowSizeNd(Dims const& windowSize) noexcept = 0;
573     virtual Dims getWindowSizeNd() const noexcept = 0;
574     virtual void setStrideNd(Dims const& stride) noexcept = 0;
575     virtual Dims getStrideNd() const noexcept = 0;
576     virtual void setPaddingNd(Dims const& padding) noexcept = 0;
577     virtual Dims getPaddingNd() const noexcept = 0;
578 };
579
580 class VLRNLayer : public VRoot
581 {
582 public:
583     virtual void setWindowSize(int64_t windowSize) noexcept = 0;
584     virtual int64_t getWindowSize() const noexcept = 0;
585     virtual void setAlpha(float alpha) noexcept = 0;
586     virtual float getAlpha() const noexcept = 0;
587     virtual void setBeta(float beta) noexcept = 0;
588     virtual float getBeta() const noexcept = 0;
589     virtual void setK(float k) noexcept = 0;
590     virtual float getK() const noexcept = 0;
591 };
592
593 class VScaleLayer : public VRoot
594 {
595 public:
596     virtual void setMode(ScaleMode mode) noexcept = 0;
597     virtual ScaleMode getMode() const noexcept = 0;
598     virtual void setShift(Weights shift) noexcept = 0;
599     virtual Weights getShift() const noexcept = 0;
600     virtual void setScale(Weights scale) noexcept = 0;
601     virtual Weights getScale() const noexcept = 0;
602     virtual void setPower(Weights power) noexcept = 0;
603     virtual Weights getPower() const noexcept = 0;
604     virtual int32_t getChannelAxis() const noexcept = 0;
605     virtual void setChannelAxis(int32_t channelAxis) noexcept = 0;
606 };
607
608 class VSoftMaxLayer : public VRoot
609 {
610 public:
611     virtual void setAxes(uint32_t axes) noexcept = 0;
612     virtual uint32_t getAxes() const noexcept = 0;
613 };
614
615 class VConcatenationLayer : public VRoot
616 {
617 public:
618     virtual void setAxis(int32_t axis) noexcept = 0;
619     virtual int32_t getAxis() const noexcept = 0;
620 };
621
622 class VDeconvolutionLayer : public VRoot
623 {
624 public:
625     virtual void setNbOutputMaps(int64_t nbOutputMaps) noexcept = 0;
626     virtual int64_t getNbOutputMaps() const noexcept = 0;
627     virtual void setNbGroups(int64_t nbGroups) noexcept = 0;
```

```
628     virtual int64_t getNbGroups() const noexcept = 0;
629     virtual void setKernelWeights(Weights weights) noexcept = 0;
630     virtual Weights getKernelWeights() const noexcept = 0;
631     virtual void setBiasWeights(Weights weights) noexcept = 0;
632     virtual Weights getBiasWeights() const noexcept = 0;
633     virtual void setPrePadding(Dims const& padding) noexcept = 0;
634     virtual Dims getPrePadding() const noexcept = 0;
635     virtual void setPostPadding(Dims const& padding) noexcept = 0;
636     virtual Dims getPostPadding() const noexcept = 0;
637     virtual void setPaddingMode(PaddingMode paddingMode) noexcept = 0;
638     virtual PaddingMode getPaddingMode() const noexcept = 0;
639     virtual void setKernelSizeNd(Dims const& kernelSize) noexcept = 0;
640     virtual Dims getKernelSizeNd() const noexcept = 0;
641     virtual void setStrideNd(Dims const& stride) noexcept = 0;
642     virtual Dims getStrideNd() const noexcept = 0;
643     virtual void setPaddingNd(Dims const& padding) noexcept = 0;
644     virtual Dims getPaddingNd() const noexcept = 0;
645     virtual void setDilationNd(Dims const& dilation) noexcept = 0;
646     virtual Dims getDilationNd() const noexcept = 0;
647 };
648
649 class VElementWiseLayer : public VRoot
650 {
651 public:
652     virtual void setOperation(ElementWiseOperation op) noexcept = 0;
653     virtual ElementWiseOperation getOperation() const noexcept = 0;
654 };
655
656 class VGatherLayer : public VRoot
657 {
658 public:
659     virtual void setGatherAxis(int32_t axis) noexcept = 0;
660     virtual int32_t getGatherAxis() const noexcept = 0;
661     virtual void setNbElementWiseDims(int32_t k) noexcept = 0;
662     virtual int32_t getNbElementWiseDims() const noexcept = 0;
663     virtual void setMode(GatherMode mode) noexcept = 0;
664     virtual GatherMode getMode() const noexcept = 0;
665 };
666
667 class VPluginLayer : public VRoot
668 {
669 public:
670     virtual IPlugin& getPlugin() noexcept = 0;
671 };
672
673 class VPluginV2Layer : public VRoot
674 {
675 public:
676     virtual IPluginV2& getPlugin() noexcept = 0;
677 };
678
679 class VPluginV3Layer : public VRoot
680 {
681 public:
682     virtual IPluginV3& getPlugin() noexcept = 0;
683 };
684
685 class VUnaryLayer : public VRoot
686 {
687 public:
688     virtual void setOperation(UnaryOperation op) noexcept = 0;
689     virtual UnaryOperation getOperation() const noexcept = 0;
690 };
691
692 class VReduceLayer : public VRoot
693 {
694 public:
695     virtual void setOperation(ReduceOperation op) noexcept = 0;
696     virtual ReduceOperation getOperation() const noexcept = 0;
697     virtual void setReduceAxes(uint32_t reduceAxes) noexcept = 0;
698     virtual uint32_t getReduceAxes() const noexcept = 0;
699     virtual void setKeepDimensions(bool keepDimensions) noexcept = 0;
700     virtual bool getKeepDimensions() const noexcept = 0;
701 };
702
703 class VPaddingLayer : public VRoot
704 {
705 public:
706     virtual void setPrePaddingNd(Dims const& padding) noexcept = 0;
707     virtual Dims getPrePaddingNd() const noexcept = 0;
708     virtual void setPostPaddingNd(Dims const& padding) noexcept = 0;
```

```
709     virtual Dims getPostPaddingNd() const noexcept = 0;
710 };
711
712 class VShuffleLayer : public VRoot
713 {
714 public:
715     virtual void setFirstTranspose(Permutation const& permutation) noexcept = 0;
716     virtual Permutation const& getFirstTranspose() const noexcept = 0;
717     virtual void setReshapeDimensions(Dims const& dimensions) noexcept = 0;
718     virtual Dims getReshapeDimensions() const noexcept = 0;
719     virtual void setSecondTranspose(Permutation const& permutation) noexcept = 0;
720     virtual Permutation const& getSecondTranspose() const noexcept = 0;
721     virtual void setZeroIsPlaceholder(bool zeroIsPlaceholder) noexcept = 0;
722     virtual bool getZeroIsPlaceholder() const noexcept = 0;
723 };
724
725 class VSliceLayer : public VRoot
726 {
727 public:
728     virtual void setStart(Dims const& start) noexcept = 0;
729     virtual Dims getStart() const noexcept = 0;
730     virtual void setSize(Dims const& size) noexcept = 0;
731     virtual Dims getSize() const noexcept = 0;
732     virtual void setStride(Dims const& stride) noexcept = 0;
733     virtual Dims getStride() const noexcept = 0;
734     virtual void setMode(SampleMode mode) noexcept = 0;
735     virtual SampleMode getMode() const noexcept = 0;
736     virtual void setAxes(Dims const& axes) noexcept = 0;
737     virtual Dims getAxes() const noexcept = 0;
738 };
739
740 class VShapeLayer : public VRoot
741 {
742 public:
743 };
744
745 class VTopKLayer : public VRoot
746 {
747 public:
748     virtual void setOperation(TopKOperation op) noexcept = 0;
749     virtual TopKOperation getOperation() const noexcept = 0;
750     virtual void setK(int32_t k) noexcept = 0;
751     virtual int32_t getK() const noexcept = 0;
752     virtual void setReduceAxes(uint32_t reduceAxes) noexcept = 0;
753     virtual uint32_t getReduceAxes() const noexcept = 0;
754 };
755
756 class VMatrixMultiplyLayer : public VRoot
757 {
758 public:
759     virtual void setOperation(int32_t index, MatrixOperation op) noexcept = 0;
760     virtual MatrixOperation getOperation(int32_t index) const noexcept = 0;
761 };
762
763 class VNonZeroLayer : public VRoot
764 {
765 public:
766 };
767
768 class VRaggedSoftMaxLayer : public VRoot
769 {
770 public:
771 };
772
773 class VIdentityLayer : public VRoot
774 {
775 public:
776 };
777
778 class VCastLayer : public VRoot
779 {
780 public:
781     virtual void setToType(DataType toType) noexcept = 0;
782     virtual DataType getToType() const noexcept = 0;
783 };
784
785 class VConstantLayer : public VRoot
786 {
787 public:
788     virtual void setWeights(Weights weights) noexcept = 0;
789     virtual Weights getWeights() const noexcept = 0;
```

```
790     virtual void setDimensions(Dims const& dimensions) noexcept = 0;
791     virtual Dims getDimensions() const noexcept = 0;
792 };
793
794 class VParametricReLULayer : public VRoot
795 {
796 public:
797 };
798
799 class VResizeLayer : public VRoot
800 {
801 public:
802     virtual void setOutputDimensions(Dims const& dimensions) noexcept = 0;
803     virtual Dims getOutputDimensions() const noexcept = 0;
804     virtual void setScales(float const* scales, int32_t nbScales) noexcept = 0;
805     virtual int32_t getScales(int32_t size, float* scales) const noexcept = 0;
806     virtual void setResizeMode(InterpolationMode interpolationMode) noexcept = 0;
807     virtual InterpolationMode getResizeMode() const noexcept = 0;
808     virtual void setCoordinateTransformation(ResizeCoordinateTransformation coordTransform) noexcept = 0;
809     virtual ResizeCoordinateTransformation getCoordinateTransformation() const noexcept = 0;
810     virtual void setSelectorForSinglePixel(ResizeSelector selector) noexcept = 0;
811     virtual ResizeSelector getSelectorForSinglePixel() const noexcept = 0;
812     virtual void setNearestRounding(ResizeRoundMode value) noexcept = 0;
813     virtual ResizeRoundMode getNearestRounding() const noexcept = 0;
814     virtual void setCubicCoeff(float value) noexcept = 0;
815     virtual float getCubicCoeff() const noexcept = 0;
816     virtual void setExcludeOutside(bool value) noexcept = 0;
817     virtual bool getExcludeOutside() const noexcept = 0;
818 };
819
820 class VLoopBoundaryLayer : public VRoot
821 {
822 public:
823     virtual ILoop* getLoop() const noexcept = 0;
824 };
825
826 class VRecurrenceLayer : public VRoot
827 {
828 public:
829 };
830
831 class VLoopOutputLayer : public VRoot
832 {
833 public:
834     virtual LoopOutput getLoopOutput() const noexcept = 0;
835     virtual void setAxis(int32_t axis) noexcept = 0;
836     virtual int32_t getAxis() const noexcept = 0;
837 };
838
839 class VTripLimitLayer : public VRoot
840 {
841 public:
842     virtual TripLimit getTripLimit() const noexcept = 0;
843 };
844
845 class VIteratorLayer : public VRoot
846 {
847 public:
848     virtual void setAxis(int32_t axis) noexcept = 0;
849     virtual int32_t getAxis() const noexcept = 0;
850     virtual void setReverse(bool reverse) noexcept = 0;
851     virtual bool getReverse() const noexcept = 0;
852 };
853 class VLoop : public VRoot
854 {
855 public:
856     virtual IRecurrenceLayer* addRecurrence(ITensor& initialValue) noexcept = 0;
857     virtual ITripLimitLayer* addTripLimit(ITensor& tensor, TripLimit limit) noexcept = 0;
858     virtual IIteratorLayer* addIterator(ITensor& tensor, int32_t axis = 0, bool reverse = false) noexcept =
    0;
859     virtual ILoopOutputLayer* addLoopOutput(ITensor& tensor, LoopOutput outputKind, int32_t axis = 0)
    noexcept = 0;
860     virtual void setName(char const* name) noexcept = 0;
861     virtual char const* getName() const noexcept = 0;
862 };
863
864 class VConditionalBoundaryLayer : public VRoot
865 {
866 public:
867     virtual IIfConditional* getConditional() const noexcept = 0;
868 };
```

```
869
870 class VConditionLayer : public VRoot
871 {
872 public:
873 };
874
875 class VConditionalInputLayer : public VRoot
876 {
877 public:
878 };
879
880 class VConditionalOutputLayer : public VRoot
881 {
882 public:
883 };
884
885 class VIfConditional : public VRoot
886 {
887 public:
888     virtual IConditionLayer* setCondition(ITensor& tensor) noexcept = 0;
889     virtual IIfConditionalInputLayer* addInput(ITensor& tensor) noexcept = 0;
890     virtual IIfConditionalOutputLayer* addOutput(ITensor& trueTensor, ITensor& falseTensor) noexcept = 0;
891     virtual void setName(char const* name) noexcept = 0;
892     virtual char const* getName() const noexcept = 0;
893 };
894
895 class VSelectLayer : public VRoot
896 {
897 };
898
899 class VAssertionLayer : public VRoot
900 {
901 public:
902     virtual void setMessage(char const* message) noexcept = 0;
903     virtual char const* getMessage() const noexcept = 0;
904 };
905
906 class VFillLayer : public VRoot
907 {
908 public:
909     virtual void setDimensions(Dims const& dimensions) noexcept = 0;
910     virtual Dims getDimensions() const noexcept = 0;
911     virtual void setOperation(FillOperation op) noexcept = 0;
912     virtual FillOperation getOperation() const noexcept = 0;
913     virtual void setAlpha(double alpha) noexcept = 0;
914     virtual double getAlpha() const noexcept = 0;
915     virtual void setBeta(double beta) noexcept = 0;
916     virtual double getBeta() const noexcept = 0;
917     virtual void setAlphaInt64(int64_t alpha) noexcept = 0;
918     virtual int64_t getAlphaInt64() const noexcept = 0;
919     virtual void setBetaInt64(int64_t beta) noexcept = 0;
920     virtual int64_t getBetaInt64() const noexcept = 0;
921     virtual bool isAlphaBetaInt64() const noexcept = 0;
922     virtual DataType getToType() const noexcept = 0;
923     virtual void setToType(DataType toType) noexcept = 0;
924 };
925
926 class VQuantizeLayer : public VRoot
927 {
928 public:
929     virtual int32_t getAxis() const noexcept = 0;
930     virtual void setAxis(int32_t axis) noexcept = 0;
931     virtual DataType getToType() const noexcept = 0;
932     virtual void setToType(DataType toType) noexcept = 0;
933 };
934
935 class VDequantizeLayer : public VRoot
936 {
937 public:
938     virtual int32_t getAxis() const noexcept = 0;
939     virtual void setAxis(int32_t axis) noexcept = 0;
940     virtual DataType getToType() const noexcept = 0;
941     virtual void setToType(DataType toType) noexcept = 0;
942 };
943
944 class VDynamicQuantizeLayer : public VRoot
945 {
946 public:
947     virtual int32_t getAxis() const noexcept = 0;
948     virtual void setAxis(int32_t axis) noexcept = 0;
949     virtual int32_t getBlockSize() const noexcept = 0;
```

```
950     virtual void setBlockSize(int32_t axis) noexcept = 0;
951     virtual DataType getScaleType() const noexcept = 0;
952     virtual void setScaleType(DataType axis) noexcept = 0;
953     virtual DataType getToType() const noexcept = 0;
954     virtual void setToType(DataType toType) noexcept = 0;
955 };
956
957 class VScatterLayer : public VRoot
958 {
959 public:
960    virtual void setMode(ScatterMode mode) noexcept = 0;
961    virtual ScatterMode getMode() const noexcept = 0;
962    virtual void setAxis(int32_t axis) noexcept = 0;
963    virtual int32_t getAxis() const noexcept = 0;
964 }; // class VScatterLayer
965
966 class VEinsumLayer : public VRoot
967 {
968 public:
969     virtual bool setEquation(char const* equation) noexcept = 0;
970     virtual char const* getEquation() const noexcept = 0;
971 };
972
973 class VOneHotLayer : public VRoot
974 {
975 public:
976     virtual int32_t getAxis() const noexcept = 0;
977     virtual void setAxis(int32_t axis) noexcept = 0;
978 }; // class VOneHotLayer
979
980 class VGridSampleLayer : public VRoot
981 {
982 public:
983     virtual void setInterpolationMode(InterpolationMode mode) noexcept = 0;
984     virtual InterpolationMode getInterpolationMode() const noexcept = 0;
985     virtual void setAlignCorners(bool alignCorners) noexcept = 0;
986     virtual bool getAlignCorners() const noexcept = 0;
987     virtual bool setSampleMode(SampleMode mode) noexcept = 0;
988     virtual SampleMode getSampleMode() const noexcept = 0;
989 }; // class VGridSampleLayer
990
991 class VNMSLayer : public VRoot
992 {
993 public:
994     virtual void setBoundingBoxFormat(BoundingBoxFormat fmt) noexcept = 0;
995     virtual BoundingBoxFormat getBoundingBoxFormat() const noexcept = 0;
996     virtual void setTopKBoxLimit(int32_t limit) noexcept = 0;
997     virtual int32_t getTopKBoxLimit() const noexcept = 0;
998 }; // class VNMSLayer
999
1000 class VReverseSequenceLayer : public VRoot
1001 {
1002 public:
1003     virtual void setBatchAxis(int32_t batchAxis) noexcept = 0;
1004     virtual int32_t getBatchAxis() const noexcept = 0;
1005
1006     virtual void setSequenceAxis(int32_t sequenceAxis) noexcept = 0;
1007     virtual int32_t getSequenceAxis() const noexcept = 0;
1008 }; // class VReverseSequenceLayer
1009
1010 class VNormalizationLayer : public VRoot
1011 {
1012 public:
1013     virtual void setEpsilon(float eps) noexcept = 0;
1014     virtual float getEpsilon() const noexcept = 0;
1015     virtual void setAxes(uint32_t axesMask) noexcept = 0;
1016     virtual uint32_t getAxes() const noexcept = 0;
1017     virtual void setNbGroups(int64_t nbGroups) noexcept = 0;
1018     virtual int64_t getNbGroups() const noexcept = 0;
1019     virtual void setComputePrecision(DataType type) noexcept = 0;
1020     virtual DataType getComputePrecision() const noexcept = 0;
1021 }; // class VNormalizationLayer
1022
1023 class VSqueezeLayer : public VRoot
1024 {
1025 };
1026
1027 class VUnsqueezeLayer : public VRoot
1028 {
1029 };
1030
```

```
1031 class VCumulativeLayer : public VRoot
1032 {
1033 public:
1034     virtual bool setOperation(CumulativeOperation op) noexcept = 0;
1035     virtual CumulativeOperation getOperation() const noexcept = 0;
1036     virtual void setExclusive(bool exclusive) noexcept = 0;
1037     virtual bool getExclusive() const noexcept = 0;
1038     virtual void setReverse(bool reverse) noexcept = 0;
1039     virtual bool getReverse() const noexcept = 0;
1040 }; // class VCumulativeLayer
1041
1042
1043 class VNetworkDefinition : public VRoot
1044 {
1045 public:
1046     virtual ITensor* addInput(char const* name, DataType type, Dims const& dimensions) noexcept = 0;
1047     virtual void markOutput(ITensor& tensor) noexcept = 0;
1048     virtual IActivationLayer* addActivation(ITensor& input, ActivationType type) noexcept = 0;
1049     virtual ILRNLayer* addLRN(ITensor& input, int64_t window, float alpha, float beta, float k) noexcept =
    0;
1050     virtual IScaleLayer* addScale(
1051         ITensor& input, ScaleMode mode, Weights shift, Weights scale, Weights power) noexcept = 0;
1052     virtual ISoftMaxLayer* addSoftMax(ITensor& input) noexcept = 0;
1053     virtual IConcatenationLayer* addConcatenation(ITensor* const* inputs, int32_t nbInputs) noexcept = 0;
1054     virtual IElementWiseLayer* addElementWise(ITensor& input1, ITensor& input2, ElementWiseOperation op)
    noexcept = 0;
1055     virtual IUnaryLayer* addUnary(ITensor& input, UnaryOperation operation) noexcept = 0;
1056     virtual IShuffleLayer* addShuffle(ITensor& input) noexcept = 0;
1057     virtual int32_t getNbLayers() const noexcept = 0;
1058     virtual ILayer* getLayer(int32_t index) const noexcept = 0;
1059     virtual int32_t getNbInputs() const noexcept = 0;
1060     virtual ITensor* getInput(int32_t index) const noexcept = 0;
1061     virtual int32_t getNbOutputs() const noexcept = 0;
1062     virtual ITensor* getOutput(int32_t index) const noexcept = 0;
1063     virtual IReduceLayer* addReduce(
1064         ITensor& input, ReduceOperation operation, uint32_t reduceAxes, bool keepDimensions) noexcept
1065         = 0;
1066     virtual ITopKLayer* addTopK(ITensor& input, TopKOperation op, int32_t k, uint32_t reduceAxes) noexcept =
    0;
1067     virtual IGatherLayer* addGather(ITensor& data, ITensor& indices, int32_t axis) noexcept = 0;
1068     virtual IRaggedSoftMaxLayer* addRaggedSoftMax(ITensor& input, ITensor& bounds) noexcept = 0;
1069     virtual IMatrixMultiplyLayer* addMatrixMultiply(
1070         ITensor& input0, MatrixOperation op0, ITensor& input1, MatrixOperation op1) noexcept = 0;
1071     virtual IConstantLayer* addConstant(Dims const& dimensions, Weights weights) noexcept = 0;
1072     virtual IIdentityLayer* addIdentity(ITensor& input) noexcept = 0;
1073     virtual void removeTensor(ITensor& tensor) noexcept = 0;
1074     virtual void unmarkOutput(ITensor& tensor) noexcept = 0;
1075     virtual IPluginV2Layer* addPluginV2(ITensor* const* inputs, int32_t nbInputs, IPluginV2& plugin)
    noexcept = 0;
1076     virtual IPluginV3Layer* addPluginV3(ITensor* const* inputs, int32_t nbInputs, ITensor* const*
    shapeInputs,
1077         int32_t nbShapeInputs, IPluginV3& plugin) noexcept = 0;
1078     virtual ISliceLayer* addSlice(ITensor& input, Dims const& start, Dims const& size, Dims const& stride)
    noexcept = 0;
1079     virtual void setName(char const* name) noexcept = 0;
1080     virtual char const* getName() const noexcept = 0;
1081     virtual IShapeLayer* addShape(ITensor& input) noexcept = 0;
1082     virtual bool hasImplicitBatchDimension() const noexcept = 0;
1083     virtual bool markOutputForShapes(ITensor& tensor) noexcept = 0;
1084     virtual bool unmarkOutputForShapes(ITensor& tensor) noexcept = 0;
1085     virtual IParametricReLULayer* addParametricReLU(ITensor& input, ITensor& slope) noexcept = 0;
1086     virtual IConvolutionLayer* addConvolutionNd(
1087         ITensor& input, int64_t nbOutputMaps, Dims const& kernelSize, Weights kernelWeights, Weights
    biasWeights) noexcept
1088         = 0;
1089     virtual IPoolingLayer* addPoolingNd(ITensor& input, PoolingType type, Dims const& windowSize) noexcept
    = 0;
1090     virtual IDeconvolutionLayer* addDeconvolutionNd(
1091         ITensor& input, int64_t nbOutputMaps, Dims const& kernelSize, Weights kernelWeights, Weights
    biasWeights) noexcept
1092         = 0;
1093     virtual IScaleLayer* addScaleNd(
1094         ITensor& input, ScaleMode mode, Weights shift, Weights scale, Weights power, int32_t channelAxis)
    noexcept = 0;
1095     virtual IResizeLayer* addResize(ITensor& input) noexcept = 0;
1096     virtual ILoop* addLoop() noexcept = 0;
1097     virtual ISelectLayer* addSelect(ITensor& condition, ITensor& thenInput, ITensor& elseInput) noexcept =
    0;
1098     virtual IFillLayer* addFill(Dims const& dimensions, FillOperation op) noexcept = 0;
1099     virtual IPaddingLayer* addPaddingNd(ITensor& input, Dims const& prePadding, Dims const& postPadding)
    noexcept = 0;
```

```
1100     virtual bool setWeightsName(Weights weights, char const* name) noexcept = 0;
1101     virtual void setErrorRecorder(IErrorRecorder* recorder) noexcept = 0;
1102     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
1103     virtual IDequantizeLayer* addDequantize(ITensor& input, ITensor& scale) noexcept = 0;
1104     virtual IQuantizeLayer* addQuantize(ITensor& input, ITensor& scale) noexcept = 0;
1105     virtual IGatherLayer* addGatherV2(ITensor& data, ITensor& indices, GatherMode mode) noexcept = 0;
1106     virtual IIfConditional* addIfConditional() noexcept = 0;
1107     virtual IScatterLayer* addScatter(ITensor& data, ITensor& indices, ITensor& updates, ScatterMode mode)
     noexcept = 0;
1108     virtual IEinsumLayer* addEinsum(ITensor* const* inputs, int32_t nbInputs, char const* equation)
     noexcept = 0;
1109     virtual IAssertionLayer* addAssertion(ITensor& condition, char const* message) noexcept = 0;
1110     virtual IOneHotLayer* addOneHot(ITensor& indices, ITensor& values, ITensor& depth, int32_t axis)
     noexcept = 0;
1111     virtual INonZeroLayer* addNonZero(ITensor& input) noexcept = 0;
1112     virtual IGridSampleLayer* addGridSample(ITensor& input, ITensor& grid) noexcept = 0;
1113     virtual INMSLayer* addNMS(ITensor& boxes, ITensor& scores, ITensor& maxOutputBoxesPerClass) noexcept =
     0;
1114     virtual IReverseSequenceLayer* addReverseSequence(ITensor& input, ITensor& sequenceLens) noexcept = 0;
1115     virtual INormalizationLayer* addNormalization(
1116         ITensor& input, ITensor& scale, ITensor& bias, uint32_t axesMask) noexcept = 0;
1117     virtual ICastLayer* addCast(ITensor& input, DataType toType) noexcept = 0;
1118     virtual IBuilder& getBuilder() const noexcept = 0;
1119     virtual NetworkDefinitionCreationFlags getFlags() const noexcept = 0;
1120     virtual bool getFlag(NetworkDefinitionCreationFlag networkDefinitionCreationFlag) const noexcept = 0;
1121     virtual IQuantizeLayer* addQuantizeV2(ITensor& input, ITensor& scale, DataType outputType) noexcept =
     0;
1122     virtual IDequantizeLayer* addDequantizeV2(ITensor& input, ITensor& scale, DataType outputType) noexcept
     = 0;
1123     virtual IFillLayer* addFillV2(Dims const& dimensions, FillOperation op, DataType outputType) noexcept =
     0;
1124     virtual bool markDebug(ITensor& tensor) noexcept = 0;
1125     virtual bool unmarkDebug(ITensor& tensor) noexcept = 0;
1126     virtual bool isDebugTensor(nvinfer1::ITensor const& tensor) const noexcept = 0;
1127     virtual bool markWeightsRefittable(char const* name) noexcept = 0;
1128     virtual bool unmarkWeightsRefittable(char const* name) noexcept = 0;
1129     virtual bool areWeightsMarkedRefittable(char const* name) const noexcept = 0;
1130     virtual ISqueezeLayer* addSqueeze(ITensor& input, ITensor& axes) noexcept = 0;
1131     virtual IUnsqueezeLayer* addUnsqueeze(ITensor& input, ITensor& axes) noexcept = 0;
1132     virtual IDynamicQuantizeLayer* addDynamicQuantize(
1133         ITensor& input, int32_t axis, int32_t blockSize, DataType toType, DataType scaleType) noexcept = 0;
1134     virtual ICumulativeLayer* addCumulative(
1135         ITensor& input, ITensor& axis, CumulativeOperation operation, bool exclusive, bool reverse)
     noexcept = 0;
1136 };
1137
1138 class VAlgorithmIOInfo : public VRoot
1139 {
1140 public:
1141     virtual DataType getDataType() const noexcept = 0;
1142     virtual Dims getStrides() const noexcept = 0;
1143     virtual int64_t getVectorizedDim() const noexcept = 0;
1144     virtual int64_t getComponentsPerElement() const noexcept = 0;
1145 };
1146
1147 class VAlgorithmVariant : public VRoot
1148 {
1149 public:
1150     virtual int64_t getImplementation() const noexcept = 0;
1151     virtual int64_t getTactic() const noexcept = 0;
1152 };
1153
1154 class VAlgorithmContext : public VRoot
1155 {
1156 public:
1157     virtual char const* getName() const noexcept = 0;
1158     virtual Dims getDimensions(int32_t index, OptProfileSelector select) const noexcept = 0;
1159     virtual int32_t getNbInputs() const noexcept = 0;
1160     virtual int32_t getNbOutputs() const noexcept = 0;
1161 };
1162
1163 class VAlgorithm : public VRoot
1164 {
1165 public:
1166     virtual IAlgorithmVariant const& getAlgorithmVariant() const noexcept = 0;
1167     virtual float getTimingMSec() const noexcept = 0;
1168     virtual std::size_t getWorkspaceSize() const noexcept = 0;
1169     virtual IAlgorithmIOInfo const* getAlgorithmIOInfoByIndex(int32_t index) const noexcept = 0;
1170 };
1171
1172 class VTimingCache : public VRoot
```

```
1173 {
1174 public:
1175     virtual nvinfer1::IHostMemory* serialize() const noexcept = 0;
1176     virtual bool combine(ITimingCache const& inputCache, bool ignoreMismatch) noexcept = 0;
1177     virtual bool reset() noexcept = 0;
1178     virtual int64_t queryKeys(TimingCacheKey* keyBuffer, int64_t capacity) const noexcept = 0;
1179     virtual TimingCacheValue query(TimingCacheKey const& key) const noexcept = 0;
1180     virtual bool update(TimingCacheKey const& key, TimingCacheValue const& value) noexcept = 0;
1181 };
1182
1183 class VBuilderConfig : public VRoot
1184 {
1185 public:
1186     virtual void setAvgTimingIterations(int32_t avgTiming) noexcept = 0;
1187     virtual int32_t getAvgTimingIterations() const noexcept = 0;
1188     virtual void setEngineCapability(EngineCapability capability) noexcept = 0;
1189     virtual EngineCapability getEngineCapability() const noexcept = 0;
1190     virtual void setInt8Calibrator(IInt8Calibrator* calibrator) noexcept = 0;
1191     virtual IInt8Calibrator* getInt8Calibrator() const noexcept = 0;
1192     virtual void setFlags(BuilderFlags builderFlags) noexcept = 0;
1193     virtual BuilderFlags getFlags() const noexcept = 0;
1194     virtual void clearFlag(BuilderFlag builderFlag) noexcept = 0;
1195     virtual void setFlag(BuilderFlag builderFlag) noexcept = 0;
1196     virtual bool getFlag(BuilderFlag builderFlag) const noexcept = 0;
1197     virtual void setDeviceType(ILayer const* layer, DeviceType deviceType) noexcept = 0;
1198     virtual DeviceType getDeviceType(ILayer const* layer) const noexcept = 0;
1199     virtual bool isDeviceTypeSet(ILayer const* layer) const noexcept = 0;
1200     virtual void resetDeviceType(ILayer const* layer) noexcept = 0;
1201     virtual bool canRunOnDLA(ILayer const* layer) const noexcept = 0;
1202     virtual void setDLACore(int32_t dlaCore) noexcept = 0;
1203     virtual int32_t getDLACore() const noexcept = 0;
1204     virtual void setDefaultDeviceType(DeviceType deviceType) noexcept = 0;
1205     virtual DeviceType getDefaultDeviceType() const noexcept = 0;
1206     virtual void reset() noexcept = 0;
1207     virtual void setProfileStream(const cudaStream_t stream) noexcept = 0;
1208     virtual cudaStream_t getProfileStream() const noexcept = 0;
1209     virtual int32_t addOptimizationProfile(IOptimizationProfile const* profile) noexcept = 0;
1210     virtual int32_t getNbOptimizationProfiles() const noexcept = 0;
1211     virtual void setProfilingVerbosity(ProfilingVerbosity verbosity) noexcept = 0;
1212     virtual ProfilingVerbosity getProfilingVerbosity() const noexcept = 0;
1213     virtual void setAlgorithmSelector(IAlgorithmSelector* selector) noexcept = 0;
1214     virtual IAlgorithmSelector* getAlgorithmSelector() const noexcept = 0;
1215     virtual bool setCalibrationProfile(IOptimizationProfile const* profile) noexcept = 0;
1216     virtual IOptimizationProfile const* getCalibrationProfile() noexcept = 0;
1217     virtual void setQuantizationFlags(QuantizationFlags flags) noexcept = 0;
1218     virtual QuantizationFlags getQuantizationFlags() const noexcept = 0;
1219     virtual void clearQuantizationFlag(QuantizationFlag flag) noexcept = 0;
1220     virtual void setQuantizationFlag(QuantizationFlag flag) noexcept = 0;
1221     virtual bool getQuantizationFlag(QuantizationFlag flag) const noexcept = 0;
1222     virtual bool setTacticSources(TacticSources tacticSources) noexcept = 0;
1223     virtual TacticSources getTacticSources() const noexcept = 0;
1224     virtual nvinfer1::ITimingCache* createTimingCache(void const* blob, std::size_t size) const noexcept =
     0;
1225     virtual bool setTimingCache(ITimingCache const& cache, bool ignoreMismatch) noexcept = 0;
1226     virtual nvinfer1::ITimingCache const* getTimingCache() const noexcept = 0;
1227     virtual void setMemoryPoolLimit(MemoryPoolType pool, std::size_t poolSize) noexcept = 0;
1228     virtual std::size_t getMemoryPoolLimit(MemoryPoolType pool) const noexcept = 0;
1229     virtual void setPreviewFeature(PreviewFeature feature, bool enable) noexcept = 0;
1230     virtual bool getPreviewFeature(PreviewFeature feature) const noexcept = 0;
1231     virtual void setBuilderOptimizationLevel(int32_t level) noexcept = 0;
1232     virtual int32_t getBuilderOptimizationLevel() const noexcept = 0;
1233     virtual void setHardwareCompatibilityLevel(HardwareCompatibilityLevel hardwareCompatibilityLevel)
    noexcept = 0;
1234     virtual HardwareCompatibilityLevel getHardwareCompatibilityLevel() const noexcept = 0;
1235     virtual void setPluginsToSerialize(char const* const* paths, int32_t nbPaths) noexcept = 0;
1236     virtual char const* getPluginToSerialize(int32_t index) const noexcept = 0;
1237     virtual int32_t getNbPluginsToSerialize() const noexcept = 0;
1238     virtual void setMaxAuxStreams(int32_t nbStreams) noexcept = 0;
1239     virtual int32_t getMaxAuxStreams() const noexcept = 0;
1240     virtual void setProgressMonitor(IProgressMonitor* monitor) noexcept = 0;
1241     virtual IProgressMonitor* getProgressMonitor() const noexcept = 0;
1242     virtual void setRuntimePlatform(RuntimePlatform runtimePlatform) noexcept = 0;
1243     virtual RuntimePlatform getRuntimePlatform() const noexcept = 0;
1244     virtual void setMaxNbTactics(int32_t maxTactics) noexcept = 0;
1245     virtual int32_t getMaxNbTactics() const noexcept = 0;
1246     virtual bool setTilingOptimizationLevel(TilingOptimizationLevel level) noexcept = 0;
1247     virtual TilingOptimizationLevel getTilingOptimizationLevel() const noexcept = 0;
1248     virtual bool setL2LimitForTiling(int64_t size) noexcept = 0;
1249     virtual int64_t getL2LimitForTiling() const noexcept = 0;
1250 };
1251
```

```
1252 class VSerializationConfig : public VRoot
1253 {
1254 public:
1255     virtual bool setFlags(SerializationFlags serializationFlags) noexcept = 0;
1256     virtual SerializationFlags getFlags() const noexcept = 0;
1257     virtual bool clearFlag(SerializationFlag serializationFlag) noexcept = 0;
1258     virtual bool setFlag(SerializationFlag serializationFlag) noexcept = 0;
1259     virtual bool getFlag(SerializationFlag serializationFlag) const noexcept = 0;
1260 };
1261
1262 class VBuilder : public VRoot
1263 {
1264 public:
1265     virtual bool platformHasFastFp16() const noexcept = 0;
1266     virtual bool platformHasFastInt8() const noexcept = 0;
1267     virtual int32_t getMaxDLABatchSize() const noexcept = 0;
1268     virtual int32_t getNbDLACores() const noexcept = 0;
1269     virtual void setGpuAllocator(IGpuAllocator* allocator) noexcept = 0;
1270     virtual nvinfer1::IBuilderConfig* createBuilderConfig() noexcept = 0;
1271     virtual nvinfer1::INetworkDefinition* createNetworkV2(NetworkDefinitionCreationFlags flags) noexcept =
     0;
1272     virtual nvinfer1::IOptimizationProfile* createOptimizationProfile() noexcept = 0;
1273     virtual void setErrorRecorder(IErrorRecorder* recorder) noexcept = 0;
1274     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
1275     virtual void reset() noexcept = 0;
1276     virtual bool platformHasTf32() const noexcept = 0;
1277     virtual nvinfer1::IHostMemory* buildSerializedNetwork(
1278         INetworkDefinition& network, IBuilderConfig& config) noexcept = 0;
1279     virtual bool isNetworkSupported(INetworkDefinition const& network, IBuilderConfig const& config) const
    noexcept = 0;
1280     virtual ILogger* getLogger() const noexcept = 0;
1281     virtual bool setMaxThreads(int32_t maxThreads) noexcept = 0;
1282     virtual int32_t getMaxThreads() const noexcept = 0;
1283     virtual IPluginRegistry& getPluginRegistry() noexcept = 0;
1284     virtual ICudaEngine* buildEngineWithConfig(INetworkDefinition& network, IBuilderConfig& config)
    noexcept = 0;
1285 };
1286
1287 } // namespace apiv
1288 } // namespace nvinfer1
1289
1290 // @endcond
1291
1292 #endif // NV_INFER_RUNTIME_IMPL_H
```

10.5 NvInferLegacyDims.h File Reference

```
#include "NvInferRuntimeBase.h"
```

Classes

- class nvinfer1::Dims2

  Descriptor for two-dimensional data.
- class nvinfer1::DimsHW

  Descriptor for two-dimensional spatial data.
- class nvinfer1::Dims3

  Descriptor for three-dimensional data.
- class nvinfer1::Dims4

  Descriptor for four-dimensional data.

Namespaces

- namespace nvinfer1

  The TensorRT API version 1 namespace.

Macros

- #define NV_INFER_INTERNAL_INCLUDE 1

10.5.1 Detailed Description

This file contains declarations of legacy dimensions types which use channel semantics in their names, and declarations on which those types rely.

10.5.2 Macro Definition Documentation

10.5.2.1 NV_INFER_INTERNAL_INCLUDE

```
#define NV_INFER_INTERNAL_INCLUDE 1
```

10.6 NvInferLegacyDims.h

Go to the documentation of this file.

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2025 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: Apache-2.0
4  *
5  * Licensed under the Apache License, Version 2.0 (the "License");
6  * you may not use this file except in compliance with the License.
7  * You may obtain a copy of the License at
8  *
9  * http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in writing, software
12  * distributed under the License is distributed on an "AS IS" BASIS,
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14  * See the License for the specific language governing permissions and
15  * limitations under the License.
16  */
17
18 #ifndef NV_INFER_LEGACY_DIMS_H
19 #define NV_INFER_LEGACY_DIMS_H
20
21 #define NV_INFER_INTERNAL_INCLUDE 1
22 #include "NvInferRuntimeBase.h"
23 #undef NV_INFER_INTERNAL_INCLUDE
24
31
37 namespace nvinfer1
38 {
44 class Dims2 : public Dims
45 {
46 public:
50     Dims2()
51         : Dims2(0, 0)
52     {
53     }
54
61     Dims2(int64_t d0, int64_t d1)
62     {
63         nbDims = 2;
64         d[0] = d0;
65         d[1] = d1;
```

```
66          for (int64_t i{nbDims}; i < Dims::MAX_DIMS; ++i)
67          {
68              d[i] = 0;
69          }
70      }
71  };
72
78  class DimsHW : public Dims2
79  {
80  public:
84      DimsHW()
85          : Dims2()
86      {
87      }
88
95      DimsHW(int64_t height, int64_t width)
96          : Dims2(height, width)
97      {
98      }
99
105      int64_t& h()
106      {
107          return d[0];
108      }
109
115      int64_t h() const
116      {
117          return d[0];
118      }
119
125      int64_t& w()
126      {
127          return d[1];
128      }
129
135      int64_t w() const
136      {
137          return d[1];
138      }
139 };
140
146 class Dims3 : public Dims2
147 {
148 public:
152      Dims3()
153          : Dims3(0, 0, 0)
154      {
155      }
156
164      Dims3(int64_t d0, int64_t d1, int64_t d2)
165          : Dims2(d0, d1)
166      {
167          nbDims = 3;
168          d[2] = d2;
169      }
170 };
171
177 class Dims4 : public Dims3
178 {
179 public:
183      Dims4()
184          : Dims4(0, 0, 0, 0)
185      {
186      }
187
196      Dims4(int64_t d0, int64_t d1, int64_t d2, int64_t d3)
197          : Dims3(d0, d1, d2)
198      {
199          nbDims = 4;
200          d[3] = d3;
201      }
202 };
203
204 } // namespace nvinfer1
205
206 #endif // NV_INFER_LEGCY_DIMS_H
```

10.7 NvInferPlugin.h File Reference

```
#include "NvInfer.h"
#include "NvInferPluginUtils.h"
```

Functions

- bool initLibNvInferPlugins (void ∗logger, char const ∗libNamespace)

  Initialize and register all the existing TensorRT plugins to the Plugin Registry with an optional namespace. The plugin library author should ensure that this function name is unique to the library. This function should be called once before accessing the Plugin Registry.

10.7.1 Detailed Description

This is the API for the Nvidia provided TensorRT plugins.

10.7.2 Function Documentation

10.7.2.1 initLibNvInferPlugins()

```
bool initLibNvInferPlugins (
            void * logger,
            char const * libNamespace )
```

Initialize and register all the existing TensorRT plugins to the Plugin Registry with an optional namespace. The plugin library author should ensure that this function name is unique to the library. This function should be called once before accessing the Plugin Registry.

Parameters

logger	Logger object to print plugin registration information
libNamespace	Namespace used to register all the plugins in this library

10.8 NvInferPlugin.h

Go to the documentation of this file.

