



**HiAI DDK V320**

# **IR Model Building Instructions**

**Issue**      **02**  
**Date**      **2019-12-31**

**Copyright © Huawei Technologies Co., Ltd. 2019. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

## **Trademarks and Permissions**



HUAWEI and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

## **Notice**

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

## **Huawei HiAI Application**

Send an application email to [developer@huawei.com](mailto:developer@huawei.com).

Email subject: HUAWEI HiAI + Company name + Product name

Email body: Cooperation company + Contact person + Phone number + Email address

We will reply to you within 5 working days.

Official website: <https://developer.huawei.com/consumer/en/>

---

# About This Document

---

## Purpose

This document describes how to build IR models, and introduces the operator APIs and offline and online building modes.

## Change History

Changes between document issues are cumulative. The latest document issue contains all the changes made in earlier issues.

Data	Version	Change Description
2019-12-31	02	Added the description of HiAI DDK V320.
2019-09-04	01	Added the description of HiAI DDK V310.

# Contents

<b>About This Document .....</b>	<b>i</b>
<b>1 Model Building .....</b>	<b>1</b>
1.1 Overview .....	1
1.2 Constraints .....	1
1.3 Supported Operators .....	1
<b>2 Operator APIs .....</b>	<b>7</b>
2.1 Input .....	7
2.2 Output .....	8
2.3 Attribute .....	9
2.3.1 Quantization .....	9
2.3.2 AIPP Attribute .....	10
<b>3 Online Model Building .....</b>	<b>12</b>
3.1 Model Building Demo .....	12
3.1.1 Dependencies .....	12
3.1.2 Procedure for Online Model Building .....	12
3.2 Building an IR Model .....	13
3.2.1 Native Model .....	13
3.2.2 Building Single-Operators .....	13
3.2.2.1 Data & Const .....	13
3.2.2.2 AIPP .....	14
3.2.2.3 Convolution .....	16
3.2.2.4 Activation .....	17
3.2.2.5 Concat .....	17
3.2.2.6 Pooling .....	18
3.2.2.7 Softmax .....	19
3.2.3 Building the Network .....	19
<b>4 Offline Model Building .....</b>	<b>23</b>
4.1 Model Building Demo .....	23

4.1.1 Dependencies.....	23
4.1.2 Procedure for Offline Model Building.....	24
4.2 Building an IR Model.....	24
4.2.1 Native Model .....	24
4.2.2 Building Single-Operators.....	25
4.2.3 Building the Model .....	25
4.2.4 Building the Network.....	25
4.2.5 Generating a Model .....	25
<b>5 API Reference .....</b>	<b>26</b>
5.1 Attribute APIs.....	27
5.1.1 SetAttr.....	27
5.1.2 GetAttr.....	28
5.1.3 HasAttr .....	29
5.1.4 DelAttr.....	29
5.1.5 SetName .....	30
5.1.6 GetName .....	31
5.1.7 GetItem .....	31
5.1.8 SetValue.....	32
5.1.9 GetValue.....	33
5.1.10 CreateFrom .....	34
5.1.11 GetValueType .....	35
5.1.12 IsEmpty .....	36
5.1.13 Copy .....	36
5.1.14 operator==.....	37
5.1.15 MutableTensor.....	38
5.1.16 MutableListTensor .....	38
5.1.17 InitDefault .....	39
5.1.18 GetProtoOwner.....	40
5.1.19 GetProtoMsg.....	40
5.1.20 CopyValueFrom .....	41
5.1.21 MoveValueFrom .....	41
5.2 Tensor APIs.....	42
5.2.1 GetTensorDesc.....	42
5.2.2 MutableTensorDesc.....	43
5.2.3 SetTensorDesc .....	43
5.2.4 GetData.....	44
5.2.5 MutableData .....	45