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2024 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: Apache-2.0
```

```
4  *
5  * Licensed under the Apache License, Version 2.0 (the "License");
6  * you may not use this file except in compliance with the License.
7  * You may obtain a copy of the License at
8  *
9  * http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17
18 #ifndef NV_INFER_PLUGIN_H
19 #define NV_INFER_PLUGIN_H
20
21 #include "NvInfer.h"
22 #include "NvInferPluginUtils.h"
28
29 extern "C"
30 {
38     TENSORRTAPI bool initLibNvInferPlugins(void* logger, char const* libNamespace);
39
40 } // extern "C"
41
42 #endif // NV_INFER_PLUGIN_H
```

10.9 NvInferPluginBase.h File Reference

`#include "NvInferRuntimeBase.h"`

Classes

- class nvinfer1::PluginField

  Structure containing plugin attribute field names and associated data This information can be parsed to decode necessary plugin metadata.
- struct nvinfer1::PluginFieldCollection

  Plugin field collection struct.
- class nvinfer1::v_1_0::IPluginCapability
- class nvinfer1::v_1_0::IPluginResource
- class nvinfer1::v_1_0::IPluginCreatorInterface
- class nvinfer1::v_1_0::IPluginV3

Namespaces

- namespace nvinfer1

  The TensorRT API version 1 namespace.
- namespace nvinfer1::v_1_0

  Forward declare IErrorRecorder for use in other interfaces.

Macros

- #define NV_INFER_INTERNAL_INCLUDE 1

Typedefs

- using nvinfer1::IPluginResource = v_1_0::IPluginResource
- using nvinfer1::IPluginCreatorInterface = v_1_0::IPluginCreatorInterface
- using nvinfer1::IPluginV3 = v_1_0::IPluginV3
- using nvinfer1::IPluginCapability = v_1_0::IPluginCapability

Enumerations

- enum class nvinfer1::PluginFieldType : int32_t { nvinfer1::kFLOAT16 = 0 , nvinfer1::kFLOAT32 = 1 , nvinfer1::kFLOAT64 = 2 , nvinfer1::kINT8 = 3 , nvinfer1::kINT16 = 4 , nvinfer1::kINT32 = 5 , nvinfer1::kCHAR = 6 , nvinfer1::kDIMS = 7 , nvinfer1::kUNKNOWN = 8 , nvinfer1::kBF16 = 9 , nvinfer1::kINT64 = 10 , nvinfer1::kFP8 = 11 , nvinfer1::kINT4 = 12 , nvinfer1::kFP4 = 13 }

  The possible field types for custom layer.
- enum class nvinfer1::TensorRTPhase : int32_t { nvinfer1::kBUILD = 0 , nvinfer1::kRUNTIME = 1 }

  Indicates a phase of operation of TensorRT.
- enum class nvinfer1::PluginCapabilityType : int32_t { nvinfer1::kCORE = 0 , nvinfer1::kBUILD = 1 , nvinfer1::kRUNTIME = 2 }

  Enumerates the different capability types a IPluginV3 object may have.

10.9.1 Macro Definition Documentation

10.9.1.1 NV_INFER_INTERNAL_INCLUDE

```
#define NV_INFER_INTERNAL_INCLUDE 1
```

10.10 NvInferPluginBase.h

Go to the documentation of this file.

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 2024 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: Apache-2.0
4  *
5  * Licensed under the Apache License, Version 2.0 (the "License");
6  * you may not use this file except in compliance with the License.
7  * You may obtain a copy of the License at
8  *
9  * http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in writing, software
12  * distributed under the License is distributed on an "AS IS" BASIS,
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14  * See the License for the specific language governing permissions and
15  * limitations under the License.
16  */
17
18 #ifndef NV_INFER_PLUGIN_BASE_H
19 #define NV_INFER_PLUGIN_BASE_H
20
21 #if !defined(NV_INFER_INTERNAL_INCLUDE)
```

```
22 static_assert(false, "Do not directly include this file. Include NvInferRuntime.h or NvInferPluginUtils.h");
23 #endif
24
25 #define NV_INFER_INTERNAL_INCLUDE 1
26 #include "NvInferRuntimeBase.h"
27 #undef NV_INFER_INTERNAL_INCLUDE
28 namespace nvinfer1
29 {
30
36 enum class PluginFieldType : int32_t
37 {
39     kFLOAT16 = 0,
41     kFLOAT32 = 1,
43     kFLOAT64 = 2,
45     kINT8 = 3,
47     kINT16 = 4,
49     kINT32 = 5,
51     kCHAR = 6,
53     kDIMS = 7,
55     kUNKNOWN = 8,
57     kBF16 = 9,
59     kINT64 = 10,
61     kFP8 = 11,
63     kINT4 = 12,
65     kFP4 = 13,
66 };
67
75 class PluginField
76 {
77 public:
79     AsciiChar const* name;
81     void const* data;
83     PluginFieldType type;
85     int32_t length;
86
87     PluginField(AsciiChar const* const name_ = nullptr, void const* const data_ = nullptr,
88         PluginFieldType const type_ = PluginFieldType::kUNKNOWN, int32_t const length_ = 0) noexcept
89         : name(name_)
90         , data(data_)
91         , type(type_)
92         , length(length_)
93     {
94     }
95 };
96
102 struct PluginFieldCollection
103 {
105     int32_t nbFields{};
107     PluginField const* fields{};
108 };
109
115 enum class TensorRTPhase : int32_t
116 {
118     kBUILD = 0,
120     kRUNTIME = 1
121 };
122
128 enum class PluginCapabilityType : int32_t
129 {
131     kCORE = 0,
133     kBUILD = 1,
135     kRUNTIME = 2
136 };
137
138 namespace v_1_0
139 {
140 class IPluginCapability : public IVersionedInterface
141 {
142 };
143
144 class IPluginResource : public IVersionedInterface
145 {
146 public:
150     InterfaceInfo getInterfaceInfo() const noexcept override
151     {
152         return InterfaceInfo{"IPluginResource", 1, 0};
153     }
167     virtual int32_t release() noexcept = 0;
168
181     virtual IPluginResource* clone() noexcept = 0;
182
```

```
183     ~IPluginResource() noexcept override = default;
184
185     IPluginResource() = default;
186     IPluginResource(IPluginResource const&) = default;
187     IPluginResource(IPluginResource&&) = default;
188     IPluginResource& operator=(IPluginResource const&) & = default;
189     IPluginResource& operator=(IPluginResource&&) & = default;
190 }; // class IPluginResource
191
192 class IPluginCreatorInterface : public IVersionedInterface
193 {
194 public:
195     ~IPluginCreatorInterface() noexcept override = default;
196
197 protected:
198     IPluginCreatorInterface() = default;
199     IPluginCreatorInterface(IPluginCreatorInterface const&) = default;
200     IPluginCreatorInterface(IPluginCreatorInterface&&) = default;
201     IPluginCreatorInterface& operator=(IPluginCreatorInterface const&) & = default;
202     IPluginCreatorInterface& operator=(IPluginCreatorInterface&&) & = default;
203 };
204
205 class IPluginV3 : public IVersionedInterface
206 {
207 public:
211     InterfaceInfo getInterfaceInfo() const noexcept override
212     {
213         return InterfaceInfo{"PLUGIN", 1, 0};
214     }
215
227     virtual IPluginCapability* getCapabilityInterface(PluginCapabilityType type) noexcept = 0;
228
239     virtual IPluginV3* clone() noexcept = 0;
240 };
241 } // namespace v_1_0
242
251 using IPluginResource = v_1_0::IPluginResource;
252
260 using IPluginCreatorInterface = v_1_0::IPluginCreatorInterface;
261
273 using IPluginV3 = v_1_0::IPluginV3;
274
288 using IPluginCapability = v_1_0::IPluginCapability;
289 } // namespace nvinfer1
290
291 #endif /* NV_INFER_PLUGIN_BASE_H */
```

10.11 NvInferPluginUtils.h File Reference

```
#include "NvInferRuntimeCommon.h"
```

Classes

- struct nvinfer1::plugin::PriorBoxParameters

  The PriorBox plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions (H x W).
- struct nvinfer1::plugin::RPROIParams

  RPROIParams is used to create the RPROIPlugin instance.
- struct nvinfer1::plugin::GridAnchorParameters

  The Anchor Generator plugin layer generates the prior boxes of designated sizes and aspect ratios across all dimensions (H x W). GridAnchorParameters defines a set of parameters for creating the plugin layer for all feature maps.
- struct nvinfer1::plugin::DetectionOutputParameters

  The DetectionOutput plugin layer generates the detection output based on location and confidence predictions by doing non maximum suppression.

- 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.
- 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) 1993-2024 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: Apache-2.0
4  *
5  * Licensed under the Apache License, Version 2.0 (the "License");
6  * you may not use this file except in compliance with the License.
7  * You may obtain a copy of the License at
8  *
9  * http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in writing, software
12  * distributed under the License is distributed on an "AS IS" BASIS,
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14  * See the License for the specific language governing permissions and
15  * limitations under the License.
16  */
17
18 #ifndef NV_INFER_PLUGIN_UTILS_H
19 #define NV_INFER_PLUGIN_UTILS_H
20
21 #include "NvInferRuntimeCommon.h"
22
23
24
25 namespace nvinfer1
26 {
27 namespace plugin
```

```
33 {
34
43 struct PriorBoxParameters
44 {
45     float *minSize;
46     float *maxSize;
47     float *aspectRatios;
48     int32_t numMinSize;
49     int32_t numMaxSize;
50     int32_t numAspectRatios;
51     bool flip;
53     bool clip;
54     float variance[4];
55     int32_t imgH;
56     int32_t imgW;
57     float stepH;
58     float stepW;
59     float offset;
60 };
61
67 struct RPROIParams
68 {
69     int32_t poolingH;
70     int32_t poolingW;
71     int32_t featureStride;
73     int32_t preNmsTop;
74     int32_t nmsMaxOut;
75     int32_t anchorsRatioCount;
76     int32_t anchorsScaleCount;
77     float iouThreshold;
78     float minBoxSize;
79     float spatialScale;
80 };
81
88 struct GridAnchorParameters
89 {
90     float minSize;
91     float maxSize;
92     float* aspectRatios;
93     int32_t numAspectRatios;
94     int32_t H;
95     int32_t W;
96     float variance[4];
97 };
98
106 enum class CodeTypeSSD : int32_t
107 {
108     CORNER TRT_DEPRECATED_ENUM = 0,
109     CENTER_SIZE TRT_DEPRECATED_ENUM = 1,
110     CORNER_SIZE TRT_DEPRECATED_ENUM = 2,
111     TF_CENTER TRT_DEPRECATED_ENUM = 3
112 };
113
126 struct TRT_DEPRECATED DetectionOutputParameters
127 {
128     bool shareLocation;
129     bool varianceEncodedInTarget;
131     int32_t backgroundLabelId;
132     int32_t numClasses;
133     int32_t topK;
135     int32_t keepTopK;
136     float confidenceThreshold;
137     float nmsThreshold;
138     CodeTypeSSD codeType;
139     int32_t inputOrder[3];
140     bool confSigmoid;
141     bool isNormalized;
142     bool isBatchAgnostic{true};
143 };
144
149 struct softmaxTree
150 {
151     int32_t* leaf;
152     int32_t n;
153     int32_t* parent;
154     int32_t* child;
155     int32_t* group;
156     char** name;
157     int32_t groups;
158     int32_t* groupSize;
159     int32_t* groupOffset;
```

```
160 };
161
170 struct RegionParameters
171 {
172     int32_t num;
173     int32_t coords;
174     int32_t classes;
175     softmaxTree* smTree;
176 };
177
184 struct TRT_DEPRECATED NMSParameters
185 {
186     bool shareLocation;
188     int32_t backgroundLabelId;
190     int32_t numClasses;
191     int32_t topK;
192     int32_t keepTopK;
194     float scoreThreshold;
195     float iouThreshold;
197     bool isNormalized;
199 };
200
201 } // namespace plugin
202 } // namespace nvinfer1
203
204 #endif // NV_INFER_PLUGIN_UTILS_H
```

10.13 NvInferRuntime.h File Reference

```
#include "NvInferImpl.h"
#include "NvInferPluginBase.h"
#include "NvInferRuntimeCommon.h"
```

Classes

- class nvinfer1::INoCopy

  Forward declaration of IEngineInspector for use by other interfaces.
- struct nvinfer1::impl::EnumMaxImpl< EngineCapability >

  Maximum number of elements in EngineCapability enum.
- class nvinfer1::Weights

  An array of weights used as a layer parameter.
- class nvinfer1::IHostMemory

  Class to handle library allocated memory that is accessible to the user.
- struct nvinfer1::impl::EnumMaxImpl< TensorLocation >

  Maximum number of elements in TensorLocation enum.
- class nvinfer1::IDimensionExpr

  An IDimensionExpr represents an integer expression constructed from constants, input dimensions, and binary operations. These expressions are can be used in overrides of IPluginV2DynamicExt::getOutputDimensions or IPluginV3OneBuild::getOutputShapes() to define output dimensions in terms of input dimensions.
- class nvinfer1::IExprBuilder

  Object for constructing IDimensionExpr.
- class nvinfer1::DimsExprs

  Analog of class Dims with expressions instead of constants for the dimensions.
- struct nvinfer1::DynamicPluginTensorDesc

  Summarizes tensors that a plugin might see for an input or output.

- class nvinfer1::IPluginV2DynamicExt

  Similar to IPluginV2Ext, but with support for dynamic shapes.
- class nvinfer1::v_1_0::IStreamReader
- class nvinfer1::v_1_0::IStreamReaderV2
- class nvinfer1::IPluginResourceContext

  Interface for plugins to access per context resources provided by TensorRT.
- class nvinfer1::v_1_0::IPluginV3OneCore
- class nvinfer1::v_1_0::IPluginV3OneBuild
- class nvinfer1::v_1_0::IPluginV3OneRuntime
- class nvinfer1::v_2_0::IPluginV3OneBuild
- class nvinfer1::v_1_0::IProfiler
- struct nvinfer1::impl::EnumMaxImpl< TensorFormat >

  Maximum number of elements in TensorFormat enum.
- struct nvinfer1::impl::EnumMaxImpl< AllocatorFlag >

  Maximum number of elements in AllocatorFlag enum.
- 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.
- class nvinfer1::v_1_0::IGpuAllocator
- class nvinfer1::IRuntime

  Allows a serialized functionally unsafe engine to be deserialized.
- class nvinfer1::IRefitter

  Updates weights in an engine.
- class nvinfer1::IOptimizationProfile

  Optimization profile for dynamic input dimensions and shape tensors.
- class nvinfer1::ISerializationConfig

  Holds properties for configuring an engine to serialize the binary.
- class nvinfer1::ICudaEngine

  An engine for executing inference on a built network, with functionally unsafe features.
- class nvinfer1::v_1_0::IOutputAllocator
- class nvinfer1::v_1_0::IDebugListener
- class nvinfer1::IExecutionContext

  Context for executing inference using an engine, with functionally unsafe features.
- class nvinfer1::IEngineInspector

  An engine inspector which prints out the layer information of an engine or an execution context.
- class nvinfer1::PluginRegistrar< T >

  Register the plugin creator to the registry The static registry object will be instantiated when the plugin library is loaded. This static object will register all creators available in the library to the registry.
- class nvinfer1::ILoggerFinder

  A virtual base class to find a logger. Allows a plugin to find an instance of a logger if it needs to emit a log message. A pointer to an instance of this class is passed to a plugin shared library on initialization when that plugin is serialized as part of a version-compatible plan. See the plugin chapter in the developer guide for details.
- class nvinfer1::v_1_0::IGpuAsyncAllocator
- class nvinfer1::v_1_0::IPluginCreatorV3One

Namespaces

- namespace nvinfer1

  The TensorRT API version 1 namespace.
- namespace nvinfer1::impl
- namespace nvinfer1::v_1_0

  Forward declare IErrorRecorder for use in other interfaces.
- namespace nvinfer1::v_2_0
- namespace nvinfer1::anonymous_namespace{NvInferRuntime.h}

Macros

- #define NV_INFER_INTERNAL_INCLUDE 1
- #define REGISTER_TENSORRT_PLUGIN(name) static nvinfer1::PluginRegistrar<name> plugin↩ Registrar##name {}

Typedefs

- using nvinfer1::IStreamReader = v_1_0::IStreamReader
- using nvinfer1::IStreamReaderV2 = v_1_0::IStreamReaderV2
- using nvinfer1::IPluginV3OneCore = v_1_0::IPluginV3OneCore
- using nvinfer1::IPluginV3OneBuild = v_1_0::IPluginV3OneBuild
- using nvinfer1::IPluginV3OneRuntime = v_1_0::IPluginV3OneRuntime
- using nvinfer1::IPluginV3OneBuildV2 = v_2_0::IPluginV3OneBuild
- using nvinfer1::IProfiler = v_1_0::IProfiler
- using nvinfer1::TempfileControlFlags = uint32_t

  Represents a collection of one or more TempfileControlFlag values combined using bitwise-OR operations.
- using nvinfer1::AllocatorFlags = uint32_t
- using nvinfer1::IGpuAllocator = v_1_0::IGpuAllocator
- using nvinfer1::TacticSources = uint32_t

  Represents a collection of one or more TacticSource values combine using bitwise-OR operations.
- using nvinfer1::SerializationFlags = uint32_t

  Represents one or more SerializationFlag values using binary OR operations, e.g., 1U << SerializationFlag::kEXCLUDE_LEAN_RUNTIME
- using nvinfer1::IOutputAllocator = v_1_0::IOutputAllocator
- using nvinfer1::IDebugListener = v_1_0::IDebugListener
- using nvinfer1::IGpuAsyncAllocator = v_1_0::IGpuAsyncAllocator
- using nvinfer1::IPluginCreatorV3One = v_1_0::IPluginCreatorV3One

Enumerations

- enum class nvinfer1::EngineCapability : int32_t { nvinfer1::kSTANDARD = 0 , nvinfer1::kSAFETY = 1 , nvinfer1::kDLA_STANDALONE = 2 }

  List of supported engine capability flows.
- enum class nvinfer1::DimensionOperation : int32_t { nvinfer1::kSUM = 0 , nvinfer1::kPROD = 1 , nvinfer1::kMAX = 2 , nvinfer1::kMIN = 3 , nvinfer1::kSUB = 4 , nvinfer1::kEQUAL = 5 , nvinfer1::kLESS = 6 , nvinfer1::kFLOOR_DIV = 7 , nvinfer1::kCEIL_DIV = 8 }

  An operation on two IDimensionExpr, which represent integer expressions used in dimension computations.
- enum class nvinfer1::TensorLocation : int32_t { nvinfer1::kDEVICE = 0 , nvinfer1::kHOST = 1 }

  The location for tensor data storage, device or host.
- enum class nvinfer1::SeekPosition : int32_t { nvinfer1::kSET = 0 , nvinfer1::kCUR = 1 , nvinfer1::kEND = 2 }

  Controls the seek mode of IStreamReaderV2.
- enum class nvinfer1::WeightsRole : int32_t { nvinfer1::kKERNEL = 0 , nvinfer1::kBIAS = 1 , nvinfer1::kSHIFT = 2 , nvinfer1::kSCALE = 3 , nvinfer1::kCONSTANT = 4 , nvinfer1::kANY = 5 }

  How a layer uses particular Weights.
- enum class nvinfer1::DeviceType : int32_t { nvinfer1::kGPU = 0 , nvinfer1::kDLA = 1 }

  The device that this layer/network will execute on.
- enum class nvinfer1::TempfileControlFlag : int32_t { nvinfer1::kALLOW_IN_MEMORY_FILES = 0 , nvinfer1::kALLOW_TEMPORARY_FILES = 1 }

  Flags used to control TensorRT's behavior when creating executable temporary files.
- enum class nvinfer1::TensorFormat : int32_t { nvinfer1::kLINEAR = 0 , nvinfer1::kCHW2 = 1 , nvinfer1::kHWC8 = 2 , nvinfer1::kCHW4 = 3 , nvinfer1::kCHW16 = 4 , nvinfer1::kCHW32 = 5 , nvinfer1::kDHWC8 = 6 , nvinfer1::kCDHW32 = 7 , nvinfer1::kHWC = 8 , nvinfer1::kDLA_LINEAR = 9 , nvinfer1::kDLA_HWC4 = 10 , nvinfer1::kHWC16 = 11 , nvinfer1::kDHWC = 12 }

  Format of the input/output tensors.
- enum class nvinfer1::AllocatorFlag : int32_t { nvinfer1::kRESIZABLE = 0 }

  Allowed type of memory allocation.
- enum class nvinfer1::OptProfileSelector : int32_t { nvinfer1::kMIN = 0 , nvinfer1::kOPT = 1 , nvinfer1::kMAX = 2 }

  When setting or querying optimization profile parameters (such as shape tensor inputs or dynamic dimensions), select whether we are interested in the minimum, optimum, or maximum values for these parameters. The minimum and maximum specify the permitted range that is supported at runtime, while the optimum value is used for the kernel selection. This should be the "typical" value that is expected to occur at runtime.
- enum class nvinfer1::TacticSource : int32_t { nvinfer1::kCUBLAS = 0 , nvinfer1::kCUBLAS_LT = 1 , nvinfer1::kCUDNN = 2 , nvinfer1::kEDGE_MASK_CONVOLUTIONS = 3 , nvinfer1::kJIT_CONVOLUTIONS = 4 }

  List of tactic sources for TensorRT.
- enum class nvinfer1::ProfilingVerbosity : int32_t { nvinfer1::kLAYER_NAMES_ONLY = 0 , nvinfer1::kNONE = 1 , nvinfer1::kDETAILED = 2 }

  List of verbosity levels of layer information exposed in NVTX annotations and in IEngineInspector.
- enum class nvinfer1::SerializationFlag : int32_t { nvinfer1::kEXCLUDE_WEIGHTS = 0 , nvinfer1::kEXCLUDE_LEAN_RUNTIM = 1 }

  List of valid flags that the engine can enable when serializing the bytes.
- enum class nvinfer1::ExecutionContextAllocationStrategy : int32_t { nvinfer1::kSTATIC = 0 , nvinfer1::kON_PROFILE_CHANGE = 1 , nvinfer1::kUSER_MANAGED = 2 }

Different memory allocation behaviors for IExecutionContext.

- enum class nvinfer1::LayerInformationFormat : int32_t { nvinfer1::kONELINE = 0 , nvinfer1::kJSON = 1 }

  The format in which the IEngineInspector prints the layer information.

Functions

- template<> constexpr int32_t nvinfer1::EnumMax< DimensionOperation > () noexcept

  Maximum number of elements in DimensionOperation enum.
- template<> constexpr int32_t nvinfer1::EnumMax< WeightsRole > () noexcept

  Maximum number of elements in WeightsRole enum.
- template<> constexpr int32_t nvinfer1::EnumMax< DeviceType > () noexcept

  Maximum number of elements in DeviceType enum.
- template<> constexpr int32_t nvinfer1::EnumMax< TempfileControlFlag > () noexcept

  Maximum number of elements in TempfileControlFlag enum.
- template<> constexpr int32_t nvinfer1::EnumMax< OptProfileSelector > () noexcept

  Number of different values of OptProfileSelector enum.
- template<> constexpr int32_t nvinfer1::EnumMax< TacticSource > () noexcept

  Maximum number of tactic sources in TacticSource enum.
- template<> constexpr int32_t nvinfer1::EnumMax< ProfilingVerbosity > () noexcept

  Maximum number of profile verbosity levels in ProfilingVerbosity enum.
- template<> constexpr int32_t nvinfer1::EnumMax< SerializationFlag > () noexcept

  Maximum number of serialization flags in SerializationFlag enum.
- template<> constexpr int32_t nvinfer1::EnumMax< ExecutionContextAllocationStrategy > () noexcept

  Maximum number of memory allocation strategies in ExecutionContextAllocationStrategy 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.
- IRuntime * nvinfer1::anonymous_namespace{NvInferRuntime.h}::createInferRuntime (ILogger &logger) noexcept

  Create an instance of an IRuntime class.
- IRefitter * nvinfer1::anonymous_namespace{NvInferRuntime.h}::createInferRefitter (ICudaEngine &engine, ILogger &logger) noexcept

  Create an instance of an IRefitter class.
- int32_t getInferLibMajorVersion () noexcept

  Return the library major version number.
- int32_t getInferLibMinorVersion () noexcept

  Return the library minor version number.
- int32_t getInferLibPatchVersion () noexcept

  Return the library patch version number.
- int32_t getInferLibBuildVersion () noexcept

  Return the library build version number.

10.13.1 Detailed Description

This is the top-level API file for TensorRT extended runtime library.

10.13.2 Macro Definition Documentation

10.13.2.1 NV_INFER_INTERNAL_INCLUDE

```
#define NV_INFER_INTERNAL_INCLUDE 1
```

10.13.2.2 REGISTER_TENSORRT_PLUGIN

```
#define REGISTER_TENSORRT_PLUGIN(
             name )  static nvinfer1::PluginRegistrar<name> pluginRegistrar##name {}
```

10.13.3 Function Documentation

10.13.3.1 getInferLibBuildVersion()

```
int32_t getInferLibBuildVersion ( )  [noexcept]
```

Return the library build version number.

10.13.3.2 getInferLibMajorVersion()

```
int32_t getInferLibMajorVersion ( )  [noexcept]
```

Return the library major version number.

10.13.3.3 getInferLibMinorVersion()

```
int32_t getInferLibMinorVersion ( )  [noexcept]
```

Return the library minor version number.

10.13.3.4 getInferLibPatchVersion()