5.2.6 SetData .....	45
5.2.7 Clone.....	46
5.2.8 operator=.....	47
5.3 TensorDesc APIs .....	47
5.3.1 Update.....	48
5.3.2 GetShape .....	49
5.3.3 MutableShape .....	49
5.3.4 SetShape.....	50
5.3.5 GetFormat.....	51
5.3.6 SetFormat.....	51
5.3.7 GetDataType .....	52
5.3.8 SetDataType.....	52
5.3.9 Clone.....	53
5.3.10 IsValid.....	54
5.3.11 operator= .....	54
5.4 Shape APIs.....	55
5.4.1 GetDimNum .....	55
5.4.2 GetDim .....	56
5.4.3 GetDims .....	57
5.4.4 SetDim.....	57
5.4.5 GetShapeSize .....	58
5.4.6 operator=.....	59
5.5 Buffer APIs.....	59
5.5.1 ClearBuffer.....	60
5.5.2 GetData.....	60
5.5.3 GetSize .....	61
5.5.4 CopyFrom.....	62
5.5.5 data .....	62
5.5.6 size.....	63
5.5.7 clear.....	64
5.5.8 operator=.....	65
5.5.9 operator[].....	65
5.6 Operator APIs.....	66
5.6.1 GetName .....	66
5.6.2 SetInput.....	67
5.6.3 GetInputDesc .....	68
5.6.4 TryGetInputDesc .....	69

5.6.5 UpdateInputDesc.....	70
5.6.6 GetOutputDesc.....	70
5.6.7 UpdateOutputDesc.....	71
5.6.8 GetDynamicInputDesc.....	72
5.6.9 UpdateDynamicInputDesc .....	73
5.6.10 GetDynamicOutputDesc.....	74
5.6.11 UpdateDynamicOutputDesc .....	75
5.6.12 SetAttr.....	76
5.6.13 GetAttr .....	76
5.7 Operator Registration APIs.....	77
5.7.1 REG_OP .....	78
5.7.2 ATTR.....	79
5.7.3 REQUIRED_ATTR.....	80
5.7.4 INPUT .....	82
5.7.5 OPTIONAL_INPUT .....	83
5.7.6 DYNAMIC_INPUT.....	85
5.7.7 OUTPUT .....	86
5.7.8 DYNAMIC_OUTPUT .....	87
5.7.9 OP_END .....	88
5.7.10 GET_INPUT_SHAPE.....	89
5.7.11 GET_DYNAMIC_INPUT_SHAPE.....	89
5.7.12 SET_OUTPUT_SHAPE.....	90
5.7.13 SET_DYNAMIC_OUTPUT_SHAPE .....	91
5.7.14 GET_ATTR .....	91
5.8 Graph APIs.....	92
5.8.1 SetInputs.....	92
5.8.2 SetOutputs.....	93
5.8.3 IsValid .....	94
5.8.4 AddOp .....	94
5.8.5 FindOpByName.....	95
5.8.6 CheckOpByName.....	96
5.8.7 GetAllOpName.....	96
5.9 Model APIs .....	97
5.9.1 SetName .....	97
5.9.2 GetName .....	98
5.9.3 SetVersion .....	98
5.9.4 GetVersion .....	99

5.9.5 SetPlatformVersion .....	100
5.9.6 GetPlatformVersion .....	100
5.9.7 GetGraph .....	101
5.9.8 SetGraph .....	102
5.9.9 Save .....	102
5.9.10 Load .....	103
5.9.11 IsValid .....	104
5.10 Model Building APIs .....	104
5.10.1 CreateModelBuff .....	105
5.10.2 CreateModelBuff .....	106
5.10.3 BuildIRModel .....	107
5.10.4 ReleaseModelBuff .....	108



# 1 Model Building

## 1.1 Overview

The Intermediate Representation (IR) model building allows users to independently build network models using the operator IR developed with HiAI.

Currently, a model can be built in either online or offline mode.

Online model building applies to third-party frameworks. Users can map third-party framework code to IR APIs by referring to the definitions of HiAI IR APIs, build models during application running, and implement inference. Users can also serialize the built model for app execution.

Offline model building is designed for advanced users with special requirements. Users can directly use IR APIs to program and build inference models.

### NOTE

In the scenario when the HiAI DDK 100.320.020.010 original model has a weight of type float32, the weight data type remains unchanged, and the IR model size is expected to be larger compared with those generated with earlier versions.

## 1.2 Constraints

The ROM and RAM capacities of a mobile chip are limited. Therefore, the model size and memory usage must be limited.

- It is recommended that the model size be less than or equal to 100 MB.
- It is recommended that the peak usage of the running memory be less than or equal to 200 MB.

## 1.3 Supported Operators

The operators are defined in the directory: `/ddk/ai_ddk_lib/include/graph/op/`

Operator Name	Category	File
Add	math_defs	math_defs.h
Mul	math_defs	math_defs.h
Expm1	math_defs	math_defs.h
Ceil	math_defs	math_defs.h
Sin	math_defs	math_defs.h
Cos	math_defs	math_defs.h
Floor	math_defs	math_defs.h
Log1p	math_defs	math_defs.h
LogicalAnd	math_defs	math_defs.h
LogicalNot	math_defs	math_defs.h
Maximum	math_defs	math_defs.h
Minimum	math_defs	math_defs.h
Acosh	math_defs	math_defs.h
Asinh	math_defs	math_defs.h
Equal	math_defs	math_defs.h
Reciprocal	math_defs	math_defs.h
Sqrt	math_defs	math_defs.h
Square	math_defs	math_defs.h
ReduceAll	math_defs	math_defs.h
Cast	math_defs	math_defs.h
Sign	math_defs	math_defs.h

Operator Name	Category	File
Cosh	math_defs	math_defs.h
Exp	math_defs	math_defs.h
FloorMod	math_defs	math_defs.h
GreaterEqual	math_defs	math_defs.h
Less	math_defs	math_defs.h
MatMul	math_defs	math_defs.h
RealDiv	math_defs	math_defs.h
Rint	math_defs	math_defs.h
Round	math_defs	math_defs.h
Rsqrt	math_defs	math_defs.h
Sinh	math_defs	math_defs.h
Sub	math_defs	math_defs.h
Range	math_defs	math_defs.h
Acos	math_defs	math_defs.h
Asin	math_defs	math_defs.h
Atanh	math_defs	math_defs.h
Log	math_defs	math_defs.h
LogicalOr	math_defs	math_defs.h
Neg	math_defs	math_defs.h
ReduceProd	math_defs	math_defs.h
ReduceSum	math_defs	math_defs.h

Operator Name	Category	File
Tan	math_defs	math_defs.h
ArgMax	math_defs	math_defs.h
FloorDiv	math_defs	math_defs.h
Const	const_defs	const_defs.h
RandomUniform	random_defs	random_defs.h
Multinomial	random_defs	random_defs.h
Permute	detection_defs	detection_defs.h
Activation	nn_defs	nn_defs.h
BatchNorm	nn_defs	nn_defs.h
Convolution	nn_defs	nn_defs.h
Deconvolution	nn_defs	nn_defs.h
BiasAdd	nn_defs	nn_defs.h
Eltwise	nn_defs	nn_defs.h
LRN	nn_defs	nn_defs.h
ConvolutionDepthwise	nn_defs	nn_defs.h
FullConnection	nn_defs	nn_defs.h
Pooling	nn_defs	nn_defs.h
Scale	nn_defs	nn_defs.h
ShuffleChannel	nn_defs	nn_defs.h
Softmax	nn_defs	nn_defs.h
TopK	nn_defs	nn_defs.h
LogSoftmax	nn_defs	nn_defs.h

Operator Name	Category	File
QuantizedConvolution	nn_defs	nn_defs.h
QuantizedFullConnection	nn_defs	nn_defs.h
QuantizedConvolutionDepthwise	nn_defs	nn_defs.h
BatchNormExt2	nn_defs	nn_defs.h
CropAndResize	image_defs	image_defs.h
ResizeBilinear	image_defs	image_defs.h
ResizeNearestNeighbor	image_defs	image_defs.h
Data	array_defs	array_defs.h
Concat	array_defs	array_defs.h
Reshape	array_defs	array_defs.h
Split	array_defs	array_defs.h
Unpack	array_defs	array_defs.h
Flatten	array_defs	array_defs.h
Slice	array_defs	array_defs.h
ExpandDims	array_defs	array_defs.h
Gather	array_defs	array_defs.h
GatherNd	array_defs	array_defs.h
Pack	array_defs	array_defs.h
SpaceToDepth	array_defs	array_defs.h
StridedSlice	array_defs	array_defs.h
SpaceToBatchND	array_defs	array_defs.h
BatchToSpaceND	array_defs	array_defs.h

Operator Name	Category	File
Tile	array_defs	array_defs.h
Size	array_defs	array_defs.h
Fill	array_defs	array_defs.h
InvertPermutation	array_defs	array_defs.h
ReverseSequence	array_defs	array_defs.h

# 2 Operator APIs

Operator APIs are used to define the input, output, and attributes of operators. For details, see [5 API Reference](#).

The following describes how to use IR APIs based on the operator definitions.

## 2.1 Input

The single-operator APIs define the operator input description names, input attributes, and supported data types.