```
int32_t getInferLibPatchVersion ( )  [noexcept]
```

Return the library patch version number.

10.13.3.5 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.

10.13.3.6 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) 1993-2025 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: Apache-2.0
4  *
5  * Licensed under the Apache License, Version 2.0 (the "License");
6  * you may not use this file except in compliance with the License.
7  * You may obtain a copy of the License at
8  *
9  * http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in writing, software
12  * distributed under the License is distributed on an "AS IS" BASIS,
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14  * See the License for the specific language governing permissions and
15  * limitations under the License.
16  */
17
18 #ifndef NV_INFER_RUNTIME_H
19 #define NV_INFER_RUNTIME_H
20
26
27 #include "NvInferImpl.h"
28 #define NV_INFER_INTERNAL_INCLUDE 1
29 #include "NvInferPluginBase.h"
30 #undef NV_INFER_INTERNAL_INCLUDE
31 #include "NvInferRuntimeCommon.h"
32
```

```
33 namespace nvinfer1
34 {
35
36 class IExecutionContext;
37 class ICudaEngine;
38 class IPluginFactory;
39 class IEngineInspector;
40
49
50 class INoCopy
51 {
52 protected:
53     INoCopy() = default;
54     virtual ~INoCopy() = default;
55     INoCopy(INoCopy const& other) = delete;
56     INoCopy& operator=(INoCopy const& other) = delete;
57     INoCopy(INoCopy&& other) = delete;
58     INoCopy& operator=(INoCopy&& other) = delete;
59 };
60
75 enum class EngineCapability : int32_t
76 {
81     kSTANDARD = 0,
82
89     kSAFETY = 1,
90
96     kDLA_STANDALONE = 2,
97 };
98
99 namespace impl
100 {
102 template <>
103 struct EnumMaxImpl<EngineCapability>
104 {
105     static constexpr int32_t kVALUE = 3;
106 };
107 } // namespace impl
108
123 class Weights
124 {
125 public:
126     DataType type;
127     void const* values;
128     int64_t count;
129 };
130
141 class IHostMemory : public INoCopy
142 {
143 public:
144     virtual ~IHostMemory() noexcept = default;
145
147     void* data() const noexcept
148     {
149         return mImpl->data();
150     }
151
153     std::size_t size() const noexcept
154     {
155         return mImpl->size();
156     }
157
159     DataType type() const noexcept
160     {
161         return mImpl->type();
162     }
163
164 protected:
165     apiv::VHostMemory* mImpl;
166 };
167
178 enum class DimensionOperation : int32_t
179 {
180     kSUM = 0,
181     kPROD = 1,
182     kMAX = 2,
183     kMIN = 3,
184     kSUB = 4,
185     kEQUAL = 5,
186     kLESS = 6,
187     kFLOOR_DIV = 7,
188     kCEIL_DIV = 8
```

```
189 };
190
192 template <>
193 constexpr inline int32_t EnumMax<DimensionOperation>() noexcept
194 {
195     return 9;
196 }
197
203 enum class TensorLocation : int32_t
204 {
205     kDEVICE = 0,
206     kHOST = 1,
207 };
208
209 namespace impl
210 {
212 template <>
213 struct EnumMaxImpl<TensorLocation>
214 {
215     static constexpr int32_t kVALUE = 2;
216 };
217 } // namespace impl
218
231 class IDimensionExpr : public INoCopy
232 {
233 public:
237     bool isConstant() const noexcept
238     {
239         return mImpl->isConstant();
240     }
241
248     int64_t getConstantValue() const noexcept
249     {
250         return mImpl->getConstantValue();
251     }
252
253 protected:
254     apiv::VDimensionExpr* mImpl;
255     virtual ~IDimensionExpr() noexcept = default;
256
257 public:
263     bool isSizeTensor() const noexcept
264     {
265         return mImpl->isSizeTensor();
266     }
267 };
268
286 class IExprBuilder : public INoCopy
287 {
288 public:
292     IDimensionExpr const* constant(int64_t value) noexcept
293     {
294         return mImpl->constant(value);
295     }
296
303     IDimensionExpr const* operation(
304         DimensionOperation op, IDimensionExpr const& first, IDimensionExpr const& second) noexcept
305     {
306         return mImpl->operation(op, first, second);
307     }
308
309 protected:
310     apiv::VExprBuilder* mImpl;
311     virtual ~IExprBuilder() noexcept = default;
312
313 public:
338     IDimensionExpr const* declareSizeTensor(int32_t outputIndex, IDimensionExpr const& opt, IDimensionExpr
    const& upper)
339     {
340         return mImpl->declareSizeTensor(outputIndex, opt, upper);
341     }
342 };
343
349 class DimsExprs
350 {
351 public:
352     int32_t nbDims;
353     IDimensionExpr const* d[Dims::MAX_DIMS];
354 };
355
361 struct DynamicPluginTensorDesc
```

```
362 {
364     PluginTensorDesc desc;
365
367     Dims min;
368
370     Dims max;
371
373     Dims opt;
374 };
375
406 class TRT_DEPRECATED IPluginV2DynamicExt : public nvinfer1::IPluginV2Ext
407 {
408 public:
409     IPluginV2DynamicExt* clone() const noexcept override = 0;
410
435     virtual DimsExprs getOutputDimensions(
436         int32_t outputIndex, DimsExprs const* inputs, int32_t nbInputs, IExprBuilder& exprBuilder) noexcept =
    0;
437
441     static constexpr int32_t kFORMAT_COMBINATION_LIMIT = 100;
442
475     virtual bool supportsFormatCombination(
476         int32_t pos, PluginTensorDesc const* inOut, int32_t nbInputs, int32_t nbOutputs) noexcept = 0;
477
515     virtual void configurePlugin(DynamicPluginTensorDesc const* in, int32_t nbInputs,
516         DynamicPluginTensorDesc const* out, int32_t nbOutputs) noexcept = 0;
517
527     virtual size_t getWorkspaceSize(PluginTensorDesc const* inputs, int32_t nbInputs, PluginTensorDesc const*
    outputs,
528         int32_t nbOutputs) const noexcept = 0;
529
542     virtual int32_t enqueue(PluginTensorDesc const* inputDesc, PluginTensorDesc const* outputDesc,
543         void const* const* inputs, void* const* outputs, void* workspace, cudaStream_t stream) noexcept = 0;
544
545 protected:
553     int32_t getTensorRTVersion() const noexcept override
554     {
555         return (static_cast<int32_t>(PluginVersion::kV2_DYNAMICEXT) << 24 | (NV_TENSORRT_VERSION & 0xFFFFFF));
556     }
557
558     virtual ~IPluginV2DynamicExt() noexcept {}
559
560 private:
561     // Following are obsolete base class methods, and must not be implemented or used.
562
566     void configurePlugin(Dims const*, int32_t, Dims const*, int32_t, DataType const*, DataType const*, bool
    const*,
567         bool const*, PluginFormat, int32_t) noexcept override final
568     {
569     }
570
574     bool supportsFormat(DataType, PluginFormat) const noexcept override final
575     {
576         return false;
577     }
578
582     Dims getOutputDimensions(int32_t, Dims const*, int32_t) noexcept override final
583     {
584         return Dims{-1, {}};
585     }
586
594     TRT_DEPRECATED bool isOutputBroadcastAcrossBatch(int32_t, bool const*, int32_t) const noexcept override
    final
595     {
596         return false;
597     }
598
606     TRT_DEPRECATED bool canBroadcastInputAcrossBatch(int32_t) const noexcept override final
607     {
608         return true;
609     }
610
614     size_t getWorkspaceSize(int32_t) const noexcept override final
615     {
616         return 0;
617     }
618
622     int32_t enqueue(int32_t, void const* const*, void* const*, void*, cudaStream_t) noexcept override final
623     {
624         return 1;
625     }
```

```
626 };
627
628 namespace v_1_0
629 {
630 class IStreamReader : public IVersionedInterface
631 {
632 public:
637     ~IStreamReader() override = default;
638     IStreamReader() = default;
639
643     InterfaceInfo getInterfaceInfo() const noexcept override
644     {
645         return InterfaceInfo{"IStreamReader", 1, 0};
646     }
647
656     virtual int64_t read(void* destination, int64_t nbBytes) = 0;
657
658 protected:
659     IStreamReader(IStreamReader const&) = default;
660     IStreamReader(IStreamReader&&) = default;
661     IStreamReader& operator=(IStreamReader const&) & = default;
662     IStreamReader& operator=(IStreamReader&&) & = default;
663 };
664 } // namespace v_1_0
665
674 using IStreamReader = v_1_0::IStreamReader;
675
680 enum class SeekPosition : int32_t
681 {
683     kSET = 0,
684
686     kCUR = 1,
687
689     kEND = 2,
690 };
691
692 namespace v_1_0
693 {
694 class IStreamReaderV2 : public IVersionedInterface
695 {
696 public:
701     ~IStreamReaderV2() override = default;
702     IStreamReaderV2() = default;
703
707     InterfaceInfo getInterfaceInfo() const noexcept override
708     {
709         return InterfaceInfo{"IStreamReaderV2", 1, 0};
710     }
711
722     virtual int64_t read(void* destination, int64_t nbBytes, cudaStream_t stream) noexcept = 0;
723
732     virtual bool seek(int64_t offset, SeekPosition where) noexcept = 0;
733
734 protected:
735     IStreamReaderV2(IStreamReaderV2 const&) = default;
736     IStreamReaderV2(IStreamReaderV2&&) = default;
737     IStreamReaderV2& operator=(IStreamReaderV2 const&) & = default;
738     IStreamReaderV2& operator=(IStreamReaderV2&&) & = default;
739 };
740 } // namespace v_1_0
741
751 using IStreamReaderV2 = v_1_0::IStreamReaderV2;
752
766 class IPluginResourceContext
767 {
768 public:
773     virtual IGpuAllocator* getGpuAllocator() const noexcept = 0;
774
779     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
780     virtual ~IPluginResourceContext() noexcept = default;
781
782 protected:
783     IPluginResourceContext() = default;
784     IPluginResourceContext(IPluginResourceContext const&) = default;
785     IPluginResourceContext(IPluginResourceContext&&) = default;
786     IPluginResourceContext& operator=(IPluginResourceContext const&) & = default;
787     IPluginResourceContext& operator=(IPluginResourceContext&&) & = default;
788 };
789
790 namespace v_1_0
791 {
```

```
792 class IPluginV3OneCore : public IPluginCapability
793 {
794 public:
798     InterfaceInfo getInterfaceInfo() const noexcept override
799     {
800         return InterfaceInfo{"PLUGIN_V3ONE_CORE", 1, 0};
801     }
802
811     virtual AsciiChar const* getPluginName() const noexcept = 0;
812
821     virtual AsciiChar const* getPluginVersion() const noexcept = 0;
822
832     virtual AsciiChar const* getPluginNamespace() const noexcept = 0;
833 };
834
835 class IPluginV3OneBuild : public IPluginCapability
836 {
837 public:
843     static constexpr int32_t kDEFAULT_FORMAT_COMBINATION_LIMIT = 100;
844
848     InterfaceInfo getInterfaceInfo() const noexcept override
849     {
850         return InterfaceInfo{"PLUGIN_V3ONE_BUILD", 1, 0};
851     }
852
872     virtual int32_t configurePlugin(DynamicPluginTensorDesc const* in, int32_t nbInputs,
873         DynamicPluginTensorDesc const* out, int32_t nbOutputs) noexcept = 0;
874
891     virtual int32_t getOutputDataTypes(
892         DataType* outputTypes, int32_t nbOutputs, const DataType* inputTypes, int32_t nbInputs) const
      noexcept = 0;
893
915     virtual int32_t getOutputShapes(DimsExprs const* inputs, int32_t nbInputs, DimsExprs const* shapeInputs,
916         int32_t nbShapeInputs, DimsExprs* outputs, int32_t nbOutputs, IExprBuilder& exprBuilder) noexcept =
      0;
917
952     virtual bool supportsFormatCombination(
953         int32_t pos, DynamicPluginTensorDesc const* inOut, int32_t nbInputs, int32_t nbOutputs) noexcept = 0;
954
960     virtual int32_t getNbOutputs() const noexcept = 0;
961
971     virtual size_t getWorkspaceSize(DynamicPluginTensorDesc const* inputs, int32_t nbInputs,
972         DynamicPluginTensorDesc const* outputs, int32_t nbOutputs) const noexcept
973     {
974         return 0;
975     }
976
1008     virtual int32_t getValidTactics(int32_t* tactics, int32_t nbTactics) noexcept
1009     {
1010         return 0;
1011     }
1012
1016     virtual int32_t getNbTactics() noexcept
1017     {
1018         return 0;
1019     }
1020
1032     virtual char const* getTimingCacheID() noexcept
1033     {
1034         return nullptr;
1035     }
1036
1040     virtual int32_t getFormatCombinationLimit() noexcept
1041     {
1042         return kDEFAULT_FORMAT_COMBINATION_LIMIT;
1043     }
1044
1051     virtual char const* getMetadataString() noexcept
1052     {
1053         return nullptr;
1054     }
1055 };
1056
1057 class IPluginV3OneRuntime : public IPluginCapability
1058 {
1059 public:
1063     InterfaceInfo getInterfaceInfo() const noexcept override
1064     {
1065         return InterfaceInfo{"PLUGIN_V3ONE_RUNTIME", 1, 0};
1066     }
1067
```

```
1075     virtual int32_t setTactic(int32_t tactic) noexcept
1076     {
1077         return 0;
1078     }
1079
1098     virtual int32_t onShapeChange(
1099         PluginTensorDesc const* in, int32_t nbInputs, PluginTensorDesc const* out, int32_t nbOutputs)
       noexcept = 0;
1100
1114     virtual int32_t enqueue(PluginTensorDesc const* inputDesc, PluginTensorDesc const* outputDesc,
1115         void const* const* inputs, void* const* outputs, void* workspace, cudaStream_t stream) noexcept =
       0;
1116
1136     virtual IPluginV3* attachToContext(IPluginResourceContext* context) noexcept = 0;
1137
1143
1147     virtual PluginFieldCollection const* getFieldsToSerialize() noexcept = 0;
1148 };
1149 } // namespace v_1_0
1150
1151 namespace v_2_0
1152 {
1153
1154 class IPluginV3OneBuild : public v_1_0::IPluginV3OneBuild
1155 {
1156 public:
1157     InterfaceInfo getInterfaceInfo() const noexcept override
1158     {
1159         return InterfaceInfo{"PLUGIN_V3ONE_BUILD", 2, 0};
1160     }
1161
1191     virtual int32_t getAliasedInput(int32_t outputIndex) noexcept
1192     {
1193         return -1;
1194     }
1195 };
1196
1197 } // namespace v_2_0
1198
1208 using IPluginV3OneCore = v_1_0::IPluginV3OneCore;
1209
1220 using IPluginV3OneBuild = v_1_0::IPluginV3OneBuild;
1221
1232 using IPluginV3OneRuntime = v_1_0::IPluginV3OneRuntime;
1233
1241 using IPluginV3OneBuildV2 = v_2_0::IPluginV3OneBuild;
1242
1243 namespace v_1_0
1244 {
1245 class IProfiler
1246 {
1247 public:
1255     virtual void reportLayerTime(char const* layerName, float ms) noexcept = 0;
1256
1257     virtual ~IProfiler() noexcept {}
1258 };
1259 } // namespace v_1_0
1260
1272 using IProfiler = v_1_0::IProfiler;
1273
1281 enum class WeightsRole : int32_t
1282 {
1283     kKERNEL = 0,
1284     kBIAS = 1,
1285     kSHIFT = 2,
1286     kSCALE = 3,
1287     kCONSTANT = 4,
1288     kANY = 5,
1289 };
1290
1292 template <>
1293 constexpr inline int32_t EnumMax<WeightsRole>() noexcept
1294 {
1295     return 6;
1296 }
1297
1303 enum class DeviceType : int32_t
1304 {
1305     kGPU = 0,
1306     kDLA = 1,
1307 };
```

```
1308
1310 template <>
1311 constexpr inline int32_t EnumMax<DeviceType>() noexcept
1312 {
1313     return 2;
1314 }
1315
1326 enum class TempfileControlFlag : int32_t
1327 {
1329     kALLOW_IN_MEMORY_FILES = 0,
1330
1334     kALLOW_TEMPORARY_FILES = 1,
1335 };
1336
1338 template <>
1339 constexpr inline int32_t EnumMax<TempfileControlFlag>() noexcept
1340 {
1341     return 2;
1342 }
1343
1350 using TempfileControlFlags = uint32_t;
1351
1382 enum class TensorFormat : int32_t
1383 {
1390     kLINEAR = 0,
1391
1396     kCHW2 = 1,
1397
1401     kHWC8 = 2,
1402
1416     kCHW4 = 3,
1417
1425     kCHW16 = 4,
1426
1434     kCHW32 = 5,
1435
1440     kDHWC8 = 6,
1441
1446     kCDHW32 = 7,
1447
1451     kHWC = 8,
1452
1461     kDLA_LINEAR = 9,
1462
1476     kDLA_HWC4 = 10,
1477
1482     kHWC16 = 11,
1483
1488     kDHWC = 12
1489 };
1490
1491 namespace impl
1492 {
1494 template <>
1495 struct EnumMaxImpl<TensorFormat>
1496 {
1498     static constexpr int32_t kVALUE = 13;
1499 };
1500 } // namespace impl
1501
1507 enum class AllocatorFlag : int32_t
1508 {
1510     kRESIZABLE = 0,
1511 };
1512
1513 namespace impl
1514 {
1516 template <>
1517 struct EnumMaxImpl<AllocatorFlag>
1518 {
1520     static constexpr int32_t kVALUE = 1;
1521 };
1522 } // namespace impl
1523
1524 using AllocatorFlags = uint32_t;
1525
1528
1541 class ILogger
1542 {
1543 public:
1549     enum class Severity : int32_t
```

```
1550     {
1552         kINTERNAL_ERROR = 0,
1554         kERROR = 1,
1556         kWARNING = 2,
1558         kINFO = 3,
1560         kVERBOSE = 4,
1561     };
1562
1581     virtual void log(Severity severity, AsciiChar const* msg) noexcept = 0;
1582
1583     ILogger() = default;
1584     virtual ~ILogger() = default;
1585
1586 protected:
1587     // @cond SuppressDoxyWarnings
1588     ILogger(ILogger const&) = default;
1589     ILogger(ILogger&&) = default;
1590     ILogger& operator=(ILogger const&) & = default;
1591     ILogger& operator=(ILogger&&) & = default;
1592     // @endcond
1593 };
1594
1595 namespace impl
1596 {
1598 template <>
1599 struct EnumMaxImpl<ILogger::Severity>
1600 {
1602     static constexpr int32_t kVALUE = 5;
1603 };
1604 } // namespace impl
1605
1606 namespace v_1_0
1607 {
1608
1609 class IGpuAllocator : public IVersionedInterface
1610 {
1611 public:
1636     TRT_DEPRECATED virtual void* allocate(
1637         uint64_t const size, uint64_t const alignment, AllocatorFlags const flags) noexcept = 0;
1638
1639     ~IGpuAllocator() override = default;
1640     IGpuAllocator() = default;
1641
1679     virtual void* reallocate(void* const /*baseAddr*/, uint64_t /*alignment*/, uint64_t /*newSize*/)
      noexcept
1680     {
1681         return nullptr;
1682     }
1683
1702     TRT_DEPRECATED virtual bool deallocate(void* const memory) noexcept = 0;
1703
1732     virtual void* allocateAsync(
1733         uint64_t const size, uint64_t const alignment, AllocatorFlags const flags, cudaStream_t /*stream*/)
      noexcept
1734     {
1735         return allocate(size, alignment, flags);
1736     }
1765     virtual bool deallocateAsync(void* const memory, cudaStream_t /*stream*/) noexcept
1766     {
1767         return deallocate(memory);
1768     }
1769
1773     InterfaceInfo getInterfaceInfo() const noexcept override
1774     {
1775         return {"IGpuAllocator", 1, 0};
1776     }
1777
1778 protected:
1779     // @cond SuppressDoxyWarnings
1780     IGpuAllocator(IGpuAllocator const&) = default;
1781     IGpuAllocator(IGpuAllocator&&) = default;
1782     IGpuAllocator& operator=(IGpuAllocator const&) & = default;
1783     IGpuAllocator& operator=(IGpuAllocator&&) & = default;
1784     // @endcond
1785 };
1786
1787 } // namespace v_1_0
1788
1809 using IGpuAllocator = v_1_0::IGpuAllocator;
1810
1818 class IRuntime : public INoCopy
```

```
1819 {
1820 public:
1821     virtual ~IRuntime() noexcept = default;
1822 
1834     void setDLACore(int32_t dlaCore) noexcept
1835     {
1836         mImpl->setDLACore(dlaCore);
1837     }
1838 
1844     int32_t getDLACore() const noexcept
1845     {
1846         return mImpl->getDLACore();
1847     }
1848 
1852     int32_t getNbDLACores() const noexcept
1853     {
1854         return mImpl->getNbDLACores();
1855     }
1856 
1868     void setGpuAllocator(IGpuAllocator* allocator) noexcept
1869     {
1870         mImpl->setGpuAllocator(allocator);
1871     }
1872 
1884     //
1887     void setErrorRecorder(IErrorRecorder* recorder) noexcept
1888     {
1889         mImpl->setErrorRecorder(recorder);
1890     }
1891 
1902     IErrorRecorder* getErrorRecorder() const noexcept
1903     {
1904         return mImpl->getErrorRecorder();
1905     }
1906 
1917     ICudaEngine* deserializeCudaEngine(void const* blob, std::size_t size) noexcept
1918     {
1919         return mImpl->deserializeCudaEngine(blob, size);
1920     }
1921 
1938     TRT_DEPRECATED ICudaEngine* deserializeCudaEngine(IStreamReader& streamReader)
1939     {
1940         return mImpl->deserializeCudaEngine(streamReader);
1941     }
1942 
1958     ICudaEngine* deserializeCudaEngine(IStreamReaderV2& streamReader)
1959     {
1960         return mImpl->deserializeCudaEngineV2(streamReader);
1961     }
1962 
1968     ILogger* getLogger() const noexcept
1969     {
1970         return mImpl->getLogger();
1971     }
1972 
1983     bool setMaxThreads(int32_t maxThreads) noexcept
1984     {
1985         return mImpl->setMaxThreads(maxThreads);
1986     }
1987 
1997     int32_t getMaxThreads() const noexcept
1998     {
1999         return mImpl->getMaxThreads();
2000     }
2001 
2032     void setTemporaryDirectory(char const* path) noexcept
2033     {
2034         return mImpl->setTemporaryDirectory(path);
2035     }
2036 
2043     char const* getTemporaryDirectory() const noexcept
2044     {
2045         return mImpl->getTemporaryDirectory();
2046     }
2047 
2059     void setTempfileControlFlags(TempfileControlFlags flags) noexcept
2060     {
2061         return mImpl->setTempfileControlFlags(flags);
2062     }
2063 
2071     TempfileControlFlags getTempfileControlFlags() const noexcept
```

```
2072     {
2073         return mImpl->getTempfileControlFlags();
2074     }
2075
2081     IPluginRegistry& getPluginRegistry() noexcept
2082     {
2083         return mImpl->getPluginRegistry();
2084     }
2085
2099     IRuntime* loadRuntime(char const* path) noexcept
2100     {
2101         return mImpl->loadRuntime(path);
2102     }
2103
2111     void setEngineHostCodeAllowed(bool allowed) noexcept
2112     {
2113         return mImpl->setEngineHostCodeAllowed(allowed);
2114     }
2115
2121     bool getEngineHostCodeAllowed() const noexcept
2122     {
2123         return mImpl->getEngineHostCodeAllowed();
2124     }
2125
2126 protected:
2127     apiv::VRuntime* mImpl;
2128 };
2129
2137 class IRefitter : public INoCopy
2138 {
2139 public:
2140     virtual ~IRefitter() noexcept = default;
2141
2157     bool setWeights(char const* layerName, WeightsRole role, Weights weights) noexcept
2158     {
2159         return mImpl->setWeights(layerName, role, weights);
2160     }
2161
2174     bool refitCudaEngine() noexcept
2175     {
2176         return mImpl->refitCudaEngine();
2177     }
2178
2195     int32_t getMissing(int32_t size, char const** layerNames, WeightsRole* roles) noexcept
2196     {
2197         return mImpl->getMissing(size, layerNames, roles);
2198     }
2199
2212     int32_t getAll(int32_t size, char const** layerNames, WeightsRole* roles) noexcept
2213     {
2214         return mImpl->getAll(size, layerNames, roles);
2215     }
2216
2234     TRT_DEPRECATED bool setDynamicRange(char const* tensorName, float min, float max) noexcept
2235     {
2236         return mImpl->setDynamicRange(tensorName, min, max);
2237     }
2238
2250     TRT_DEPRECATED float getDynamicRangeMin(char const* tensorName) const noexcept
2251     {
2252         return mImpl->getDynamicRangeMin(tensorName);
2253     }
2254
2266     TRT_DEPRECATED float getDynamicRangeMax(char const* tensorName) const noexcept
2267     {
2268         return mImpl->getDynamicRangeMax(tensorName);
2269     }
2270
2284     TRT_DEPRECATED int32_t getTensorsWithDynamicRange(int32_t size, char const** tensorNames) const noexcept
2285     {
2286         return mImpl->getTensorsWithDynamicRange(size, tensorNames);
2287     }
2288
2300     //
2303     void setErrorRecorder(IErrorRecorder* recorder) noexcept
2304     {
2305         mImpl->setErrorRecorder(recorder);
2306     }
2307
2318     IErrorRecorder* getErrorRecorder() const noexcept
2319     {
```

```
2320         return mImpl->getErrorRecorder();
2321     }
2322
2343     bool setNamedWeights(char const* name, Weights weights) noexcept
2344     {
2345         return mImpl->setNamedWeights(name, weights);
2346     }
2347
2363     int32_t getMissingWeights(int32_t size, char const** weightsNames) noexcept
2364     {
2365         return mImpl->getMissingWeights(size, weightsNames);
2366     }
2367
2379     int32_t getAllWeights(int32_t size, char const** weightsNames) noexcept
2380     {
2381         return mImpl->getAllWeights(size, weightsNames);
2382     }
2383
2389     ILogger* getLogger() const noexcept
2390     {
2391         return mImpl->getLogger();
2392     }
2393
2405     bool setMaxThreads(int32_t maxThreads) noexcept
2406     {
2407         return mImpl->setMaxThreads(maxThreads);
2408     }
2409
2419     int32_t getMaxThreads() const noexcept
2420     {
2421         return mImpl->getMaxThreads();
2422     }
2423
2446     bool setNamedWeights(char const* name, Weights weights, TensorLocation location) noexcept
2447     {
2448         return mImpl->setNamedWeightsWithLocation(name, weights, location);
2449     }
2450
2462     Weights getNamedWeights(char const* weightsName) const noexcept
2463     {
2464         return mImpl->getNamedWeights(weightsName);
2465     }
2466
2478     TensorLocation getWeightsLocation(char const* weightsName) const noexcept
2479     {
2480         return mImpl->getWeightsLocation(weightsName);
2481     }
2482
2494     bool unsetNamedWeights(char const* weightsName) noexcept
2495     {
2496         return mImpl->unsetNamedWeights(weightsName);
2497     }
2498
2510     void setWeightsValidation(bool weightsValidation) noexcept
2511     {
2512         return mImpl->setWeightsValidation(weightsValidation);
2513     }
2514
2518     bool getWeightsValidation() const noexcept
2519     {
2520         return mImpl->getWeightsValidation();
2521     }
2522
2540     bool refitCudaEngineAsync(cudaStream_t stream) noexcept
2541     {
2542         return mImpl->refitCudaEngineAsync(stream);
2543     }
2544
2558     Weights getWeightsPrototype(char const* weightsName) const noexcept
2559     {
2560         return mImpl->getWeightsPrototype(weightsName);
2561     }
2562
2563 protected:
2564     apiv::VRefitter* mImpl;
2565 };
2566
2577 enum class OptProfileSelector : int32_t
2578 {
2579     kMIN = 0,
2580     kOPT = 1,
```

```
2581     kMAX = 2
2582 };
2583
2589 template <>
2590 constexpr inline int32_t EnumMax<OptProfileSelector>() noexcept
2591 {
2592     return 3;
2593 }
2594
2617 class IOptimizationProfile : public INoCopy
2618 {
2619 public:
2647     bool setDimensions(char const* inputName, OptProfileSelector select, Dims const& dims) noexcept
2648     {
2649         return mImpl->setDimensions(inputName, select, dims);
2650     }
2651
2659     Dims getDimensions(char const* inputName, OptProfileSelector select) const noexcept
2660     {
2661         return mImpl->getDimensions(inputName, select);
2662     }
2663
2706     bool setShapeValues(
2707         char const* inputName, OptProfileSelector select, int32_t const* values, int32_t nbValues) noexcept
2708     {
2709         return mImpl->setShapeValues(inputName, select, values, nbValues);
2710     }
2711
2720     int32_t getNbShapeValues(char const* inputName) const noexcept
2721     {
2722         return mImpl->getNbShapeValues(inputName);
2723     }
2724
2732     int32_t const* getShapeValues(char const* inputName, OptProfileSelector select) const noexcept
2733     {
2734         return mImpl->getShapeValues(inputName, select);
2735     }
2736
2750     bool setExtraMemoryTarget(float target) noexcept
2751     {
2752         return mImpl->setExtraMemoryTarget(target);
2753     }
2754
2762     float getExtraMemoryTarget() const noexcept
2763     {
2764         return mImpl->getExtraMemoryTarget();
2765     }
2766
2779     bool isValid() const noexcept
2780     {
2781         return mImpl->isValid();
2782     }
2783
2784 protected:
2785     apiv::VOptimizationProfile* mImpl;
2786     virtual ~IOptimizationProfile() noexcept = default;
2787 };
2788
2796 enum class TacticSource : int32_t
2797 {
2801     kCUBLAS TRT_DEPRECATED_ENUM = 0,
2802
2805     kCUBLAS_LT TRT_DEPRECATED_ENUM = 1,
2806
2810     kCUDNN TRT_DEPRECATED_ENUM = 2,
2811
2815     kEDGE_MASK_CONVOLUTIONS = 3,
2816
2819     kJIT_CONVOLUTIONS = 4,
2820 };
2821
2822 template <>
2823 constexpr inline int32_t EnumMax<TacticSource>() noexcept
2824 {
2825     return 5;
2826 }
2827
2834 using TacticSources = uint32_t;
2835
2845 enum class ProfilingVerbosity : int32_t
2846 {
```

```
2847     kLAYER_NAMES_ONLY = 0,
2848     kNONE = 1,
2849     kDETAILED = 2,
2850 };
2851
2853 template <>
2854 constexpr inline int32_t EnumMax<ProfilingVerbosity>() noexcept
2855 {
2856     return 3;
2857 }
2858
2865 using SerializationFlags = uint32_t;
2866
2874 enum class SerializationFlag : int32_t
2875 {
2876     kEXCLUDE_WEIGHTS = 0,
2877     kEXCLUDE_LEAN_RUNTIME = 1,
2878 };
2879
2881 template <>
2882 constexpr inline int32_t EnumMax<SerializationFlag>() noexcept
2883 {
2884     return 2;
2885 }
2886
2894 class ISerializationConfig : public INoCopy
2895 {
2896 public:
2897     virtual ~ISerializationConfig() noexcept = default;
2898
2910     bool setFlags(SerializationFlags serializationFlags) noexcept
2911     {
2912         return mImpl->setFlags(serializationFlags);
2913     }
2914
2922     SerializationFlags getFlags() const noexcept
2923     {
2924         return mImpl->getFlags();
2925     }
2926
2934     bool clearFlag(SerializationFlag serializationFlag) noexcept
2935     {
2936         return mImpl->clearFlag(serializationFlag);
2937     }
2938
2946     bool setFlag(SerializationFlag serializationFlag) noexcept
2947     {
2948         return mImpl->setFlag(serializationFlag);
2949     }
2950
2958     bool getFlag(SerializationFlag serializationFlag) const noexcept
2959     {
2960         return mImpl->getFlag(serializationFlag);
2961     }
2962
2963 protected:
2964     apiv::VSerializationConfig* mImpl;
2965 };
2966
2978 enum class ExecutionContextAllocationStrategy : int32_t
2979 {
2980     kSTATIC = 0,
2981     kON_PROFILE_CHANGE = 1,
2982     kUSER_MANAGED = 2,
2983 };
2984
2990 template <>
2991 constexpr inline int32_t EnumMax<ExecutionContextAllocationStrategy>() noexcept
2992 {
2993     return 3;
2994 }
2995
3003 class ICudaEngine : public INoCopy
3004 {
3005 public:
3006     virtual ~ICudaEngine() noexcept = default;
3007
3018     Dims getTensorShape(char const* tensorName) const noexcept
3019     {
3020         return mImpl->getTensorShape(tensorName);
3021     }
```

```
3022
3033     DataType getTensorDataType(char const* tensorName) const noexcept
3034     {
3035         return mImpl->getTensorDataType(tensorName);
3036     }
3037
3047     int32_t getNbLayers() const noexcept
3048     {
3049         return mImpl->getNbLayers();
3050     }
3051
3061     IHostMemory* serialize() const noexcept
3062     {
3063         return mImpl->serialize();
3064     }
3065
3078     IExecutionContext* createExecutionContext(
3079         ExecutionContextAllocationStrategy strategy = ExecutionContextAllocationStrategy::kSTATIC) noexcept
3080     {
3081         return mImpl->createExecutionContext(strategy);
3082     }
3083
3096     TensorLocation getTensorLocation(char const* tensorName) const noexcept
3097     {
3098         return mImpl->getTensorLocation(tensorName);
3099     }
3100
3116     bool isShapeInferenceIO(char const* tensorName) const noexcept
3117     {
3118         return mImpl->isShapeInferenceIO(tensorName);
3119     }
3120
3130     TensorIOMode getTensorIOMode(char const* tensorName) const noexcept
3131     {
3132         return mImpl->getTensorIOMode(tensorName);
3133     }
3134
3142     TRT_DEPRECATED IExecutionContext* createExecutionContextWithoutDeviceMemory() noexcept
3143     {
3144         return mImpl->createExecutionContextWithoutDeviceMemory();
3145     }
3146
3154     TRT_DEPRECATED size_t getDeviceMemorySize() const noexcept
3155     {
3156         return mImpl->getDeviceMemorySize();
3157     }
3158
3166     TRT_DEPRECATED size_t getDeviceMemorySizeForProfile(int32_t profileIndex) const noexcept
3167     {
3168         return mImpl->getDeviceMemorySizeForProfile(profileIndex);
3169     }
3170
3182     int64_t getDeviceMemorySizeV2() const noexcept
3183     {
3184         return mImpl->getDeviceMemorySizeV2();
3185     }
3186
3198     int64_t getDeviceMemorySizeForProfileV2(int32_t profileIndex) const noexcept
3199     {
3200         return mImpl->getDeviceMemorySizeForProfileV2(profileIndex);
3201     }
3202
3208     bool isRefittable() const noexcept
3209     {
3210         return mImpl->isRefittable();
3211     }
3212
3229     int32_t getTensorBytesPerComponent(char const* tensorName) const noexcept
3230     {
3231         return mImpl->getTensorBytesPerComponent(tensorName);
3232     }
3233
3247     int32_t getTensorBytesPerComponent(char const* tensorName, int32_t profileIndex) const noexcept
3248     {
3249         return mImpl->getTensorBytesPerComponentV2(tensorName, profileIndex);
3250     }
3251
3268     int32_t getTensorComponentsPerElement(char const* tensorName) const noexcept
3269     {
3270         return mImpl->getTensorComponentsPerElement(tensorName);
3271     }
```

```
3272
3286     int32_t getTensorComponentsPerElement(char const* tensorName, int32_t profileIndex) const noexcept
3287     {
3288         return mImpl->getTensorComponentsPerElementV2(tensorName, profileIndex);
3289     }
3290
3301     TensorFormat getTensorFormat(char const* tensorName) const noexcept
3302     {
3303         return mImpl->getTensorFormat(tensorName);
3304     }
3305
3315     TensorFormat getTensorFormat(char const* tensorName, int32_t profileIndex) const noexcept
3316     {
3317         return mImpl->getTensorFormatV2(tensorName, profileIndex);
3318     }
3319
3339     char const* getTensorFormatDesc(char const* tensorName) const noexcept
3340     {
3341         return mImpl->getTensorFormatDesc(tensorName);
3342     }
3343
3362     char const* getTensorFormatDesc(char const* tensorName, int32_t profileIndex) const noexcept
3363     {
3364         return mImpl->getTensorFormatDescV2(tensorName, profileIndex);
3365     }
3366
3379     int32_t getTensorVectorizedDim(char const* tensorName) const noexcept
3380     {
3381         return mImpl->getTensorVectorizedDim(tensorName);
3382     }
3383
3395     int32_t getTensorVectorizedDim(char const* tensorName, int32_t profileIndex) const noexcept
3396     {
3397         return mImpl->getTensorVectorizedDimV2(tensorName, profileIndex);
3398     }
3399
3410     char const* getName() const noexcept
3411     {
3412         return mImpl->getName();
3413     }
3414
3421     int32_t getNbOptimizationProfiles() const noexcept
3422     {
3423         return mImpl->getNbOptimizationProfiles();
3424     }
3425
3441     Dims getProfileShape(char const* tensorName, int32_t profileIndex, OptProfileSelector select) const
    noexcept
3442     {
3443         return mImpl->getProfileShape(tensorName, profileIndex, select);
3444     }
3445
3463     int32_t const* getProfileTensorValues(char const* tensorName, int32_t profileIndex, OptProfileSelector
    select) const
3464         noexcept
3465     {
3466         return mImpl->getProfileTensorValues(tensorName, profileIndex, select);
3467     }
3468
3479     EngineCapability getEngineCapability() const noexcept
3480     {
3481         return mImpl->getEngineCapability();
3482     }
3483
3498     void setErrorRecorder(IErrorRecorder* recorder) noexcept
3499     {
3500         return mImpl->setErrorRecorder(recorder);
3501     }
3502
3513     IErrorRecorder* getErrorRecorder() const noexcept
3514     {
3515         return mImpl->getErrorRecorder();
3516     }
3517
3527     TRT_DEPRECATED bool hasImplicitBatchDimension() const noexcept
3528     {
3529         return mImpl->hasImplicitBatchDimension();
3530     }
3531
3543     TacticSources getTacticSources() const noexcept
3544     {
```

```
3545         return mImpl->getTacticSources();
3546     }
3547
3555     ProfilingVerbosity getProfilingVerbosity() const noexcept
3556     {
3557         return mImpl->getProfilingVerbosity();
3558     }
3559
3565     IEngineInspector* createEngineInspector() const noexcept
3566     {
3567         return mImpl->createEngineInspector();
3568     }
3569
3578     int32_t getNbIOTensors() const noexcept
3579     {
3580         return mImpl->getNbIOTensors();
3581     }
3582
3590     char const* getIOTensorName(int32_t index) const noexcept
3591     {
3592         return mImpl->getIOTensorName(index);
3593     }
3594
3601     HardwareCompatibilityLevel getHardwareCompatibilityLevel() const noexcept
3602     {
3603         return mImpl->getHardwareCompatibilityLevel();
3604     }
3605
3616     int32_t getNbAuxStreams() const noexcept
3617     {
3618         return mImpl->getNbAuxStreams();
3619     }
3620
3626     ISerializationConfig* createSerializationConfig() noexcept
3627     {
3628         return mImpl->createSerializationConfig();
3629     }
3630
3642     IHostMemory* serializeWithConfig(ISerializationConfig& config) const noexcept
3643     {
3644         return mImpl->serializeWithConfig(config);
3645     }
3646
3687     TRT_DEPRECATED bool setWeightStreamingBudget(int64_t gpuMemoryBudget) noexcept
3688     {
3689         return mImpl->setWeightStreamingBudget(gpuMemoryBudget);
3690     }
3691
3707     TRT_DEPRECATED int64_t getWeightStreamingBudget() const noexcept
3708     {
3709         return mImpl->getWeightStreamingBudget();
3710     }
3711
3730     TRT_DEPRECATED int64_t getMinimumWeightStreamingBudget() const noexcept
3731     {
3732         return mImpl->getMinimumWeightStreamingBudget();
3733     }
3734
3746     int64_t getStreamableWeightsSize() const noexcept
3747     {
3748         return mImpl->getStreamableWeightsSize();
3749     }
3750
3788     bool setWeightStreamingBudgetV2(int64_t gpuMemoryBudget) noexcept
3789     {
3790         return mImpl->setWeightStreamingBudgetV2(gpuMemoryBudget);
3791     }
3792
3806     int64_t getWeightStreamingBudgetV2() const noexcept
3807     {
3808         return mImpl->getWeightStreamingBudgetV2();
3809     }
3810
3831     int64_t getWeightStreamingAutomaticBudget() const noexcept
3832     {
3833         return mImpl->getWeightStreamingAutomaticBudget();
3834     }
3835
3859     int64_t getWeightStreamingScratchMemorySize() const noexcept
3860     {
3861         return mImpl->getWeightStreamingScratchMemorySize();
```

```
3862     }
3863
3873     bool isDebugTensor(char const* name) const noexcept
3874     {
3875         return mImpl->isDebugTensor(name);
3876     }
3877
3878 protected:
3879     apiv::VCudaEngine* mImpl;
3880 };
3881
3882 namespace v_1_0
3883 {
3884 class IOutputAllocator : public IVersionedInterface
3885 {
3886 public:
3890     InterfaceInfo getInterfaceInfo() const noexcept override
3891     {
3892         return {"IOutputAllocator", 1, 0};
3893     }
3894
3914     TRT_DEPRECATED virtual void* reallocateOutput(
3915         char const* tensorName, void* currentMemory, uint64_t size, uint64_t alignment) noexcept
3916     {
3917         return nullptr;
3918     }
3919
3942     virtual void* reallocateOutputAsync(
3943         char const* tensorName, void* currentMemory, uint64_t size, uint64_t alignment, cudaStream_t
    /*stream*/)
3944     {
3945         return reallocateOutput(tensorName, currentMemory, size, alignment);
3946     }
3947
3956     virtual void notifyShape(char const* tensorName, Dims const& dims) noexcept = 0;
3957 };
3958 } // namespace v_1_0
3959
3967 using IOutputAllocator = v_1_0::IOutputAllocator;
3968
3969 namespace v_1_0
3970 {
3971 class IDebugListener : public IVersionedInterface
3972 {
3973 public:
3977     InterfaceInfo getInterfaceInfo() const noexcept override
3978     {
3979         return {"IDebugListener", 1, 0};
3980     }
3981
3995     virtual bool processDebugTensor(void const* addr, TensorLocation location, DataType type, Dims const&
    shape,
3996         char const* name, cudaStream_t stream)
3997         = 0;
3998
3999     ~IDebugListener() override = default;
4000 };
4001 } // namespace v_1_0
4002
4008 using IDebugListener = v_1_0::IDebugListener;
4009
4020 class IExecutionContext : public INoCopy
4021 {
4022 public:
4023     virtual ~IExecutionContext() noexcept = default;
4024
4033     void setDebugSync(bool sync) noexcept
4034     {
4035         mImpl->setDebugSync(sync);
4036     }
4037
4043     bool getDebugSync() const noexcept
4044     {
4045         return mImpl->getDebugSync();
4046     }
4047
4053     void setProfiler(IProfiler* profiler) noexcept
4054     {
4055         mImpl->setProfiler(profiler);
4056     }
4057
```

```
4063     IProfiler* getProfiler() const noexcept
4064     {
4065         return mImpl->getProfiler();
4066     }
4067
4073     ICudaEngine const& getEngine() const noexcept
4074     {
4075         return mImpl->getEngine();
4076     }
4077
4087     void setName(char const* name) noexcept
4088     {
4089         mImpl->setName(name);
4090     }
4091
4097     char const* getName() const noexcept
4098     {
4099         return mImpl->getName();
4100     }
4101
4124     void setDeviceMemory(void* memory) noexcept
4125     {
4126         mImpl->setDeviceMemory(memory);
4127     }
4128
4146     void setDeviceMemoryV2(void* memory, int64_t size) noexcept
4147     {
4148         return mImpl->setDeviceMemoryV2(memory, size);
4149     }
4150
4167     Dims getTensorStrides(char const* tensorName) const noexcept
4168     {
4169         return mImpl->getTensorStrides(tensorName);
4170     }
4171
4172 public:
4182     int32_t getOptimizationProfile() const noexcept
4183     {
4184         return mImpl->getOptimizationProfile();
4185     }
4186
4200     bool setInputShape(char const* tensorName, Dims const& dims) noexcept
4201     {
4202         return mImpl->setInputShape(tensorName, dims);
4203     }
4204
4237     Dims getTensorShape(char const* tensorName) const noexcept
4238     {
4239         return mImpl->getTensorShape(tensorName);
4240     }
4241
4253     bool allInputDimensionsSpecified() const noexcept
4254     {
4255         return mImpl->allInputDimensionsSpecified();
4256     }
4257
4270     TRT_DEPRECATED bool allInputShapesSpecified() const noexcept
4271     {
4272         return mImpl->allInputShapesSpecified();
4273     }
4274
4289     void setErrorRecorder(IErrorRecorder* recorder) noexcept
4290     {
4291         mImpl->setErrorRecorder(recorder);
4292     }
4293
4304     IErrorRecorder* getErrorRecorder() const noexcept
4305     {
4306         return mImpl->getErrorRecorder();
4307     }
4308
4321     bool executeV2(void* const* bindings) noexcept
4322     {
4323         return mImpl->executeV2(bindings);
4324     }
4325
4365     bool setOptimizationProfileAsync(int32_t profileIndex, cudaStream_t stream) noexcept
4366     {
4367         return mImpl->setOptimizationProfileAsync(profileIndex, stream);
4368     }
4369
```

```
4381     void setEnqueueEmitsProfile(bool enqueueEmitsProfile) noexcept
4382     {
4383         mImpl->setEnqueueEmitsProfile(enqueueEmitsProfile);
4384     }
4385
4393     bool getEnqueueEmitsProfile() const noexcept
4394     {
4395         return mImpl->getEnqueueEmitsProfile();
4396     }
4397
4423     bool reportToProfiler() const noexcept
4424     {
4425         return mImpl->reportToProfiler();
4426     }
4427
4467     bool setTensorAddress(char const* tensorName, void* data) noexcept
4468     {
4469         return mImpl->setTensorAddress(tensorName, data);
4470     }
4471
4484     void const* getTensorAddress(char const* tensorName) const noexcept
4485     {
4486         return mImpl->getTensorAddress(tensorName);
4487     }
4488
4507     bool setOutputTensorAddress(char const* tensorName, void* data) noexcept
4508     {
4509         return mImpl->setOutputTensorAddress(tensorName, data);
4510     }
4511
4529     bool setInputTensorAddress(char const* tensorName, void const* data) noexcept
4530     {
4531         return mImpl->setInputTensorAddress(tensorName, data);
4532     }
4533
4548     void* getOutputTensorAddress(char const* tensorName) const noexcept
4549     {
4550         return mImpl->getOutputTensorAddress(tensorName);
4551     }
4552
4581     int32_t inferShapes(int32_t nbMaxNames, char const** tensorNames) noexcept
4582     {
4583         return mImpl->inferShapes(nbMaxNames, tensorNames);
4584     }
4585
4598     size_t updateDeviceMemorySizeForShapes() noexcept
4599     {
4600         return mImpl->updateDeviceMemorySizeForShapes();
4601     }
4602
4614     bool setInputConsumedEvent(cudaEvent_t event) noexcept
4615     {
4616         return mImpl->setInputConsumedEvent(event);
4617     }
4618
4624     cudaEvent_t getInputConsumedEvent() const noexcept
4625     {
4626         return mImpl->getInputConsumedEvent();
4627     }
4628
4643     bool setOutputAllocator(char const* tensorName, IOutputAllocator* outputAllocator) noexcept
4644     {
4645         return mImpl->setOutputAllocator(tensorName, outputAllocator);
4646     }
4647
4656     IOutputAllocator* getOutputAllocator(char const* tensorName) const noexcept
4657     {
4658         return mImpl->getOutputAllocator(tensorName);
4659     }
4660
4674     int64_t getMaxOutputSize(char const* tensorName) const noexcept
4675     {
4676         return mImpl->getMaxOutputSize(tensorName);
4677     }
4678
4694     bool setTemporaryStorageAllocator(IGpuAllocator* allocator) noexcept
4695     {
4696         return mImpl->setTemporaryStorageAllocator(allocator);
4697     }
4698
4704     IGpuAllocator* getTemporaryStorageAllocator() const noexcept
```

```
4705     {
4706         return mImpl->getTemporaryStorageAllocator();
4707     }
4708
4728     bool enqueueV3(cudaStream_t stream) noexcept
4729     {
4730         return mImpl->enqueueV3(stream);
4731     }
4732
4744     void setPersistentCacheLimit(size_t size) noexcept
4745     {
4746         mImpl->setPersistentCacheLimit(size);
4747     }
4748
4755     size_t getPersistentCacheLimit() const noexcept
4756     {
4757         return mImpl->getPersistentCacheLimit();
4758     }
4759
4779     bool setNvtxVerbosity(ProfilingVerbosity verbosity) noexcept
4780     {
4781         return mImpl->setNvtxVerbosity(verbosity);
4782     }
4783
4791     ProfilingVerbosity getNvtxVerbosity() const noexcept
4792     {
4793         return mImpl->getNvtxVerbosity();
4794     }
4795
4822     void setAuxStreams(cudaStream_t* auxStreams, int32_t nbStreams) noexcept
4823     {
4824         mImpl->setAuxStreams(auxStreams, nbStreams);
4825     }
4826
4834     bool setDebugListener(IDebugListener* listener) noexcept
4835     {
4836         return mImpl->setDebugListener(listener);
4837     }
4838
4844     IDebugListener* getDebugListener() noexcept
4845     {
4846         return mImpl->getDebugListener();
4847     }
4848
4863     bool setTensorDebugState(char const* name, bool flag) noexcept
4864     {
4865         return mImpl->setTensorDebugState(name, flag);
4866     }
4867
4876     bool setAllTensorsDebugState(bool flag) noexcept
4877     {
4878         return mImpl->setAllTensorsDebugState(flag);
4879     }
4880
4886     bool getDebugState(char const* name) const noexcept
4887     {
4888         return mImpl->getDebugState(name);
4889     }
4890
4891 protected:
4892     apiv::VExecutionContext* mImpl;
4893 }; // class IExecutionContext
4894
4902 enum class LayerInformationFormat : int32_t
4903 {
4904     kONELINE = 0,
4905     kJSON = 1,
4906 };
4907
4910 template <>
4911 constexpr inline int32_t EnumMax<LayerInformationFormat>() noexcept
4912 {
4913     return 2;
4914 }
4915
4931 class IEngineInspector : public INoCopy
4932 {
4933 public:
4934     virtual ~IEngineInspector() noexcept = default;
4935
4948     bool setExecutionContext(IExecutionContext const* context) noexcept
```

```
4949     {
4950         return mImpl->setExecutionContext(context);
4951     }
4952
4960     IExecutionContext const* getExecutionContext() const noexcept
4961     {
4962         return mImpl->getExecutionContext();
4963     }
4964
4985     char const* getLayerInformation(int32_t layerIndex, LayerInformationFormat format) const noexcept
4986     {
4987         return mImpl->getLayerInformation(layerIndex, format);
4988     }
4989
5008     char const* getEngineInformation(LayerInformationFormat format) const noexcept
5009     {
5010         return mImpl->getEngineInformation(format);
5011     }
5012
5027     void setErrorRecorder(IErrorRecorder* recorder) noexcept
5028     {
5029         mImpl->setErrorRecorder(recorder);
5030     }
5031
5042     IErrorRecorder* getErrorRecorder() const noexcept
5043     {
5044         return mImpl->getErrorRecorder();
5045     }
5046
5047 protected:
5048     apiv::VEngineInspector* mImpl;
5049 }; // class IEngineInspector
5050
5051 } // namespace nvinfer1
5052
5057 extern "C" TENSORRTAPI void* createInferRuntime_INTERNAL(void* logger, int32_t version) noexcept;
5058
5063 extern "C" TENSORRTAPI void* createInferRefitter_INTERNAL(void* engine, void* logger, int32_t version)
       noexcept;
5064
5068 extern "C" TENSORRTAPI nvinfer1::IPluginRegistry* getPluginRegistry() noexcept;
5069
5075 extern "C" TENSORRTAPI nvinfer1::ILogger* getLogger() noexcept;
5076
5077 namespace nvinfer1
5078 {
5079 namespace // unnamed namespace avoids linkage surprises when linking objects built with different versions
       of this
5080           // header.
5081 {
5087 inline IRuntime* createInferRuntime(ILogger& logger) noexcept
5088 {
5089     return static_cast<IRuntime*>(createInferRuntime_INTERNAL(&logger, NV_TENSORRT_VERSION));
5090 }
5091
5098 inline IRefitter* createInferRefitter(ICudaEngine& engine, ILogger& logger) noexcept
5099 {
5100     return static_cast<IRefitter*>(createInferRefitter_INTERNAL(&engine, &logger, NV_TENSORRT_VERSION));
5101 }
5102
5103 } // namespace
5104
5116 template <typename T>
5117 class PluginRegistrar
5118 {
5119 public:
5120     PluginRegistrar()
5121     {
5122         getPluginRegistry()->registerCreator(instance, "");
5123     }
5124
5125 private:
5127     T instance{};
5128 };
5129
5130 } // namespace nvinfer1
5131
5132 #define REGISTER_TENSORRT_PLUGIN(name)
                \
5133     static nvinfer1::PluginRegistrar<name> pluginRegistrar##name {}
5134
```

```
5135 namespace nvinfer1
5136 {
5145 class ILoggerFinder
5146 {
5147 public:
5155     virtual ILogger* findLogger() = 0;
5156
5157 protected:
5158     virtual ~ILoggerFinder() = default;
5159 };
5160
5163 namespace v_1_0
5164 {
5165
5166 class IGpuAsyncAllocator : public IGpuAllocator
5167 {
5168 public:
5169     IGpuAsyncAllocator() = default;
5170     ~IGpuAsyncAllocator() override = default;
5171
5201     void* allocateAsync(uint64_t const size, uint64_t const alignment, AllocatorFlags const flags,
5202         cudaStream_t /*stream*/) noexcept override = 0;
5203
5229     bool deallocateAsync(void* const memory, cudaStream_t /*stream*/) noexcept override = 0;
5230
5253     TRT_DEPRECATED void* allocate(
5254         uint64_t const size, uint64_t const alignment, AllocatorFlags const flags) noexcept override
5255     {
5256         return allocateAsync(size, alignment, flags, nullptr);
5257     }
5258
5277     TRT_DEPRECATED bool deallocate(void* const memory) noexcept override
5278     {
5279         return deallocateAsync(memory, nullptr);
5280     }
5281
5285     InterfaceInfo getInterfaceInfo() const noexcept override
5286     {
5287         return {"IGpuAllocator", 1, 0};
5288     }
5289 };
5290
5291 class IPluginCreatorV3One : public IPluginCreatorInterface
5292 {
5293 public:
5297     InterfaceInfo getInterfaceInfo() const noexcept override
5298     {
5299         return InterfaceInfo{"PLUGIN CREATOR_V3ONE", 1, 0};
5300     }
5301
5318     virtual IPluginV3* createPlugin(
5319         AsciiChar const* name, PluginFieldCollection const* fc, TensorRTPhase phase) noexcept = 0;
5320
5327     virtual PluginFieldCollection const* getFieldNames() noexcept = 0;
5328
5335     virtual AsciiChar const* getPluginName() const noexcept = 0;
5336
5343     virtual AsciiChar const* getPluginVersion() const noexcept = 0;
5344
5351     virtual AsciiChar const* getPluginNamespace() const noexcept = 0;
5352
5353     IPluginCreatorV3One() = default;
5354     virtual ~IPluginCreatorV3One() = default;
5355
5356 protected:
5357     IPluginCreatorV3One(IPluginCreatorV3One const&) = default;
5358     IPluginCreatorV3One(IPluginCreatorV3One&&) = default;
5359     IPluginCreatorV3One& operator=(IPluginCreatorV3One const&) & = default;
5360     IPluginCreatorV3One& operator=(IPluginCreatorV3One&&) & = default;
5361 };
5362
5363 } // namespace v_1_0
5364
5378 using IGpuAsyncAllocator = v_1_0::IGpuAsyncAllocator;
5379
5388 using IPluginCreatorV3One = v_1_0::IPluginCreatorV3One;
5389
5390 } // namespace nvinfer1
5391
5395 extern "C" TENSORRTAPI int32_t getInferLibMajorVersion() noexcept;
5399 extern "C" TENSORRTAPI int32_t getInferLibMinorVersion() noexcept;
```

```
5403 extern "C" TENSORRTAPI int32_t getInferLibPatchVersion() noexcept;
5407 extern "C" TENSORRTAPI int32_t getInferLibBuildVersion() noexcept;
5408
5409 #endif // NV_INFER_RUNTIME_H
```

10.15 NvInferRuntimeBase.h File Reference

```
#include "NvInferVersion.h"
#include <cstddef>
#include <cstdint>
#include <cuda_runtime_api.h>
```

Classes

- struct nvinfer1::impl::EnumMaxImpl< DataType >

  Maximum number of elements in DataType enum.
- class nvinfer1::Dims64
- class nvinfer1::InterfaceInfo

  Version information associated with a TRT interface.
- struct nvinfer1::impl::EnumMaxImpl< APILanguage >

  Maximum number of elements in APILanguage enum.
- class nvinfer1::IVersionedInterface

  An Interface class for version control.
- struct nvinfer1::impl::EnumMaxImpl< ErrorCode >

  Maximum number of elements in ErrorCode enum.
- class nvinfer1::v_1_0::IErrorRecorder
- struct nvinfer1::impl::EnumMaxImpl< TensorIOMode >

  Maximum number of elements in TensorIOMode enum.

Namespaces

- namespace nvinfer1

  The TensorRT API version 1 namespace.
- namespace nvinfer1::v_1_0

  Forward declare IErrorRecorder for use in other interfaces.
- namespace nvinfer1::impl

Macros

- #define TRT_DEPRECATED __attribute__((deprecated))
- #define TRT_DEPRECATED_ENUM
- #define TRT_DEPRECATED_API __attribute__((deprecated, visibility("default")))
- #define TENSORRTAPI
- #define TRTNOEXCEPT
- #define NV_TENSORRT_VERSION_INT(major, minor, patch) ((major) *10000L + (minor) *100L + (patch) *1L)
- #define NV_TENSORRT_VERSION NV_TENSORRT_VERSION_INT(NV_TENSORRT_MAJOR, NV_TENSORRT_MINOR, NV_TENSORRT_PATCH)

Typedefs

- using nvinfer1::char_t = char

  char_t is the type used by TensorRT to represent all valid characters.
- using nvinfer1::AsciiChar = char_t
- using nvinfer1::IErrorRecorder = v_1_0::IErrorRecorder
- using nvinfer1::Dims = Dims64
- using nvinfer1::InterfaceKind = char const *

Enumerations

- enum class nvinfer1::DataType : int32_t { nvinfer1::kFLOAT = 0 , nvinfer1::kHALF = 1 , nvinfer1::kINT8 = 2 , nvinfer1::kINT32 = 3 , nvinfer1::kBOOL = 4 , nvinfer1::kUINT8 = 5 , nvinfer1::kFP8 = 6 , nvinfer1::kBF16 = 7 , nvinfer1::kINT64 = 8 , nvinfer1::kINT4 = 9 , nvinfer1::kFP4 = 10 }

  The type of weights and tensors.
- enum class nvinfer1::APILanguage : int32_t { nvinfer1::kCPP = 0 , nvinfer1::kPYTHON = 1 }

  Programming language used in the implementation of a TRT interface.
- enum class nvinfer1::ErrorCode : int32_t { nvinfer1::kSUCCESS = 0 , nvinfer1::kUNSPECIFIED_ERROR = 1 , nvinfer1::kINTERNAL_ERROR = 2 , nvinfer1::kINVALID_ARGUMENT = 3 , nvinfer1::kINVALID_CONFIG = 4 , nvinfer1::kFAILED_ALLOCATION = 5 , nvinfer1::kFAILED_INITIALIZATION = 6 , nvinfer1::kFAILED_EXECUTION = 7 , nvinfer1::kFAILED_COMPUTATION = 8 , nvinfer1::kINVALID_STATE = 9 , nvinfer1::kUNSUPPORTED_STATE = 10 }

  Error codes that can be returned by TensorRT during execution.
- enum class nvinfer1::TensorIOMode : int32_t { nvinfer1::kNONE = 0 , nvinfer1::kINPUT = 1 , nvinfer1::kOUTPUT = 2 }

  Definition of tensor IO Mode.

Functions

- template<typename T >
  constexpr int32_t nvinfer1::EnumMax () noexcept

  Maximum number of elements in an enumeration type.
- int32_t getInferLibVersion () noexcept

  Return the library version number.

10.15.1 Detailed Description

This file contains common definitions, data structures and interfaces shared between the standard and safe runtime.

Warning

Do not directly include this file. Instead include one of:

- NvInferRuntime.h (for the standard runtime)
- NvInferPluginUtils.h (for plugin utilities)

10.15.2 Macro Definition Documentation

10.15.2.1 NV_TENSORRT_VERSION

```
#define NV_TENSORRT_VERSION NV_TENSORRT_VERSION_INT(NV_TENSORRT_MAJOR, NV_TENSORRT_MINOR, NV_TENSORRT_PATCH)
```

TensorRT version as a single integer. Usable in preprocessor expressions.

10.15.2.2 NV_TENSORRT_VERSION_INT

```
#define NV_TENSORRT_VERSION_INT(
            major,
            minor,
            patch ) ((major) *10000L + (minor) *100L + (patch) *1L)
```

Construct a single integer denoting TensorRT version. Usable in preprocessor expressions.

10.15.2.3 TENSORRTAPI

```
#define TENSORRTAPI
```

10.15.2.4 TRT_DEPRECATED

```
#define TRT_DEPRECATED __attribute__((deprecated))
```

10.15.2.5 TRT_DEPRECATED_API

```
#define TRT_DEPRECATED_API __attribute__((deprecated, visibility("default")))
```

10.15.2.6 TRT_DEPRECATED_ENUM

```
#define TRT_DEPRECATED_ENUM
```

10.15.2.7 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: (MAJOR * 100 + MINOR) * 100 + PATCH

10.16 NvInferRuntimeBase.h

Go to the documentation of this file.