An input can be marked as optional (**OPTIONAL\_INPUT**) or dynamic (**DYNAMIC\_INPUT**), according to its nature. **OPTIONAL\_INPUT** is generally used for weight configuration. If the number of inputs is not fixed, use the **DYNAMIC\_INPUT** flag. Several examples are provided as follows:

Example 1: Convolution operator

```
REG_OP(Convolution)
.INPUT(x, TensorType ({ DT_FLOAT } ))
.INPUT(w, TensorType ({ DT_FLOAT, DT_INT8 } ))
.OPTIONAL_INPUT(b, TensorType ({ DT_FLOAT } ))
```

Weight **b** of the operator is optional, that is, marked as **OPTIONAL\_INPUT**. An input, optional or not, is configured through the `set_input_` API.

```
auto conv_op= op::Convolution("convolution")
.set_input_x(data)
.set_input_w(conv1_const_0)
.set_input_b(conv1_const_1).
```

Example 2: Concat operator

```
REG_OP(Concat)
.DYNAMIC_INPUT(x, TensorType({ DT_FLOAT, DT_INT32 }))
.OUTPUT(y, TensorType({ DT_FLOAT, DT_INT32 })))
```

Input **x** of the operator is dynamic, that is, marked as **DYNAMIC\_INPUT**. It is configured using the `create_dynamic_input_` and `set_dynamic_input_` APIs.

```
auto fire2_concat = op::Concat("fire2/concat")
.create_dynamic_input_x(2)
.set_dynamic_input_x(1, fire2_relu_expand1x1)
.set_dynamic_input_x(2, fire2_relu_expand3x3)
```

- In `create_dynamic_input_x(2)`, **x** indicates the operator input name, and argument **2** indicates the number of dynamic inputs.
- In `set_dynamic_input_x(1, fire2_relu_expand1x1)`, **x** indicates the operator input name, **1** indicates the input index (starting from 1 by default), and **fire2\_relu\_expand1x1** indicates the input value.

## 2.2 Output

The single-operator APIs define the operator output description names, output attributes, and supported data types.

If the number of outputs is not fixed, the **DYNAMIC\_OUTPUT** output flag can be used.

Example: Split operator

```
REG_OP(Split)
.INPUT(x, TensorType({ DT_FLOAT, DT_INT8,
.DYNAMIC_OUTPUT(y, TensorType({ DT_FLOAT,
```

The output **y** of the operator is dynamic, that is, marked as **DYNAMIC\_OUTPUT**. It is set by using the `create_dynamic_output_` API.

Create a dynamic output.

```
auto split_op = op::Split("split")
.set_input_x(data)
.create_dynamic_output_y(2)
.set_attr_axis(0)
```



For a dynamic output:

```
auto acosh_op= op::Acosh("acosh")
.set_input_x(split_op.get_output(0))
```

- In `create_dynamic_output_y(2)`, **y** indicates the operator output name, and argument **2** indicates the number of dynamic outputs.
- In `set_input_x(split_op.get_output(0))`, **split\_op** indicates the input value, and **get\_output(0)** indicates the first dynamic output. The **get\_output(0)** name is the combination of **y** in `create_dynamic_output_y` and the output index (starting from 1 by default).

## 2.3 Attribute

The single-operator APIs define the operator attribute names, attribute types, and default values and value ranges. A common attribute is marked as **ATTR**. A mandatory attribute is marked as **REQUIRED\_ATTR**.

Example: Reshape operator

```
REG_OP(Reshape)
.INPUT(tensor, TensorType({ DT_FLOAT, DT_INT32, DT_INT64, DT_BOOL }))
.INPUT(w, TensorType({ DT_FLOAT, DT_INT32, DT_INT64, DT_BOOL }))
.OUTPUT(output, TensorType({ DT_FLOAT, DT_INT32, DT_INT64, DT_BOOL }))
.REQUIRED_ATTR(shape, AttrValue::LIST_INT)
.ATTR(axis, AttrValue::INT { 0 })
.ATTR(num_axes, AttrValue::INT { -1 })
.OP_END()
```

This operator has a mandatory attribute—**shape**. An attribute, mandatory or not, is configured through the `set_attr_` API.

```
.set_attr_shape(1)
.set_attr_axis(0)
.set_attr_num_axes(0)
```