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2025 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: Apache-2.0
4  *
5  * Licensed under the Apache License, Version 2.0 (the "License");
6  * you may not use this file except in compliance with the License.
7  * You may obtain a copy of the License at
8  *
9  * http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in writing, software
12  * distributed under the License is distributed on an "AS IS" BASIS,
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14  * See the License for the specific language governing permissions and
15  * limitations under the License.
16  */
17
18 #ifndef NV_INFER_RUNTIME_BASE_H
19 #define NV_INFER_RUNTIME_BASE_H
20
21 #include "NvInferVersion.h"
22 #include <cstddef>
23 #include <cstdint>
24 #include <cuda_runtime_api.h>
25
26 // Items that are marked as deprecated will be removed in a future release.
27 #if __cplusplus >= 201402L
28 #define TRT_DEPRECATED [[deprecated]]
29 #define TRT_DEPRECATED_ENUM TRT_DEPRECATED
30 #ifdef _MSC_VER
31 #define TRT_DEPRECATED_API __declspec(dllexport)
32 #else
33 #define TRT_DEPRECATED_API [[deprecated]] __attribute__((visibility("default")))
34 #endif
35 #else
36 #ifdef _MSC_VER
37 #define TRT_DEPRECATED
38 #define TRT_DEPRECATED_ENUM
39 #define TRT_DEPRECATED_API __declspec(dllexport)
40 #else
41 #define TRT_DEPRECATED __attribute__((deprecated))
42 #define TRT_DEPRECATED_ENUM
43 #define TRT_DEPRECATED_API __attribute__((deprecated, visibility("default")))
```

```
44 #endif
45 #endif
46
47 // Defines which symbols are exported
48 #ifdef TENSORRT_BUILD_LIB
49 #ifdef _MSC_VER
50 #define TENSORRTAPI __declspec(dllexport)
51 #else
52 #define TENSORRTAPI __attribute__((visibility("default")))
53 #endif
54 #else
55 #define TENSORRTAPI
56 #endif
57 #define TRTNOEXCEPT
67 #if !defined(NV_INFER_INTERNAL_INCLUDE)
68 static_assert(false, "Do not directly include this file. Include NvInferRuntime.h or NvInferPluginUtils.h");
69 #endif
70
72
73 extern "C"
74 {
76     struct cublasContext;
78     struct cudnnContext;
79 }
80
83 #define NV_TENSORRT_VERSION_INT(major, minor, patch) ((major) *10000L + (minor) *100L + (patch) *1L)
84
87 #define NV_TENSORRT_VERSION NV_TENSORRT_VERSION_INT(NV_TENSORRT_MAJOR, NV_TENSORRT_MINOR, NV_TENSORRT_PATCH)
88
94 namespace nvinfer1
95 {
97 using char_t = char;
98
101 using AsciiChar = char_t;
102
104 namespace v_1_0
105 {
106 class IErrorRecorder;
107 }
108 using IErrorRecorder = v_1_0::IErrorRecorder;
109
110 namespace impl
111 {
113 template <typename T>
114 struct EnumMaxImpl;
115 } // namespace impl
116
118 template <typename T>
119 constexpr int32_t EnumMax() noexcept
120 {
121     return impl::EnumMaxImpl<T>::kVALUE;
122 }
123
128 enum class DataType : int32_t
129 {
131     kFLOAT = 0,
132
134     kHALF = 1,
135
137     kINT8 = 2,
138
140     kINT32 = 3,
141
143     kBOOL = 4,
144
157     kUINT8 = 5,
158
161     kFP8 = 6,
162
164     kBF16 = 7,
165
167     kINT64 = 8,
168
170     kINT4 = 9,
171
174     kFP4 = 10,
175
176 };
177
178 namespace impl
179 {
```

```
181 template <>
182 struct EnumMaxImpl<DataType>
183 {
185     static constexpr int32_t kVALUE = 11;
186 };
187 } // namespace impl
188
199 class Dims64
200 {
201 public:
203     static constexpr int32_t MAX_DIMS{8};
204
206     int32_t nbDims;
207
209     int64_t d[MAX_DIMS];
210 };
211
215 using Dims = Dims64;
216
217 using InterfaceKind = char const*;
218
224 class InterfaceInfo
225 {
226 public:
227     InterfaceKind kind;
228     int32_t major;
229     int32_t minor;
230 };
231
237 enum class APILanguage : int32_t
238 {
239     kCPP = 0,
240     kPYTHON = 1
241 };
242
243 namespace impl
244 {
246 template <>
247 struct EnumMaxImpl<APILanguage>
248 {
250     static constexpr int32_t kVALUE = 2;
251 };
252 } // namespace impl
253
259 class IVersionedInterface
260 {
261 public:
267     virtual APILanguage getAPILanguage() const noexcept
268     {
269         return APILanguage::kCPP;
270     }
271
275     virtual InterfaceInfo getInterfaceInfo() const noexcept = 0;
276
277     virtual ~IVersionedInterface() noexcept = default;
278
279 protected:
280     IVersionedInterface() = default;
281     IVersionedInterface(IVersionedInterface const&) = default;
282     IVersionedInterface(IVersionedInterface&&) = default;
283     IVersionedInterface& operator=(IVersionedInterface const&) & = default;
284     IVersionedInterface& operator=(IVersionedInterface&&) & = default;
285 };
286
292 enum class ErrorCode : int32_t
293 {
297     kSUCCESS = 0,
298
302     kUNSPECIFIED_ERROR = 1,
303
308     kINTERNAL_ERROR = 2,
309
314     kINVALID_ARGUMENT = 3,
315
323     kINVALID_CONFIG = 4,
324
330     kFAILED_ALLOCATION = 5,
331
336     kFAILED_INITIALIZATION = 6,
337
344     kFAILED_EXECUTION = 7,
```

```
345
353     kFAILED_COMPUTATION = 8,
354
367     kINVALID_STATE = 9,
368
379     kUNSUPPORTED_STATE = 10,
380
381 };
382
383 namespace impl
384 {
386 template <>
387 struct EnumMaxImpl<ErrorCode>
388 {
390     static constexpr int32_t kVALUE = 11;
391 };
392 } // namespace impl
393
394 namespace v_1_0
395 {
396 class IErrorRecorder : public IVersionedInterface
397 {
398 public:
402     InterfaceInfo getInterfaceInfo() const noexcept override
403     {
404         return InterfaceInfo{"IErrorRecorder", 1, 0};
405     }
406
410     using ErrorDesc = char const*;
411
417     static constexpr size_t kMAX_DESC_LENGTH{127U};
418
422     using RefCount = int32_t;
423
424     IErrorRecorder() = default;
425     ~IErrorRecorder() noexcept override = default;
426
427     // Public API used to retrieve information from the error recorder.
428
452     virtual int32_t getNbErrors() const noexcept = 0;
453
472     virtual ErrorCode getErrorCode(int32_t errorIdx) const noexcept = 0;
473
495     virtual ErrorDesc getErrorDesc(int32_t errorIdx) const noexcept = 0;
496
511     virtual bool hasOverflowed() const noexcept = 0;
512
527     virtual void clear() noexcept = 0;
528
529     // API used by TensorRT to report Error information to the application.
530
557     virtual bool reportError(ErrorCode val, ErrorDesc desc) noexcept = 0;
558
575     virtual RefCount incRefCount() noexcept = 0;
576
593     virtual RefCount decRefCount() noexcept = 0;
594
595 protected:
596     // @cond SuppressDoxyWarnings
597     IErrorRecorder(IErrorRecorder const&) = default;
598     IErrorRecorder(IErrorRecorder&&) = default;
599     IErrorRecorder& operator=(IErrorRecorder const&) & = default;
600     IErrorRecorder& operator=(IErrorRecorder&&) & = default;
601     // @endcond
602 }; // class IErrorRecorder
603 } // namespace v_1_0
604
632 using IErrorRecorder = v_1_0::IErrorRecorder;
633
639 enum class TensorIOMode : int32_t
640 {
642     kNONE = 0,
643
645     kINPUT = 1,
646
648     kOUTPUT = 2
649 };
650
651 namespace impl
652 {
654 template <>
```

```
655 struct EnumMaxImpl<TensorIOMode>
656 {
657     // Declaration of kVALUE that represents maximum number of elements in TensorIOMode enum
658     static constexpr int32_t kVALUE = 3;
659 };
660 } // namespace impl
661 } // namespace nvinfer1
662
668 extern "C" TENSORRTAPI int32_t getInferLibVersion() noexcept;
669
670 #endif // NV_INFER_RUNTIME_BASE_H
```

10.17 NvInferRuntimeCommon.h File Reference

```
#include "NvInferPluginBase.h"
#include "NvInferRuntimePlugin.h"
```

Classes

- class nvinfer1::IPluginRegistry

  Single registration point for all plugins in an application. It is used to find plugin implementations during engine deserialization. Internally, the plugin registry is considered to be a singleton so all plugins in an application are part of the same global registry. Note that the plugin registry is only supported for plugins of type IPluginV2 and should also have a corresponding IPluginCreator implementation.

Namespaces

- namespace nvinfer1

  The TensorRT API version 1 namespace.

Macros

- #define NV_INFER_INTERNAL_INCLUDE 1

10.17.1 Detailed Description

This file provides the nvinfer1::IPluginRegistry interface, which will be moved to the NvInferRuntime.h header in a future release.

> Warning
>
> This file will be removed in a future release.
>
> Do not directly include this file. Instead include NvInferRuntime.h

10.17.2 Macro Definition Documentation

10.17.2.1 NV_INFER_INTERNAL_INCLUDE

```
#define NV_INFER_INTERNAL_INCLUDE 1
```

10.18 NvInferRuntimeCommon.h

Go to the documentation of this file.

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2025 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: Apache-2.0
4  *
5  * Licensed under the Apache License, Version 2.0 (the "License");
6  * you may not use this file except in compliance with the License.
7  * You may obtain a copy of the License at
8  *
9  * http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in writing, software
12  * distributed under the License is distributed on an "AS IS" BASIS,
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14  * See the License for the specific language governing permissions and
15  * limitations under the License.
16  */
17
18 #ifndef NV_INFER_RUNTIME_COMMON_H
19 #define NV_INFER_RUNTIME_COMMON_H
20
31 #define NV_INFER_INTERNAL_INCLUDE 1
32 #include "NvInferPluginBase.h"
33 #undef NV_INFER_INTERNAL_INCLUDE
34 #include "NvInferRuntimePlugin.h"
35
36 namespace nvinfer1
37 {
55 class IPluginRegistry
56 {
57 public:
61     using PluginLibraryHandle = void*;
62
77     TRT_DEPRECATED virtual bool registerCreator(
78         IPluginCreator& creator, AsciiChar const* const pluginNamespace) noexcept = 0;
79
93     TRT_DEPRECATED virtual IPluginCreator* const* getPluginCreatorList(int32_t* const numCreators) const
      noexcept = 0;
94
112     TRT_DEPRECATED virtual IPluginCreator* getPluginCreator(AsciiChar const* const pluginName,
113         AsciiChar const* const pluginVersion, AsciiChar const* const pluginNamespace = "") noexcept = 0;
114
115     // @cond SuppressDoxyWarnings
116     IPluginRegistry() = default;
117     IPluginRegistry(IPluginRegistry const&) = delete;
118     IPluginRegistry(IPluginRegistry&&) = delete;
119     IPluginRegistry& operator=(IPluginRegistry const&) & = delete;
120     IPluginRegistry& operator=(IPluginRegistry&&) & = delete;
121     // @endcond
122
123 protected:
124     virtual ~IPluginRegistry() noexcept = default;
125
126 public:
143     virtual void setErrorRecorder(IErrorRecorder* const recorder) noexcept = 0;
144
160     virtual IErrorRecorder* getErrorRecorder() const noexcept = 0;
161
179     TRT_DEPRECATED virtual bool deregisterCreator(IPluginCreator const& creator) noexcept = 0;
180
189     virtual bool isParentSearchEnabled() const = 0;
190
198     virtual void setParentSearchEnabled(bool const enabled) = 0;
199
208     virtual PluginLibraryHandle loadLibrary(AsciiChar const* pluginPath) noexcept = 0;
209
216     virtual void deregisterLibrary(PluginLibraryHandle handle) noexcept = 0;
```

```
217
229     virtual bool registerCreator(IPluginCreatorInterface& creator, AsciiChar const* const pluginNamespace)
        noexcept = 0;
230
241     virtual IPluginCreatorInterface* const* getAllCreators(int32_t* const numCreators) const noexcept = 0;
242
254     virtual IPluginCreatorInterface* getCreator(AsciiChar const* const pluginName, AsciiChar const* const
        pluginVersion,
255         AsciiChar const* const pluginNamespace = "") noexcept = 0;
256
271     virtual bool deregisterCreator(IPluginCreatorInterface const& creator) noexcept = 0;
272
285     virtual IPluginResource* acquirePluginResource(AsciiChar const* key, IPluginResource* resource) noexcept
        = 0;
286
300     virtual int32_t releasePluginResource(AsciiChar const* key) noexcept = 0;
301
317     virtual IPluginCreatorInterface* const* getAllCreatorsRecursive(int32_t* const numCreators) noexcept =
        0;
318 };
319
320 } // namespace nvinfer1
321
322 #endif /* NV_INFER_RUNTIME_COMMON_H */
```

10.19 NvInferRuntimePlugin.h File Reference

```
#include "NvInferPluginBase.h"
```

Classes

- struct nvinfer1::PluginTensorDesc

  Fields that a plugin might see for an input or output.
- class nvinfer1::IPluginV2

  Plugin class for user-implemented layers.
- class nvinfer1::IPluginV2Ext

  Plugin class for user-implemented layers.
- class nvinfer1::IPluginV2IOExt

  Plugin class for user-implemented layers.
- class nvinfer1::v_1_0::IPluginCreator

Namespaces

- namespace nvinfer1

  The TensorRT API version 1 namespace.
- namespace nvinfer1::v_1_0

  Forward declare IErrorRecorder for use in other interfaces.

Macros

- #define NV_INFER_INTERNAL_INCLUDE 1

Typedefs

- using nvinfer1::PluginFormat = TensorFormat

  PluginFormat is reserved for backward compatibility.
- using nvinfer1::IPluginCreator = v_1_0::IPluginCreator

Enumerations

- enum class nvinfer1::PluginVersion : uint8_t { nvinfer1::kV2 = 0 , nvinfer1::kV2_EXT = 1 , nvinfer1::kV2_IOEXT = 2 , nvinfer1::kV2_DYNAMICEXT = 3 , nvinfer1::kV2_DYNAMICEXT_PYTHON = kPLUGIN_VERSION_PYTHON_BIT | 3 }
- enum class nvinfer1::PluginCreatorVersion : int32_t { nvinfer1::kV1 = 0 , nvinfer1::kV1_PYTHON = kPLUGIN_VERSION_PYTHON_BIT }

  Enum to identify version of the plugin creator.

10.19.1 Detailed Description

This file contains common definitions, data structures and interfaces that relate to plugins and are shared between the standard and safe runtime.

> Warning
>
> Do not directly include this file. Instead include NvInferRuntime.h

10.19.2 Macro Definition Documentation

10.19.2.1 NV_INFER_INTERNAL_INCLUDE

```
#define NV_INFER_INTERNAL_INCLUDE 1
```

10.20 NvInferRuntimePlugin.h

Go to the documentation of this file.

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2025 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: Apache-2.0
4  *
5  * Licensed under the Apache License, Version 2.0 (the "License");
6  * you may not use this file except in compliance with the License.
7  * You may obtain a copy of the License at
8  *
9  * http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in writing, software
12  * distributed under the License is distributed on an "AS IS" BASIS,
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14  * See the License for the specific language governing permissions and
15  * limitations under the License.
```

```
16  */
17
18 #ifndef NV_INFER_RUNTIME_PLUGIN_H
19 #define NV_INFER_RUNTIME_PLUGIN_H
20
21 #define NV_INFER_INTERNAL_INCLUDE 1
22 #include "NvInferPluginBase.h"
23 #undef NV_INFER_INTERNAL_INCLUDE
24
33
39 namespace nvinfer1
40 {
41
42 enum class TensorFormat : int32_t;
43 namespace v_1_0
44 {
45 class IGpuAllocator;
46 }
47 using IGpuAllocator = v_1_0::IGpuAllocator;
48
54 using PluginFormat = TensorFormat;
55
59 static constexpr int32_t kPLUGIN_VERSION_PYTHON_BIT = 0x40;
60
72 struct PluginTensorDesc
73 {
75     Dims dims;
77     DataType type;
79     TensorFormat format;
81     float scale;
82 };
83
94 enum class PluginVersion : uint8_t
95 {
97     kV2 TRT_DEPRECATED_ENUM = 0,
99     kV2_EXT TRT_DEPRECATED_ENUM = 1,
101     kV2_IOEXT TRT_DEPRECATED_ENUM = 2,
103     kV2_DYNAMICEXT TRT_DEPRECATED_ENUM = 3,
105     kV2_DYNAMICEXT_PYTHON TRT_DEPRECATED_ENUM = kPLUGIN_VERSION_PYTHON_BIT | 3
106 };
107
116 enum class PluginCreatorVersion : int32_t
117 {
119     kV1 TRT_DEPRECATED_ENUM = 0,
121     kV1_PYTHON TRT_DEPRECATED_ENUM = kPLUGIN_VERSION_PYTHON_BIT
122 };
123
138 class TRT_DEPRECATED IPluginV2
139 {
140 public:
153     virtual int32_t getTensorRTVersion() const noexcept
154     {
155         return NV_TENSORRT_VERSION;
156     }
157
171     virtual AsciiChar const* getPluginType() const noexcept = 0;
172
186     virtual AsciiChar const* getPluginVersion() const noexcept = 0;
187
201     virtual int32_t getNbOutputs() const noexcept = 0;
202
226     virtual Dims getOutputDimensions(int32_t index, Dims const* inputs, int32_t nbInputDims) noexcept = 0;
227
251     virtual bool supportsFormat(DataType type, PluginFormat format) const noexcept = 0;
252
285     virtual void configureWithFormat(Dims const* inputDims, int32_t nbInputs, Dims const* outputDims, int32_t
    nbOutputs,
286         DataType type, PluginFormat format, int32_t maxBatchSize) noexcept
287         = 0;
288
300     virtual int32_t initialize() noexcept = 0;
301
315     virtual void terminate() noexcept = 0;
316
334     virtual size_t getWorkspaceSize(int32_t maxBatchSize) const noexcept = 0;
335
357     virtual int32_t enqueue(int32_t batchSize, void const* const* inputs, void* const* outputs, void*
    workspace,
358         cudaStream_t stream) noexcept
359         = 0;
360
```

```
371     virtual size_t getSerializationSize() const noexcept = 0;
372
386     virtual void serialize(void* buffer) const noexcept = 0;
387
396     virtual void destroy() noexcept = 0;
397
415     virtual IPluginV2* clone() const noexcept = 0;
416
431     virtual void setPluginNamespace(AsciiChar const* pluginNamespace) noexcept = 0;
432
444     virtual AsciiChar const* getPluginNamespace() const noexcept = 0;
445
446     // @cond SuppressDoxyWarnings
447     IPluginV2() = default;
448     virtual ~IPluginV2() noexcept = default;
449 // @endcond
450
451 protected:
452 // @cond SuppressDoxyWarnings
453     IPluginV2(IPluginV2 const&) = default;
454     IPluginV2(IPluginV2&&) = default;
455     IPluginV2& operator=(IPluginV2 const&) & = default;
456     IPluginV2& operator=(IPluginV2&&) & = default;
457 // @endcond
458 };
459
473 class TRT_DEPRECATED IPluginV2Ext : public IPluginV2
474 {
475 public:
499     virtual nvinfer1::DataType getOutputDataType(
500         int32_t index, nvinfer1::DataType const* inputTypes, int32_t nbInputs) const noexcept
501         = 0;
502
523     TRT_DEPRECATED virtual bool isOutputBroadcastAcrossBatch(
524         int32_t outputIndex, bool const* inputIsBroadcasted, int32_t nbInputs) const noexcept
525         = 0;
526
552     TRT_DEPRECATED virtual bool canBroadcastInputAcrossBatch(int32_t inputIndex) const noexcept = 0;
553
591     virtual void configurePlugin(Dims const* inputDims, int32_t nbInputs, Dims const* outputDims, int32_t
    nbOutputs,
592         DataType const* inputTypes, DataType const* outputTypes, bool const* inputIsBroadcast,
593         bool const* outputIsBroadcast, PluginFormat floatFormat, int32_t maxBatchSize) noexcept
594         = 0;
595
596     IPluginV2Ext() = default;
597     ~IPluginV2Ext() override = default;
598
633     virtual void attachToContext(
634         cudnnContext* /*cudnn*/, cublasContext* /*cublas*/, IGpuAllocator* /*allocator*/) noexcept
635     {
636     }
637
651     virtual void detachFromContext() noexcept {}
652
667     IPluginV2Ext* clone() const noexcept override = 0;
668
669 protected:
670     // @cond SuppressDoxyWarnings
671     IPluginV2Ext(IPluginV2Ext const&) = default;
672     IPluginV2Ext(IPluginV2Ext&&) = default;
673     IPluginV2Ext& operator=(IPluginV2Ext const&) & = default;
674     IPluginV2Ext& operator=(IPluginV2Ext&&) & = default;
675 // @endcond
676
692     int32_t getTensorRTVersion() const noexcept override
693     {
694         return static_cast<int32_t>((static_cast<uint32_t>(PluginVersion::kV2_EXT) << 24U)
695             | (static_cast<uint32_t>(NV_TENSORRT_VERSION) & 0xFFFFFFU));
696     }
697
704     void configureWithFormat(Dims const* /*inputDims*/, int32_t /*nbInputs*/, Dims const* /*outputDims*/,
705         int32_t /*nbOutputs*/, DataType /*type*/, PluginFormat /*format*/, int32_t /*maxBatchSize*/) noexcept
    override
706     {
707     }
708 };
709
722 class TRT_DEPRECATED IPluginV2IOExt : public IPluginV2Ext
723 {
724 public:
```

```
742     virtual void configurePlugin(
743         PluginTensorDesc const* in, int32_t nbInput, PluginTensorDesc const* out, int32_t nbOutput) noexcept
744         = 0;
745
783     virtual bool supportsFormatCombination(
784         int32_t pos, PluginTensorDesc const* inOut, int32_t nbInputs, int32_t nbOutputs) const noexcept
785         = 0;
786
787     // @cond SuppressDoxyWarnings
788     IPluginV2IOExt() = default;
789     ~IPluginV2IOExt() override = default;
790 // @endcond
791
792 protected:
793 // @cond SuppressDoxyWarnings
794     IPluginV2IOExt(IPluginV2IOExt const&) = default;
795     IPluginV2IOExt(IPluginV2IOExt&&) = default;
796     IPluginV2IOExt& operator=(IPluginV2IOExt const&) & = default;
797     IPluginV2IOExt& operator=(IPluginV2IOExt&&) & = default;
798 // @endcond
799
811     int32_t getTensorRTVersion() const noexcept override
812     {
813         return static_cast<int32_t>((static_cast<uint32_t>(PluginVersion::kV2_IOEXT) << 24U)
814             | (static_cast<uint32_t>(NV_TENSORRT_VERSION) & 0xFFFFFFU));
815     }
816
817 private:
818     // Following are obsolete base class methods, and must not be implemented or used.
819
823     void configurePlugin(Dims const*, int32_t, Dims const*, int32_t, DataType const*, DataType const*, bool
    const*,
824         bool const*, PluginFormat, int32_t) noexcept final
825     {
826     }
827
831     bool supportsFormat(DataType, PluginFormat) const noexcept final
832     {
833         return false;
834     }
835 };
836
837 namespace v_1_0
838 {
839 class TRT_DEPRECATED IPluginCreator : public IPluginCreatorInterface
840 {
841 public:
854     virtual AsciiChar const* getPluginName() const noexcept = 0;
855
868     virtual AsciiChar const* getPluginVersion() const noexcept = 0;
869
881     virtual PluginFieldCollection const* getFieldNames() noexcept = 0;
882
895     virtual IPluginV2* createPlugin(AsciiChar const* name, PluginFieldCollection const* fc) noexcept = 0;
896
912     virtual IPluginV2* deserializePlugin(AsciiChar const* name, void const* serialData, size_t serialLength)
    noexcept
913         = 0;
914
929     virtual void setPluginNamespace(AsciiChar const* pluginNamespace) noexcept = 0;
930
943     virtual AsciiChar const* getPluginNamespace() const noexcept = 0;
944
945     IPluginCreator() = default;
946     ~IPluginCreator() override = default;
947
948 protected:
949     // @cond SuppressDoxyWarnings
950     IPluginCreator(IPluginCreator const&) = default;
951     IPluginCreator(IPluginCreator&&) = default;
952     IPluginCreator& operator=(IPluginCreator const&) & = default;
953     IPluginCreator& operator=(IPluginCreator&&) & = default;
954     // @endcond
955 public:
959     InterfaceInfo getInterfaceInfo() const noexcept override
960     {
961         return InterfaceInfo{"PLUGIN CREATOR_V1", 1, 0};
962     }
963 };
964 } // namespace v_1_0
965
```

```
976 using IPluginCreator = v_1_0::IPluginCreator;
977
978 } // namespace nvinfer1
979
980 #endif // NV_INFER_RUNTIME_PLUGIN_H
```

10.21 NvInferSerialize.h File Reference

`#include "NvInfer.h"`

Namespaces

- namespace nvinfer1

  The TensorRT API version 1 namespace.
- namespace nvinfer1::serialize

Functions

- nvinfer1::IHostMemory * nvinfer1::serialize::serializeNetwork (nvinfer1::INetworkDefinition const &network) noexcept

  Serialize TensorRT INetworkDefinition.

10.22 NvInferSerialize.h

Go to the documentation of this file.

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2024 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: Apache-2.0
4  *
5  * Licensed under the Apache License, Version 2.0 (the "License");
6  * you may not use this file except in compliance with the License.
7  * You may obtain a copy of the License at
8  *
9  * http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in writing, software
12  * distributed under the License is distributed on an "AS IS" BASIS,
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14  * See the License for the specific language governing permissions and
15  * limitations under the License.
16  */
17
18 #ifndef NV_INFER_SERIALIZE_H
19 #define NV_INFER_SERIALIZE_H
20
21 #include "NvInfer.h"
22
27 extern "C" TENSORRTAPI void* serializeNetwork_INTERNAL(void const* network) noexcept;
28
29 namespace nvinfer1
30 {
31 namespace serialize
32 {
33
42 inline nvinfer1::IHostMemory* serializeNetwork(nvinfer1::INetworkDefinition const& network) noexcept
43 {
44     return static_cast<nvinfer1::IHostMemory*>(serializeNetwork_INTERNAL(&network));
45 }
46
47 } // namespace serialize
48 } // namespace nvinfer1
49
50 #endif // NV_INFER_SERIALIZE_H
```

10.23 NvInferVersion.h File Reference

Macros

- #define NV_TENSORRT_MAJOR 10

  TensorRT major version.
- #define NV_TENSORRT_MINOR 10

  TensorRT minor version.
- #define NV_TENSORRT_PATCH 10

  TensorRT patch version.
- #define NV_TENSORRT_BUILD 1

  TensorRT build number.
- #define NV_TENSORRT_LWS_MAJOR 0

  TensorRT LWS major version.
- #define NV_TENSORRT_LWS_MINOR 0

  TensorRT LWS minor version.
- #define NV_TENSORRT_LWS_PATCH 0

  TensorRT LWS patch version.
- #define NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS 0

  An early access release.
- #define NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE 1

  A release candidate.
- #define NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY 2

  A final release.
- #define NV_TENSORRT_RELEASE_TYPE NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY

  TensorRT release type.

10.23.1 Detailed Description

Defines the TensorRT version

10.23.2 Macro Definition Documentation

10.23.2.1 NV_TENSORRT_BUILD

```
#define NV_TENSORRT_BUILD 1
```

TensorRT build number.

10.23.2.2 NV_TENSORRT_LWS_MAJOR

```
#define NV_TENSORRT_LWS_MAJOR 0
```

TensorRT LWS major version.

10.23.2.3 NV_TENSORRT_LWS_MINOR

```
#define NV_TENSORRT_LWS_MINOR 0
```

TensorRT LWS minor version.

10.23.2.4 NV_TENSORRT_LWS_PATCH

```
#define NV_TENSORRT_LWS_PATCH 0
```

TensorRT LWS patch version.

10.23.2.5 NV_TENSORRT_MAJOR

```
#define NV_TENSORRT_MAJOR 10
```

TensorRT major version.

10.23.2.6 NV_TENSORRT_MINOR

```
#define NV_TENSORRT_MINOR 10
```

TensorRT minor version.

10.23.2.7 NV_TENSORRT_PATCH

```
#define NV_TENSORRT_PATCH 10
```

TensorRT patch version.

10.23.2.8 NV_TENSORRT_RELEASE_TYPE

```
#define NV_TENSORRT_RELEASE_TYPE NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY
```

TensorRT release type.

10.23.2.9 NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS

```
#define NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS 0
```

An early access release.

10.23.2.10 NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY

```
#define NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY 2
```

A final release.

10.23.2.11 NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE

```
#define NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE 1
```

A release candidate.

10.24 NvInferVersion.h

Go to the documentation of this file.

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2025 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: Apache-2.0
4  *
5  * Licensed under the Apache License, Version 2.0 (the "License");
6  * you may not use this file except in compliance with the License.
7  * You may obtain a copy of the License at
8  *
9  * http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in writing, software
12  * distributed under the License is distributed on an "AS IS" BASIS,
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14  * See the License for the specific language governing permissions and
15  * limitations under the License.
16  */
17
23 #ifndef NV_INFER_VERSION_H
24 #define NV_INFER_VERSION_H
25
```

```
26 #define NV_TENSORRT_MAJOR 10
27 #define NV_TENSORRT_MINOR 10
28 #define NV_TENSORRT_PATCH 10
29 #define NV_TENSORRT_BUILD 1
30
31 #define NV_TENSORRT_LWS_MAJOR 0
32 #define NV_TENSORRT_LWS_MINOR 0
33 #define NV_TENSORRT_LWS_PATCH 0
34
35 #define NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS 0
36 #define NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE 1
37 #define NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY 2
38
39 #define NV_TENSORRT_RELEASE_TYPE NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY
40
41 #endif // NV_INFER_VERSION_H
```

10.25 NvOnnxConfig.h File Reference

```
#include "NvInfer.h"
```

Classes

- class nvonnxparser::IOnnxConfig

  Configuration Manager Class.

Namespaces

- namespace nvonnxparser

  The TensorRT ONNX parser API namespace.

Functions

- IOnnxConfig * nvonnxparser::createONNXConfig ()

10.25.1 Detailed Description

This is the API file for the Configuration Manager for ONNX Parser for Nvidia TensorRT.

10.26 NvOnnxConfig.h

Go to the documentation of this file.

```
1 /*
2  * SPDX-FileCopyrightText: Copyright (c) 1993-2024 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
3  * SPDX-License-Identifier: Apache-2.0
4  *
5  * Licensed under the Apache License, Version 2.0 (the "License");
6  * you may not use this file except in compliance with the License.
7  * You may obtain a copy of the License at
8  *
9  * http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in writing, software
12  * distributed under the License is distributed on an "AS IS" BASIS,
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14  * See the License for the specific language governing permissions and
15  * limitations under the License.
16  */
17
18 #ifndef NV_ONNXCONFIG_H
19 #define NV_ONNXCONFIG_H
20
21 #include "NvInfer.h"
22
23 namespace nvonnxparser
24 {
25
35
41
46 class IOnnxConfig
47 {
48 public:
49     virtual ~IOnnxConfig() noexcept = default;
55     typedef int32_t Verbosity;
56
65     virtual void setModelDtype(const nvinfer1::DataType) noexcept = 0;
66
74     virtual nvinfer1::DataType getModelDtype() const noexcept = 0;
75
83     virtual char const* getModelFileName() const noexcept = 0;
84
96     virtual void setModelFileName(char const* onnxFilename) noexcept = 0;
97
105     virtual Verbosity getVerbosityLevel() const noexcept = 0;
106
114     virtual void addVerbosity() noexcept = 0;
115
121     virtual void reduceVerbosity() noexcept = 0;
122
128     virtual void setVerbosityLevel(Verbosity) noexcept = 0;
129
138     virtual char const* getTextFileName() const noexcept = 0;
139
151     virtual void setTextFileName(char const* textFileName) noexcept = 0;
152
161     virtual char const* getFullTextFileName() const noexcept = 0;
162
174     virtual void setFullTextFileName(char const* fullTextFileName) noexcept = 0;
175
183     virtual bool getPrintLayerInfo() const noexcept = 0;
184
190     virtual void setPrintLayerInfo(bool) noexcept = 0;
191
192 }; // class IOnnxConfig
193
194 TENSORRTAPI IOnnxConfig* createONNXConfig();
195
196 } // namespace nvonnxparser
197
198 #endif
```

10.27 NvOnnxParser.h File Reference

```
#include "NvInfer.h"
#include <stddef.h>
#include <string>
#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
- class nvonnxparser::IParserRefitter

  An interface designed to refit weights from an ONNX model.

Namespaces

- namespace nvonnxparser

  The TensorRT ONNX parser API namespace.
- namespace nvonnxparser::anonymous_namespace{NvOnnxParser.h}

Macros

- #define NV_ONNX_PARSER_MAJOR 0
- #define NV_ONNX_PARSER_MINOR 1
- #define NV_ONNX_PARSER_PATCH 0

Typedefs

- typedef std::pair< std::vector< size_t >, bool > SubGraph_t

  The data structure containing the parsing capability of a set of nodes in an ONNX graph.
- typedef std::vector< SubGraph_t > SubGraphCollection_t

  The data structure containing all SubGraph_t partitioned out of an ONNX graph.
- using nvonnxparser::OnnxParserFlags = uint32_t

  Represents one or more OnnxParserFlag values using binary OR operations, e.g., 1U << OnnxParserFlag::kNATIVE_INSTANCENORM.

Enumerations

- enum class nvonnxparser::ErrorCode : int {
  nvonnxparser::kSUCCESS = 0 , nvonnxparser::kINTERNAL_ERROR = 1 , nvonnxparser::kMEM_ALLOC_FAILED = 2 , nvonnxparser::kMODEL_DESERIALIZE_FAILED = 3 ,
  nvonnxparser::kINVALID_VALUE = 4 , nvonnxparser::kINVALID_GRAPH = 5 , nvonnxparser::kINVALID_NODE = 6 , nvonnxparser::kUNSUPPORTED_GRAPH = 7 ,
  nvonnxparser::kUNSUPPORTED_NODE = 8 , nvonnxparser::kUNSUPPORTED_NODE_ATTR = 9 , nvonnxparser::kUNSUPPORTED_NODE_INPUT = 10 , nvonnxparser::kUNSUPPORTED_NODE_DATATYPE = 11 ,
  nvonnxparser::kUNSUPPORTED_NODE_DYNAMIC = 12 , nvonnxparser::kUNSUPPORTED_NODE_SHAPE = 13 , nvonnxparser::kREFIT_FAILED = 14 }

  The type of error that the parser or refitter may return.
- enum class nvonnxparser::OnnxParserFlag : int32_t { nvonnxparser::kNATIVE_INSTANCENORM = 0 }

Functions

- template<typename T >
  constexpr int32_t nvonnxparser::EnumMax () noexcept
- template<> constexpr int32_t nvonnxparser::EnumMax< ErrorCode > () noexcept
- template<> constexpr int32_t nvonnxparser::EnumMax< OnnxParserFlag > () noexcept
- TENSORRTAPI void * createNvOnnxParser_INTERNAL (void *network, void *logger, int version) noexcept
- TENSORRTAPI void * createNvOnnxParserRefitter_INTERNAL (void *refitter, void *logger, int32_t version) noexcept
- TENSORRTAPI int getNvOnnxParserVersion () noexcept
- IParser * nvonnxparser::anonymous_namespace{NvOnnxParser.h}::createParser (nvinfer1::INetworkDefinition &network, nvinfer1::ILogger &logger) noexcept

  Create a new parser object.
- IParserRefitter * nvonnxparser::anonymous_namespace{NvOnnxParser.h}::createParserRefitter (nvinfer1::IRefitter &refitter, nvinfer1::ILogger &logger) noexcept

  Create a new ONNX refitter object.

10.27.1 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 )  [noexcept]
```

10.27.4.2 createNvOnnxParserRefitter_INTERNAL()

```
TENSORRTAPI void * createNvOnnxParserRefitter_INTERNAL (
            void * refitter,
            void * logger,
            int32_t version )  [noexcept]
```

10.27.4.3 getNvOnnxParserVersion()

```
TENSORRTAPI int getNvOnnxParserVersion ( )  [noexcept]
```

10.28 NvOnnxParser.h

Go to the documentation of this file.

```
1 /*
2  * Copyright (c) 1993-2024, 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 <string>
29 #include <vector>
30
36
37 #define NV_ONNX_PARSER_MAJOR 0
38 #define NV_ONNX_PARSER_MINOR 1
39 #define NV_ONNX_PARSER_PATCH 0
40
41 static constexpr int32_t NV_ONNX_PARSER_VERSION
42     = ((NV_ONNX_PARSER_MAJOR * 10000) + (NV_ONNX_PARSER_MINOR * 100) + NV_ONNX_PARSER_PATCH);
43
50 typedef std::pair<std::vector<size_t>, bool> SubGraph_t;
51
58 typedef std::vector<SubGraph_t> SubGraphCollection_t;
59
65 namespace nvonnxparser
66 {
67
68 template <typename T>
69 constexpr inline int32_t EnumMax() noexcept;
70
76 enum class ErrorCode : int
77 {
```

```
78      kSUCCESS = 0,
79      kINTERNAL_ERROR = 1,
80      kMEM_ALLOC_FAILED = 2,
81      kMODEL_DESERIALIZE_FAILED = 3,
82      kINVALID_VALUE = 4,
83      kINVALID_GRAPH = 5,
84      kINVALID_NODE = 6,
85      kUNSUPPORTED_GRAPH = 7,
86      kUNSUPPORTED_NODE = 8,
87      kUNSUPPORTED_NODE_ATTR = 9,
88      kUNSUPPORTED_NODE_INPUT = 10,
89      kUNSUPPORTED_NODE_DATATYPE = 11,
90      kUNSUPPORTED_NODE_DYNAMIC = 12,
91      kUNSUPPORTED_NODE_SHAPE = 13,
92      kREFIT_FAILED = 14
93  };
94
100 template <>
101 constexpr inline int32_t EnumMax<ErrorCode>() noexcept
102 {
103     return 14;
104 }
105
112 using OnnxParserFlags = uint32_t;
113
114 enum class OnnxParserFlag : int32_t
115 {
120     kNATIVE_INSTANCENORM = 0
121 };
122
128 template <>
129 constexpr inline int32_t EnumMax<OnnxParserFlag>() noexcept
130 {
131     return 1;
132 }
133
139 class IParserError
140 {
141 public:
145     virtual ErrorCode code() const = 0;
149     virtual char const* desc() const = 0;
153     virtual char const* file() const = 0;
157     virtual int line() const = 0;
161     virtual char const* func() const = 0;
165     virtual int node() const = 0;
169     virtual char const* nodeName() const = 0;
173     virtual char const* nodeOperator() const = 0;
179     virtual char const* const* localFunctionStack() const = 0;
183     //              a stack size of 0.
185     virtual int32_t localFunctionStackSize() const = 0;
186
187 protected:
188     virtual ~IParserError() {}
189 };
190
201 class IParser
202 {
203 public:
218     virtual bool parse(
219         void const* serialized_onnx_model, size_t serialized_onnx_model_size, const char* model_path = nullptr)
    noexcept
220         = 0;
221
232     virtual bool parseFromFile(const char* onnxModelFile, int verbosity) noexcept = 0;
233
247     TRT_DEPRECATED virtual bool supportsModel(void const* serialized_onnx_model, size_t
    serialized_onnx_model_size,
248         SubGraphCollection_t& sub_graph_collection, const char* model_path = nullptr) noexcept = 0;
249
260     virtual bool parseWithWeightDescriptors(
261         void const* serialized_onnx_model, size_t serialized_onnx_model_size) noexcept
262         = 0;
263
273     virtual bool supportsOperator(const char* op_name) const noexcept = 0;
274
281     virtual int getNbErrors() const noexcept = 0;
282
288     virtual IParserError const* getError(int index) const noexcept = 0;
289
295     virtual void clearErrors() noexcept = 0;
296
```

```
297     virtual ~IParser() noexcept = default;
298
315     virtual char const* const* getUsedVCPluginLibraries(int64_t& nbPluginLibs) const noexcept = 0;
316
328     virtual void setFlags(OnnxParserFlags onnxParserFlags) noexcept = 0;
329
337     virtual OnnxParserFlags getFlags() const noexcept = 0;
338
346     virtual void clearFlag(OnnxParserFlag onnxParserFlag) noexcept = 0;
347
355     virtual void setFlag(OnnxParserFlag onnxParserFlag) noexcept = 0;
356
364     virtual bool getFlag(OnnxParserFlag onnxParserFlag) const noexcept = 0;
365
378     virtual nvinfer1::ITensor const* getLayerOutputTensor(char const* name, int64_t i) noexcept = 0;
379
392     virtual bool supportsModelV2(
393         void const* serializedOnnxModel, size_t serializedOnnxModelSize, char const* modelPath = nullptr)
        noexcept = 0;
394
402     virtual int64_t getNbSubgraphs() noexcept = 0;
403
412     virtual bool isSubgraphSupported(int64_t const index) noexcept = 0;
413
424     virtual int64_t* getSubgraphNodes(int64_t const index, int64_t& subgraphLength) noexcept = 0;
425 };
426
434 class IParserRefitter
435 {
436 public:
449     virtual bool refitFromBytes(
450         void const* serializedOnnxModel, size_t serializedOnnxModelSize, char const* modelPath = nullptr)
        noexcept
451         = 0;
452
463     virtual bool refitFromFile(char const* onnxModelFile) noexcept = 0;
464
470     virtual int32_t getNbErrors() const noexcept = 0;
471
477     virtual IParserError const* getError(int32_t index) const noexcept = 0;
478
484     virtual void clearErrors() = 0;
485
486     virtual ~IParserRefitter() noexcept = default;
487 };
488
489 } // namespace nvonnxparser
490
491 extern "C" TENSORRTAPI void* createNvOnnxParser_INTERNAL(void* network, void* logger, int version) noexcept;
492 extern "C" TENSORRTAPI void* createNvOnnxParserRefitter_INTERNAL(
493     void* refitter, void* logger, int32_t version) noexcept;
494 extern "C" TENSORRTAPI int getNvOnnxParserVersion() noexcept;
495
496 namespace nvonnxparser
497 {
498
499 namespace
500 {
501
517 inline IParser* createParser(nvinfer1::INetworkDefinition& network, nvinfer1::ILogger& logger) noexcept
518 {
519     try
520     {
521         return static_cast<IParser*>(createNvOnnxParser_INTERNAL(&network, &logger, NV_ONNX_PARSER_VERSION));
522     }
523     catch (std::exception& e)
524     {
525         logger.log(nvinfer1::ILogger::Severity::kINTERNAL_ERROR, e.what());
526     }
527
528     return nullptr;
529 }
530
540 inline IParserRefitter* createParserRefitter(nvinfer1::IRefitter& refitter, nvinfer1::ILogger& logger)
    noexcept
541 {
542     try
543     {
544         return static_cast<IParserRefitter*>(
545             createNvOnnxParserRefitter_INTERNAL(&refitter, &logger, NV_ONNX_PARSER_VERSION));
546     }
```