### 2.3.1 Quantization

IR APIs provide quantization factors for users to train. Take QuantizedConvolution as an example.

```
REG_OP(QuantizedConvolution)
.INPUT(x, TensorType ({ DT_FLOAT }))
.INPUT(filter, TensorType({ DT_FLOAT, DT_INT8 }))
.OPTIONAL_INPUT(bias, TensorType ({ DT_FLOAT, DT_INT32 }))
```

```
.OUTPUT(y, TensorType ({ DT_FLOAT }))
.ATTR(group, AttrValue::INT { 1 })
.ATTR(num_output, AttrValue::INT { 0 })
.ATTR(pad, AttrValue::LIST_INT({ 0, 0, 0, 0 }))
.ATTR(stride, AttrValue::LIST_INT({ 1, 1 }))
.ATTR(dilation, AttrValue::LIST_INT({ 1, 1 }))
.ATTR(kernel, AttrValue::LIST_INT({ 0, 0 }))
.ATTR(pad_mode, AttrValue::INT { 0 })
.ATTR(format, AttrValue::INT { 0 })
.ATTR(x_quant_type, AttrValue::INT { 0 })
.ATTR(filter_quant_type, AttrValue::INT { 0 })
.ATTR(x_quant_scale, AttrValue::FLOAT { 1.0 })
.ATTR(x_quant_offset, AttrValue::INT { 0 })
.ATTR(filter_quant_scales, AttrValue::LIST_FLOAT({}))
.OP_END()
```

- The **x\_quant\_type**, **filter\_quant\_type**, **x\_quant\_scale**, **x\_quant\_offset**, and **filter\_quant\_scales** fields are added to support quantization.
- In an unquantized convolution process, **x\_quant\_type** and **filter\_quant\_type** are set to **0** or left empty.
- In a quantized convolution process, only 8-bit quantization is supported, and **x\_quant\_type** and **filter\_quant\_type** can only be set to **1**.
- The scale parameter **x\_quant\_scale** must be greater than **0**.
- **filter\_quant\_scales** indicates the weight scale. At least one **filter\_quant\_scales** parameter is required, whose value must be greater than **0**.
- The values of the scale type parameters must be the same for all operators that need to be quantized on the entire network.
- The weight is quantified by the user. The quantized weight is of data type **DT\_INT8**, passed through input **w**.
- The bias (if applicable) is quantified by the user. The quantized bias is of data type **DT\_INT32**, passed through input **b**.

## 2.3.2 AIPP Attribute

A Da Vinci model can be generated with the artificial intelligence pre-processing (AIPP) function based on IR definition. For details about the IR operator definition, see `/ddk/ai_ddk_lib/include/graph/op/image_defs.h`.

Take ImageData as an example:

```
REG_OP(ImageData)
.INPUT(x, TensorType({ DT_UINT8 }))
.OUTPUT(y, TensorType({ DT_UINT8 }))
.REQUIRED_ATTR(input_format, AttrValue::STR)
.REQUIRED_ATTR(src_image_size_w, AttrValue::INT)
.REQUIRED_ATTR(src_image_size_h, AttrValue::INT)
.ATTR(image_type, AttrValue::STR { "JPEG" })
.OP_END()
```

- **input\_format**: mandatory attribute, indicating the format of the source image. Currently, YUV420SP\_U8, YUV422SP\_U8, AYUV444\_U8, YUYV\_U8, YUV400\_U8, XRGB8888\_U8, and ARGB8888\_U8 are supported.
- **src\_image\_size\_w** and **src\_image\_size\_h**: mandatory attributes, indicating the length and width of the source image.
- **image\_type**: mandatory attribute, indicating the image type. The JPEG, BT\_601\_NARROW, BT\_601\_FULL, and BT\_709\_NARROW formats are supported. The default value is **JPEG**.

---

# 3 Online Model Building

---

## 3.1 Model Building Demo

### 3.1.1 Dependencies

- Tool: Android Studio
- Source code package directory:  
**ddk/app\_sample/IR\_model\_demo/IR\_Demo\_Soure\_Code.rar**  
It is an online building demo package, containing the complete code of the demo APK and project file.
- Other dependency: **ddk/ddk/ai\_ddk\_lib/**  
It contains the header files and library files required for app development and compilation.  
Components:
  - **include**: header files that the project depends on (including the IR API and operator definition header files)
  - **lib/lib64**: APK integration library files

### 3.1.2 Procedure for Online Model Building

- Step 1** Decompress the source code package **IR\_Demo\_Soure\_Code.rar**.
- Step 2** In Android Studio, open the source code project **IR\_Demo\_Soure\_Code**.
- Step 3** Modify the parameters and configurations in the demo **createModel.cpp** based on the native model.
1. Find all the single-operator models from the native model.
  2. Build single-operators: Call the operator APIs described in [5.8 Graph APIs](#) to build single-operators and cascade them according to the native model.
  3. Build a graph and set the inputs and outputs of the graph.