```
547     catch (std::exception& e)
548     {
549         logger.log(nvinfer1::ILogger::Severity::kINTERNAL_ERROR, e.what());
550     }
551
552     return nullptr;
553 }
554
555 } // namespace
556
557 } // namespace nvonnxparser
558
559 #endif // NV_ONNX_PARSER_H
```

Index

~IActivationLayer
nvinfer1::IActivationLayer, 128
~IAlgorithm
nvinfer1::IAlgorithm, 132
~IAlgorithmContext
nvinfer1::IAlgorithmContext, 135
~IAlgorithmIOInfo
nvinfer1::IAlgorithmIOInfo, 137
~IAlgorithmSelector
nvinfer1::v_1_0::IAlgorithmSelector, 140
~IAlgorithmVariant
nvinfer1::IAlgorithmVariant, 143
~IAssertionLayer
nvinfer1::IAssertionLayer, 145
~IBuilder
nvinfer1::IBuilder, 147
~IBuilderConfig
nvinfer1::IBuilderConfig, 158
~ICastLayer
nvinfer1::ICastLayer, 184
~IConcatenationLayer
nvinfer1::IConcatenationLayer, 186
~IConditionLayer
nvinfer1::IConditionLayer, 188
~IConstantLayer
nvinfer1::IConstantLayer, 190
~IConvolutionLayer
nvinfer1::IConvolutionLayer, 194
~ICudaEngine
nvinfer1::ICudaEngine, 204
~ICumulativeLayer
nvinfer1::ICumulativeLayer, 229
~IDebugListener
nvinfer1::v_1_0::IDebugListener, 232
~IDeconvolutionLayer
nvinfer1::IDeconvolutionLayer, 235
~IDequantizeLayer
nvinfer1::IDequantizeLayer, 245
~IDimensionExpr
nvinfer1::IDimensionExpr, 248
~IDynamicQuantizeLayer
nvinfer1::IDynamicQuantizeLayer, 250
~IEinsumLayer
nvinfer1::IEinsumLayer, 255
~IElementWiseLayer
nvinfer1::IElementWiseLayer, 257
~IEngineInspector
nvinfer1::IEngineInspector, 259
~IErrorRecorder
nvinfer1::v_1_0::IErrorRecorder, 265
~IExecutionContext
nvinfer1::IExecutionContext, 273
~IExprBuilder
nvinfer1::IExprBuilder, 298
~IFillLayer
nvinfer1::IFillLayer, 302
~IGatherLayer
nvinfer1::IGatherLayer, 311
~IGpuAllocator
nvinfer1::v_1_0::IGpuAllocator, 315
~IGpuAsyncAllocator
nvinfer1::v_1_0::IGpuAsyncAllocator, 321
~IGridSampleLayer
nvinfer1::IGridSampleLayer, 326
~IHostMemory
nvinfer1::IHostMemory, 330
~IIdentityLayer
nvinfer1::IIdentityLayer, 332
~IIfConditional
nvinfer1::IIfConditional, 333
~IIfConditionalBoundaryLayer
nvinfer1::IIfConditionalBoundaryLayer, 337
~IIfConditionalInputLayer
nvinfer1::IIfConditionalInputLayer, 338
~IIfConditionalOutputLayer
nvinfer1::IIfConditionalOutputLayer, 340
~IInt8Calibrator
nvinfer1::IInt8Calibrator, 341
~IInt8EntropyCalibrator
nvinfer1::v_1_0::IInt8EntropyCalibrator, 345
~IInt8EntropyCalibrator2
nvinfer1::v_1_0::IInt8EntropyCalibrator2, 347
~IInt8LegacyCalibrator
nvinfer1::v_1_0::IInt8LegacyCalibrator, 349
~IInt8MinMaxCalibrator
nvinfer1::v_1_0::IInt8MinMaxCalibrator, 352
~IIteratorLayer
nvinfer1::IIteratorLayer, 354
~ILRNLayer
nvinfer1::ILRNLayer, 378

~ILayer
nvinfer1::ILayer, 357
~ILogger
nvinfer1::ILogger, 367
~ILoggerFinder
nvinfer1::ILoggerFinder, 368
~ILoop
nvinfer1::ILoop, 370
~ILoopBoundaryLayer
nvinfer1::ILoopBoundaryLayer, 373
~ILoopOutputLayer
nvinfer1::ILoopOutputLayer, 375
~IMatrixMultiplyLayer
nvinfer1::IMatrixMultiplyLayer, 382
~INMSLayer
nvinfer1::INMSLayer, 433
~INetworkDefinition
nvinfer1::INetworkDefinition, 388
~INoCopy
nvinfer1::INoCopy, 437
~INonZeroLayer
nvinfer1::INonZeroLayer, 439
~INormalizationLayer
nvinfer1::INormalizationLayer, 440
~IOneHotLayer
nvinfer1::IOneHotLayer, 446
~IOnnxConfig
nvonnxparser::IOnnxConfig, 449
~IOptimizationProfile
nvinfer1::IOptimizationProfile, 455
~IPaddingLayer
nvinfer1::IPaddingLayer, 464
~IParametricReLULayer
nvinfer1::IParametricReLULayer, 467
~IParser
nvonnxparser::IParser, 469
~IParserError
nvonnxparser::IParserError, 477
~IParserRefitter
nvonnxparser::IParserRefitter, 480
~IPluginCreator
nvinfer1::v_1_0::IPluginCreator, 485
~IPluginCreatorInterface
nvinfer1::v_1_0::IPluginCreatorInterface, 490
~IPluginCreatorV3One
nvinfer1::v_1_0::IPluginCreatorV3One, 492
~IPluginRegistry
nvinfer1::IPluginRegistry, 497
~IPluginResource
nvinfer1::v_1_0::IPluginResource, 507
~IPluginResourceContext
nvinfer1::IPluginResourceContext, 510
~IPluginV2DynamicExt
nvinfer1::IPluginV2DynamicExt, 526
~IPluginV2Ext
nvinfer1::IPluginV2Ext, 532
~IPluginV2Layer
nvinfer1::IPluginV2Layer, 543
~IPluginV3Layer
nvinfer1::IPluginV3Layer, 546
~IPoolingLayer
nvinfer1::IPoolingLayer, 566
~IProfiler
nvinfer1::v_1_0::IProfiler, 572
~IProgressMonitor
nvinfer1::v_1_0::IProgressMonitor, 574
~IQuantizeLayer
nvinfer1::IQuantizeLayer, 579
~IRaggedSoftMaxLayer
nvinfer1::IRaggedSoftMaxLayer, 581
~IRecurrenceLayer
nvinfer1::IRecurrenceLayer, 583
~IReduceLayer
nvinfer1::IReduceLayer, 585
~IRefitter
nvinfer1::IRefitter, 589
~IResizeLayer
nvinfer1::IResizeLayer, 603
~IReverseSequenceLayer
nvinfer1::IReverseSequenceLayer, 611
~IRuntime
nvinfer1::IRuntime, 614
~IScaleLayer
nvinfer1::IScaleLayer, 625
~IScatterLayer
nvinfer1::IScatterLayer, 630
~ISelectLayer
nvinfer1::ISelectLayer, 632
~ISerializationConfig
nvinfer1::ISerializationConfig, 634
~IShapeLayer
nvinfer1::IShapeLayer, 637
~IShuffleLayer
nvinfer1::IShuffleLayer, 639
~ISliceLayer
nvinfer1::ISliceLayer, 645
~ISoftMaxLayer
nvinfer1::ISoftMaxLayer, 651
~ISqueezeLayer
nvinfer1::ISqueezeLayer, 653
~IStreamReader
nvinfer1::v_1_0::IStreamReader, 655
~IStreamReaderV2
nvinfer1::v_1_0::IStreamReaderV2, 658
~ITensor
nvinfer1::ITensor, 662
~ITimingCache
nvinfer1::ITimingCache, 674

~ITopKLayer
nvinfer1::ITopKLayer, 678
~ITripLimitLayer
nvinfer1::ITripLimitLayer, 682
~IUnaryLayer
nvinfer1::IUnaryLayer, 684
~IUnsqueezeLayer
nvinfer1::IUnsqueezeLayer, 686
~IVersionedInterface
nvinfer1::IVersionedInterface, 688

acquirePluginResource
nvinfer1::IPluginRegistry, 497
ActivationType
nvinfer1, 44
addActivation
nvinfer1::INetworkDefinition, 388
addAssertion
nvinfer1::INetworkDefinition, 388
addCast
nvinfer1::INetworkDefinition, 389
addConcatenation
nvinfer1::INetworkDefinition, 389
addConstant
nvinfer1::INetworkDefinition, 390
addConvolutionNd
nvinfer1::INetworkDefinition, 391
addCumulative
nvinfer1::INetworkDefinition, 391
addDeconvolutionNd
nvinfer1::INetworkDefinition, 392
addDequantize
nvinfer1::INetworkDefinition, 393
addDynamicQuantize
nvinfer1::INetworkDefinition, 394
addEinsum
nvinfer1::INetworkDefinition, 395
addElementWise
nvinfer1::INetworkDefinition, 395
addFill
nvinfer1::INetworkDefinition, 396, 397
addGather
nvinfer1::INetworkDefinition, 398
addGatherV2
nvinfer1::INetworkDefinition, 398
addGridSample
nvinfer1::INetworkDefinition, 399
addIdentity
nvinfer1::INetworkDefinition, 399
addIfConditional
nvinfer1::INetworkDefinition, 400
addInput
nvinfer1::IIfConditional, 334
nvinfer1::INetworkDefinition, 400
addIterator
nvinfer1::ILoop, 370
addLoop
nvinfer1::INetworkDefinition, 401
addLoopOutput
nvinfer1::ILoop, 370
addLRN
nvinfer1::INetworkDefinition, 401
addMatrixMultiply
nvinfer1::INetworkDefinition, 402
addNMS
nvinfer1::INetworkDefinition, 403
addNonZero
nvinfer1::INetworkDefinition, 403
addNormalization
nvinfer1::INetworkDefinition, 404
addOneHot
nvinfer1::INetworkDefinition, 404
addOptimizationProfile
nvinfer1::IBuilderConfig, 159
addOutput
nvinfer1::IIfConditional, 334
addPaddingNd
nvinfer1::INetworkDefinition, 405
addParametricReLU
nvinfer1::INetworkDefinition, 406
addPluginV2
nvinfer1::INetworkDefinition, 406
addPluginV3
nvinfer1::INetworkDefinition, 407
addPoolingNd
nvinfer1::INetworkDefinition, 407
addQuantize
nvinfer1::INetworkDefinition, 408, 409
addRaggedSoftMax
nvinfer1::INetworkDefinition, 409
addRecurrence
nvinfer1::ILoop, 370
addReduce
nvinfer1::INetworkDefinition, 410
addResize
nvinfer1::INetworkDefinition, 411
addReverseSequence
nvinfer1::INetworkDefinition, 411
addScale
nvinfer1::INetworkDefinition, 412
addScaleNd
nvinfer1::INetworkDefinition, 413
addScatter
nvinfer1::INetworkDefinition, 414
addSelect
nvinfer1::INetworkDefinition, 414
addShape
nvinfer1::INetworkDefinition, 415

addShuffle
nvinfer1::INetworkDefinition, 416
addSlice
nvinfer1::INetworkDefinition, 416
addSoftMax
nvinfer1::INetworkDefinition, 417
addSqueeze
nvinfer1::INetworkDefinition, 417
addTopK
nvinfer1::INetworkDefinition, 418
addTripLimit
nvinfer1::ILoop, 371
addUnary
nvinfer1::INetworkDefinition, 419
addUnsqueeze
nvinfer1::INetworkDefinition, 419
addVerbosity
nvonnxparser::IOnnxConfig, 449
allInputDimensionsSpecified
nvinfer1::IExecutionContext, 274
allInputShapesSpecified
nvinfer1::IExecutionContext, 274
allocate
nvinfer1::v_1_0::IGpuAllocator, 316
nvinfer1::v_1_0::IGpuAsyncAllocator, 322
allocateAsync
nvinfer1::v_1_0::IGpuAllocator, 316
nvinfer1::v_1_0::IGpuAsyncAllocator, 322
AllocatorFlag
nvinfer1, 45
AllocatorFlags
nvinfer1, 38
anchorsRatioCount
nvinfer1::plugin::RPROIParams, 703
anchorsScaleCount
nvinfer1::plugin::RPROIParams, 703
APILanguage
nvinfer1, 45
areWeightsMarkedRefittable
nvinfer1::INetworkDefinition, 420
AsciiChar
nvinfer1, 38
aspectRatios
nvinfer1::plugin::GridAnchorParameters, 126
nvinfer1::plugin::PriorBoxParameters, 698
attachToContext
nvinfer1::IPluginV2Ext, 532
nvinfer1::v_1_0::IPluginV3OneRuntime, 561

backgroundLabelId
nvinfer1::plugin::DetectionOutputParameters, 96
nvinfer1::plugin::NMSParameters, 690
BoundingBoxFormat
nvinfer1, 45

buildEngineWithConfig
nvinfer1::IBuilder, 148
BuilderFlag
nvinfer1, 45
BuilderFlags
nvinfer1, 38
buildSerializedNetwork
nvinfer1::IBuilder, 148

CalibrationAlgoType
nvinfer1, 49
canBroadcastInputAcrossBatch
nvinfer1::IPluginV2Ext, 533
canRunOnDLA
nvinfer1::IBuilderConfig, 159
CENTER_SIZE
nvinfer1::plugin, 88
char_t
nvinfer1, 38
child
nvinfer1::plugin::softmaxTree, 705
classes
nvinfer1::plugin::RegionParameters, 701
clear
nvinfer1::v_1_0::IErrorRecorder, 265
clearErrors
nvonnxparser::IParser, 469
nvonnxparser::IParserRefitter, 480
clearFlag
nvinfer1::IBuilderConfig, 159
nvinfer1::ISerializationConfig, 634
nvonnxparser::IParser, 469
clearQuantizationFlag
nvinfer1::IBuilderConfig, 160
clip
nvinfer1::plugin::PriorBoxParameters, 698
clone
nvinfer1::IPluginV2, 513
nvinfer1::IPluginV2DynamicExt, 526
nvinfer1::IPluginV2Ext, 534
nvinfer1::v_1_0::IPluginResource, 508
nvinfer1::v_1_0::IPluginV3, 544
code
nvonnxparser::IParserError, 477
codeType
nvinfer1::plugin::DetectionOutputParameters, 96
CodeTypeSSD
nvinfer1::plugin, 88
combine
nvinfer1::ITimingCache, 674
confidenceThreshold
nvinfer1::plugin::DetectionOutputParameters, 96
configurePlugin
nvinfer1::IPluginV2DynamicExt, 526

nvinfer1::IPluginV2Ext, 534
nvinfer1::IPluginV2IOExt, 539
nvinfer1::v_1_0::IPluginV3OneBuild, 549
configureWithFormat
nvinfer1::IPluginV2, 514
nvinfer1::IPluginV2Ext, 536
confSigmoid
nvinfer1::plugin::DetectionOutputParameters, 96
constant
nvinfer1::IExprBuilder, 298
coords
nvinfer1::plugin::RegionParameters, 702
CORNER
nvinfer1::plugin, 88
CORNER_SIZE
nvinfer1::plugin, 88
count
nvinfer1::Weights, 710
createBuilderConfig
nvinfer1::IBuilder, 149
createEngineInspector
nvinfer1::ICudaEngine, 205
createExecutionContext
nvinfer1::ICudaEngine, 205
createExecutionContextWithoutDeviceMemory
nvinfer1::ICudaEngine, 205
createInferBuilder
nvinfer1::anonymous_namespace{NvInfer.h}, 85
createInferRefitter
nvinfer1::anonymous_namespace{NvInferRuntime.h}, 86
createInferRuntime
nvinfer1::anonymous_namespace{NvInferRuntime.h}, 86
createNetworkV2
nvinfer1::IBuilder, 149
createNvOnnxParser_INTERNAL
NvOnnxParser.h, 843
createNvOnnxParserRefitter_INTERNAL
NvOnnxParser.h, 843
createONNXConfig
nvonnxparser, 92
createOptimizationProfile
nvinfer1::IBuilder, 150
createParser
nvonnxparser::anonymous_namespace{NvOnnxParser.h}, 93
createParserRefitter
nvonnxparser::anonymous_namespace{NvOnnxParser.h}, 94
createPlugin
nvinfer1::v_1_0::IPluginCreator, 485
nvinfer1::v_1_0::IPluginCreatorV3One, 493
createSerializationConfig
nvinfer1::ICudaEngine, 205
createTimingCache
nvinfer1::IBuilderConfig, 160
CumulativeOperation
nvinfer1, 49

d
nvinfer1::Dims64, 103
nvinfer1::DimsExprs, 104
data
nvinfer1::IHostMemory, 330
nvinfer1::PluginField, 693
nvinfer1::v_1_0::TimingCacheKey, 707
DataType
nvinfer1, 49
deallocate
nvinfer1::v_1_0::IGpuAllocator, 317
nvinfer1::v_1_0::IGpuAsyncAllocator, 323
deallocateAsync
nvinfer1::v_1_0::IGpuAllocator, 318
nvinfer1::v_1_0::IGpuAsyncAllocator, 324
declareSizeTensor
nvinfer1::IExprBuilder, 298
decRefCount
nvinfer1::v_1_0::IErrorRecorder, 266
deregisterCreator
nvinfer1::IPluginRegistry, 498
deregisterLibrary
nvinfer1::IPluginRegistry, 499
desc
nvinfer1::DynamicPluginTensorDesc, 108
nvonnxparser::IParserError, 478
deserializeCudaEngine
nvinfer1::IRuntime, 615, 616
deserializePlugin
nvinfer1::v_1_0::IPluginCreator, 485
destroy
nvinfer1::IPluginV2, 515
detachFromContext
nvinfer1::IPluginV2Ext, 536
DeviceType
nvinfer1, 50
DimensionOperation
nvinfer1, 50
Dims, 98
nvinfer1, 39
dims
nvinfer1::PluginTensorDesc, 696
Dims2
nvinfer1::Dims2, 99
Dims3
nvinfer1::Dims3, 100, 101
Dims4
nvinfer1::Dims4, 102

DimsHW
nvinfer1::DimsHW, 106
dynamicRangeIsSet
nvinfer1::ITensor, 663

ElementWiseOperation
nvinfer1, 51
EngineCapability
nvinfer1, 52
enqueue
nvinfer1::IPluginV2, 515
nvinfer1::IPluginV2DynamicExt, 528
nvinfer1::v_1_0::IPluginV3OneRuntime, 562
enqueueV3
nvinfer1::IExecutionContext, 274
EnumMax
nvinfer1, 77
nvonnxparser, 92
EnumMax< BoundingBoxFormat >
nvinfer1, 77
EnumMax< BuilderFlag >
nvinfer1, 77
EnumMax< CalibrationAlgoType >
nvinfer1, 78
EnumMax< DeviceType >
nvinfer1, 78
EnumMax< DimensionOperation >
nvinfer1, 78
EnumMax< ErrorCode >
nvonnxparser, 93
EnumMax< ExecutionContextAllocationStrategy >
nvinfer1, 78
EnumMax< FillOperation >
nvinfer1, 79
EnumMax< GatherMode >
nvinfer1, 79
EnumMax< LayerInformationFormat >
nvinfer1, 79
EnumMax< LayerType >
nvinfer1, 79
EnumMax< LoopOutput >
nvinfer1, 80
EnumMax< MatrixOperation >
nvinfer1, 80
EnumMax< MemoryPoolType >
nvinfer1, 80
EnumMax< NetworkDefinitionCreationFlag >
nvinfer1, 80
EnumMax< OnnxParserFlag >
nvonnxparser, 93
EnumMax< OptProfileSelector >
nvinfer1, 81
EnumMax< ProfilingVerbosity >
nvinfer1, 81
EnumMax< QuantizationFlag >
nvinfer1, 81
EnumMax< ReduceOperation >
nvinfer1, 81
EnumMax< SampleMode >
nvinfer1, 82
EnumMax< ScaleMode >
nvinfer1, 82
EnumMax< ScatterMode >
nvinfer1, 82
EnumMax< SerializationFlag >
nvinfer1, 82
EnumMax< TacticSource >
nvinfer1, 83
EnumMax< TempfileControlFlag >
nvinfer1, 83
EnumMax< TopKOperation >
nvinfer1, 83
EnumMax< TripLimit >
nvinfer1, 83
EnumMax< UnaryOperation >
nvinfer1, 84
EnumMax< WeightsRole >
nvinfer1, 84
ErrorCode
nvinfer1, 52
nvonnxparser, 91
ErrorDesc
nvinfer1::v_1_0::IErrorRecorder, 265
executeV2
nvinfer1::IExecutionContext, 275
ExecutionContextAllocationStrategy
nvinfer1, 53

featureStride
nvinfer1::plugin::RPROIParams, 703
fields
nvinfer1::PluginFieldCollection, 694
file
nvonnxparser::IParserError, 478
FillOperation
nvinfer1, 54
findLogger
nvinfer1::ILoggerFinder, 368
flip
nvinfer1::plugin::PriorBoxParameters, 699
format
nvinfer1::PluginTensorDesc, 696
func
nvonnxparser::IParserError, 478

GatherMode
nvinfer1, 55
getActivationType
nvinfer1::IActivationLayer, 129

getAlgorithm
nvinfer1::IInt8Calibrator, 341
nvinfer1::v_1_0::IInt8EntropyCalibrator, 345
nvinfer1::v_1_0::IInt8EntropyCalibrator2, 347
nvinfer1::v_1_0::IInt8LegacyCalibrator, 349
nvinfer1::v_1_0::IInt8MinMaxCalibrator, 352
getAlgorithmIOInfoByIndex
nvinfer1::IAlgorithm, 132
getAlgorithmSelector
nvinfer1::IBuilderConfig, 161
getAlgorithmVariant
nvinfer1::IAlgorithm, 133
getAliasedInput
nvinfer1::v_2_0::IPluginV3OneBuild, 556
getAlignCorners
nvinfer1::IGridSampleLayer, 327
getAll
nvinfer1::IRefitter, 589
getAllCreators
nvinfer1::IPluginRegistry, 499
getAllCreatorsRecursive
nvinfer1::IPluginRegistry, 500
getAllowedFormats
nvinfer1::ITensor, 663
getAllWeights
nvinfer1::IRefitter, 590
getAlpha
nvinfer1::IActivationLayer, 129
nvinfer1::IFillLayer, 302
nvinfer1::ILRNLayer, 378
getAlphaInt64
nvinfer1::IFillLayer, 302
getAPILanguage
nvinfer1::IVersionedInterface, 688
getAverageCountExcludesPadding
nvinfer1::IPoolingLayer, 566
getAvgTimingIterations
nvinfer1::IBuilderConfig, 161
getAxes
nvinfer1::INormalizationLayer, 441
nvinfer1::ISliceLayer, 646
nvinfer1::ISoftMaxLayer, 652
getAxis
nvinfer1::IConcatenationLayer, 187
nvinfer1::IDequantizeLayer, 245
nvinfer1::IDynamicQuantizeLayer, 250
nvinfer1::IIteratorLayer, 354
nvinfer1::ILoopOutputLayer, 375
nvinfer1::IOneHotLayer, 446
nvinfer1::IQuantizeLayer, 579
nvinfer1::IScatterLayer, 630
getBatch
nvinfer1::IInt8Calibrator, 342
getBatchAxis
nvinfer1::IReverseSequenceLayer, 611
getBatchSize
nvinfer1::IInt8Calibrator, 342
getBeta
nvinfer1::IActivationLayer, 129
nvinfer1::IFillLayer, 303
nvinfer1::ILRNLayer, 378
getBetaInt64
nvinfer1::IFillLayer, 303
getBiasWeights
nvinfer1::IConvolutionLayer, 194
nvinfer1::IDeconvolutionLayer, 235
getBlendFactor
nvinfer1::IPoolingLayer, 566
getBlockSize
nvinfer1::IDynamicQuantizeLayer, 251
getBoundingBoxFormat
nvinfer1::INMSLayer, 433
getBroadcastAcrossBatch
nvinfer1::ITensor, 663
getBuilder
nvinfer1::INetworkDefinition, 420
getBuilderOptimizationLevel
nvinfer1::IBuilderConfig, 161
getBuilderPluginRegistry
nvinfer1, 84
getBuilderSafePluginRegistry
nvinfer1, 84
getCalibrationProfile
nvinfer1::IBuilderConfig, 162
getCapabilityInterface
nvinfer1::v_1_0::IPluginV3, 545
getChannelAxis
nvinfer1::IScaleLayer, 625
getComponentsPerElement
nvinfer1::IAlgorithmIOInfo, 138
getComputePrecision
nvinfer1::INormalizationLayer, 441
getConditional
nvinfer1::IIfConditionalBoundaryLayer, 337
getConstantValue
nvinfer1::IDimensionExpr, 248
getCoordinateTransformation
nvinfer1::IResizeLayer, 604
getCreator
nvinfer1::IPluginRegistry, 500
getCubicCoeff
nvinfer1::IResizeLayer, 604
getDataType
nvinfer1::IAlgorithmIOInfo, 138
getDebugListener
nvinfer1::IExecutionContext, 275
getDebugState
nvinfer1::IExecutionContext, 276

getDebugSync
nvinfer1::IExecutionContext, 276
getDefaultDeviceType
nvinfer1::IBuilderConfig, 162
getDeviceMemorySize
nvinfer1::ICudaEngine, 206
getDeviceMemorySizeForProfile
nvinfer1::ICudaEngine, 206
getDeviceMemorySizeForProfileV2
nvinfer1::ICudaEngine, 206
getDeviceMemorySizeV2
nvinfer1::ICudaEngine, 207
getDeviceType
nvinfer1::IBuilderConfig, 162
getDilationNd
nvinfer1::IConvolutionLayer, 194
nvinfer1::IDeconvolutionLayer, 236
getDimensionName
nvinfer1::ITensor, 663
getDimensions
nvinfer1::IAlgorithmContext, 135
nvinfer1::IConstantLayer, 190
nvinfer1::IFillLayer, 303
nvinfer1::IOptimizationProfile, 455
nvinfer1::ITensor, 664
getDLACore
nvinfer1::IBuilderConfig, 162
nvinfer1::IRuntime, 616
getDynamicRangeMax
nvinfer1::IRefitter, 590
nvinfer1::ITensor, 664
getDynamicRangeMin
nvinfer1::IRefitter, 591
nvinfer1::ITensor, 664
getEngine
nvinfer1::IExecutionContext, 276
getEngineCapability
nvinfer1::IBuilderConfig, 163
nvinfer1::ICudaEngine, 207
getEngineHostCodeAllowed
nvinfer1::IRuntime, 616
getEngineInformation
nvinfer1::IEngineInspector, 260
getEnqueueEmitsProfile
nvinfer1::IExecutionContext, 276
getEpsilon
nvinfer1::INormalizationLayer, 441
getEquation
nvinfer1::IEinsumLayer, 255
getError
nvonnxparser::IParser, 469
nvonnxparser::IParserRefitter, 480
getErrorCode
nvinfer1::v_1_0::IErrorRecorder, 266
getErrorDesc
nvinfer1::v_1_0::IErrorRecorder, 267
getErrorRecorder
nvinfer1::IBuilder, 150
nvinfer1::ICudaEngine, 208
nvinfer1::IEngineInspector, 260
nvinfer1::IExecutionContext, 277
nvinfer1::INetworkDefinition, 421
nvinfer1::IPluginRegistry, 501
nvinfer1::IPluginResourceContext, 511
nvinfer1::IRefitter, 591
nvinfer1::IRuntime, 617
getExcludeOutside
nvinfer1::IResizeLayer, 604
getExclusive
nvinfer1::ICumulativeLayer, 229
getExecutionContext
nvinfer1::IEngineInspector, 261
getExtraMemoryTarget
nvinfer1::IOptimizationProfile, 456
getFieldNames
nvinfer1::v_1_0::IPluginCreator, 486
nvinfer1::v_1_0::IPluginCreatorV3One, 493
getFieldsToSerialize
nvinfer1::v_1_0::IPluginV3OneRuntime, 562
getFirstTranspose
nvinfer1::IShuffleLayer, 639
getFlag
nvinfer1::IBuilderConfig, 163
nvinfer1::INetworkDefinition, 421
nvinfer1::ISerializationConfig, 634
nvonnxparser::IParser, 470
getFlags
nvinfer1::IBuilderConfig, 163
nvinfer1::INetworkDefinition, 421
nvinfer1::ISerializationConfig, 635
nvonnxparser::IParser, 470
getFormatCombinationLimit
nvinfer1::v_1_0::IPluginV3OneBuild, 550
getFullTextFileName
nvonnxparser::IOnnxConfig, 449
getGatherAxis
nvinfer1::IGatherLayer, 311
getGpuAllocator
nvinfer1::IPluginResourceContext, 511
getHardwareCompatibilityLevel
nvinfer1::IBuilderConfig, 164
nvinfer1::ICudaEngine, 208
getImplementation
nvinfer1::IAlgorithmVariant, 143
getInferLibBuildVersion
NvInferRuntime.h, 797
getInferLibMajorVersion
NvInferRuntime.h, 797

getInferLibMinorVersion
NvInferRuntime.h, 797
getInferLibPatchVersion
NvInferRuntime.h, 797
getInferLibVersion
NvInferRuntimeBase.h, 824
getInput
nvinfer1::ILayer, 358
nvinfer1::INetworkDefinition, 422
getInputConsumedEvent
nvinfer1::IExecutionContext, 277
getInt8Calibrator
nvinfer1::IBuilderConfig, 164
getInterfaceInfo
nvinfer1::IVersionedInterface, 689
nvinfer1::v_1_0::IAlgorithmSelector, 140
nvinfer1::v_1_0::IDebugListener, 232
nvinfer1::v_1_0::IErrorRecorder, 268
nvinfer1::v_1_0::IGpuAllocator, 319
nvinfer1::v_1_0::IGpuAsyncAllocator, 325
nvinfer1::v_1_0::IInt8EntropyCalibrator, 345
nvinfer1::v_1_0::IInt8EntropyCalibrator2, 347
nvinfer1::v_1_0::IInt8LegacyCalibrator, 349
nvinfer1::v_1_0::IInt8MinMaxCalibrator, 352
nvinfer1::v_1_0::IOutputAllocator, 461
nvinfer1::v_1_0::IPluginCreator, 486
nvinfer1::v_1_0::IPluginCreatorV3One, 494
nvinfer1::v_1_0::IPluginResource, 508
nvinfer1::v_1_0::IPluginV3, 545
nvinfer1::v_1_0::IPluginV3OneBuild, 550
nvinfer1::v_1_0::IPluginV3OneCore, 558
nvinfer1::v_1_0::IPluginV3OneRuntime, 563
nvinfer1::v_1_0::IProgressMonitor, 575
nvinfer1::v_1_0::IStreamReader, 656
nvinfer1::v_1_0::IStreamReaderV2, 659
nvinfer1::v_2_0::IPluginV3OneBuild, 556
getInterpolationMode
nvinfer1::IGridSampleLayer, 327
getIOTensorName
nvinfer1::ICudaEngine, 208
getK
nvinfer1::ILRNLayer, 378
nvinfer1::ITopKLayer, 679
getKeepDimensions
nvinfer1::IReduceLayer, 585
getKernelSizeNd
nvinfer1::IConvolutionLayer, 194
nvinfer1::IDeconvolutionLayer, 236
getKernelWeights
nvinfer1::IConvolutionLayer, 195
nvinfer1::IDeconvolutionLayer, 236
getL2LimitForTiling
nvinfer1::IBuilderConfig, 164
getLayer
nvinfer1::INetworkDefinition, 422
getLayerInformation
nvinfer1::IEngineInspector, 261
getLayerOutputTensor
nvonnxparser::IParser, 470
getLocation
nvinfer1::ITensor, 665
getLogger
nvinfer1::IBuilder, 150
nvinfer1::IRefitter, 591
nvinfer1::IRuntime, 617
NvInferRuntime.h, 798
getLoop
nvinfer1::ILoopBoundaryLayer, 373
getLoopOutput
nvinfer1::ILoopOutputLayer, 375
getMaxAuxStreams
nvinfer1::IBuilderConfig, 165
getMaxDLABatchSize
nvinfer1::IBuilder, 151
getMaxNbTactics
nvinfer1::IBuilderConfig, 165
getMaxOutputSize
nvinfer1::IExecutionContext, 277
getMaxThreads
nvinfer1::IBuilder, 151
nvinfer1::IRefitter, 592
nvinfer1::IRuntime, 617
getMemoryPoolLimit
nvinfer1::IBuilderConfig, 165
getMessage
nvinfer1::IAssertionLayer, 145
getMetadata
nvinfer1::ILayer, 358
getMetadataString
nvinfer1::v_1_0::IPluginV3OneBuild, 550
getMinimumWeightStreamingBudget
nvinfer1::ICudaEngine, 209
getMissing
nvinfer1::IRefitter, 592
getMissingWeights
nvinfer1::IRefitter, 593
getMode
nvinfer1::IGatherLayer, 312
nvinfer1::IScaleLayer, 625
nvinfer1::IScatterLayer, 630
nvinfer1::ISliceLayer, 646
getModelDtype
nvonnxparser::IOnnxConfig, 449
getModelFileName
nvonnxparser::IOnnxConfig, 450
getName
nvinfer1::IAlgorithmContext, 135
nvinfer1::ICudaEngine, 209

nvinfer1::IExecutionContext, 278
nvinfer1::IIfConditional, 335
nvinfer1::ILayer, 358
nvinfer1::ILoop, 371
nvinfer1::INetworkDefinition, 423
nvinfer1::ITensor, 665
getNamedWeights
nvinfer1::IRefitter, 593
getNbAuxStreams
nvinfer1::ICudaEngine, 209
getNbDLACores
nvinfer1::IBuilder, 151
nvinfer1::IRuntime, 618
getNbElementWiseDims
nvinfer1::IGatherLayer, 312
getNbErrors
nvinfer1::v_1_0::IErrorRecorder, 268
nvonnxparser::IParser, 471
nvonnxparser::IParserRefitter, 481
getNbGroups
nvinfer1::IConvolutionLayer, 195
nvinfer1::IDeconvolutionLayer, 236
nvinfer1::INormalizationLayer, 441
getNbInputs
nvinfer1::IAlgorithmContext, 135
nvinfer1::ILayer, 358
nvinfer1::INetworkDefinition, 423
getNbIOTensors
nvinfer1::ICudaEngine, 210
getNbLayers
nvinfer1::ICudaEngine, 210
nvinfer1::INetworkDefinition, 423
getNbOptimizationProfiles
nvinfer1::IBuilderConfig, 166
nvinfer1::ICudaEngine, 210
getNbOutputMaps
nvinfer1::IConvolutionLayer, 195
nvinfer1::IDeconvolutionLayer, 237
getNbOutputs
nvinfer1::IAlgorithmContext, 136
nvinfer1::ILayer, 359
nvinfer1::INetworkDefinition, 424
nvinfer1::IPluginV2, 516
nvinfer1::v_1_0::IPluginV3OneBuild, 550
getNbPluginsToSerialize
nvinfer1::IBuilderConfig, 166
getNbShapeValues
nvinfer1::IOptimizationProfile, 456
getNbSubgraphs
nvonnxparser::IParser, 471
getNbTactics
nvinfer1::v_1_0::IPluginV3OneBuild, 551
getNearestRounding
nvinfer1::IResizeLayer, 604
getNvOnnxParserVersion
NvOnnxParser.h, 844
getNvtxVerbosity
nvinfer1::IExecutionContext, 278
getOperation
nvinfer1::ICumulativeLayer, 229
nvinfer1::IElementWiseLayer, 257
nvinfer1::IFillLayer, 304
nvinfer1::IMatrixMultiplyLayer, 382
nvinfer1::IReduceLayer, 585
nvinfer1::ITopKLayer, 679
nvinfer1::IUnaryLayer, 684
getOptimizationProfile
nvinfer1::IExecutionContext, 278
getOutput
nvinfer1::ILayer, 359
nvinfer1::INetworkDefinition, 424
getOutputAllocator
nvinfer1::IExecutionContext, 279
getOutputDataType
nvinfer1::IPluginV2Ext, 536
getOutputDataTypes
nvinfer1::v_1_0::IPluginV3OneBuild, 551
getOutputDimensions
nvinfer1::IPluginV2, 516
nvinfer1::IPluginV2DynamicExt, 528
nvinfer1::IResizeLayer, 605
getOutputShapes
nvinfer1::v_1_0::IPluginV3OneBuild, 552
getOutputTensorAddress
nvinfer1::IExecutionContext, 279
getOutputType
nvinfer1::ILayer, 359
getPaddingMode
nvinfer1::IConvolutionLayer, 195
nvinfer1::IDeconvolutionLayer, 237
nvinfer1::IPoolingLayer, 566
getPaddingNd
nvinfer1::IConvolutionLayer, 196
nvinfer1::IDeconvolutionLayer, 237
nvinfer1::IPoolingLayer, 567
getPersistentCacheLimit
nvinfer1::IExecutionContext, 280
getPlugin
nvinfer1::IPluginV2Layer, 543
nvinfer1::IPluginV3Layer, 547
getPluginCreator
nvinfer1::IPluginRegistry, 501
getPluginCreatorList
nvinfer1::IPluginRegistry, 502
getPluginName
nvinfer1::v_1_0::IPluginCreator, 487
nvinfer1::v_1_0::IPluginCreatorV3One, 494
nvinfer1::v_1_0::IPluginV3OneCore, 559

getPluginNamespace
nvinfer1::IPluginV2, 518
nvinfer1::v_1_0::IPluginCreator, 487
nvinfer1::v_1_0::IPluginCreatorV3One, 494
nvinfer1::v_1_0::IPluginV3OneCore, 559
getPluginRegistry
nvinfer1::IBuilder, 152
nvinfer1::IRuntime, 618
NvInferRuntime.h, 798
getPluginToSerialize
nvinfer1::IBuilderConfig, 166
getPluginType
nvinfer1::IPluginV2, 518
getPluginVersion
nvinfer1::IPluginV2, 519
nvinfer1::v_1_0::IPluginCreator, 487
nvinfer1::v_1_0::IPluginCreatorV3One, 495
nvinfer1::v_1_0::IPluginV3OneCore, 559
getPoolingType
nvinfer1::IPoolingLayer, 567
getPostPadding
nvinfer1::IConvolutionLayer, 196
nvinfer1::IDeconvolutionLayer, 237
nvinfer1::IPoolingLayer, 567
getPostPaddingNd
nvinfer1::IPaddingLayer, 464
getPower
nvinfer1::IScaleLayer, 626
getPrecision
nvinfer1::ILayer, 360
getPrePadding
nvinfer1::IConvolutionLayer, 196
nvinfer1::IDeconvolutionLayer, 238
nvinfer1::IPoolingLayer, 567
getPrePaddingNd
nvinfer1::IPaddingLayer, 464
getPreviewFeature
nvinfer1::IBuilderConfig, 167
getPrintLayerInfo
nvonnxparser::IOnnxConfig, 450
getProfiler
nvinfer1::IExecutionContext, 280
getProfileShape
nvinfer1::ICudaEngine, 211
getProfileStream
nvinfer1::IBuilderConfig, 167
getProfileTensorValues
nvinfer1::ICudaEngine, 211
getProfilingVerbosity
nvinfer1::IBuilderConfig, 167
nvinfer1::ICudaEngine, 212
getProgressMonitor
nvinfer1::IBuilderConfig, 168
getQuantile
nvinfer1::v_1_0::IInt8LegacyCalibrator, 349
getQuantizationFlag
nvinfer1::IBuilderConfig, 168
getQuantizationFlags
nvinfer1::IBuilderConfig, 168
getReduceAxes
nvinfer1::IReduceLayer, 586
nvinfer1::ITopKLayer, 679
getRegressionCutoff
nvinfer1::v_1_0::IInt8LegacyCalibrator, 349
getReshapeDimensions
nvinfer1::IShuffleLayer, 639
getResizeMode
nvinfer1::IResizeLayer, 605
getReverse
nvinfer1::ICumulativeLayer, 229
nvinfer1::IIteratorLayer, 354
getRuntimePlatform
nvinfer1::IBuilderConfig, 169
getSampleMode
nvinfer1::IGridSampleLayer, 327
getScale
nvinfer1::IScaleLayer, 626
getScales
nvinfer1::IResizeLayer, 605
getScaleType
nvinfer1::IDynamicQuantizeLayer, 251
getSecondTranspose
nvinfer1::IShuffleLayer, 640
getSelectorForSinglePixel
nvinfer1::IResizeLayer, 606
getSequenceAxis
nvinfer1::IReverseSequenceLayer, 612
getSerializationSize
nvinfer1::IPluginV2, 519
getShapeValues
nvinfer1::IOptimizationProfile, 456
getShift
nvinfer1::IScaleLayer, 626
getSize
nvinfer1::ISliceLayer, 646
getStart
nvinfer1::ISliceLayer, 647
getStreamableWeightsSize
nvinfer1::ICudaEngine, 212
getStride
nvinfer1::ISliceLayer, 647
getStrideNd
nvinfer1::IConvolutionLayer, 196
nvinfer1::IDeconvolutionLayer, 238
nvinfer1::IPoolingLayer, 568
getStrides
nvinfer1::IAlgorithmIOInfo, 138
getSubgraphNodes

nvonnxparser::IParser, 471
getTactic
nvinfer1::IAlgorithmVariant, 143
getTacticSources
nvinfer1::IBuilderConfig, 169
nvinfer1::ICudaEngine, 212
getTempfileControlFlags
nvinfer1::IRuntime, 618
getTemporaryDirectory
nvinfer1::IRuntime, 618
getTemporaryStorageAllocator
nvinfer1::IExecutionContext, 280
getTensorAddress
nvinfer1::IExecutionContext, 280
getTensorBytesPerComponent
nvinfer1::ICudaEngine, 213
getTensorComponentsPerElement
nvinfer1::ICudaEngine, 214
getTensorDataType
nvinfer1::ICudaEngine, 215
getTensorFormat
nvinfer1::ICudaEngine, 215, 216
getTensorFormatDesc
nvinfer1::ICudaEngine, 216, 217
getTensorIOMode
nvinfer1::ICudaEngine, 217
getTensorLocation
nvinfer1::ICudaEngine, 218
getTensorRTVersion
nvinfer1::IPluginV2, 520
nvinfer1::IPluginV2DynamicExt, 529
nvinfer1::IPluginV2Ext, 537
nvinfer1::IPluginV2IOExt, 540
getTensorShape
nvinfer1::ICudaEngine, 218
nvinfer1::IExecutionContext, 281
getTensorStrides
nvinfer1::IExecutionContext, 282
getTensorsWithDynamicRange
nvinfer1::IRefitter, 594
getTensorVectorizedDim
nvinfer1::ICudaEngine, 219
getTextFileName
nvonnxparser::IOnnxConfig, 450
getTilingOptimizationLevel
nvinfer1::IBuilderConfig, 169
getTimingCache
nvinfer1::IBuilderConfig, 170
getTimingCacheID
nvinfer1::v_1_0::IPluginV3OneBuild, 552
getTimingMSec
nvinfer1::IAlgorithm, 133
getTopKBoxLimit
nvinfer1::INMSLayer, 433
getToType
nvinfer1::ICastLayer, 185
nvinfer1::IDequantizeLayer, 245
nvinfer1::IDynamicQuantizeLayer, 251
nvinfer1::IFillLayer, 304
nvinfer1::IQuantizeLayer, 579
getTripLimit
nvinfer1::ITripLimitLayer, 682
getType
nvinfer1::ILayer, 360
nvinfer1::ITensor, 665
getUsedVCPluginLibraries
nvonnxparser::IParser, 472
getValidTactics
nvinfer1::v_1_0::IPluginV3OneBuild, 553
getVectorizedDim
nvinfer1::IAlgorithmIOInfo, 138
getVerbosityLevel
nvonnxparser::IOnnxConfig, 451
getWeights
nvinfer1::IConstantLayer, 190
getWeightsLocation
nvinfer1::IRefitter, 594
getWeightsPrototype
nvinfer1::IRefitter, 595
getWeightStreamingAutomaticBudget
nvinfer1::ICudaEngine, 220
getWeightStreamingBudget
nvinfer1::ICudaEngine, 220
getWeightStreamingBudgetV2
nvinfer1::ICudaEngine, 221
getWeightStreamingScratchMemorySize
nvinfer1::ICudaEngine, 221
getWeightsValidation
nvinfer1::IRefitter, 595
getWindowSize
nvinfer1::ILRNLayer, 379
getWindowSizeNd
nvinfer1::IPoolingLayer, 568
getWorkspaceSize
nvinfer1::IAlgorithm, 133
nvinfer1::IPluginV2, 520
nvinfer1::IPluginV2DynamicExt, 529
nvinfer1::v_1_0::IPluginV3OneBuild, 554
getZeroIsPlaceholder
nvinfer1::IShuffleLayer, 640
group
nvinfer1::plugin::softmaxTree, 706
groupOffset
nvinfer1::plugin::softmaxTree, 706
groups
nvinfer1::plugin::softmaxTree, 706
groupSize
nvinfer1::plugin::softmaxTree, 706

H
nvinfer1::plugin::GridAnchorParameters, 126
h
nvinfer1::DimsHW, 106
HardwareCompatibilityLevel
nvinfer1, 55
hasImplicitBatchDimension
nvinfer1::ICudaEngine, 222
nvinfer1::INetworkDefinition, 425
hasOverflowed
nvinfer1::v_1_0::IErrorRecorder, 268

IAlgorithmSelector, 139
nvinfer1, 39
IDebugListener, 231
nvinfer1, 39
IErrorRecorder, 263
nvinfer1, 39
nvinfer1::v_1_0::IErrorRecorder, 265
IGpuAllocator, 314
nvinfer1, 39
nvinfer1::v_1_0::IGpuAllocator, 315
IGpuAsyncAllocator, 320
nvinfer1, 39
nvinfer1::v_1_0::IGpuAsyncAllocator, 321
IInt8EntropyCalibrator, 344
nvinfer1, 39
IInt8EntropyCalibrator2, 346
nvinfer1, 40
IInt8LegacyCalibrator, 347
nvinfer1, 40
IInt8MinMaxCalibrator, 351
nvinfer1, 40
ILogger
nvinfer1::ILogger, 366
imgH
nvinfer1::plugin::PriorBoxParameters, 699
imgW
nvinfer1::plugin::PriorBoxParameters, 699
incRefCount
nvinfer1::v_1_0::IErrorRecorder, 269
inferShapes
nvinfer1::IExecutionContext, 282
initialize
nvinfer1::IPluginV2, 521
initLibNvInferPlugins
NvInferPlugin.h, 785
INoCopy
nvinfer1::INoCopy, 436, 437
INonZero, 438
inputOrder
nvinfer1::plugin::DetectionOutputParameters, 96
InterfaceKind
nvinfer1, 40
InterpolationMode
nvinfer1, 56
iouThreshold
nvinfer1::plugin::NMSParameters, 690
nvinfer1::plugin::RPROIParams, 704
IOutputAllocator, 460
nvinfer1, 40
IPluginCapability, 482
nvinfer1, 40
IPluginCreator, 483
nvinfer1, 40
nvinfer1::v_1_0::IPluginCreator, 484
IPluginCreatorInterface, 489
nvinfer1, 41
nvinfer1::v_1_0::IPluginCreatorInterface, 490
IPluginCreatorV3One, 491
nvinfer1, 41
nvinfer1::v_1_0::IPluginCreatorV3One, 492, 493
IPluginResource, 506
nvinfer1, 41
nvinfer1::v_1_0::IPluginResource, 507
IPluginResourceContext
nvinfer1::IPluginResourceContext, 510
IPluginV2Ext
nvinfer1::IPluginV2Ext, 532
IPluginV3, 543
nvinfer1, 41
IPluginV3OneBuild, 547
nvinfer1, 41
IPluginV3OneBuildV2, 557
nvinfer1, 41
IPluginV3OneCore, 557
nvinfer1, 41
IPluginV3OneRuntime, 560
nvinfer1, 42
IProfiler, 572
nvinfer1, 42
IProgressMonitor, 573
nvinfer1, 42
nvinfer1::v_1_0::IProgressMonitor, 574
isAlphaBetaInt64
nvinfer1::IFillLayer, 304
isBatchAgnostic
nvinfer1::plugin::DetectionOutputParameters, 97
isConstant
nvinfer1::IDimensionExpr, 248
isDebugTensor
nvinfer1::ICudaEngine, 222
nvinfer1::INetworkDefinition, 425
isDeviceTypeSet
nvinfer1::IBuilderConfig, 170
isExecutionTensor
nvinfer1::ITensor, 666
isNetworkInput

nvinfer1::ITensor, 666
isNetworkOutput
nvinfer1::ITensor, 666
isNetworkSupported
nvinfer1::IBuilder, 152
isNormalized
nvinfer1::plugin::DetectionOutputParameters, 97
nvinfer1::plugin::NMSParameters, 690
isOutputBroadcastAcrossBatch
nvinfer1::IPluginV2Ext, 537
isParentSearchEnabled
nvinfer1::IPluginRegistry, 502
isRefittable
nvinfer1::ICudaEngine, 223
isShapeInferenceIO
nvinfer1::ICudaEngine, 223
isShapeTensor
nvinfer1::ITensor, 666
isSizeTensor
nvinfer1::IDimensionExpr, 248
isSubgraphSupported
nvonnxparser::IParser, 472
IStreamReader, 654
nvinfer1, 42
nvinfer1::v_1_0::IStreamReader, 656
IStreamReaderV2, 657
nvinfer1, 42
nvinfer1::v_1_0::IStreamReaderV2, 658, 659
isValid
nvinfer1::IOptimizationProfile, 457
IVersionedInterface
nvinfer1::IVersionedInterface, 688

kABS
nvinfer1, 76
kACOS
nvinfer1, 76
kACOSH
nvinfer1, 76
kACTIVATION
nvinfer1, 57
kALIASED_PLUGIN_IO_10_03
nvinfer1, 65
kALIGN_CORNERS
nvinfer1, 67
kALLOW_IN_MEMORY_FILES
nvinfer1, 72
kALLOW_TEMPORARY_FILES
nvinfer1, 72
kAMPERE_PLUS
nvinfer1, 55
kAND
nvinfer1, 51
kANY
nvinfer1, 77
kASIN
nvinfer1, 76
kASINH
nvinfer1, 76
kASSERTION
nvinfer1, 58
kASYMMETRIC
nvinfer1, 67
kATAN
nvinfer1, 76
kATANH
nvinfer1, 76
kAVERAGE
nvinfer1, 65
kAVG
nvinfer1, 67
kBF16
nvinfer1, 47, 50, 64
kBIAS
nvinfer1, 77
kBOOL
nvinfer1, 50
kBUILD
nvinfer1, 63, 74
kCALIBRATE_BEFORE_FUSION
nvinfer1, 66
kCAST
nvinfer1, 57
kCDHW32
nvinfer1, 73
kCEIL
nvinfer1, 68, 76
kCEIL_DIV
nvinfer1, 51
kCENTER_SIZES
nvinfer1, 45
kCHANNEL
nvinfer1, 70
kCHAR
nvinfer1, 64
kCHW16
nvinfer1, 73
kCHW2
nvinfer1, 73
kCHW32
nvinfer1, 73
kCHW4
nvinfer1, 73
kCLAMP
nvinfer1, 69
kCLIP
nvinfer1, 44
kCONCATENATE

nvinfer1, 58
kCONCATENATION
nvinfer1, 57
kCONDITION
nvinfer1, 58
kCONDITIONAL_INPUT
nvinfer1, 58
kCONDITIONAL_OUTPUT
nvinfer1, 58
kCONSTANT
nvinfer1, 57, 77
kCONVOLUTION
nvinfer1, 57
kCORE
nvinfer1, 63
kCORNER_PAIRS
nvinfer1, 45
kCOS
nvinfer1, 76
kCOSH
nvinfer1, 76
kCOUNT
nvinfer1, 75
kCPP
nvinfer1, 45
kCUBIC
nvinfer1, 56
kCUBLAS
nvinfer1, 71
kCUBLAS_LT
nvinfer1, 71
kCUDNN
nvinfer1, 71
kCUMULATIVE
nvinfer1, 58
kCUR
nvinfer1, 70
kDEBUG
nvinfer1, 46
kDECONVOLUTION
nvinfer1, 57
kDEFAULT
nvinfer1, 55
kDEFAULT_FORMAT_COMBINATION_LIMIT
nvinfer1::v_1_0::IPluginV3OneBuild, 555
kDEQUANTIZE
nvinfer1, 58
kDETAILED
nvinfer1, 66
kDEVICE
nvinfer1, 74
kDHWC
nvinfer1, 73
kDHWC8
nvinfer1, 73
kDIMS
nvinfer1, 64
kDIRECT_IO
nvinfer1, 46
kDISABLE_COMPILATION_CACHE
nvinfer1, 47
kDISABLE_TIMING_CACHE
nvinfer1, 46
kDIV
nvinfer1, 51
kDLA
nvinfer1, 50
kDLA_GLOBAL_DRAM
nvinfer1, 59
kDLA_HWC4
nvinfer1, 73
kDLA_LINEAR
nvinfer1, 73
kDLA_LOCAL_DRAM
nvinfer1, 59
kDLA_MANAGED_SRAM
nvinfer1, 59
kDLA_STANDALONE
nvinfer1, 52
kDYNAMIC_QUANTIZE
nvinfer1, 58
kEDGE_MASK_CONVOLUTIONS
nvinfer1, 71
kEDITABLE_TIMING_CACHE
nvinfer1, 48
keepTopK
nvinfer1::plugin::DetectionOutputParameters, 97
nvinfer1::plugin::NMSParameters, 690
kEINSUM
nvinfer1, 58
kELEMENT
nvinfer1, 55, 70
kELEMENTWISE
nvinfer1, 57, 70
kELU
nvinfer1, 44
kEND
nvinfer1, 70
kENTROPY_CALIBRATION
nvinfer1, 49
kENTROPY_CALIBRATION_2
nvinfer1, 49
kEQUAL
nvinfer1, 51
kERF
nvinfer1, 76
kERROR
nvinfer1::ILogger, 366

kERROR_ON_TIMING_CACHE_MISS
nvinfer1, 47
kEXCLUDE_LEAN_RUNTIME
nvinfer1, 47, 71
kEXCLUDE_WEIGHTS
nvinfer1, 71
kEXP
nvinfer1, 76
kEXPLICIT_BATCH
nvinfer1, 60
kEXPLICIT_ROUND_DOWN
nvinfer1, 63
kEXPLICIT_ROUND_UP
nvinfer1, 63
kFAILED_ALLOCATION
nvinfer1, 53
kFAILED_COMPUTATION
nvinfer1, 53
kFAILED_EXECUTION
nvinfer1, 53
kFAILED_INITIALIZATION
nvinfer1, 53
kFAST
nvinfer1, 75
kFILL
nvinfer1, 57, 69
kFLOAT
nvinfer1, 50
kFLOAT16
nvinfer1, 64
kFLOAT32
nvinfer1, 64
kFLOAT64
nvinfer1, 64
kFLOOR
nvinfer1, 68, 76
kFLOOR_DIV
nvinfer1, 51
kFORMAT_COMBINATION_LIMIT
nvinfer1::IPluginV2DynamicExt, 530
kFORMULA
nvinfer1, 68
kFP16
nvinfer1, 46
kFP4
nvinfer1, 48, 50, 64
kFP8
nvinfer1, 47, 50, 64
kFULL
nvinfer1, 75
kGATHER
nvinfer1, 57
kGELU_ERF
nvinfer1, 44
kGELU_TANH
nvinfer1, 44
kGPU
nvinfer1, 50