4. Build a model and add the graph object to the model.
5. Build the network: Call the IR building APIs described in [5.10 Model Building APIs](#) to build a model generation file.

**Step 4** Compile and generate an APK in Android Studio.

**Step 5** Install the APK on the mobile phone and run the app to generate an OM.

----End

## 3.2 Building an IR Model

### 3.2.1 Native Model

The following uses the SqueezeNet as an example. For details about the code, see the C++ demo **createModel.cpp** in the **jni** directory.

This model contains the following operators:

- Data
- Const
- Convolution
- Activation
- Pooling
- Concat
- Softmax

### 3.2.2 Building Single-Operators

Call the operator APIs described in [5.8 Graph APIs](#) to build single-operators and cascade them according to the native model.

#### 3.2.2.1 Data & Const

You can build a Data operator as the input of the entire network. An example is as follows:

```
TensorDesc desc(Shape({in_channels, out_channels, h, w}), format, datatype);  
string data_name = op_name;  
data = op::Data(data_name);  
data.update_input_desc_x(desc);
```

The Const operator can be used as the input of the operator for information such as the weight. An example is as follows:

```
ge::op::Const conv10_const_1= op::Const("conv10_filter").set_attr_value(weightList0[0]);
```

### 3.2.2.2 AIPP

#### NOTE

This function is newly added in HiAI DDK V320.

AIPP operators include AIPP input operators and AIPP functional operators.

AIPP input operators: ImageData and DynamicImageData.

AIPP functional operators: ImageCrop, ImageChannelSwap, ImageColorSpaceConversion, ImageResize, ImageDataTypeConversion, and ImagePadding.

### Static and Dynamic AIPP

**Table 3-1** Precautions for using AIPP input operators

Operator Name	Application Scenario	Whether AIPP Functional Operators Can Be Cascaded
ImageData	Used for static AIPP. The type, length, and width of the input image are defined by <b>input_format</b> , <b>src_image_size_w</b> , and <b>src_image_size_h</b> .	Yes
DynamicImageData	Used for dynamic AIPP. The type, length, and width of the input image are uncertain.	No

The IR model definition of AIPP supports multiple input scenarios. Multiple ImageData operators can be defined on the network. In addition, multiple groups of different AIPP functional operators can be connected after an ImageData operator is defined.

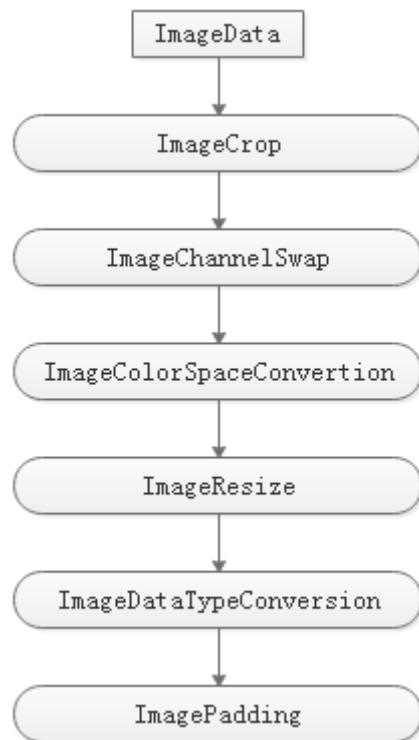
### Cascading Sequence of Static AIPP Operators

For all AIPP functions, the hardware supports only a fixed processing sequence. The operators must be cascaded during IR model building according to the sequence shown in [Figure 3-1](#).

#### NOTE

Unnecessary AIPP functional operators can be omitted. However, operators of other types cannot be inserted, and the same AIPP functional operator cannot be added repeatedly.

**Figure 3-1** Cascading sequence of AIPP operators



## Example

Assume that the model input needs to be cropped. The IR code implementation is as follows:

```
// Step 1. Build the ImageData node.
TensorDesc desc(Shape({n, channels, h, w}), format, datatype);
ge::op::ImageData data =
op::ImageData(data_name).set_attr_input_format("XRGB8888_U8").set_attr_src_image_size_w(32).set_attr_src_image_size_h(32);
data.update_input_desc_x(desc);
// Step 2. Build the AIPP functional operators cropOp and resizeOp.
auto cropOp =
op::ImageCrop("crop").set_input_x(data).set_attr_load_start_pos_w(0).set_attr_load_start_pos_h(0).set_attr_crop_size_w(16).set_attr_crop_size_h(16);
auto resizeOp =
op::ImageResize("resize").set_input_x(cropOp).set_attr_resize_output_h(32).set_attr_resize_output_w(32);
// Step 3. Concatenate with other functional operators, for example, pool op
auto poolOp = op::Pooling("pool").set_input_x(resizeOp).
```

```
set_attr_data_mode(0).set_attr_pad_mode(4).set_attr_ceil_mode(1).set_attr_mode(0).set_attr_pad(AttrValue::LIST_INT({ 0, 0, 0, 0})).set_attr_window(AttrValue::LIST_INT({ 3, 3})).set_attr_stride(AttrValue::LIST_INT({ 2, 2})).set_attr_global_pooling(0);
```

### 3.2.2.3 Convolution

According to the operator definition, the Convolution operator has the following inputs and attributes:

```
REG_OP(Convolution)
.INPUT(x, TensorType ({ DT_FLOAT } )) ...../
.INPUT(w, TensorType ({ DT_FLOAT, DT_INT8 } )) .....//
.OPTIONAL_INPUT(b, TensorType ({ DT_FLOAT } )) ...../
.OUTPUT(y, TensorType ({ DT_FLOAT } )) ...../
.ATTR(mode, AttrValue::INT { 1 })
.ATTR(group, AttrValue::INT { 1 }) .....
.ATTR(num_output, AttrValue::INT { 0 }) .....
.ATTR(pad, AttrValue::LIST_INT({ 0, 0, 0, 0 } )) .....
.ATTR(stride, AttrValue::LIST_INT({ 1, 1 } )) .....
.ATTR(dilation, AttrValue::LIST_INT({ 1, 1 } )) .....
.ATTR(kernel, AttrValue::LIST_INT({ 0, 0 } )) .....
.ATTR(pad_mode, AttrValue::INT { 0 }) .....
.OP_END()
```

For a specific node, call the single-operator model APIs described in [5.8 Graph APIs](#) to set the input and attribute values based on the detailed information about the node in the network model.

The first Convolution node on the network has one input and two weights. Set them using the `set_input_x`, `set_input_w`, and `set_input_b` APIs, respectively. Input **b** is optional. You can choose whether to set it as required.

The attributes such as **stride** and **num\_output** are specified in the model. For other unspecified attributes, the default values are provided and can be changed by using the `set_attr_attribute` name API.

```
auto conv0 = op::Convolution("conv0")
.set_input_x(data)
.set_input_w(conv0_const_0)
.set_attr_kernel(AttrValue::LIST_INT({7,7}))
.set_attr_mode(1)
.set_attr_stride(AttrValue::LIST_INT({2,2}))
.set_attr_dilation(AttrValue::LIST_INT({1,1}))
.set_attr_group(1)
.set_attr_pad(AttrValue::LIST_INT({3, 3, 3, 3}))
```



```
.set_attr_pad_mode(4)
.set_attr_num_output(64)
```

### 3.2.2.4 Activation

According to the operator definition, the Activation operator has the following input and attributes:

```
REG_OP(Convolution)
.INPUT(x, TensorType ({ DT_FLOAT } )) ...../
.INPUT(w, TensorType ({ DT_FLOAT, DT_INT8 } )) .....//
.OPTIONAL_INPUT(b, TensorType ({ DT_FLOAT } )) ...../
.OUTPUT(y, TensorType ({ DT_FLOAT } )) ...../
.ATTR(mode, AttrValue::INT { 1 })
.ATTR(group, AttrValue::INT { 1 }) .....
.ATTR(num_output, AttrValue::INT { 0 }) .....
.ATTR(pad, AttrValue::LIST_INT ({ 0, 0, 0, 0 } ))
.ATTR(stride, AttrValue::LIST_INT ({ 1, 1 } )) .....
.ATTR(dilation, AttrValue::LIST_INT ({ 1, 1 } )) .....
.ATTR(kernel, AttrValue::LIST_INT ({ 0, 0 } )) .....
.ATTR(pad_mode, AttrValue::INT { 0 }) .....
.OP_END()
```

For a specific node, call the single-operator model APIs described in [5.8 Graph APIs](#) to set the input and attribute values based on the detailed information about the node in the network model.

The first Activation node on the network has one input. Set it using the `set_input_x` API.