kGPU_FALLBACK
nvinfer1, 46
kGREATER
nvinfer1, 51
kGRID_SAMPLE
nvinfer1, 58
kHALF
nvinfer1, 50
kHALF_DOWN
nvinfer1, 68
kHALF_PIXEL
nvinfer1, 67
kHALF_UP
nvinfer1, 68
kHARD_SIGMOID
nvinfer1, 44
kHOST
nvinfer1, 74
kHWC
nvinfer1, 73
kHWC16
nvinfer1, 73
kHWC8
nvinfer1, 73
kIDENTITY
nvinfer1, 57
kind
nvinfer1::InterfaceInfo, 444
kINFO
nvinfer1::ILogger, 366
kINPUT
nvinfer1, 74
kINT16
nvinfer1, 64
kINT32
nvinfer1, 50, 64
kINT4
nvinfer1, 48, 50, 64
kINT64
nvinfer1, 50, 64
kINT8
nvinfer1, 46, 50, 64
kINTERNAL_ERROR
nvinfer1, 52
nvinfer1::ILogger, 366
nvonnxparser, 92
kINVALID_ARGUMENT
nvinfer1, 52
kINVALID_CONFIG
nvinfer1, 53

kINVALID_GRAPH
nvonnxparser, 92
kINVALID_NODE
nvonnxparser, 92
kINVALID_STATE
nvinfer1, 53
kINVALID_TACTIC_HASH
nvinfer1::v_1_0::TimingCacheValue, 708
kINVALID_VALUE
nvonnxparser, 92
kISINF
nvinfer1, 76
kISNAN
nvinfer1, 76
kITERATOR
nvinfer1, 57
kJIT_CONVOLUTIONS
nvinfer1, 71
kJSON
nvinfer1, 56
kKERNEL
nvinfer1, 77
kLAST_VALUE
nvinfer1, 58
kLAYER_NAMES_ONLY
nvinfer1, 66
kLEAKY_RELU
nvinfer1, 44
kLEGACY_CALIBRATION
nvinfer1, 49
kLESS
nvinfer1, 51
kLINEAR
nvinfer1, 56, 73
kLINSPACE
nvinfer1, 54
kLOG
nvinfer1, 76
kLOOP_OUTPUT
nvinfer1, 57
kLRN
nvinfer1, 57
kMATRIX_MULTIPLY
nvinfer1, 57
kMAX
nvinfer1, 51, 61, 65, 67, 75
kMAX_AVERAGE_BLEND
nvinfer1, 65
kMAX_DESC_LENGTH
nvinfer1::v_1_0::IErrorRecorder, 270
kMEM_ALLOC_FAILED
nvonnxparser, 92
kMIN
nvinfer1, 51, 61, 67, 75
kMINMAX_CALIBRATION
nvinfer1, 49
kMODEL_DESERIALIZE_FAILED
nvonnxparser, 92
kMODERATE
nvinfer1, 75
kMONITOR_MEMORY
nvinfer1, 48
kNATIVE_INSTANCENORM
nvonnxparser, 92
kND
nvinfer1, 55, 70
kNEAREST
nvinfer1, 56
kNEG
nvinfer1, 76
kNMS
nvinfer1, 58
kNON_ZERO
nvinfer1, 58
kNONE
nvinfer1, 55, 59, 66, 74, 75
kNORMALIZATION
nvinfer1, 58
kNOT
nvinfer1, 76
kOBEY_PRECISION_CONSTRAINTS
nvinfer1, 46
kON_PROFILE_CHANGE
nvinfer1, 54
kONE_HOT
nvinfer1, 58
kONELINE
nvinfer1, 56
kOPT
nvinfer1, 61
kOR
nvinfer1, 51
kOUTPUT
nvinfer1, 74
kPADDING
nvinfer1, 57
kPARAMETRIC_RELU
nvinfer1, 57
kPLUGIN
nvinfer1, 57
kPLUGIN_V2
nvinfer1, 57
kPLUGIN_V3
nvinfer1, 58
kPOOLING
nvinfer1, 57
kPOW
nvinfer1, 51

kPREFER_PRECISION_CONSTRAINTS
nvinfer1, 46
kPROD
nvinfer1, 51, 67
kPROFILE_SHARING_0806
nvinfer1, 65
kPYTHON
nvinfer1, 45
kQUANTIZE
nvinfer1, 57
kRAGGED_SOFTMAX
nvinfer1, 57
kRANDOM_NORMAL
nvinfer1, 54
kRANDOM_UNIFORM
nvinfer1, 54
kRECIP
nvinfer1, 76
kRECURRENCE
nvinfer1, 57
kREDUCE
nvinfer1, 57
kREFIT
nvinfer1, 46
kREFIT_FAILED
nvonnxparser, 92
kREFIT_IDENTICAL
nvinfer1, 47
kREFIT_INDIVIDUAL
nvinfer1, 48
kREFLECT
nvinfer1, 69
kREJECT_EMPTY_ALGORITHMS
nvinfer1, 46
kRELU
nvinfer1, 44
kRESIZABLE
nvinfer1, 45
kRESIZE
nvinfer1, 57
kREVERSE
nvinfer1, 58
kREVERSE_SEQUENCE
nvinfer1, 58
kROUND
nvinfer1, 76
kRUNTIME
nvinfer1, 63, 74
kRUNTIME_ACTIVATION_RESIZE_10_10
nvinfer1, 65
kSAFETY
nvinfer1, 52
kSAFETY_SCOPE
nvinfer1, 46
kSAME_AS_BUILD
nvinfer1, 69
kSAME_COMPUTE_CAPABILITY
nvinfer1, 56
kSAME_LOWER
nvinfer1, 63
kSAME_UPPER
nvinfer1, 63
kSCALE
nvinfer1, 57, 77
kSCALED_TANH
nvinfer1, 44
kSCATTER
nvinfer1, 58
kSELECT
nvinfer1, 57
kSELU
nvinfer1, 44
kSET
nvinfer1, 70
kSHAPE
nvinfer1, 57
kSHIFT
nvinfer1, 77
kSHUFFLE
nvinfer1, 57
kSIGMOID
nvinfer1, 44
kSIGN
nvinfer1, 76
kSIN
nvinfer1, 76
kSINH
nvinfer1, 76
kSLICE
nvinfer1, 57
kSOFTMAX
nvinfer1, 57
kSOFTPLUS
nvinfer1, 44
kSOFTSIGN
nvinfer1, 44
kSPARSE_WEIGHTS
nvinfer1, 46
kSQRT
nvinfer1, 76
kSQUEEZE
nvinfer1, 58
kSTANDARD
nvinfer1, 52
kSTATIC
nvinfer1, 54
kSTRICT_BOUNDS
nvinfer1, 69

kSTRICT_NANS
nvinfer1, 48
kSTRIP_PLAN
nvinfer1, 47
kSTRONGLY_TYPED
nvinfer1, 60
kSUB
nvinfer1, 51
kSUCCESS
nvinfer1, 52
nvonnxparser, 92
kSUM
nvinfer1, 49, 51, 67
kTACTIC_DRAM
nvinfer1, 59
kTACTIC_SHARED_MEMORY
nvinfer1, 60
kTAN
nvinfer1, 76
kTANH
nvinfer1, 44
kTF32
nvinfer1, 46
kTHRESHOLDED_RELU
nvinfer1, 44
kTOPK
nvinfer1, 57
kTRANSPOSE
nvinfer1, 59
kTRIP_LIMIT
nvinfer1, 57
kUINT8
nvinfer1, 50
kUNARY
nvinfer1, 57
kUNIFORM
nvinfer1, 70
kUNKNOWN
nvinfer1, 64
kUNSPECIFIED_ERROR
nvinfer1, 52
kUNSQUEEZE
nvinfer1, 58
kUNSUPPORTED_GRAPH
nvonnxparser, 92
kUNSUPPORTED_NODE
nvonnxparser, 92
kUNSUPPORTED_NODE_ATTR
nvonnxparser, 92
kUNSUPPORTED_NODE_DATATYPE
nvonnxparser, 92
kUNSUPPORTED_NODE_DYNAMIC
nvonnxparser, 92
kUNSUPPORTED_NODE_INPUT
nvonnxparser, 92
kUNSUPPORTED_NODE_SHAPE
nvonnxparser, 92
kUNSUPPORTED_STATE
nvinfer1, 53
kUPPER
nvinfer1, 68
kUSER_MANAGED
nvinfer1, 54
kV1
nvinfer1, 64
kV1_PYTHON
nvinfer1, 64
kV2
nvinfer1, 64
kV2_DYNAMICEXT
nvinfer1, 64
kV2_DYNAMICEXT_PYTHON
nvinfer1, 64
kV2_EXT
nvinfer1, 64
kV2_IOEXT
nvinfer1, 64
kVALUE
nvinfer1::impl::EnumMaxImpl< ActivationType >, 110
nvinfer1::impl::EnumMaxImpl< AllocatorFlag >, 110
nvinfer1::impl::EnumMaxImpl< APILanguage >, 111
nvinfer1::impl::EnumMaxImpl< CumulativeOperation >, 112
nvinfer1::impl::EnumMaxImpl< DataType >, 113
nvinfer1::impl::EnumMaxImpl< ElementWiseOperation >, 113
nvinfer1::impl::EnumMaxImpl< EngineCapability >, 114
nvinfer1::impl::EnumMaxImpl< ErrorCode >, 115
nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel >, 115
nvinfer1::impl::EnumMaxImpl< ILogger::Severity >, 116
nvinfer1::impl::EnumMaxImpl< InterpolationMode >, 117
nvinfer1::impl::EnumMaxImpl< PaddingMode >, 118
nvinfer1::impl::EnumMaxImpl< PoolingType >, 118
nvinfer1::impl::EnumMaxImpl< PreviewFeature >, 119
nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation >, 120
nvinfer1::impl::EnumMaxImpl< ResizeRoundMode >, 121

nvinfer1::impl::EnumMaxImpl< ResizeSelector >, 121
nvinfer1::impl::EnumMaxImpl< RuntimePlatform >, 122
nvinfer1::impl::EnumMaxImpl< TensorFormat >, 123
nvinfer1::impl::EnumMaxImpl< TensorIOMode >, 123
nvinfer1::impl::EnumMaxImpl< TensorLocation >, 124
nvinfer1::impl::EnumMaxImpl< TilingOptimizationLevel >, 125
kVECTOR
nvinfer1, 59
kVERBOSE
nvinfer1::ILogger, 366
kVERSION_COMPATIBLE
nvinfer1, 46
kWARNING
nvinfer1::ILogger, 366
kWEIGHT_STREAMING
nvinfer1, 48
kWEIGHTLESS
nvinfer1, 47
kWHILE
nvinfer1, 75
kWINDOWS_AMD64
nvinfer1, 69
kWORKSPACE
nvinfer1, 59
kWRAP
nvinfer1, 69
kXOR
nvinfer1, 51

LayerInformationFormat
nvinfer1, 56
LayerType
nvinfer1, 56
leaf
nvinfer1::plugin::softmaxTree, 706
length
nvinfer1::PluginField, 693
line
nvonnxparser::IParserError, 478
loadLibrary
nvinfer1::IPluginRegistry, 503
loadRuntime
nvinfer1::IRuntime, 619
localFunctionStack
nvonnxparser::IParserError, 478
localFunctionStackSize
nvonnxparser::IParserError, 478
log
nvinfer1::ILogger, 367
LoopOutput
nvinfer1, 58

major
nvinfer1::InterfaceInfo, 444
markDebug
nvinfer1::INetworkDefinition, 425
markOutput
nvinfer1::INetworkDefinition, 426
markOutputForShapes
nvinfer1::INetworkDefinition, 426
markWeightsRefittable
nvinfer1::INetworkDefinition, 427
MatrixOperation
nvinfer1, 58
max
nvinfer1::DynamicPluginTensorDesc, 108
MAX_DIMS
nvinfer1::Dims64, 103
maxSize
nvinfer1::plugin::GridAnchorParameters, 126
nvinfer1::plugin::PriorBoxParameters, 699
mBoundary
nvinfer1::IIfConditionalBoundaryLayer, 337
nvinfer1::ILoopBoundaryLayer, 373
MemoryPoolType
nvinfer1, 59
mImpl
nvinfer1::IActivationLayer, 131
nvinfer1::IAlgorithm, 133
nvinfer1::IAlgorithmContext, 136
nvinfer1::IAlgorithmIOInfo, 139
nvinfer1::IAlgorithmVariant, 143
nvinfer1::IAssertionLayer, 145
nvinfer1::IBuilder, 155
nvinfer1::IBuilderConfig, 183
nvinfer1::ICastLayer, 185
nvinfer1::IConcatenationLayer, 187
nvinfer1::IConditionLayer, 189
nvinfer1::IConstantLayer, 191
nvinfer1::IConvolutionLayer, 201
nvinfer1::ICudaEngine, 227
nvinfer1::ICumulativeLayer, 231
nvinfer1::IDeconvolutionLayer, 242
nvinfer1::IDequantizeLayer, 246
nvinfer1::IDimensionExpr, 248
nvinfer1::IDynamicQuantizeLayer, 253
nvinfer1::IEinsumLayer, 256
nvinfer1::IElementWiseLayer, 258
nvinfer1::IEngineInspector, 263
nvinfer1::IExecutionContext, 297
nvinfer1::IExprBuilder, 300
nvinfer1::IFillLayer, 309

nvinfer1::IGatherLayer, 314
nvinfer1::IGridSampleLayer, 328
nvinfer1::IHostMemory, 330
nvinfer1::IIdentityLayer, 332
nvinfer1::IIfConditional, 336
nvinfer1::IIfConditionalInputLayer, 339
nvinfer1::IIfConditionalOutputLayer, 340
nvinfer1::IIteratorLayer, 355
nvinfer1::ILoop, 372
nvinfer1::ILoopOutputLayer, 376
nvinfer1::ILRNLayer, 380
nvinfer1::IMatrixMultiplyLayer, 383
nvinfer1::INetworkDefinition, 431
nvinfer1::INMSLayer, 435
nvinfer1::INonZeroLayer, 439
nvinfer1::INormalizationLayer, 443
nvinfer1::IOneHotLayer, 447
nvinfer1::IOptimizationProfile, 459
nvinfer1::IPaddingLayer, 466
nvinfer1::IParametricReLULayer, 467
nvinfer1::IPluginV2Layer, 543
nvinfer1::IPluginV3Layer, 547
nvinfer1::IPoolingLayer, 571
nvinfer1::IQuantizeLayer, 580
nvinfer1::IRaggedSoftMaxLayer, 582
nvinfer1::IRecurrenceLayer, 584
nvinfer1::IReduceLayer, 587
nvinfer1::IRefitter, 601
nvinfer1::IResizeLayer, 610
nvinfer1::IReverseSequenceLayer, 613
nvinfer1::IRuntime, 623
nvinfer1::IScaleLayer, 628
nvinfer1::IScatterLayer, 631
nvinfer1::ISelectLayer, 633
nvinfer1::ISerializationConfig, 636
nvinfer1::IShapeLayer, 637
nvinfer1::IShuffleLayer, 643
nvinfer1::ISliceLayer, 650
nvinfer1::ISoftMaxLayer, 652
nvinfer1::ISqueezeLayer, 654
nvinfer1::ITensor, 672
nvinfer1::ITimingCache, 677
nvinfer1::ITopKLayer, 681
nvinfer1::ITripLimitLayer, 682
nvinfer1::IUnaryLayer, 684
nvinfer1::IUnsqueezeLayer, 686

min
nvinfer1::DynamicPluginTensorDesc, 108

minBoxSize
nvinfer1::plugin::RPROIParams, 704

minor
nvinfer1::InterfaceInfo, 444

minSize
nvinfer1::plugin::GridAnchorParameters, 126
nvinfer1::plugin::PriorBoxParameters, 699

mLayer
nvinfer1::ILayer, 365

n
nvinfer1::plugin::softmaxTree, 706

name
nvinfer1::plugin::softmaxTree, 706
nvinfer1::PluginField, 693

nbDims
nvinfer1::Dims64, 104
nvinfer1::DimsExprs, 105

nbFields
nvinfer1::PluginFieldCollection, 694

NetworkDefinitionCreationFlag
nvinfer1, 60

NetworkDefinitionCreationFlags
nvinfer1, 42

nmsMaxOut
nvinfer1::plugin::RPROIParams, 704

nmsThreshold
nvinfer1::plugin::DetectionOutputParameters, 97

node
nvonnxparser::IParserError, 479

nodeName
nvonnxparser::IParserError, 479

nodeOperator
nvonnxparser::IParserError, 479

notifyShape
nvinfer1::v_1_0::IOutputAllocator, 461

num
nvinfer1::plugin::RegionParameters, 702

numAspectRatios
nvinfer1::plugin::GridAnchorParameters, 126
nvinfer1::plugin::PriorBoxParameters, 699

numClasses
nvinfer1::plugin::DetectionOutputParameters, 97
nvinfer1::plugin::NMSParameters, 691

numMaxSize
nvinfer1::plugin::PriorBoxParameters, 700

numMinSize
nvinfer1::plugin::PriorBoxParameters, 700

NV_INFER_INTERNAL_INCLUDE
NvInferLegacyDims.h, 783
NvInferPluginBase.h, 787
NvInferRuntime.h, 797
NvInferRuntimeCommon.h, 828
NvInferRuntimePlugin.h, 831

NV_ONNX_PARSER_MAJOR
NvOnnxParser.h, 842

NV_ONNX_PARSER_MINOR
NvOnnxParser.h, 842

NV_ONNX_PARSER_PATCH
NvOnnxParser.h, 843

NV_TENSORRT_BUILD
NvInferVersion.h, 836
NV_TENSORRT_LWS_MAJOR
NvInferVersion.h, 836
NV_TENSORRT_LWS_MINOR
NvInferVersion.h, 837
NV_TENSORRT_LWS_PATCH
NvInferVersion.h, 837
NV_TENSORRT_MAJOR
NvInferVersion.h, 837
NV_TENSORRT_MINOR
NvInferVersion.h, 837
NV_TENSORRT_PATCH
NvInferVersion.h, 837
NV_TENSORRT_RELEASE_TYPE
NvInferVersion.h, 837
NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS
NvInferVersion.h, 838
NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY
NvInferVersion.h, 838
NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE
NvInferVersion.h, 838
NV_TENSORRT_VERSION
NvInferRuntimeBase.h, 823
NV_TENSORRT_VERSION_INT
NvInferRuntimeBase.h, 823
NvInfer.h, 711, 719
nvinfer1, 27
ActivationType, 44
AllocatorFlag, 45
AllocatorFlags, 38
APILanguage, 45
AsciiChar, 38
BoundingBoxFormat, 45
BuilderFlag, 45
BuilderFlags, 38
CalibrationAlgoType, 49
char_t, 38
CumulativeOperation, 49
DataType, 49
DeviceType, 50
DimensionOperation, 50
Dims, 39
ElementWiseOperation, 51
EngineCapability, 52
EnumMax, 77
EnumMax< BoundingBoxFormat >, 77
EnumMax< BuilderFlag >, 77
EnumMax< CalibrationAlgoType >, 78
EnumMax< DeviceType >, 78
EnumMax< DimensionOperation >, 78
EnumMax< ExecutionContextAllocationStrategy >, 78
EnumMax< FillOperation >, 79
EnumMax< GatherMode >, 79
EnumMax< LayerInformationFormat >, 79
EnumMax< LayerType >, 79
EnumMax< LoopOutput >, 80
EnumMax< MatrixOperation >, 80
EnumMax< MemoryPoolType >, 80
EnumMax< NetworkDefinitionCreationFlag >, 80
EnumMax< OptProfileSelector >, 81
EnumMax< ProfilingVerbosity >, 81
EnumMax< QuantizationFlag >, 81
EnumMax< ReduceOperation >, 81
EnumMax< SampleMode >, 82
EnumMax< ScaleMode >, 82
EnumMax< ScatterMode >, 82
EnumMax< SerializationFlag >, 82
EnumMax< TacticSource >, 83
EnumMax< TempfileControlFlag >, 83
EnumMax< TopKOperation >, 83
EnumMax< TripLimit >, 83
EnumMax< UnaryOperation >, 84
EnumMax< WeightsRole >, 84
ErrorCode, 52
ExecutionContextAllocationStrategy, 53
FillOperation, 54
GatherMode, 55
getBuilderPluginRegistry, 84
getBuilderSafePluginRegistry, 84
HardwareCompatibilityLevel, 55
IAlgorithmSelector, 39
IDebugListener, 39
IErrorRecorder, 39
IGpuAllocator, 39
IGpuAsyncAllocator, 39
IInt8EntropyCalibrator, 39
IInt8EntropyCalibrator2, 40
IInt8LegacyCalibrator, 40
IInt8MinMaxCalibrator, 40
InterfaceKind, 40
InterpolationMode, 56
IOutputAllocator, 40
IPluginCapability, 40
IPluginCreator, 40
IPluginCreatorInterface, 41
IPluginCreatorV3One, 41
IPluginResource, 41
IPluginV3, 41
IPluginV3OneBuild, 41
IPluginV3OneBuildV2, 41
IPluginV3OneCore, 41
IPluginV3OneRuntime, 42
IProfiler, 42
IProgressMonitor, 42
IStreamReader, 42
IStreamReaderV2, 42

kABS, 76
kACOS, 76
kACOSH, 76
kACTIVATION, 57
kALIASED_PLUGIN_IO_10_03, 65
kALIGN_CORNERS, 67
kALLOW_IN_MEMORY_FILES, 72
kALLOW_TEMPORARY_FILES, 72
kAMPERE_PLUS, 55
kAND, 51
kANY, 77
kASIN, 76
kASINH, 76
kASSERTION, 58
kASYMMETRIC, 67
kATAN, 76
kATANH, 76
kAVERAGE, 65
kAVG, 67
kBF16, 47, 50, 64
kBIAS, 77
kBOOL, 50
kBUILD, 63, 74
kCALIBRATE_BEFORE_FUSION, 66
kCAST, 57
kCDHW32, 73
kCEIL, 68, 76
kCEIL_DIV, 51
kCENTER_SIZES, 45
kCHANNEL, 70
kCHAR, 64
kCHW16, 73
kCHW2, 73
kCHW32, 73
kCHW4, 73
kCLAMP, 69
kCLIP, 44
kCONCATENATE, 58
kCONCATENATION, 57
kCONDITION, 58
kCONDITIONAL_INPUT, 58
kCONDITIONAL_OUTPUT, 58
kCONSTANT, 57, 77
kCONVOLUTION, 57
kCORE, 63
kCORNER_PAIRS, 45
kCOS, 76
kCOSH, 76
kCOUNT, 75
kCPP, 45
kCUBIC, 56
kCUBLAS, 71
kCUBLAS_LT, 71
kCUDNN, 71
kCUMULATIVE, 58
kCUR, 70
kDEBUG, 46
kDECONVOLUTION, 57
kDEFAULT, 55
kDEQUANTIZE, 58
kDETAILED, 66
kDEVICE, 74
kDHWC, 73
kDHWC8, 73
kDIMS, 64
kDIRECT_IO, 46
kDISABLE_COMPILATION_CACHE, 47
kDISABLE_TIMING_CACHE, 46
kDIV, 51
kDLA, 50
kDLA_GLOBAL_DRAM, 59
kDLA_HWC4, 73
kDLA_LINEAR, 73
kDLA_LOCAL_DRAM, 59
kDLA_MANAGED_SRAM, 59
kDLA_STANDALONE, 52
kDYNAMIC_QUANTIZE, 58
kEDGE_MASK_CONVOLUTIONS, 71
kEDITABLE_TIMING_CACHE, 48
kEINSUM, 58
kELEMENT, 55, 70
kELEMENTWISE, 57, 70
kELU, 44
kEND, 70
kENTROPY_CALIBRATION, 49
kENTROPY_CALIBRATION_2, 49
kEQUAL, 51
kERF, 76
kERROR_ON_TIMING_CACHE_MISS, 47
kEXCLUDE_LEAN_RUNTIME, 47, 71
kEXCLUDE_WEIGHTS, 71
kEXP, 76
kEXPLICIT_BATCH, 60
kEXPLICIT_ROUND_DOWN, 63
kEXPLICIT_ROUND_UP, 63
kFAILED_ALLOCATION, 53
kFAILED_COMPUTATION, 53
kFAILED_EXECUTION, 53
kFAILED_INITIALIZATION, 53
kFAST, 75
kFILL, 57, 69
kFLOAT, 50
kFLOAT16, 64
kFLOAT32, 64
kFLOAT64, 64
kFLOOR, 68, 76
kFLOOR_DIV, 51
kFORMULA, 68

kFP16, 46
kFP4, 48, 50, 64
kFP8, 47, 50, 64
kFULL, 75
kGATHER, 57
kGELU_ERF, 44
kGELU_TANH, 44
kGPU, 50
kGPU_FALLBACK, 46
kGREATER, 51
kGRID_SAMPLE, 58
kHALF, 50
kHALF_DOWN, 68
kHALF_PIXEL, 67
kHALF_UP, 68
kHARD_SIGMOID, 44
kHOST, 74
kHWC, 73
kHWC16, 73
kHWC8, 73
kIDENTITY, 57
kINPUT, 74
kINT16, 64
kINT32, 50, 64
kINT4, 48, 50, 64
kINT64, 50, 64
kINT8, 46, 50, 64
kINTERNAL_ERROR, 52
kINVALID_ARGUMENT, 52
kINVALID_CONFIG, 53
kINVALID_STATE, 53
kISINF, 76
kISNAN, 76
kITERATOR, 57
kJIT_CONVOLUTIONS, 71
kJSON, 56
kKERNEL, 77
kLAST_VALUE, 58
kLAYER_NAMES_ONLY, 66
kLEAKY_RELU, 44
kLEGACY_CALIBRATION, 49
kLESS, 51
kLINEAR, 56, 73
kLINSPACE, 54
kLOG, 76
kLOOP_OUTPUT, 57
kLRN, 57
kMATRIX_MULTIPLY, 57
kMAX, 51, 61, 65, 67, 75
kMAX_AVERAGE_BLEND, 65
kMIN, 51, 61, 67, 75
kMINMAX_CALIBRATION, 49
kMODERATE, 75
kMONITOR_MEMORY, 48
kND, 55, 70
kNEAREST, 56
kNEG, 76
kNMS, 58
kNON_ZERO, 58
kNONE, 55, 59, 66, 74, 75
kNORMALIZATION, 58
kNOT, 76
kOBEY_PRECISION_CONSTRAINTS, 46
kON_PROFILE_CHANGE, 54
kONE_HOT, 58
kONELINE, 56
kOPT, 61
kOR, 51
kOUTPUT, 74
kPADDING, 57
kPARAMETRIC_RELU, 57
kPLUGIN, 57
kPLUGIN_V2, 57
kPLUGIN_V3, 58
kPOOLING, 57
kPOW, 51
kPREFER_PRECISION_CONSTRAINTS, 46
kPROD, 51, 67
kPROFILE_SHARING_0806, 65
kPYTHON, 45
kQUANTIZE, 57
kRAGGED_SOFTMAX, 57
kRANDOM_NORMAL, 54
kRANDOM_UNIFORM, 54
kRECIP, 76
kRECURRENCE, 57
kREDUCE, 57
kREFIT, 46
kREFIT_IDENTICAL, 47
kREFIT_INDIVIDUAL, 48
kREFLECT, 69
kREJECT_EMPTY_ALGORITHMS, 46
kRELU, 44
kRESIZABLE, 45
kRESIZE, 57
kREVERSE, 58
kREVERSE_SEQUENCE, 58
kROUND, 76
kRUNTIME, 63, 74
kRUNTIME_ACTIVATION_RESIZE_10_10, 65
kSAFETY, 52
kSAFETY_SCOPE, 46
kSAME_AS_BUILD, 69
kSAME_COMPUTE_CAPABILITY, 56
kSAME_LOWER, 63
kSAME_UPPER, 63
kSCALE, 57, 77
kSCALED_TANH, 44

kSCATTER, 58
kSELECT, 57
kSELU, 44
kSET, 70
kSHAPE, 57
kSHIFT, 77
kSHUFFLE, 57
kSIGMOID, 44
kSIGN, 76
kSIN, 76
kSINH, 76
kSLICE, 57
kSOFTMAX, 57
kSOFTPLUS, 44
kSOFTSIGN, 44
kSPARSE_WEIGHTS, 46
kSQRT, 76
kSQUEEZE, 58
kSTANDARD, 52
kSTATIC, 54
kSTRICT_BOUNDS, 69
kSTRICT_NANS, 48
kSTRIP_PLAN, 47
kSTRONGLY_TYPED, 60
kSUB, 51
kSUCCESS, 52
kSUM, 49, 51, 67
kTACTIC_DRAM, 59
kTACTIC_SHARED_MEMORY, 60
kTAN, 76
kTANH, 44
kTF32, 46
kTHRESHOLDED_RELU, 44
kTOPK, 57
kTRANSPOSE, 59
kTRIP_LIMIT, 57
kUINT8, 50
kUNARY, 57
kUNIFORM, 70
kUNKNOWN, 64
kUNSPECIFIED_ERROR, 52
kUNSQUEEZE, 58
kUNSUPPORTED_STATE, 53
kUPPER, 68
kUSER_MANAGED, 54
kV1, 64
kV1_PYTHON, 64
kV2, 64
kV2_DYNAMICEXT, 64
kV2_DYNAMICEXT_PYTHON, 64
kV2_EXT, 64
kV2_IOEXT, 64
kVECTOR, 59
kVERSION_COMPATIBLE, 46
kWEIGHT_STREAMING, 48
kWEIGHTLESS, 47
kWHILE, 75
kWINDOWS_AMD64, 69
kWORKSPACE, 59
kWRAP, 69
kXOR, 51
LayerInformationFormat, 56
LayerType, 56
LoopOutput, 58
MatrixOperation, 58
MemoryPoolType, 59
NetworkDefinitionCreationFlag, 60
NetworkDefinitionCreationFlags, 42
OptProfileSelector, 60
PaddingMode, 61
PluginCapabilityType, 63
PluginCreatorVersion, 63
PluginFieldType, 64
PluginFormat, 42
PluginVersion, 64
PoolingType, 65
PreviewFeature, 65
ProfilingVerbosity, 65
QuantizationFlag, 66
QuantizationFlags, 43
ReduceOperation, 66
ResizeCoordinateTransformation, 67
ResizeRoundMode, 67
ResizeSelector, 68
RuntimePlatform, 68
SampleMode, 69
ScaleMode, 69
ScatterMode, 70
SeekPosition, 70
SerializationFlag, 70
SerializationFlags, 43
TacticSource, 71
TacticSources, 43
TempfileControlFlag, 72
TempfileControlFlags, 43
TensorFormat, 72
TensorFormats, 44
TensorIOMode, 74
TensorLocation, 74
TensorRTPhase, 74
TilingOptimizationLevel, 74
TopKOperation, 75
TripLimit, 75
UnaryOperation, 75
WeightsRole, 77
nvinfer1::anonymous_namespace{NvInfer.h}, 85
createInferBuilder, 85
nvinfer1::anonymous_namespace{NvInferRuntime.h}, 85

createInferRefitter, 86
createInferRuntime, 86
nvinfer1::Dims2, 99
Dims2, 99
nvinfer1::Dims3, 100
Dims3, 100, 101
nvinfer1::Dims4, 101
Dims4, 102
nvinfer1::Dims64, 103
d, 103
MAX_DIMS, 103
nbDims, 104
nvinfer1::DimsExprs, 104
d, 104
nbDims, 105
nvinfer1::DimsHW, 105
DimsHW, 106
h, 106
w, 107
nvinfer1::DynamicPluginTensorDesc, 107
desc, 108
max, 108
min, 108
opt, 108
nvinfer1::IActivationLayer, 127
~IActivationLayer, 128
getActivationType, 129
getAlpha, 129
getBeta, 129
mImpl, 131
setActivationType, 129
setAlpha, 130
setBeta, 130
nvinfer1::IAlgorithm, 131
~IAlgorithm, 132
getAlgorithmIOInfoByIndex, 132
getAlgorithmVariant, 133
getTimingMSec, 133
getWorkspaceSize, 133
mImpl, 133
nvinfer1::IAlgorithmContext, 134
~IAlgorithmContext, 135
getDimensions, 135
getName, 135
getNbInputs, 135
getNbOutputs, 136
mImpl, 136
nvinfer1::IAlgorithmIOInfo, 136
~IAlgorithmIOInfo, 137
getComponentsPerElement, 138
getDataType, 138
getStrides, 138
getVectorizedDim, 138
mImpl, 139
nvinfer1::IAlgorithmVariant, 141
~IAlgorithmVariant, 143
getImplementation, 143
getTactic, 143
mImpl, 143
nvinfer1::IAssertionLayer, 144
~IAssertionLayer, 145
getMessage, 145
mImpl, 145
setMessage, 145
nvinfer1::IBuilder, 146
~IBuilder, 147
buildEngineWithConfig, 148
buildSerializedNetwork, 148
createBuilderConfig, 149
createNetworkV2, 149
createOptimizationProfile, 150
getErrorRecorder, 150
getLogger, 150
getMaxDLABatchSize, 151
getMaxThreads, 151
getNbDLACores, 151
getPluginRegistry, 152
isNetworkSupported, 152
mImpl, 155
platformHasFastFp16, 152
platformHasFastInt8, 153
platformHasTf32, 153
reset, 153
setErrorRecorder, 153
setGpuAllocator, 154
setMaxThreads, 154
nvinfer1::IBuilderConfig, 155
~IBuilderConfig, 158
addOptimizationProfile, 159
canRunOnDLA, 159
clearFlag, 159
clearQuantizationFlag, 160
createTimingCache, 160
getAlgorithmSelector, 161
getAvgTimingIterations, 161
getBuilderOptimizationLevel, 161
getCalibrationProfile, 162
getDefaultDeviceType, 162
getDeviceType, 162
getDLACore, 162
getEngineCapability, 163
getFlag, 163
getFlags, 163
getHardwareCompatibilityLevel, 164
getInt8Calibrator, 164
getL2LimitForTiling, 164
getMaxAuxStreams, 165
getMaxNbTactics, 165

getMemoryPoolLimit, 165
getNbOptimizationProfiles, 166
getNbPluginsToSerialize, 166
getPluginToSerialize, 166
getPreviewFeature, 167
getProfileStream, 167
getProfilingVerbosity, 167
getProgressMonitor, 168
getQuantizationFlag, 168
getQuantizationFlags, 168
getRuntimePlatform, 169
getTacticSources, 169
getTilingOptimizationLevel, 169
getTimingCache, 170
isDeviceTypeSet, 170
mImpl, 183
reset, 170
resetDeviceType, 171
setAlgorithmSelector, 171
setAvgTimingIterations, 171
setBuilderOptimizationLevel, 172
setCalibrationProfile, 172
setDefaultDeviceType, 173
setDeviceType, 173
setDLACore, 174
setEngineCapability, 174
setFlag, 174
setFlags, 175
setHardwareCompatibilityLevel, 175
setInt8Calibrator, 176
setL2LimitForTiling, 176
setMaxAuxStreams, 176
setMaxNbTactics, 177
setMemoryPoolLimit, 177
setPluginsToSerialize, 178
setPreviewFeature, 179
setProfileStream, 179
setProfilingVerbosity, 179
setProgressMonitor, 180
setQuantizationFlag, 180
setQuantizationFlags, 180
setRuntimePlatform, 181
setTacticSources, 181
setTilingOptimizationLevel, 182
setTimingCache, 182
nvinfer1::ICastLayer, 184
~ICastLayer, 184
getToType, 185
mImpl, 185
setToType, 185
nvinfer1::IConcatenationLayer, 186
~IConcatenationLayer, 186
getAxis, 187
mImpl, 187
setAxis, 187
nvinfer1::IConditionLayer, 188
~IConditionLayer, 188
mImpl, 189
nvinfer1::IConstantLayer, 189
~IConstantLayer, 190
getDimensions, 190
getWeights, 190
mImpl, 191
setDimensions, 191
setWeights, 191
nvinfer1::IConvolutionLayer, 192
~IConvolutionLayer, 194
getBiasWeights, 194
getDilationNd, 194
getKernelSizeNd, 194
getKernelWeights, 195
getNbGroups, 195
getNbOutputMaps, 195
getPaddingMode, 195
getPaddingNd, 196
getPostPadding, 196
getPrePadding, 196
getStrideNd, 196
mImpl, 201
setBiasWeights, 197
setDilationNd, 197
setInput, 197
setKernelSizeNd, 198
setKernelWeights, 198
setNbGroups, 198
setNbOutputMaps, 199
setPaddingMode, 199
setPaddingNd, 199
setPostPadding, 200
setPrePadding, 200
setStrideNd, 200
nvinfer1::ICudaEngine, 201
~ICudaEngine, 204
createEngineInspector, 205
createExecutionContext, 205
createExecutionContextWithoutDeviceMemory, 205
createSerializationConfig, 205
getDeviceMemorySize, 206
getDeviceMemorySizeForProfile, 206
getDeviceMemorySizeForProfileV2, 206
getDeviceMemorySizeV2, 207
getEngineCapability, 207
getErrorRecorder, 208
getHardwareCompatibilityLevel, 208
getIOTensorName, 208
getMinimumWeightStreamingBudget, 209
getName, 209
getNbAuxStreams, 209

getNbIOTensors, 210
getNbLayers, 210
getNbOptimizationProfiles, 210
getProfileShape, 211
getProfileTensorValues, 211
getProfilingVerbosity, 212
getStreamableWeightsSize, 212
getTacticSources, 212
getTensorBytesPerComponent, 213
getTensorComponentsPerElement, 214
getTensorDataType, 215
getTensorFormat, 215, 216
getTensorFormatDesc, 216, 217
getTensorIOMode, 217
getTensorLocation, 218
getTensorShape, 218
getTensorVectorizedDim, 219
getWeightStreamingAutomaticBudget, 220
getWeightStreamingBudget, 220
getWeightStreamingBudgetV2, 221
getWeightStreamingScratchMemorySize, 221
hasImplicitBatchDimension, 222
isDebugTensor, 222
isRefittable, 223
isShapeInferenceIO, 223
mImpl, 227
serialize, 223
serializeWithConfig, 224
setErrorRecorder, 224
setWeightStreamingBudget, 225
setWeightStreamingBudgetV2, 226
nvinfer1::ICumulativeLayer, 227
~ICumulativeLayer, 229
getExclusive, 229
getOperation, 229
getReverse, 229
mImpl, 231
setExclusive, 230
setOperation, 230
setReverse, 231
nvinfer1::IDeconvolutionLayer, 233
~IDeconvolutionLayer, 235
getBiasWeights, 235
getDilationNd, 236
getKernelSizeNd, 236
getKernelWeights, 236
getNbGroups, 236
getNbOutputMaps, 237
getPaddingMode, 237
getPaddingNd, 237
getPostPadding, 237
getPrePadding, 238
getStrideNd, 238
mImpl, 242
setBiasWeights, 238
setDilationNd, 238
setInput, 239
setKernelSizeNd, 239
setKernelWeights, 239
setNbGroups, 240
setNbOutputMaps, 240
setPaddingMode, 240
setPaddingNd, 241
setPostPadding, 241
setPrePadding, 241
setStrideNd, 242
nvinfer1::IDequantizeLayer, 243
~IDequantizeLayer, 245
getAxis, 245
getToType, 245
mImpl, 246
setAxis, 245
setToType, 246
nvinfer1::IDimensionExpr, 247
~IDimensionExpr, 248
getConstantValue, 248
isConstant, 248
isSizeTensor, 248
mImpl, 248
nvinfer1::IDynamicQuantizeLayer, 249
~IDynamicQuantizeLayer, 250
getAxis, 250
getBlockSize, 251
getScaleType, 251
getToType, 251
mImpl, 253
setAxis, 251
setBlockSize, 252
setInput, 252
setScaleType, 253
setToType, 253
nvinfer1::IEinsumLayer, 254
~IEinsumLayer, 255
getEquation, 255
mImpl, 256
setEquation, 255
nvinfer1::IElementWiseLayer, 256
~IElementWiseLayer, 257
getOperation, 257
mImpl, 258
setOperation, 257
nvinfer1::IEngineInspector, 258
~IEngineInspector, 259
getEngineInformation, 260
getErrorRecorder, 260
getExecutionContext, 261
getLayerInformation, 261
mImpl, 263

setErrorRecorder, 262
setExecutionContext, 262
nvinfer1::IExecutionContext, 271
~IExecutionContext, 273
allInputDimensionsSpecified, 274
allInputShapesSpecified, 274
enqueueV3, 274
executeV2, 275
getDebugListener, 275
getDebugState, 276
getDebugSync, 276
getEngine, 276
getEnqueueEmitsProfile, 276
getErrorRecorder, 277
getInputConsumedEvent, 277
getMaxOutputSize, 277
getName, 278
getNvtxVerbosity, 278
getOptimizationProfile, 278
getOutputAllocator, 279
getOutputTensorAddress, 279
getPersistentCacheLimit, 280
getProfiler, 280
getTemporaryStorageAllocator, 280
getTensorAddress, 280
getTensorShape, 281
getTensorStrides, 282
inferShapes, 282
mImpl, 297
reportToProfiler, 284
setAllTensorsDebugState, 284
setAuxStreams, 285
setDebugListener, 286
setDebugSync, 286
setDeviceMemory, 286
setDeviceMemoryV2, 287
setEnqueueEmitsProfile, 287
setErrorRecorder, 288
setInputConsumedEvent, 288
setInputShape, 289
setInputTensorAddress, 289
setName, 290
setNvtxVerbosity, 290
setOptimizationProfileAsync, 291
setOutputAllocator, 292
setOutputTensorAddress, 292
setPersistentCacheLimit, 293
setProfiler, 294
setTemporaryStorageAllocator, 294
setTensorAddress, 294
setTensorDebugState, 296
updateDeviceMemorySizeForShapes, 296
nvinfer1::IExprBuilder, 297
~IExprBuilder, 298
constant, 298
declareSizeTensor, 298
mImpl, 300
operation, 299
nvinfer1::IFillLayer, 300
~IFillLayer, 302
getAlpha, 302
getAlphaInt64, 302
getBeta, 303
getBetaInt64, 303
getDimensions, 303
getOperation, 304
getToType, 304
isAlphaBetaInt64, 304
mImpl, 309
setAlpha, 304
setAlphaInt64, 305
setBeta, 305
setBetaInt64, 306
setDimensions, 306
setInput, 307
setOperation, 308
setToType, 308
nvinfer1::IGatherLayer, 309
~IGatherLayer, 311
getGatherAxis, 311
getMode, 312
getNbElementWiseDims, 312
mImpl, 314
setGatherAxis, 312
setMode, 313
setNbElementWiseDims, 313
nvinfer1::IGridSampleLayer, 325
~IGridSampleLayer, 326
getAlignCorners, 327
getInterpolationMode, 327
getSampleMode, 327
mImpl, 328
setAlignCorners, 327
setInterpolationMode, 328
setSampleMode, 328
nvinfer1::IHostMemory, 329
~IHostMemory, 330
data, 330
mImpl, 330
size, 330
type, 330
nvinfer1::IIdentityLayer, 331
~IIdentityLayer, 332
mImpl, 332
nvinfer1::IIfConditional, 332
~IIfConditional, 333
addInput, 334
addOutput, 334

getName, 335
mImpl, 336
setCondition, 335
setName, 335
nvinfer1::IIfConditionalBoundaryLayer, 336
~IIfConditionalBoundaryLayer, 337
getConditional, 337
mBoundary, 337
nvinfer1::IIfConditionalInputLayer, 338
~IIfConditionalInputLayer, 338
mImpl, 339
nvinfer1::IIfConditionalOutputLayer, 339
~IIfConditionalOutputLayer, 340
mImpl, 340
nvinfer1::IInt8Calibrator, 340
~IInt8Calibrator, 341
getAlgorithm, 341
getBatch, 342
getBatchSize, 342
readCalibrationCache, 343
writeCalibrationCache, 343
nvinfer1::IIteratorLayer, 353
~IIteratorLayer, 354
getAxis, 354
getReverse, 354
mImpl, 355
setAxis, 354
setReverse, 354
nvinfer1::ILayer, 355
~ILayer, 357
getInput, 358
getMetadata, 358
getName, 358
getNbInputs, 358
getNbOutputs, 359
getOutput, 359
getOutputType, 359
getPrecision, 360
getType, 360
mLayer, 365
outputTypeIsSet, 360
precisionIsSet, 361
resetOutputType, 361
resetPrecision, 361
setInput, 362
setMetadata, 362
setName, 363
setOutputType, 363
setPrecision, 364
nvinfer1::ILogger, 365
~ILogger, 367
ILogger, 366
kERROR, 366
kINFO, 366
kINTERNAL_ERROR, 366
kVERBOSE, 366
kWARNING, 366
log, 367
Severity, 366
nvinfer1::ILoggerFinder, 368
~ILoggerFinder, 368
findLogger, 368
nvinfer1::ILoop, 369
~ILoop, 370
addIterator, 370
addLoopOutput, 370
addRecurrence, 370
addTripLimit, 371
getName, 371
mImpl, 372
setName, 371
nvinfer1::ILoopBoundaryLayer, 372
~ILoopBoundaryLayer, 373
getLoop, 373
mBoundary, 373
nvinfer1::ILoopOutputLayer, 374
~ILoopOutputLayer, 375
getAxis, 375
getLoopOutput, 375
mImpl, 376
setAxis, 375
setInput, 376
nvinfer1::ILRNLayer, 377
~ILRNLayer, 378
getAlpha, 378
getBeta, 378
getK, 378
getWindowSize, 379
mImpl, 380
setAlpha, 379
setBeta, 379
setK, 379
setWindowSize, 380
nvinfer1::IMatrixMultiplyLayer, 381
~IMatrixMultiplyLayer, 382
getOperation, 382
mImpl, 383
setOperation, 382
nvinfer1::impl, 86
nvinfer1::impl::EnumMaxImpl< ActivationType >, 109
kVALUE, 110
nvinfer1::impl::EnumMaxImpl< AllocatorFlag >, 110
kVALUE, 110
nvinfer1::impl::EnumMaxImpl< APILanguage >, 111
kVALUE, 111
nvinfer1::impl::EnumMaxImpl< CumulativeOperation >, 111
kVALUE, 112

nvinfer1::impl::EnumMaxImpl< DataType >, 112
kVALUE, 113
nvinfer1::impl::EnumMaxImpl< ElementWiseOperation >, 113
kVALUE, 113
nvinfer1::impl::EnumMaxImpl< EngineCapability >, 114
kVALUE, 114
nvinfer1::impl::EnumMaxImpl< ErrorCode >, 114
kVALUE, 115
nvinfer1::impl::EnumMaxImpl< HardwareCompatibilityLevel >, 115
kVALUE, 115
nvinfer1::impl::EnumMaxImpl< ILogger::Severity >, 116
kVALUE, 116
nvinfer1::impl::EnumMaxImpl< InterpolationMode >, 117
kVALUE, 117
nvinfer1::impl::EnumMaxImpl< PaddingMode >, 117
kVALUE, 118
nvinfer1::impl::EnumMaxImpl< PoolingType >, 118
kVALUE, 118
nvinfer1::impl::EnumMaxImpl< PreviewFeature >, 119
kVALUE, 119
nvinfer1::impl::EnumMaxImpl< ResizeCoordinateTransformation >, 119
kVALUE, 120
nvinfer1::impl::EnumMaxImpl< ResizeRoundMode >, 120
kVALUE, 121
nvinfer1::impl::EnumMaxImpl< ResizeSelector >, 121
kVALUE, 121
nvinfer1::impl::EnumMaxImpl< RuntimePlatform >, 122
kVALUE, 122
nvinfer1::impl::EnumMaxImpl< T >, 109
nvinfer1::impl::EnumMaxImpl< TensorFormat >, 122
kVALUE, 123
nvinfer1::impl::EnumMaxImpl< TensorIOMode >, 123
kVALUE, 123
nvinfer1::impl::EnumMaxImpl< TensorLocation >, 124
kVALUE, 124
nvinfer1::impl::EnumMaxImpl< TilingOptimizationLevel >, 124
kVALUE, 125
nvinfer1::INetworkDefinition, 383
~INetworkDefinition, 388
addActivation, 388
addAssertion, 388
addCast, 389
addConcatenation, 389
addConstant, 390
addConvolutionNd, 391
addCumulative, 391
addDeconvolutionNd, 392
addDequantize, 393
addDynamicQuantize, 394
addEinsum, 395
addElementWise, 395
addFill, 396, 397
addGather, 398
addGatherV2, 398
addGridSample, 399
addIdentity, 399
addIfConditional, 400
addInput, 400
addLoop, 401
addLRN, 401
addMatrixMultiply, 402
addNMS, 403
addNonZero, 403
addNormalization, 404
addOneHot, 404
addPaddingNd, 405
addParametricReLU, 406
addPluginV2, 406
addPluginV3, 407
addPoolingNd, 407
addQuantize, 408, 409
addRaggedSoftMax, 409
addReduce, 410
addResize, 411
addReverseSequence, 411
addScale, 412
addScaleNd, 413
addScatter, 414
addSelect, 414
addShape, 415
addShuffle, 416
addSlice, 416
addSoftMax, 417
addSqueeze, 417
addTopK, 418
addUnary, 419
addUnsqueeze, 419
areWeightsMarkedRefittable, 420
getBuilder, 420
getErrorRecorder, 421
getFlag, 421
getFlags, 421
getInput, 422
getLayer, 422
getName, 423
getNbInputs, 423
getNbLayers, 423
getNbOutputs, 424
getOutput, 424

hasImplicitBatchDimension, 425
isDebugTensor, 425
markDebug, 425
markOutput, 426
markOutputForShapes, 426
markWeightsRefittable, 427
mImpl, 431
removeTensor, 427
setErrorRecorder, 427
setName, 428
setWeightsName, 428
unmarkDebug, 429
unmarkOutput, 430
unmarkOutputForShapes, 430
unmarkWeightsRefittable, 430
nvinfer1::INMSLayer, 431
~INMSLayer, 433
getBoundingBoxFormat, 433
getTopKBoxLimit, 433
mImpl, 435
setBoundingBoxFormat, 433
setInput, 434
setTopKBoxLimit, 434
nvinfer1::INoCopy, 435
~INoCopy, 437
INoCopy, 436, 437
operator=, 437
nvinfer1::INonZeroLayer, 438
~INonZeroLayer, 439
mImpl, 439
nvinfer1::INormalizationLayer, 439
~INormalizationLayer, 440
getAxes, 441
getComputePrecision, 441
getEpsilon, 441
getNbGroups, 441
mImpl, 443
setAxes, 442
setComputePrecision, 442
setEpsilon, 443
setNbGroups, 443
nvinfer1::InterfaceInfo, 444
kind, 444
major, 444
minor, 444
nvinfer1::IOneHotLayer, 445
~IOneHotLayer, 446
getAxis, 446
mImpl, 447
setAxis, 446
nvinfer1::IOptimizationProfile, 454
~IOptimizationProfile, 455
getDimensions, 455
getExtraMemoryTarget, 456
getNbShapeValues, 456
getShapeValues, 456
isValid, 457
mImpl, 459
setDimensions, 457
setExtraMemoryTarget, 458
setShapeValues, 458
nvinfer1::IPaddingLayer, 463
~IPaddingLayer, 464
getPostPaddingNd, 464
getPrePaddingNd, 464
mImpl, 466
setPostPaddingNd, 465
setPrePaddingNd, 465
nvinfer1::IParametricReLULayer, 466
~IParametricReLULayer, 467
mImpl, 467
nvinfer1::IPluginRegistry, 495
~IPluginRegistry, 497
acquirePluginResource, 497
deregisterCreator, 498
deregisterLibrary, 499
getAllCreators, 499
getAllCreatorsRecursive, 500
getCreator, 500
getErrorRecorder, 501
getPluginCreator, 501
getPluginCreatorList, 502
isParentSearchEnabled, 502
loadLibrary, 503
PluginLibraryHandle, 497
registerCreator, 503, 504
releasePluginResource, 504
setErrorRecorder, 505
setParentSearchEnabled, 505
nvinfer1::IPluginResourceContext, 509
~IPluginResourceContext, 510
getErrorRecorder, 511
getGpuAllocator, 511
IPluginResourceContext, 510
operator=, 511
nvinfer1::IPluginV2, 512
clone, 513
configureWithFormat, 514
destroy, 515
enqueue, 515
getNbOutputs, 516
getOutputDimensions, 516
getPluginNamespace, 518
getPluginType, 518
getPluginVersion, 519
getSerializationSize, 519
getTensorRTVersion, 520
getWorkspaceSize, 520

initialize, 521
serialize, 521
setPluginNamespace, 522
supportsFormat, 522
terminate, 523
nvinfer1::IPluginV2DynamicExt, 524
~IPluginV2DynamicExt, 526
clone, 526
configurePlugin, 526
enqueue, 528
getOutputDimensions, 528
getTensorRTVersion, 529
getWorkspaceSize, 529
kFORMAT_COMBINATION_LIMIT, 530
supportsFormatCombination, 529
nvinfer1::IPluginV2Ext, 531
~IPluginV2Ext, 532
attachToContext, 532
canBroadcastInputAcrossBatch, 533
clone, 534
configurePlugin, 534
configureWithFormat, 536
detachFromContext, 536
getOutputDataType, 536
getTensorRTVersion, 537
IPluginV2Ext, 532
isOutputBroadcastAcrossBatch, 537
nvinfer1::IPluginV2IOExt, 538
configurePlugin, 539
getTensorRTVersion, 540
supportsFormatCombination, 540
nvinfer1::IPluginV2Layer, 542
~IPluginV2Layer, 543
getPlugin, 543
mImpl, 543
nvinfer1::IPluginV3Layer, 546
~IPluginV3Layer, 546
getPlugin, 547
mImpl, 547
nvinfer1::IPoolingLayer, 564
~IPoolingLayer, 566
getAverageCountExcludesPadding, 566
getBlendFactor, 566
getPaddingMode, 566
getPaddingNd, 567
getPoolingType, 567
getPostPadding, 567
getPrePadding, 567
getStrideNd, 568
getWindowSizeNd, 568
mImpl, 571
setAverageCountExcludesPadding, 568
setBlendFactor, 568
setPaddingMode, 569
setPaddingNd, 569
setPoolingType, 569
setPostPadding, 570
setPrePadding, 570
setStrideNd, 570
setWindowSizeNd, 571
nvinfer1::IQuantizeLayer, 577
~IQuantizeLayer, 579
getAxis, 579
getToType, 579
mImpl, 580
setAxis, 579
setToType, 580
nvinfer1::IRaggedSoftMaxLayer, 581
~IRaggedSoftMaxLayer, 581
mImpl, 582
nvinfer1::IRecurrenceLayer, 582
~IRecurrenceLayer, 583
mImpl, 584
setInput, 583
nvinfer1::IReduceLayer, 584
~IReduceLayer, 585
getKeepDimensions, 585
getOperation, 585
getReduceAxes, 586
mImpl, 587
setKeepDimensions, 586
setOperation, 586
setReduceAxes, 586
nvinfer1::IRefitter, 587
~IRefitter, 589
getAll, 589
getAllWeights, 590
getDynamicRangeMax, 590
getDynamicRangeMin, 591
getErrorRecorder, 591
getLogger, 591
getMaxThreads, 592
getMissing, 592
getMissingWeights, 593
getNamedWeights, 593
getTensorsWithDynamicRange, 594
getWeightsLocation, 594
getWeightsPrototype, 595
getWeightsValidation, 595
mImpl, 601
refitCudaEngine, 595
refitCudaEngineAsync, 596
setDynamicRange, 596
setErrorRecorder, 597
setMaxThreads, 597
setNamedWeights, 598, 599
setWeights, 599
setWeightsValidation, 600

unsetNamedWeights, 600
nvinfer1::IResizeLayer, 601
~IResizeLayer, 603
getCoordinateTransformation, 604
getCubicCoeff, 604
getExcludeOutside, 604
getNearestRounding, 604
getOutputDimensions, 605
getResizeMode, 605
getScales, 605
getSelectorForSinglePixel, 606
mImpl, 610
setCoordinateTransformation, 606
setCubicCoeff, 606
setExcludeOutside, 607
setInput, 607
setNearestRounding, 608
setOutputDimensions, 608
setResizeMode, 609
setScales, 609
setSelectorForSinglePixel, 610
nvinfer1::IReverseSequenceLayer, 610
~IReverseSequenceLayer, 611
getBatchAxis, 611
getSequenceAxis, 612
mImpl, 613
setBatchAxis, 612
setSequenceAxis, 612
nvinfer1::IRuntime, 613
~IRuntime, 614
deserializeCudaEngine, 615, 616
getDLACore, 616
getEngineHostCodeAllowed, 616
getErrorRecorder, 617
getLogger, 617
getMaxThreads, 617
getNbDLACores, 618
getPluginRegistry, 618
getTempfileControlFlags, 618
getTemporaryDirectory, 618
loadRuntime, 619
mImpl, 623
setDLACore, 619
setEngineHostCodeAllowed, 620
setErrorRecorder, 620
setGpuAllocator, 621
setMaxThreads, 621
setTempfileControlFlags, 621
setTemporaryDirectory, 622
nvinfer1::IScaleLayer, 623
~IScaleLayer, 625
getChannelAxis, 625
getMode, 625
getPower, 626
getScale, 626
getShift, 626
mImpl, 628
setChannelAxis, 626
setMode, 627
setPower, 627
setScale, 627
setShift, 628
nvinfer1::IScatterLayer, 628
~IScatterLayer, 630
getAxis, 630
getMode, 630
mImpl, 631
setAxis, 631
setMode, 631
nvinfer1::ISelectLayer, 632
~ISelectLayer, 632
mImpl, 633
nvinfer1::ISerializationConfig, 633
~ISerializationConfig, 634
clearFlag, 634
getFlag, 634
getFlags, 635
mImpl, 636
setFlag, 635
setFlags, 635
nvinfer1::IShapeLayer, 636
~IShapeLayer, 637
mImpl, 637
nvinfer1::IShuffleLayer, 638
~IShuffleLayer, 639
getFirstTranspose, 639
getReshapeDimensions, 639
getSecondTranspose, 640
getZeroIsPlaceholder, 640
mImpl, 643
setFirstTranspose, 640
setInput, 641
setReshapeDimensions, 641
setSecondTranspose, 642
setZeroIsPlaceholder, 642
nvinfer1::ISliceLayer, 643
~ISliceLayer, 645
getAxes, 646
getMode, 646
getSize, 646
getStart, 647
getStride, 647
mImpl, 650
setAxes, 647
setInput, 648
setMode, 648
setSize, 649
setStart, 649

setStride, 650
nvinfer1::ISoftMaxLayer, 650
~ISoftMaxLayer, 651
getAxes, 652
mImpl, 652
setAxes, 652
nvinfer1::ISqueezeLayer, 653
~ISqueezeLayer, 653
mImpl, 654
setInput, 654
nvinfer1::ITensor, 660
~ITensor, 662
dynamicRangeIsSet, 663
getAllowedFormats, 663
getBroadcastAcrossBatch, 663
getDimensionName, 663
getDimensions, 664
getDynamicRangeMax, 664
getDynamicRangeMin, 664
getLocation, 665
getName, 665
getType, 665
isExecutionTensor, 666
isNetworkInput, 666
isNetworkOutput, 666
isShapeTensor, 666
mImpl, 672
resetDynamicRange, 667
setAllowedFormats, 667
setBroadcastAcrossBatch, 668
setDimensionName, 668
setDimensions, 670
setDynamicRange, 670
setLocation, 671
setName, 671
setType, 672
nvinfer1::ITimingCache, 673
~ITimingCache, 674
combine, 674
mImpl, 677
query, 675
queryKeys, 675
reset, 675
serialize, 676
update, 676
nvinfer1::ITopKLayer, 677
~ITopKLayer, 678
getK, 679
getOperation, 679
getReduceAxes, 679
mImpl, 681
setInput, 679
setK, 680
setOperation, 680
setReduceAxes, 680
nvinfer1::ITripLimitLayer, 681
~ITripLimitLayer, 682
getTripLimit, 682
mImpl, 682
nvinfer1::IUnaryLayer, 683
~IUnaryLayer, 684
getOperation, 684
mImpl, 684
setOperation, 684
nvinfer1::IUnsqueezeLayer, 685
~IUnsqueezeLayer, 686
mImpl, 686
setInput, 686
nvinfer1::IVersionedInterface, 687
~IVersionedInterface, 688
getAPILanguage, 688
getInterfaceInfo, 689
IVersionedInterface, 688
operator=, 689
nvinfer1::Permutation, 691
order, 692
nvinfer1::plugin, 87
CENTER_SIZE, 88
CodeTypeSSD, 88
CORNER, 88
CORNER_SIZE, 88
TF_CENTER, 88
nvinfer1::plugin::DetectionOutputParameters, 95
backgroundLabelId, 96
codeType, 96
confidenceThreshold, 96
confSigmoid, 96
inputOrder, 96
isBatchAgnostic, 97
isNormalized, 97
keepTopK, 97
nmsThreshold, 97
numClasses, 97
shareLocation, 97
topK, 98
varianceEncodedInTarget, 98
nvinfer1::plugin::GridAnchorParameters, 125
aspectRatios, 126
H, 126
maxSize, 126
minSize, 126
numAspectRatios, 126
variance, 127
W, 127
nvinfer1::plugin::NMSParameters, 689
backgroundLabelId, 690
iouThreshold, 690
isNormalized, 690

keepTopK, 690
numClasses, 691
scoreThreshold, 691
shareLocation, 691
topK, 691
nvinfer1::plugin::PriorBoxParameters, 697
aspectRatios, 698
clip, 698
flip, 699
imgH, 699
imgW, 699
maxSize, 699
minSize, 699
numAspectRatios, 699
numMaxSize, 700
numMinSize, 700
offset, 700
stepH, 700
stepW, 700
variance, 700
nvinfer1::plugin::RegionParameters, 701
classes, 701
coords, 702
num, 702
smTree, 702
nvinfer1::plugin::RPROIParams, 702
anchorsRatioCount, 703
anchorsScaleCount, 703
featureStride, 703
iouThreshold, 704
minBoxSize, 704
nmsMaxOut, 704
poolingH, 704
poolingW, 704
preNmsTop, 704
spatialScale, 705
nvinfer1::plugin::softmaxTree, 705
child, 705
group, 706
groupOffset, 706
groups, 706
groupSize, 706
leaf, 706
n, 706
name, 706
parent, 707
nvinfer1::PluginField, 692
data, 693
length, 693
name, 693
PluginField, 693
type, 693
nvinfer1::PluginFieldCollection, 694
fields, 694
nbFields, 694
nvinfer1::PluginRegistrar< T >, 695
PluginRegistrar, 695
nvinfer1::PluginTensorDesc, 696
dims, 696
format, 696
scale, 696
type, 697
nvinfer1::safe, 88
nvinfer1::serialize, 88
serializeNetwork, 88
nvinfer1::v_1_0, 89
nvinfer1::v_1_0::IAlgorithmSelector, 139
~IAlgorithmSelector, 140
getInterfaceInfo, 140
reportAlgorithms, 140
selectAlgorithms, 141
nvinfer1::v_1_0::IDebugListener, 232
~IDebugListener, 232
getInterfaceInfo, 232
processDebugTensor, 233
nvinfer1::v_1_0::IErrorRecorder, 264
~IErrorRecorder, 265
clear, 265
decRefCount, 266
ErrorDesc, 265
getErrorCode, 266
getErrorDesc, 267
getInterfaceInfo, 268
getNbErrors, 268
hasOverflowed, 268
IErrorRecorder, 265
incRefCount, 269
kMAX_DESC_LENGTH, 270
RefCount, 265
reportError, 269
nvinfer1::v_1_0::IGpuAllocator, 315
~IGpuAllocator, 315
allocate, 316
allocateAsync, 316
deallocate, 317
deallocateAsync, 318
getInterfaceInfo, 319
IGpuAllocator, 315
reallocate, 319
nvinfer1::v_1_0::IGpuAsyncAllocator, 321
~IGpuAsyncAllocator, 321
allocate, 322
allocateAsync, 322
deallocate, 323
deallocateAsync, 324
getInterfaceInfo, 325
IGpuAsyncAllocator, 321
nvinfer1::v_1_0::IInt8EntropyCalibrator, 344

~IInt8EntropyCalibrator, 345
getAlgorithm, 345
getInterfaceInfo, 345
nvinfer1::v_1_0::IInt8EntropyCalibrator2, 346
~IInt8EntropyCalibrator2, 347
getAlgorithm, 347
getInterfaceInfo, 347
nvinfer1::v_1_0::IInt8LegacyCalibrator, 348
~IInt8LegacyCalibrator, 349
getAlgorithm, 349
getInterfaceInfo, 349
getQuantile, 349
getRegressionCutoff, 349
readHistogramCache, 350
writeHistogramCache, 350
nvinfer1::v_1_0::IInt8MinMaxCalibrator, 351
~IInt8MinMaxCalibrator, 352
getAlgorithm, 352
getInterfaceInfo, 352
nvinfer1::v_1_0::IOutputAllocator, 460
getInterfaceInfo, 461
notifyShape, 461
reallocateOutput, 461
reallocateOutputAsync, 462
nvinfer1::v_1_0::IPluginCapability, 483
nvinfer1::v_1_0::IPluginCreator, 484
~IPluginCreator, 485
createPlugin, 485
deserializePlugin, 485
getFieldNames, 486
getInterfaceInfo, 486
getPluginName, 487
getPluginNamespace, 487
getPluginVersion, 487
IPluginCreator, 484
setPluginNamespace, 488
nvinfer1::v_1_0::IPluginCreatorInterface, 489
~IPluginCreatorInterface, 490
IPluginCreatorInterface, 490
operator=, 490
nvinfer1::v_1_0::IPluginCreatorV3One, 491
~IPluginCreatorV3One, 492
createPlugin, 493
getFieldNames, 493
getInterfaceInfo, 494
getPluginName, 494
getPluginNamespace, 494
getPluginVersion, 495
IPluginCreatorV3One, 492, 493
operator=, 495
nvinfer1::v_1_0::IPluginResource, 506
~IPluginResource, 507
clone, 508
getInterfaceInfo, 508
IPluginResource, 507
operator=, 508
release, 509
nvinfer1::v_1_0::IPluginV3, 544
clone, 544
getCapabilityInterface, 545
getInterfaceInfo, 545
nvinfer1::v_1_0::IPluginV3OneBuild, 548
configurePlugin, 549
getFormatCombinationLimit, 550
getInterfaceInfo, 550
getMetadataString, 550
getNbOutputs, 550
getNbTactics, 551
getOutputDataTypes, 551
getOutputShapes, 552
getTimingCacheID, 552
getValidTactics, 553
getWorkspaceSize, 554
kDEFAULT_FORMAT_COMBINATION_LIMIT, 555
supportsFormatCombination, 554
nvinfer1::v_1_0::IPluginV3OneCore, 558
getInterfaceInfo, 558
getPluginName, 559
getPluginNamespace, 559
getPluginVersion, 559
nvinfer1::v_1_0::IPluginV3OneRuntime, 560
attachToContext, 561
enqueue, 562
getFieldsToSerialize, 562
getInterfaceInfo, 563
onShapeChange, 563
setTactic, 564
nvinfer1::v_1_0::IProfiler, 572
~IProfiler, 572
reportLayerTime, 572
nvinfer1::v_1_0::IProgressMonitor, 574
~IProgressMonitor, 574
getInterfaceInfo, 575
IProgressMonitor, 574
phaseFinish, 575
phaseStart, 575
stepComplete, 576
nvinfer1::v_1_0::IStreamReader, 655
~IStreamReader, 655
getInterfaceInfo, 656
IStreamReader, 656
operator=, 656
read, 657
nvinfer1::v_1_0::IStreamReaderV2, 658
~IStreamReaderV2, 658
getInterfaceInfo, 659
IStreamReaderV2, 658, 659

operator=, 659
read, 659
seek, 660
nvinfer1::v_1_0::TimingCacheKey, 707
data, 707
nvinfer1::v_1_0::TimingCacheValue, 708
kINVALID_TACTIC_HASH, 708
tacticHash, 708
timingMSec, 708
nvinfer1::v_2_0, 90
nvinfer1::v_2_0::IPluginV3OneBuild, 555
getAliasedInput, 556
getInterfaceInfo, 556
nvinfer1::Weights, 709
count, 710
type, 710
values, 710
NvInferImpl.h, 766
NvInferLegacyDims.h, 782, 783
NV_INFER_INTERNAL_INCLUDE, 783
NvInferPlugin.h, 785
initLibNvInferPlugins, 785
NvInferPluginBase.h, 786, 787
NV_INFER_INTERNAL_INCLUDE, 787
NvInferPluginUtils.h, 789, 790
NvInferRuntime.h, 792, 798
getInferLibBuildVersion, 797
getInferLibMajorVersion, 797
getInferLibMinorVersion, 797
getInferLibPatchVersion, 797
getLogger, 798
getPluginRegistry, 798
NV_INFER_INTERNAL_INCLUDE, 797
REGISTER_TENSORRT_PLUGIN, 797
NvInferRuntimeBase.h, 821, 824
getInferLibVersion, 824
NV_TENSORRT_VERSION, 823
NV_TENSORRT_VERSION_INT, 823
TENSORRTAPI, 823
TRT_DEPRECATED, 823
TRT_DEPRECATED_API, 823
TRT_DEPRECATED_ENUM, 823
TRTNOEXCEPT, 823
NvInferRuntimeCommon.h, 828, 829
NV_INFER_INTERNAL_INCLUDE, 828
NvInferRuntimePlugin.h, 830, 831
NV_INFER_INTERNAL_INCLUDE, 831
NvInferSerialize.h, 835
NvInferVersion.h, 836, 838
NV_TENSORRT_BUILD, 836
NV_TENSORRT_LWS_MAJOR, 836
NV_TENSORRT_LWS_MINOR, 837
NV_TENSORRT_LWS_PATCH, 837
NV_TENSORRT_MAJOR, 837
NV_TENSORRT_MINOR, 837
NV_TENSORRT_PATCH, 837
NV_TENSORRT_RELEASE_TYPE, 837
NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS, 838
NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY, 838
NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE, 838
NvOnnxConfig.h, 839, 840
nvonnxparser, 90
createONNXConfig, 92
EnumMax, 92
EnumMax< ErrorCode >, 93
EnumMax< OnnxParserFlag >, 93
ErrorCode, 91
kINTERNAL_ERROR, 92
kINVALID_GRAPH, 92
kINVALID_NODE, 92
kINVALID_VALUE, 92
kMEM_ALLOC_FAILED, 92
kMODEL_DESERIALIZE_FAILED, 92
kNATIVE_INSTANCENORM, 92
kREFIT_FAILED, 92
kSUCCESS, 92
kUNSUPPORTED_GRAPH, 92
kUNSUPPORTED_NODE, 92
kUNSUPPORTED_NODE_ATTR, 92
kUNSUPPORTED_NODE_DATATYPE, 92
kUNSUPPORTED_NODE_DYNAMIC, 92
kUNSUPPORTED_NODE_INPUT, 92
kUNSUPPORTED_NODE_SHAPE, 92
OnnxParserFlag, 92
OnnxParserFlags, 91
NvOnnxParser.h, 841, 844
createNvOnnxParser_INTERNAL, 843
createNvOnnxParserRefitter_INTERNAL, 843
getNvOnnxParserVersion, 844
NV_ONNX_PARSER_MAJOR, 842
NV_ONNX_PARSER_MINOR, 842
NV_ONNX_PARSER_PATCH, 843
SubGraph_t, 843
SubGraphCollection_t, 843
nvonnxparser::anonymous_namespace{NvOnnxParser.h}, 93
createParser, 93
createParserRefitter, 94
nvonnxparser::IOnnxConfig, 447
~IOnnxConfig, 449
addVerbosity, 449
getFullTextFileName, 449
getModelDtype, 449
getModelFileName, 450
getPrintLayerInfo, 450

getTextFileName, 450
getVerbosityLevel, 451
reduceVerbosity, 451
setFullTextFileName, 451
setModelDtype, 452
setModelFileName, 452
setPrintLayerInfo, 452
setTextFileName, 453
setVerbosityLevel, 453
Verbosity, 448
nvonnxparser::IParser, 467
~IParser, 469
clearErrors, 469
clearFlag, 469
getError, 469
getFlag, 470
getFlags, 470
getLayerOutputTensor, 470
getNbErrors, 471
getNbSubgraphs, 471
getSubgraphNodes, 471
getUsedVCPluginLibraries, 472
isSubgraphSupported, 472
parse, 472
parseFromFile, 473
parseWithWeightDescriptors, 473
setFlag, 474
setFlags, 474
supportsModel, 475
supportsModelV2, 475
supportsOperator, 476
nvonnxparser::IParserError, 476
~IParserError, 477
code, 477
desc, 478
file, 478
func, 478
line, 478
localFunctionStack, 478
localFunctionStackSize, 478
node, 479
nodeName, 479
nodeOperator, 479
nvonnxparser::IParserRefitter, 479
~IParserRefitter, 480
clearErrors, 480
getError, 480
getNbErrors, 481
refitFromBytes, 481
refitFromFile, 481

offset
nvinfer1::plugin::PriorBoxParameters, 700
OnnxParserFlag
nvonnxparser, 92
OnnxParserFlags
nvonnxparser, 91
onShapeChange
nvinfer1::v_1_0::IPluginV3OneRuntime, 563
operation
nvinfer1::IExprBuilder, 299
operator=
nvinfer1::INoCopy, 437
nvinfer1::IPluginResourceContext, 511
nvinfer1::IVersionedInterface, 689
nvinfer1::v_1_0::IPluginCreatorInterface, 490
nvinfer1::v_1_0::IPluginCreatorV3One, 495
nvinfer1::v_1_0::IPluginResource, 508
nvinfer1::v_1_0::IStreamReader, 656
nvinfer1::v_1_0::IStreamReaderV2, 659
opt
nvinfer1::DynamicPluginTensorDesc, 108
OptProfileSelector
nvinfer1, 60
order
nvinfer1::Permutation, 692
outputTypeIsSet
nvinfer1::ILayer, 360

PaddingMode
nvinfer1, 61
parent
nvinfer1::plugin::softmaxTree, 707
parse
nvonnxparser::IParser, 472
parseFromFile
nvonnxparser::IParser, 473
parseWithWeightDescriptors
nvonnxparser::IParser, 473
phaseFinish
nvinfer1::v_1_0::IProgressMonitor, 575
phaseStart
nvinfer1::v_1_0::IProgressMonitor, 575
platformHasFastFp16
nvinfer1::IBuilder, 152
platformHasFastInt8
nvinfer1::IBuilder, 153
platformHasTf32
nvinfer1::IBuilder, 153
PluginCapabilityType
nvinfer1, 63
PluginCreatorVersion
nvinfer1, 63
PluginField
nvinfer1::PluginField, 693
PluginFieldType
nvinfer1, 64
PluginFormat

nvinfer1, 42
PluginLibraryHandle
nvinfer1::IPluginRegistry, 497
PluginRegistrar
nvinfer1::PluginRegistrar< T >, 695
PluginVersion, 697
nvinfer1, 64
poolingH
nvinfer1::plugin::RPROIParams, 704
PoolingType
nvinfer1, 65
poolingW
nvinfer1::plugin::RPROIParams, 704
precisionIsSet
nvinfer1::ILayer, 361
preNmsTop
nvinfer1::plugin::RPROIParams, 704
PreviewFeature
nvinfer1, 65
processDebugTensor
nvinfer1::v_1_0::IDebugListener, 233
ProfilingVerbosity
nvinfer1, 65

QuantizationFlag
nvinfer1, 66
QuantizationFlags
nvinfer1, 43
query
nvinfer1::ITimingCache, 675
queryKeys
nvinfer1::ITimingCache, 675

read
nvinfer1::v_1_0::IStreamReader, 657
nvinfer1::v_1_0::IStreamReaderV2, 659
readCalibrationCache
nvinfer1::IInt8Calibrator, 343
readHistogramCache
nvinfer1::v_1_0::IInt8LegacyCalibrator, 350
reallocate
nvinfer1::v_1_0::IGpuAllocator, 319
reallocateOutput
nvinfer1::v_1_0::IOutputAllocator, 461
reallocateOutputAsync
nvinfer1::v_1_0::IOutputAllocator, 462
ReduceOperation
nvinfer1, 66
reduceVerbosity
nvonnxparser::IOnnxConfig, 451
RefCount
nvinfer1::v_1_0::IErrorRecorder, 265
refitCudaEngine
nvinfer1::IRefitter, 595
refitCudaEngineAsync
nvinfer1::IRefitter, 596
refitFromBytes
nvonnxparser::IParserRefitter, 481
refitFromFile
nvonnxparser::IParserRefitter, 481
REGISTER_TENSORRT_PLUGIN
NvInferRuntime.h, 797
registerCreator
nvinfer1::IPluginRegistry, 503, 504
release
nvinfer1::v_1_0::IPluginResource, 509
releasePluginResource
nvinfer1::IPluginRegistry, 504
removeTensor
nvinfer1::INetworkDefinition, 427
reportAlgorithms
nvinfer1::v_1_0::IAlgorithmSelector, 140
reportError
nvinfer1::v_1_0::IErrorRecorder, 269
reportLayerTime
nvinfer1::v_1_0::IProfiler, 572
reportToProfiler
nvinfer1::IExecutionContext, 284
reset
nvinfer1::IBuilder, 153
nvinfer1::IBuilderConfig, 170
nvinfer1::ITimingCache, 675
resetDeviceType
nvinfer1::IBuilderConfig, 171
resetDynamicRange
nvinfer1::ITensor, 667
resetOutputType
nvinfer1::ILayer, 361
resetPrecision
nvinfer1::ILayer, 361
ResizeCoordinateTransformation
nvinfer1, 67
ResizeRoundMode
nvinfer1, 67
ResizeSelector
nvinfer1, 68
RuntimePlatform
nvinfer1, 68

SampleMode
nvinfer1, 69
scale
nvinfer1::PluginTensorDesc, 696
ScaleMode
nvinfer1, 69
ScatterMode
nvinfer1, 70
scoreThreshold
nvinfer1::plugin::NMSParameters, 691

seek
nvinfer1::v_1_0::IStreamReaderV2, 660
SeekPosition
nvinfer1, 70
selectAlgorithms
nvinfer1::v_1_0::IAlgorithmSelector, 141
SerializationFlag
nvinfer1, 70
SerializationFlags
nvinfer1, 43
serialize
nvinfer1::ICudaEngine, 223
nvinfer1::IPluginV2, 521
nvinfer1::ITimingCache, 676
serializeNetwork
nvinfer1::serialize, 88
serializeWithConfig
nvinfer1::ICudaEngine, 224
setActivationType
nvinfer1::IActivationLayer, 129
setAlgorithmSelector
nvinfer1::IBuilderConfig, 171
setAlignCorners
nvinfer1::IGridSampleLayer, 327
setAllowedFormats
nvinfer1::ITensor, 667
setAllTensorsDebugState
nvinfer1::IExecutionContext, 284
setAlpha
nvinfer1::IActivationLayer, 130
nvinfer1::IFillLayer, 304
nvinfer1::ILRNLayer, 379
setAlphaInt64
nvinfer1::IFillLayer, 305
setAuxStreams
nvinfer1::IExecutionContext, 285
setAverageCountExcludesPadding
nvinfer1::IPoolingLayer, 568
setAvgTimingIterations
nvinfer1::IBuilderConfig, 171
setAxes
nvinfer1::INormalizationLayer, 442
nvinfer1::ISliceLayer, 647
nvinfer1::ISoftMaxLayer, 652
setAxis
nvinfer1::IConcatenationLayer, 187
nvinfer1::IDequantizeLayer, 245
nvinfer1::IDynamicQuantizeLayer, 251
nvinfer1::IIteratorLayer, 354
nvinfer1::ILoopOutputLayer, 375
nvinfer1::IOneHotLayer, 446
nvinfer1::IQuantizeLayer, 579
nvinfer1::IScatterLayer, 631
setBatchAxis
nvinfer1::IReverseSequenceLayer, 612
setBeta
nvinfer1::IActivationLayer, 130
nvinfer1::IFillLayer, 305
nvinfer1::ILRNLayer, 379
setBetaInt64
nvinfer1::IFillLayer, 306
setBiasWeights
nvinfer1::IConvolutionLayer, 197
nvinfer1::IDeconvolutionLayer, 238
setBlendFactor
nvinfer1::IPoolingLayer, 568
setBlockSize
nvinfer1::IDynamicQuantizeLayer, 252
setBoundingBoxFormat
nvinfer1::INMSLayer, 433
setBroadcastAcrossBatch
nvinfer1::ITensor, 668
setBuilderOptimizationLevel
nvinfer1::IBuilderConfig, 172
setCalibrationProfile
nvinfer1::IBuilderConfig, 172
setChannelAxis
nvinfer1::IScaleLayer, 626
setComputePrecision
nvinfer1::INormalizationLayer, 442
setCondition
nvinfer1::IIfConditional, 335
setCoordinateTransformation
nvinfer1::IResizeLayer, 606
setCubicCoeff
nvinfer1::IResizeLayer, 606
setDebugListener
nvinfer1::IExecutionContext, 286
setDebugSync
nvinfer1::IExecutionContext, 286
setDefaultDeviceType
nvinfer1::IBuilderConfig, 173
setDeviceMemory
nvinfer1::IExecutionContext, 286
setDeviceMemoryV2
nvinfer1::IExecutionContext, 287
setDeviceType
nvinfer1::IBuilderConfig, 173
setDilationNd
nvinfer1::IConvolutionLayer, 197
nvinfer1::IDeconvolutionLayer, 238
setDimensionName
nvinfer1::ITensor, 668
setDimensions
nvinfer1::IConstantLayer, 191
nvinfer1::IFillLayer, 306
nvinfer1::IOptimizationProfile, 457
nvinfer1::ITensor, 670

setDLACore
nvinfer1::IBuilderConfig, 174
nvinfer1::IRuntime, 619
setDynamicRange
nvinfer1::IRefitter, 596
nvinfer1::ITensor, 670
setEngineCapability
nvinfer1::IBuilderConfig, 174
setEngineHostCodeAllowed
nvinfer1::IRuntime, 620
setEnqueueEmitsProfile
nvinfer1::IExecutionContext, 287
setEpsilon
nvinfer1::INormalizationLayer, 443
setEquation
nvinfer1::IEinsumLayer, 255
setErrorRecorder
nvinfer1::IBuilder, 153
nvinfer1::ICudaEngine, 224
nvinfer1::IEngineInspector, 262
nvinfer1::IExecutionContext, 288
nvinfer1::INetworkDefinition, 427
nvinfer1::IPluginRegistry, 505
nvinfer1::IRefitter, 597
nvinfer1::IRuntime, 620
setExcludeOutside
nvinfer1::IResizeLayer, 607
setExclusive
nvinfer1::ICumulativeLayer, 230
setExecutionContext
nvinfer1::IEngineInspector, 262
setExtraMemoryTarget
nvinfer1::IOptimizationProfile, 458
setFirstTranspose
nvinfer1::IShuffleLayer, 640
setFlag
nvinfer1::IBuilderConfig, 174
nvinfer1::ISerializationConfig, 635
nvonnxparser::IParser, 474
setFlags
nvinfer1::IBuilderConfig, 175
nvinfer1::ISerializationConfig, 635
nvonnxparser::IParser, 474
setFullTextFileName
nvonnxparser::IOnnxConfig, 451
setGatherAxis
nvinfer1::IGatherLayer, 312
setGpuAllocator
nvinfer1::IBuilder, 154
nvinfer1::IRuntime, 621
setHardwareCompatibilityLevel
nvinfer1::IBuilderConfig, 175
setInput
nvinfer1::IConvolutionLayer, 197
nvinfer1::IDeconvolutionLayer, 239
nvinfer1::IDynamicQuantizeLayer, 252
nvinfer1::IFillLayer, 307
nvinfer1::ILayer, 362
nvinfer1::ILoopOutputLayer, 376
nvinfer1::INMSLayer, 434
nvinfer1::IRecurrenceLayer, 583
nvinfer1::IResizeLayer, 607
nvinfer1::IShuffleLayer, 641
nvinfer1::ISliceLayer, 648
nvinfer1::ISqueezeLayer, 654
nvinfer1::ITopKLayer, 679
nvinfer1::IUnsqueezeLayer, 686
setInputConsumedEvent
nvinfer1::IExecutionContext, 288
setInputShape
nvinfer1::IExecutionContext, 289
setInputTensorAddress
nvinfer1::IExecutionContext, 289
setInt8Calibrator
nvinfer1::IBuilderConfig, 176
setInterpolationMode
nvinfer1::IGridSampleLayer, 328
setK
nvinfer1::ILRNLayer, 379
nvinfer1::ITopKLayer, 680
setKeepDimensions
nvinfer1::IReduceLayer, 586
setKernelSizeNd
nvinfer1::IConvolutionLayer, 198
nvinfer1::IDeconvolutionLayer, 239
setKernelWeights
nvinfer1::IConvolutionLayer, 198
nvinfer1::IDeconvolutionLayer, 239
setL2LimitForTiling
nvinfer1::IBuilderConfig, 176
setLocation
nvinfer1::ITensor, 671
setMaxAuxStreams
nvinfer1::IBuilderConfig, 176
setMaxNbTactics
nvinfer1::IBuilderConfig, 177
setMaxThreads
nvinfer1::IBuilder, 154
nvinfer1::IRefitter, 597
nvinfer1::IRuntime, 621
setMemoryPoolLimit
nvinfer1::IBuilderConfig, 177
setMessage
nvinfer1::IAssertionLayer, 145
setMetadata
nvinfer1::ILayer, 362
setMode
nvinfer1::IGatherLayer, 313

nvinfer1::IScaleLayer, 627
nvinfer1::IScatterLayer, 631
nvinfer1::ISliceLayer, 648
setModelDtype
nvonnxparser::IOnnxConfig, 452
setModelFileName
nvonnxparser::IOnnxConfig, 452
setName
nvinfer1::IExecutionContext, 290
nvinfer1::IIfConditional, 335
nvinfer1::ILayer, 363
nvinfer1::ILoop, 371
nvinfer1::INetworkDefinition, 428
nvinfer1::ITensor, 671
setNamedWeights
nvinfer1::IRefitter, 598, 599
setNbElementWiseDims
nvinfer1::IGatherLayer, 313
setNbGroups
nvinfer1::IConvolutionLayer, 198
nvinfer1::IDeconvolutionLayer, 240
nvinfer1::INormalizationLayer, 443
setNbOutputMaps
nvinfer1::IConvolutionLayer, 199
nvinfer1::IDeconvolutionLayer, 240
setNearestRounding
nvinfer1::IResizeLayer, 608
setNvtxVerbosity
nvinfer1::IExecutionContext, 290
setOperation
nvinfer1::ICumulativeLayer, 230
nvinfer1::IElementWiseLayer, 257
nvinfer1::IFillLayer, 308
nvinfer1::IMatrixMultiplyLayer, 382
nvinfer1::IReduceLayer, 586
nvinfer1::ITopKLayer, 680
nvinfer1::IUnaryLayer, 684
setOptimizationProfileAsync
nvinfer1::IExecutionContext, 291
setOutputAllocator
nvinfer1::IExecutionContext, 292
setOutputDimensions
nvinfer1::IResizeLayer, 608
setOutputTensorAddress
nvinfer1::IExecutionContext, 292
setOutputType
nvinfer1::ILayer, 363
setPaddingMode
nvinfer1::IConvolutionLayer, 199
nvinfer1::IDeconvolutionLayer, 240
nvinfer1::IPoolingLayer, 569
setPaddingNd
nvinfer1::IConvolutionLayer, 199
nvinfer1::IDeconvolutionLayer, 241
nvinfer1::IPoolingLayer, 569
setParentSearchEnabled
nvinfer1::IPluginRegistry, 505
setPersistentCacheLimit
nvinfer1::IExecutionContext, 293
setPluginNamespace
nvinfer1::IPluginV2, 522
nvinfer1::v_1_0::IPluginCreator, 488
setPluginsToSerialize
nvinfer1::IBuilderConfig, 178
setPoolingType
nvinfer1::IPoolingLayer, 569
setPostPadding
nvinfer1::IConvolutionLayer, 200
nvinfer1::IDeconvolutionLayer, 241
nvinfer1::IPoolingLayer, 570
setPostPaddingNd
nvinfer1::IPaddingLayer, 465
setPower
nvinfer1::IScaleLayer, 627
setPrecision
nvinfer1::ILayer, 364
setPrePadding
nvinfer1::IConvolutionLayer, 200
nvinfer1::IDeconvolutionLayer, 241
nvinfer1::IPoolingLayer, 570
setPrePaddingNd
nvinfer1::IPaddingLayer, 465
setPreviewFeature
nvinfer1::IBuilderConfig, 179
setPrintLayerInfo
nvonnxparser::IOnnxConfig, 452
setProfiler
nvinfer1::IExecutionContext, 294
setProfileStream
nvinfer1::IBuilderConfig, 179
setProfilingVerbosity
nvinfer1::IBuilderConfig, 179
setProgressMonitor
nvinfer1::IBuilderConfig, 180
setQuantizationFlag
nvinfer1::IBuilderConfig, 180
setQuantizationFlags
nvinfer1::IBuilderConfig, 180
setReduceAxes
nvinfer1::IReduceLayer, 586
nvinfer1::ITopKLayer, 680
setReshapeDimensions
nvinfer1::IShuffleLayer, 641
setResizeMode
nvinfer1::IResizeLayer, 609
setReverse
nvinfer1::ICumulativeLayer, 231
nvinfer1::IIteratorLayer, 354

setRuntimePlatform
nvinfer1::IBuilderConfig, 181
setSampleMode
nvinfer1::IGridSampleLayer, 328
setScale
nvinfer1::IScaleLayer, 627
setScales
nvinfer1::IResizeLayer, 609
setScaleType
nvinfer1::IDynamicQuantizeLayer, 253
setSecondTranspose
nvinfer1::IShuffleLayer, 642
setSelectorForSinglePixel
nvinfer1::IResizeLayer, 610
setSequenceAxis
nvinfer1::IReverseSequenceLayer, 612
setShapeValues
nvinfer1::IOptimizationProfile, 458
setShift
nvinfer1::IScaleLayer, 628
setSize
nvinfer1::ISliceLayer, 649
setStart
nvinfer1::ISliceLayer, 649
setStride
nvinfer1::ISliceLayer, 650
setStrideNd
nvinfer1::IConvolutionLayer, 200
nvinfer1::IDeconvolutionLayer, 242
nvinfer1::IPoolingLayer, 570
setTactic
nvinfer1::v_1_0::IPluginV3OneRuntime, 564
setTacticSources
nvinfer1::IBuilderConfig, 181
setTempfileControlFlags
nvinfer1::IRuntime, 621
setTemporaryDirectory
nvinfer1::IRuntime, 622
setTemporaryStorageAllocator
nvinfer1::IExecutionContext, 294
setTensorAddress
nvinfer1::IExecutionContext, 294
setTensorDebugState
nvinfer1::IExecutionContext, 296
setTextFileName
nvonnxparser::IOnnxConfig, 453
setTilingOptimizationLevel
nvinfer1::IBuilderConfig, 182
setTimingCache
nvinfer1::IBuilderConfig, 182
setTopKBoxLimit
nvinfer1::INMSLayer, 434
setToType
nvinfer1::ICastLayer, 185
nvinfer1::IDequantizeLayer, 246
nvinfer1::IDynamicQuantizeLayer, 253
nvinfer1::IFillLayer, 308
nvinfer1::IQuantizeLayer, 580
setType
nvinfer1::ITensor, 672
setVerbosityLevel
nvonnxparser::IOnnxConfig, 453
setWeights
nvinfer1::IConstantLayer, 191
nvinfer1::IRefitter, 599
setWeightsName
nvinfer1::INetworkDefinition, 428
setWeightStreamingBudget
nvinfer1::ICudaEngine, 225
setWeightStreamingBudgetV2
nvinfer1::ICudaEngine, 226
setWeightsValidation
nvinfer1::IRefitter, 600
setWindowSize
nvinfer1::ILRNLayer, 380
setWindowSizeNd
nvinfer1::IPoolingLayer, 571
setZeroIsPlaceholder
nvinfer1::IShuffleLayer, 642
Severity
nvinfer1::ILogger, 366
shareLocation
nvinfer1::plugin::DetectionOutputParameters, 97
nvinfer1::plugin::NMSParameters, 691
size
nvinfer1::IHostMemory, 330
smTree
nvinfer1::plugin::RegionParameters, 702
spatialScale
nvinfer1::plugin::RPROIParams, 705
stepComplete
nvinfer1::v_1_0::IProgressMonitor, 576
stepH
nvinfer1::plugin::PriorBoxParameters, 700
stepW
nvinfer1::plugin::PriorBoxParameters, 700
SubGraph_t
NvOnnxParser.h, 843
SubGraphCollection_t
NvOnnxParser.h, 843
supportsFormat
nvinfer1::IPluginV2, 522
supportsFormatCombination
nvinfer1::IPluginV2DynamicExt, 529
nvinfer1::IPluginV2IOExt, 540
nvinfer1::v_1_0::IPluginV3OneBuild, 554
supportsModel
nvonnxparser::IParser, 475

supportsModelV2
nvonnxparser::IParser, 475
supportsOperator
nvonnxparser::IParser, 476

tacticHash
nvinfer1::v_1_0::TimingCacheValue, 708
TacticSource
nvinfer1, 71
TacticSources
nvinfer1, 43
TempfileControlFlag
nvinfer1, 72
TempfileControlFlags
nvinfer1, 43
TensorFormat
nvinfer1, 72
TensorFormats
nvinfer1, 44
TensorIOMode
nvinfer1, 74
TensorLocation
nvinfer1, 74
TENSORRTAPI
NvInferRuntimeBase.h, 823
TensorRTPhase
nvinfer1, 74
terminate
nvinfer1::IPluginV2, 523
TF_CENTER
nvinfer1::plugin, 88
TilingOptimizationLevel
nvinfer1, 74
timingMSec
nvinfer1::v_1_0::TimingCacheValue, 708
topK
nvinfer1::plugin::DetectionOutputParameters, 98
nvinfer1::plugin::NMSParameters, 691
TopKOperation
nvinfer1, 75
TripLimit
nvinfer1, 75
TRT_DEPRECATED
NvInferRuntimeBase.h, 823
TRT_DEPRECATED_API
NvInferRuntimeBase.h, 823
TRT_DEPRECATED_ENUM
NvInferRuntimeBase.h, 823
TRTNOEXCEPT
NvInferRuntimeBase.h, 823
type
nvinfer1::IHostMemory, 330
nvinfer1::PluginField, 693
nvinfer1::PluginTensorDesc, 697
nvinfer1::Weights, 710

UnaryOperation
nvinfer1, 75
unmarkDebug
nvinfer1::INetworkDefinition, 429
unmarkOutput
nvinfer1::INetworkDefinition, 430
unmarkOutputForShapes
nvinfer1::INetworkDefinition, 430
unmarkWeightsRefittable
nvinfer1::INetworkDefinition, 430
unsetNamedWeights
nvinfer1::IRefitter, 600
update
nvinfer1::ITimingCache, 676
updateDeviceMemorySizeForShapes
nvinfer1::IExecutionContext, 296

Value, 709
values
nvinfer1::Weights, 710
variance
nvinfer1::plugin::GridAnchorParameters, 127
nvinfer1::plugin::PriorBoxParameters, 700
varianceEncodedInTarget
nvinfer1::plugin::DetectionOutputParameters, 98
Verbosity
nvonnxparser::IOnnxConfig, 448

W
nvinfer1::plugin::GridAnchorParameters, 127
w
nvinfer1::DimsHW, 107
WeightsRole
nvinfer1, 77
writeCalibrationCache
nvinfer1::IInt8Calibrator, 343
writeHistogramCache
nvinfer1::v_1_0::IInt8LegacyCalibrator, 350

NVIDIA TensorRT Standard Python API Documentation

Release 10.10.10

NVIDIA

Apr 16, 2025

OVERVIEW

1 Getting Started with TensorRT **1**
- 1.1 Installation . . . 1
- 1.2 Samples . . . 1
- 1.3 Operator Documentation . . . 1
- 1.4 Installing `cuda-python` . . . 1

2 Core Concepts **3**
- 2.1 TensorRT Workflow . . . 3
- 2.2 Classes Overview . . . 3
  - 2.2.1 Logger . . . 3
  - 2.2.2 Parsers . . . 3
  - 2.2.3 Network . . . 3
  - 2.2.4 Builder . . . 4
  - 2.2.5 Engine and Context . . . 4

3 Writing custom operators with TensorRT Python plugins **5**
- 3.1 Composition of a plugin . . . 5
- 3.2 Example: Circular padding plugin . . . 6
  - 3.2.1 Providing an Ahead-of-Time (AOT) implementation . . . 7
  - 3.2.2 Picking the most performant plugin configuration: Autotuning . . . 10
  - 3.2.3 Adding the plugin to a TensorRT network . . . 11
    - 3.2.3.1 Adding the plugin using TRT Python APIs . . . 11
    - 3.2.3.2 Loading an ONNX model with the custom operator . . . 11
- 3.3 Advanced usage . . . 12
  - 3.3.1 Example: Operators with data-dependent output shapes - Non-zero . . . 12
  - 3.3.2 Example: Implementing in-place custom ops with I/O aliasing . . . 13
  - 3.3.3 Example: Plugins with multiple backends: Using custom tactics . . . 13
- 3.4 FAQ . . . 14
- 3.5 Limitations . . . 14
- 3.6 Additional resources . . . 15

4 Foundational Types **17**
- 4.1 DataType . . . 17
- 4.2 Weights . . . 17
- 4.3 Dims . . . 17
  - 4.3.1 Volume . . . 17
  - 4.3.2 Dims . . . 17
  - 4.3.3 Dims2 . . . 17
  - 4.3.4 DimsHW . . . 17
  - 4.3.5 Dims3 . . . 17

4.3.6 Dims4 . . . 17
4.4 IHostMemory . . . 17

5 Core **19**
5.1 Logger . . . 20
5.2 Profiler . . . 20
5.3 IOptimizationProfile . . . 20
5.4 IBuilderConfig . . . 20
5.5 Builder . . . 20
5.5.1 NetworkDefinitionCreationFlag . . . 20
5.5.2 Builder . . . 20
5.6 Debug Listener . . . 20
5.6.1 IDebugListener . . . 20
5.7 ICudaEngine . . . 20
5.8 IExecutionContext . . . 20
5.9 Runtime . . . 20
5.10 Refitter . . . 20
5.11 IErrorRecorder . . . 20
5.12 ITimingCache . . . 20
5.13 GPU Allocator . . . 20
5.13.1 AllocatorFlag . . . 20
5.13.2 IGpuAllocator . . . 20
5.14 EngineInspector . . . 20
5.15 ISerializationConfig . . . 20

6 Network **21**
6.1 INetworkDefinition . . . 21
6.2 Layer Base Classes . . . 21
6.2.1 ITensor . . . 21
6.2.2 ILayer . . . 21
6.3 Layers . . . 21
6.3.1 PaddingMode . . . 23
6.3.2 ICastLayer . . . 23
6.3.3 IConvolutionLayer . . . 23
6.3.4 IGridSampleLayer . . . 23
6.3.5 IActivationLayer . . . 23
6.3.6 IPoolingLayer . . . 23
6.3.7 ILRNLayer . . . 23
6.3.8 IScaleLayer . . . 23
6.3.9 ISoftMaxLayer . . . 23
6.3.10 IConcatenationLayer . . . 23
6.3.11 IDeconvolutionLayer . . . 23
6.3.12 IElementWiseLayer . . . 23
6.3.13 IGatherLayer . . . 23
6.3.14 IPluginV2Layer . . . 23
6.3.15 IPluginV3Layer . . . 23
6.3.16 IUnaryLayer . . . 23
6.3.17 IReduceLayer . . . 23
6.3.18 IPaddingLayer . . . 23
6.3.19 IParametricReLULayer . . . 23
6.3.20 ISelectLayer . . . 23
6.3.21 IShuffleLayer . . . 23
6.3.22 ISliceLayer . . . 23
6.3.23 IShapeLayer . . . 23

6.3.24 ITopKLayer . . . 23
6.3.25 IMatrixMultiplyLayer . . . 23
6.3.26 IRaggedSoftMaxLayer . . . 23
6.3.27 IIdentityLayer . . . 23
6.3.28 IConstantLayer . . . 23
6.3.29 IResizeLayer . . . 23
6.3.30 ILoop . . . 23
6.3.30.1 ILoopBoundaryLayer . . . 23
6.3.30.1.1 ITripLimitLayer . . . 23
6.3.30.1.2 IRecurrenceLayer . . . 23
6.3.30.1.3 IIteratorLayer . . . 23
6.3.30.1.4 ILoopOutputLayer . . . 23
6.3.31 IFillLayer . . . 23
6.3.32 IQuantizeLayer . . . 23
6.3.33 IDequantizeLayer . . . 23
6.3.34 IDynamicQuantizeLayer . . . 23
6.3.35 IScatterLayer . . . 23
6.3.36 IIfConditional . . . 23
6.3.37 IConditionLayer . . . 23
6.3.38 IIfConditionalOutputLayer . . . 23
6.3.39 IIfConditionalInputLayer . . . 23
6.3.40 IEinsumLayer . . . 23
6.3.41 IAssertionLayer . . . 23
6.3.42 IOneHotLayer . . . 23
6.3.43 INonZeroLayer . . . 23
6.3.44 INMSLayer . . . 23
6.3.45 IReverseSequenceLayer . . . 23
6.3.46 INormalizationLayer . . . 23
6.3.47 ISqueezeLayer . . . 23
6.3.48 IUnsqueezeLayer . . . 23
6.3.49 ICumulativeLayer . . . 23

7 Plugin **25**
7.1 IPluginCreator . . . 25
7.2 IPluginRegistry . . . 25
7.3 IPluginV2DynamicExt . . . 25
7.4 IPluginV3 . . . 25

8 tensorrt.plugin **27**
8.1 tensorrt.plugin.register . . . 27
8.2 tensorrt.plugin.impl . . . 27
8.3 tensorrt.plugin.aot_impl . . . 27
8.3.1 KernelLaunchParams . . . 27
8.4 tensorrt.plugin.autotune . . . 27
8.4.1 AutoTuneCombination . . . 27
8.5 Shape . . . 27
8.5.1 Shape . . . 27
8.5.2 ShapeExpr . . . 27
8.6 Tensor . . . 27
8.6.1 TensorDesc . . . 27
8.6.2 Tensor . . . 27
8.7 Functions . . . 27

9 Int8 **29**

9.1 IInt8Calibrator 29
9.2 IInt8LegacyCalibrator 29
9.3 IInt8EntropyCalibrator 29
9.4 IInt8EntropyCalibrator2 29
9.5 IInt8MinMaxCalibrator 29

10 Algorithm Selector **31**

11 Onnx Parser **33**

CHAPTER

ONE

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 Operator Documentation

For more information, including examples, refer to the TensorRT Operator's Reference documentation.

1.4 Installing `cuda-python`

Although not required by the TensorRT Python API, `cuda-python` is used in several samples. For installation instructions, refer to the CUDA Python Installation documentation.

CHAPTER

TWO

CORE CONCEPTS

2.1 TensorRT Workflow

The general TensorRT workflow consists of 3 steps:

1. Populate a `tensorrt.INetworkDefinition` either with a parser or by using the TensorRT Network API (see `tensorrt.INetworkDefinition` for more details). The `tensorrt.Builder` can be used to generate an empty `tensorrt.INetworkDefinition` .
2. Use the `tensorrt.Builder` to build a `tensorrt.ICudaEngine` using the populated `tensorrt.INetworkDefinition` .
3. Create a `tensorrt.IExecutionContext` from the `tensorrt.ICudaEngine` and use it to perform optimized inference.

2.2 Classes Overview

2.2.1 Logger

Most other TensorRT classes use a logger to report errors, warnings and informative messages. TensorRT provides a basic `tensorrt.Logger` implementation, but you can write your own implementation by deriving from `tensorrt.ILogger` for more advanced functionality.

2.2.2 Parsers

Parsers are used to populate a `tensorrt.INetworkDefinition` from a model trained in a Deep Learning framework.

2.2.3 Network

The `tensorrt.INetworkDefinition` represents a computational graph. In order to populate the network, TensorRT provides a suite of parsers for a variety of Deep Learning frameworks. It is also possible to populate the network manually using the Network API.

2.2.4 Builder

The `tensorrt.Builder` is used to build a `tensorrt.ICudaEngine` . In order to do so, it must be provided a populated `tensorrt.INetworkDefinition` .

2.2.5 Engine and Context

The `tensorrt.ICudaEngine` is the output of the TensorRT optimizer. It is used to generate a `tensorrt.IExecutionContext` that can perform inference.

CHAPTER

THREE

WRITING CUSTOM OPERATORS WITH TENSORRT PYTHON PLUGINS

TensorRT (TRT) offers a wide array of built-in operators that fit most use cases. However, you may wish to introduce a custom operator for a variety of reasons, including:

- Supporting an entirely new operator that TRT does not support out-of-the-box
- Adding an operator with a slightly different behavior to an operator that is already supported

TRT provides support for custom operators through plugins. This guide shows how Python functions that define the behavior of a plugin can be implemented and wrapped such that they can be added to a network as a custom operator.

3.1 Composition of a plugin

Primarily, a plugin definition needs two functions, wrapped with decorators provided by the `tensorrt.plugin` module:

1. `tensorrt.plugin.register()`: Returns shape and type characteristics of output tensors. The function signature also defines the input tensors and any attributes the plugin needs to function.
2. A function defining the plugin computation. This can be either (or both)

- `tensorrt.plugin.impl()`: A Just-in-Time (JIT) compute definition
- `tensorrt.plugin.aot_impl()`: An Ahead-of-Time (AOT) compute definition

It is possible to specify both an AOT and JIT implementation for the same plugin at the same time. However, based on user choice, TensorRT will only use one of these implementations for a given plugin or plugin layer in a single build phase.

Optionally, if a plugin is able to support multiple data type/tensor layout combinations for its I/O, or is able to support multiple backends (e.g. kernels), it can also take advantage of TensorRT's auototuning capabilities to let it pick the most performant configuration of the plugin on the target platform. To enable autotuning, simply define a function wrapped with `tensorrt.plugin.autotune()`.

Let us consider a few examples to understand the specifics of these functions.

3.2 Example: Circular padding plugin

Circular padding is useful for ops like circular convolution. The following image denotes how the original image (red) is circular padded once (green) and twice (blue):



The plugin shall have the following characteristics:

- Input: N-dimensional input
- Attribute(s): m-dimensional parameter `pads` where m is even and $m/2 \leq N$. `pads` denotes the amount of padding to apply before and after each of the $m/2$ last dimensions of the input tensor.
- Output: Padded tensor, which has the same data type as the input. The `i`-th to-last dimension of the output, d^{out}_{N-i-1} can be expressed in terms of the corresponding input dimension, d^{in}_{N-i-1}, as:

$$d^{\text{out}}_{N-i-1} = d^{\text{in}}_{N-i-1} + \text{pads}_{2i} + \text{pads}_{2i+1}$$

Let's capture this information in the `tensorrt.plugin.register` function:

```
import tensorrt.plugin as trtp
import numpy.typing as npt

@trtp.register("example::circ_pad_plugin")
def circ_pad_plugin_desc(
    inp0: trtp.TensorDesc, pads: npt.NDArray[np.int32]
) -> trtp.TensorDesc:
    ndim = inp0.ndim
    out_desc = inp0.like()

    for i in range(np.size(pads) // 2):
        out_desc.shape_expr[ndim - i - 1] += int(
            pads[i * 2] + pads[i * 2 + 1]
        )

    return out_desc
```

The argument `example::circ_pad_plugin` defines the namespace ("example") and name ("circ_pad_plugin") of the plugin. Input arguments annotated with `tensorrt.plugin.TensorDesc` denote the input tensors; all others are interpreted as plugin attributes. Supported attribute types are:

- `int`, `float`, `str`, `bool`, `bytes`. Lists/tuples of these types are not supported.
- 1-D Numpy arrays of the following types: `int8`, `int16`, `int32`, `int64`, `float16`, `float32`, `float64`, `bool`. These must be annotated with `numpy.typing.NDArray[dtype]`, where `dtype` is the expected Numpy data type.

The output signature is a `trt.plugin.TensorDesc` describing the output. To construct the output tensor descriptor, we start with `inp0.like()`, which returns a tensor descriptor with identical shape and type characteristics to `inp0`. Since the output shape is in fact different from the input, symbolic expressions for the output shape is accessed through `tensorrt.plugin.TensorDesc.shape_expr`, to which the expected shape is written.

Now let's define the computation function, decorated with `tensorrt.plugin.impl()`. For simplicity, let's leverage PyTorch's `torch.nn.functional.pad` to do the computation.

```
import tensorrt.plugin as trtp

@trtp.impl("example::circ_pad_plugin")
def circ_pad_plugin_impl(
    inp0: trtp.Tensor,
    pads: npt.NDArray[np.int32],
    outputs: Tuple[trtp.Tensor],
    stream: int
) -> None:
    inp_t = torch.as_tensor(inp0, device="cuda")
    out_t = torch.as_tensor(outputs[0], device="cuda")

    out = torch.nn.functional.pad(inp_t, pads.tolist(), mode="circular")
    out_t.copy_(out)
```

Note that the decorated function receives `tensorrt.plugin.Tensors` for each input and output. In contrast to `TensorDescs`, a `Tensor` references an underlying data buffer, directly accessible through `tensorrt.plugin.Tensor.data_ptr()`. When working with Torch and OpenAI Triton kernels, it is easier to use `torch.as_tensor()` to zero-copy construct a `torch.Tensor` corresponding to the `tensorrt.plugin.Tensor`.

3.2.1 Providing an Ahead-of-Time (AOT) implementation

In contrast to a JIT implementation provided by `tensorrt.plugin.impl()`, an AOT implementation provides the following benefits:

1. Building a TRT engine with a JIT computation for a plugin requires both its `tensorrt.plugin.register()` and `tensorrt.plugin.impl()` Python definitions to be present at runtime. With an AOT implementation, the plugin is fully embedded in the engine such that no user-provided definition of the plugin is required at runtime.
2. Python-independence of the plugin layer at runtime. This means that if the TRT engine only consists of AOT plugins, it can be executed on the standard TRT runtime as you would an engine with compiled C++ plugins, for example, through `trtexec`.

`tensorrt.plugin.aot_impl()` can be used to define the AOT implementation of a plugin. For the above circular padding plugin, let us define an AOT implementation using an OpenAI Triton kernel:

```
import triton
```

(continues on next page)

(continued from previous page)

```
import triton.language as tl

@triton.jit
def circ_pad_kernel(
    # input tensor
    X,
    # extra scalar args in between input and output tensors
    all_pads_0,
    all_pads_2,
    all_pads_4,
    all_pads_6,
    orig_dims_0,
    orig_dims_1,
    orig_dims_2,
    orig_dims_3,
    Y_shape_1,
    Y_shape_2,
    Y_shape_3,
    X_len,
    Y_len,
    # output tensor
    Y,
    BLOCK_SIZE: tl.constexpr,
):
    pid = tl.program_id(0)
    i = pid * BLOCK_SIZE + tl.arange(0, BLOCK_SIZE)

    mask_y = i < Y_len

    i3 = i % Y_shape_3
    i2 = (i // Y_shape_3) % Y_shape_2
    i1 = (i // Y_shape_3 // Y_shape_2) % Y_shape_1
    i0 = i // Y_shape_3 // Y_shape_2 // Y_shape_1

    j0 = (i0 - all_pads_0 + orig_dims_0) % orig_dims_0
    j1 = (i1 - all_pads_2 + orig_dims_1) % orig_dims_1
    j2 = (i2 - all_pads_4 + orig_dims_2) % orig_dims_2
    j3 = (i3 - all_pads_6 + orig_dims_3) % orig_dims_3

    load_idx = (
        orig_dims_3 * orig_dims_2 * orig_dims_1 * j0
        + orig_dims_3 * orig_dims_2 * j1
        + orig_dims_3 * j2
        + j3
    )
    mask_x = load_idx < X_len

    x = tl.load(X + load_idx, mask=mask_x)

    tl.store(Y + i, x, mask=mask_y)

@trtp.aot_impl("example::circ_pad_plugin")
```

(continues on next page)

(continued from previous page)

```
def circ_pad_plugin_aot_impl(
    inp0: trtp.TensorDesc, pads: npt.NDArray[np.int32], outputs: Tuple[trtp.TensorDesc],␣
↪tactic: int
) -> Tuple[Union[str, bytes], Union[str, bytes], trtp.KernelLaunchParams, trtp.SymExprs]:

    assert tactic == 0
    block_size = 256

    type_str = "fp32" if inp0.dtype == trt.float32 else "fp16"

    src = triton.compiler.ASTSource(
        fn=circ_pad_kernel,
        signature=f"*{type_str},{','.join(['i32']*13)},*{type_str}",
        constants={
            "BLOCK_SIZE": block_size,
        },
    )

    compiled_kernel = triton.compile(src)
    launch_params = trtp.KernelLaunchParams()

    N = inp0.ndim
    all_pads = np.zeros((N * 2,), dtype=np.int32)
    inp_dims = inp0.shape_expr
    out_dims = outputs[0].shape_expr

    for i in range(np.size(pads) // 2):
        all_pads[N * 2 - 2 * i - 2] = pads[i * 2]
        all_pads[N * 2 - 2 * i - 1] = pads[i * 2 + 1]

    # grid dims
    launch_params.grid_x = trtp.cdiv(out_dims.numel(), block_size)
    # block dims
    launch_params.block_x = compiled_kernel.metadata.num_warps * 32
    # shared memory
    launch_params.shared_mem = compiled_kernel.metadata.shared

    all_pads = all_pads.tolist()

    # Representing all int32 scalar inputs as symbolic expressions.
    # These inputs are either constants or derivatives of input/output shapes.
    # The symbolic expressions are resolved after the full shape context becomes␣
↪available at runtime.
    # For the `circ_pad_kernel`, there are 13 such scalar extra-arguments, corresponding␣
↪to the 13 arguments
    # between the mandatory input and output tensors in the triton kernel's function␣
↪signature.
    extra_args = trtp.SymIntExprs.from_tuple(
        [
            trtp.SymInt32(e)
            for e in [
                all_pads[0],
```

(continues on next page)

(continued from previous page)

```
                all_pads[2],
                all_pads[4],
                all_pads[6],
                inp_dims[0],
                inp_dims[1],
                inp_dims[2],
                inp_dims[3],
                out_dims[1],
                out_dims[2],
                out_dims[3],
                inp_dims.numel(),
                out_dims.numel(),
            ]
        ]
    )

    return compiled_kernel.metadata.name, compiled_kernel.asm["ptx"], launch_params,␣
↪extra_args
```

The function decorated with `tensorrt.plugin.aot_impl()` will be invoked for each I/O type/format combination and for each custom tactic advertised with `tensorrt.plugin.autotune()`. For each such combination, it should return the name of the kernel, the compiled form of the kernel, the kernel launch parameters, and any extra scalar arguments to pass to the kernel. Note that

- When dealing with dynamic shapes, concrete I/O dimensions may not be known when `tensorrt.plugin.aot_impl()` is invoked. Therefore, the kernel launch parameters and extra kernel arguments are specified symbolically.
- The kernel should always be written such that the input tensors come first, any scalar args (specified as above) come next, with the output tensors at the end.

3.2.2 Picking the most performant plugin configuration: Autotuning

Let's assume the plugin is able to support both FP32 and FP16 I/O, in linear tensor layouts (tensor formats). If performance is key, and it is uncertain whether FP32 or FP16 would perform better on the target platform, we can define a function decorated with `tensorrt.plugin.autotune()`.

```
@trtp.autotune("example::circ_pad_plugin")
def circ_pad_plugin_autotune(
    inp0: trtp.TensorDesc,
    pads: npt.NDArray[np.int32],
    outputs: Tuple[trtp.TensorDesc],
) -> List[trtp.AutoTuneCombination]:
    return [trtp.AutoTuneCombination("FP32|FP16, FP32|FP16", "LINEAR")]
```

The decorated function must return a list of `tensorrt.plugin.AutoTuneCombination`s. In this case, we define a single combination `AutoTuneCombination("FP32|FP16, FP32|FP16", "LINEAR")`; this indicates that the input and output must be either both FP32 or both FP16, and each have linear formats. When an autotune function is defined, during the engine build, TRT may execute the plugin (invoke its `impl` function) with random inputs, for each of the type/format combinations.

3.2.3 Adding the plugin to a TensorRT network

Now that we have defined the plugin, it can be added to a TensorRT network. Depending on your workflow, you may need to add the plugin directly to a TRT `tensorrt.INetworkDefinition` through the TRT Python API, or you may want the plugin to be added by the TRT ONNX parser when loading an ONNX graph containing the custom operator.

3.2.3.1 Adding the plugin using TRT Python APIs

The `tensorrt.INetworkDefinition.add_plugin()` API can be used to add a plugin to a network definition (an instance of `tensorrt.INetworkDefinition`):

```
input_tensor = network.add_input(name="x", dtype=trt.DataType.FLOAT, shape=x.shape)
plugin_layer = network.add_plugin(trt.plugin.op.example.circ_pad_plugin(input_tensor,␣
↪pads = pads), aot = False)
```

Note that

1. The registered plugin is findable in `tensorrt.plugin.op` under its registered namespace and name.
2. `tensorrt.INetworkDefinition.add_plugin()` takes a boolean argument `aot`, which can be used to choose either the AOT or JIT implementation of a plugin that has both definitions. If a plugin has only one implementation (either AOT or JIT), then omitting the `aot` argument will result in that implementation being picked by TensorRT.

When a plugin has both AOT and JIT implementations defined, it is also possible to apply a global setting to control which implementation is chosen. `tensorrt.NetworkDefinitionCreationFlag.PREFER_AOT_PYTHON_PLUGINS` and `tensorrt.NetworkDefinitionCreationFlag.PREFER_JIT_PYTHON_PLUGINS` are mutually exclusive flags that can be applied to the `tensorrt.INetworkDefinition` as follows:

```
builder = trt.Builder(trt.Logger(trt.Logger.INFO))
# Always choose AOT implementations wherever possible
network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.PREFER_AOT_
↪PYTHON_PLUGINS))
...
# Now the `aot` argument can be omitted (and TRT will choose the AOT implementation)
plugin_layer = network.add_plugin(trt.plugin.op.example.circ_pad_plugin(input_tensor,␣
↪pads = pads))
```

3.2.3.2 Loading an ONNX model with the custom operator

It is possible to load an ONNX model with a custom op node which you need to run through the TRT plugin. To allow the TRT ONNX parser to correctly recognize your plugin as being mapped to the ONNX node(s) of interest, ensure that:

- The `op` property of the ONNX node is exactly the same as your plugin name.
- The node contains a string attribute called `plugin_namespace` with the namespace of your plugin.
- For a plugin with both AOT and JIT implementations, either
  - The node should contain a boolean attribute called `aot` to communicate which implementation should be chosen (`true` for AOT, `false` for JIT), or
  - `NetworkDefinitionCreationFlag.PREFER_AOT_PYTHON_PLUGINS()` or `NetworkDefinitionCreationFlag.PREFER_JIT_PYTHON_PLUGINS()` should be specified (see above for an example)

For example, if using ONNX Graphsurgeon, the custom op node can be constructed as follows:

```
import onnx_graphsurgeon as gs

var_x = gs.Variable(name="x", shape=inp_shape, dtype=np.float32)
var_y = gs.Variable(name="y", dtype=np.float32)

circ_pad_node = gs.Node(
    name="circ_pad_plugin",
    op="circ_pad_plugin",
    inputs=[var_x],
    outputs=[var_y],
    attrs={"pads": pads, "plugin_namespace": "example"},
)
```

3.3 Advanced usage

3.3.1 Example: Operators with data-dependent output shapes - Non-zero

Non-zero is an operation where the indices of the non-zero elements of the input tensor is found. Therefore, it has data-dependent output shapes (DDS), for which typical shape calculations cannot be done with input shapes.

To handle DDS, the extent of each data-dependent output dimension must be expressed in terms of a size tensor, which is a scalar that communicates to TRT an upper-bound and an autotune value for that dimension, in terms of the input shapes. The TRT engine build may be optimized for the autotune value, but the extent of that dimension may stretch up to the upper-bound at runtime.

In this example, we consider a 2D input tensor `inp0`; the output will be an $N \times 2$ tensor (a set of N 2D indices), where N is the number of non-zero indices. At maximum, all elements could be non-zero, and so the upper-bound could be expressed as `upper_bound = inp0.shape_expr[0] * inp0.shape_expr[1]`.

On average, we can expect half of the input to be filled with zero, so a size tensor can be constructed with that as the autotune value:

```
st = trt.plugin.size_tensor(opt = upper_bound // 2, upper_bound = upper_bound)
```

Now we're ready to construct the output shape. `st.expr()` returns a shape expression for the size tensor, so a tensor descriptor for the output shape can be constructed as `trt.plugin.from_shape_expr((st.expr(), 2), dtype=trt.int32)`. TRT requires that any size tensors also be made outputs of the plugin. Putting things together, we arrive at the following:

```
import tensorrt.plugin as trtp

@trtp.register("example::non_zero_plugin")
def non_zero_plugin_desc(
    inp0: trtp.TensorDesc,
) -> Tuple[trtp.TensorDesc, trtp.TensorDesc]:
    upper_bound = inp0.shape_expr[0] * inp0.shape_expr[1]
    st = trtp.size_tensor(upper_bound // 2, upper_bound)
    return trtp.from_shape_expr((st.expr(), 2), dtype=trt.int32), st
```

3.3.2 Example: Implementing in-place custom ops with I/O aliasing

In-place computations can be accomplished with TRT plugins via aliased I/O. i.e. An input that needs to be modified in-place can be represented by an input-output pair, where the output is aliased to the input. For example, consider a simple elementwise addition plugin. If an in-place addition is needed, that can be achieved as below:

```
import tensorrt.plugin as trtp

@trtp.register("sample::elemwise_add_plugin_")
def add_plugin_desc_(inp0: trtp.TensorDesc) -> trtp.TensorDesc:
    return inp0.aliased()
```

Note that `inp0.aliased()` produces an output `TensorDesc` that is aliased to `inp0`.

3.3.3 Example: Plugins with multiple backends: Using custom tactics

It may be the case that there are multiple kernels or libraries available to perform the plugin computation (for the same IO data type/tensor layout combination) but it is not predeterminable which backend would be the fastest on the target platform. Such alternate backends are termed tactics of the plugin. To let TRT figure out the fastest tactic, it is possible to leverage the autotune function.

Let's assume that we have an OpenAI Triton kernel `circ_pad_kernel` which can perform the circular padding operation in the above example. We can ask TRT to pick the fastest between the OpenAI Triton kernel and `torch.nn.functional.pad` as below:

```
import tensorrt.plugin as trtp
from enum import IntEnum

class Tactic(IntEnum):
    TORCH = 1
    TRITON = 2

@trt.plugin.autotune("sample::circ_pad_plugin")
def circ_pad_plugin_autotune(inp0: trtp.TensorDesc, pads: npt.NDArray[np.int32],␣
↪outputs: Tuple[trtp.TensorDesc]) -> List[trtp.AutoTuneCombination]:
    c = trtp.AutoTuneCombination()
    c.pos([0, 1], "FP32|FP16")
    c.tactics([int(Tactic.TORCH), int(Tactic.TRITON)])
    return [c]
```

Note that we're using another way of constructing a `trt.plugin.AutoTuneCombination` here – namely, through `pos(...)` to populate the type/format information and `tactics(...)` to specify the tactics.

Now, the `impl` function can take an additional `int` argument called `tactic` to use the appropriate backend:

```
@trtp.impl("example::circ_pad_plugin")
def circ_pad_plugin_impl(
    inp0: trtp.Tensor,
    pads: npt.NDArray[np.int32],
    outputs: Tuple[trtp.Tensor],
    stream: int,
    tactic: int
) -> None:
    inp_t = torch.as_tensor(inp0, device="cuda")
```

(continues on next page)

(continued from previous page)

```
    out_t = torch.as_tensor(outputs[0], device="cuda")

    if tactic == Tactic.TORCH:
        out = torch.nn.functional.pad(inp_t, pads.tolist(), mode="circular")
        out_t.copy_(out)
    elif tactic == Tactic.TRITON:
        N = inp0.ndim
        all_pads = np.zeros((N * 2,), dtype=np.int32)
        out_dims = trtp.Shape(tuple(inp0.shape))

        block_size = 256
        num_blocks = tuple(
            [int((np.prod(out_dims) + block_size - 1) // block_size)]
        )

        circ_pad_kernel[num_blocks](inp_t, ..., BLOCK_SIZE=block_size)
```

3.4 FAQ

1. What happened to plugin creators, plugin registration and plugin libraries?

You may have this question if you have previously dealt with class-based TRT plugins. For decorator based plugins, registration is automatically handled by the `tensorrt.plugin` module as long as the `tensorrt.plugin.register` function is defined.

To simulate the experience of plugin libraries, you may define plugins under a common namespace on a separate Python module, and then load the module when that plugin "library" needs to be loaded.

2. How is the serialization/deserialization of plugin attributes handled?

The `tensorrt.plugin` module automatically serializes any plugin attributes included in the `tensorrt.plugin.impl` function signature into the TRT engine. When that engine is loaded, those attributes are deserialized and passed back to the `tensorrt.plugin.impl` function.

Note: If only a subset of the plugin attributes included in `tensorrt.plugin.register` is required for the plugin computation, only include those in `tensorrt.plugin.impl` function signature. This should avoid the unnecessary serialization of data, and therefore produce leaner TRT engines.

3.5 Limitations

- TRT plugins defined in Python result in TRT engines that, when executed, rely on the availability of a Python environment with those plugin definitions. To build Python-independent TRT engines, it is recommended to use the TRT C++ plugin interfaces.
- This guide describes plugin implementations through the `tensorrt.plugin` module, which supports some of the most common use cases for plugin usage. For more advanced use cases, such as passing shape inputs, it is recommended to define plugins as direct implementations of the `tensorrt.IPluginV3` interface.

3.6 Additional resources

- Extending TensorRT with Custom Layers

CHAPTER

FOUR

FOUNDATIONAL TYPES

4.1 DataType

TensorRT also exposes some short-hand, NumPy-style `DataType` aliases that can be used across the library:

Type	Alias
`tensorrt.DataType.FLOAT`	`tensorrt.float32`
`tensorrt.DataType.HALF`	`tensorrt.float16`
`tensorrt.DataType.INT32`	`tensorrt.int32`
`tensorrt.DataType.INT8`	`tensorrt.int8`
`tensorrt.DataType.BOOL`	`tensorrt.bool`
`tensorrt.DataType.UINT8`	`tensorrt.uint8`
`tensorrt.DataType.FP8`	`tensorrt.fp8`
`tensorrt.DataType.BF16`	`tensorrt.bfloat16`
`tensorrt.DataType.INT64`	`tensorrt.int64`

4.2 Weights

4.3 Dims

4.3.1 Volume

4.3.2 Dims

4.3.3 Dims2

4.3.4 DimsHW

4.3.5 Dims3

4.3.6 Dims4

4.4 IHostMemory

CHAPTER

FIVE

CORE

5.1 Logger

5.2 Profiler

5.3 IOptimizationProfile

5.4 IBuilderConfig

5.5 Builder

5.5.1 NetworkDefinitionCreationFlag

5.5.2 Builder

5.6 Debug Listener

5.6.1 IDebugListener

5.7 ICudaEngine

5.8 IExecutionContext

5.9 Runtime

5.10 Refitter

5.11 IErrorRecorder

5.12 ITimingCache

5.13 GPU Allocator

5.13.1 AllocatorFlag

5.13.2 IGpuAllocator

CHAPTER

SIX

NETWORK

6.1 INetworkDefinition

6.2 Layer Base Classes

6.2.1 ITensor

6.2.2 ILayer

6.3 Layers

For more information, including examples, refer to the TensorRT Operator's Reference documentation.

Note that layer weight properties may be represented as NumPy arrays or Weights objects depending on whether the underlying datatype is supported by NumPy. Explicitly construct a Weights object from the property if you want a consistent type:

```
conv_layer = network.add_convolution_nd(...)
bias_weights = trt.Weights(conv_layer.bias)
```

6.3.1 PaddingMode

6.3.2 ICastLayer

6.3.3 IConvolutionLayer

6.3.4 IGridSampleLayer

6.3.5 IActivationLayer

6.3.6 IPoolingLayer

6.3.7 ILRNLayer

6.3.8 IScaleLayer

6.3.9 ISoftMaxLayer

6.3.10 IConcatenationLayer

6.3.11 IDeconvolutionLayer

6.3.12 IElementWiseLayer

6.3.13 IGatherLayer

6.3.14 IPluginV2Layer

6.3.15 IPluginV3Layer

6.3.16 IUnaryLayer

6.3.17 IReduceLayer

6.3.18 IPaddingLayer

6.3.19 IParametricReLULayer

6.3.20 ISelectLayer

6.3.21 IShuffleLayer

6.3.22 ISliceLayer

6.3.23 IShapeLayer

6.3.24 ITopKLayer

6.3.25 IMatrixMultiplyLayer

6.3.26 IRaggedSoftMaxLayer

6.3.27 IIdentityLayer

6.3.28 IConstantLayer

CHAPTER

SEVEN

PLUGIN

7.1 IPluginCreator

7.2 IPluginRegistry

7.3 IPluginV2DynamicExt

7.4 IPluginV3

CHAPTER

EIGHT

TENSORRT.PLUGIN

8.1 tensorrt.plugin.register

8.2 tensorrt.plugin.impl

8.3 tensorrt.plugin.aot_impl

See also:

8.3.1 KernelLaunchParams

8.4 tensorrt.plugin.autotune

See also:

8.4.1 AutoTuneCombination

8.5 Shape

8.5.1 Shape

8.5.2 ShapeExpr

8.6 Tensor

8.6.1 TensorDesc

8.6.2 Tensor

8.7 Functions

CHAPTER

NINE

INT8

9.1 IInt8Calibrator

9.2 IInt8LegacyCalibrator

9.3 IInt8EntropyCalibrator

9.4 IInt8EntropyCalibrator2

9.5 IInt8MinMaxCalibrator

CHAPTER

TEN

ALGORITHM SELECTOR

Warning: Algorithm Selector is deprecated in TensorRT 10.8. Please use editable mode in ITimingCache instead.

CHAPTER

ELEVEN

ONNX PARSER

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

ARM

ARM, AMBA and ARM Powered are registered trademarks of ARM Limited. Cortex, MPCore and Mali are trademarks of ARM Limited. "ARM" is used to represent ARM Holdings plc; its operating company ARM Limited; and the regional subsidiaries ARM Inc.; ARM KK; ARM Korea Limited.; ARM Taiwan Limited; ARM France SAS; ARM Consulting (Shanghai) Co. Ltd.; ARM Germany GmbH; ARM Embedded Technologies Pvt. Ltd.; ARM Norway, AS and ARM Sweden AB.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

Blackberry/QNX

Copyright © 2020 BlackBerry Limited. All rights reserved.

Trademarks, including but not limited to BLACKBERRY, EMBLEM Design, QNX, AVIAGE, MOMENTICS, NEUTRINO and QNX CAR are the trademarks or registered trademarks of BlackBerry Limited, used under license, and the exclusive rights to such trademarks are expressly reserved.

Google

Android, Android TV, Google Play and the Google Play logo are trademarks of Google, Inc.

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com

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

© 2017-2025 NVIDIA Corporation & affiliates. All rights reserved.

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com