No attribute value is specified in the model, the default values are set by using the `set_attr_Attribute name` API.

```
auto relu_conv1 = op::Activation("relu_conv1")
.set_input_x(conv1)
.set_attr_coef(0.000000)
.set_attr_mode(1);
```

### 3.2.2.5 Concat

According to the operator definition, the Concat operator has the following input and attributes:

```
REG_OP(Concat)
.DYNAMIC_INPUT(x, TensorType({ DT_FLOAT, DT_INT32 }))
.OUTPUT(y, TensorType({ DT_FLOAT, DT_INT32 }))
.ATTR(axis, AttrValue::INT { 1 })
.ATTR(N, AttrValue::INT { 1 })
.OP_END()
```

For a specific node, call the single-operator model APIs described in [5.8 Graph APIs](#) to set the input and attribute values based on the detailed information about the node in the network model.

The first Concat node on the network has two inputs. Limit the number of inputs by using the `create_dynamic_input_x` API.

No attribute value is specified in the model, the default values are set by using the `set_attr_Attribute name` API.

```
auto fire2_concat = op::Concat("fire2/concat")
.create_dynamic_input_x(2)
.set_dynamic_input_x(1, fire2_relu_expand1x1)
.set_dynamic_input_x(2, fire2_relu_expand3x3)
.set_attr_axis(1);
```

### 3.2.2.6 Pooling

According to the operator definition, the Pooling operator has the following input and attributes:

```
REG_OP(Pooling)
.INPUT(x, TensorType({ DT_FLOAT }))
.OUTPUT(y, TensorType({ DT_FLOAT }))
.ATTR(mode, AttrValue::INT { 0 })
.ATTR(pad_mode, AttrValue::INT { 0 })
.ATTR(global_pooling, AttrValue::BOOL { false })
.ATTR(window, AttrValue::LIST_INT({ 1, 1 }))
.ATTR(pad, AttrValue::LIST_INT({ 0, 0, 0, 0 }))
.ATTR(stride, AttrValue::LIST_INT({ 1, 1 }))
.ATTR(ceil_mode, AttrValue::INT { 0 })
.ATTR(data_mode, AttrValue::INT { 1 })
.OP_END()
```

For a specific node, call the single-operator model APIs described in [5.8 Graph APIs](#) to set the input and attribute values based on the detailed information about the node in the network model.

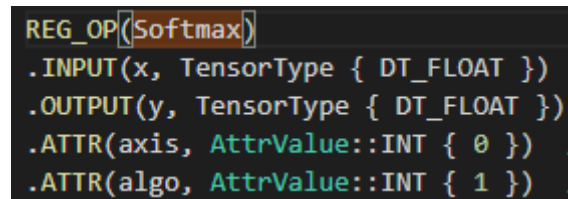
The first Pooling node on the network has one input. Set it using the `set_input_x` API.

The attribute **stride** is specified in the model. For other unspecified attributes, the default values are provided and can be changed by using the `set_attr_Attribute name` API.

```
auto pool1 = op::Pooling("pool1").set_input_x(relu_conv1)
.set_attr_data_mode(0)
.set_attr_pad_mode(4)
.set_attr_ceil_mode(1)
.set_attr_mode(0)
.set_attr_pad(AttrValue::LIST_INT({ 0, 0, 0, 0}))
.set_attr_window(AttrValue::LIST_INT({ 3, 3}))
.set_attr_stride(AttrValue::LIST_INT({ 2, 2}))
.set_attr_global_pooling(false);
```

### 3.2.2.7 Softmax

According to the operator definition, the Softmax operator has the following input and attributes:



```
REG_OP(Softmax)
.INPUT(x, TensorType { DT_FLOAT })
.OUTPUT(y, TensorType { DT_FLOAT })
.ATTR(axis, AttrValue::INT { 0 })
.ATTR(algo, AttrValue::INT { 1 })
```

For a specific node, call the single-operator model APIs described in [5.8 Graph APIs](#) to set the input and attribute values based on the detailed information about the node in the network model.

The first Softmax node in the network has one input. Set it using the `set_input_x` API.

No attribute value is specified in the model, the default values are set by using the `set_attr_Attribute name` API.

```
auto prob = op::Softmax("prob").set_input_x(pool10)
.set_attr_axis(1)
.set_attr_algo(1);
```

## 3.2.3 Building the Network

In the demo, Java invokes the native code through the JNI. The native code invokes the functions of the **libhiai\_ir.so** and **libhiai\_ir\_build.so** libraries to build

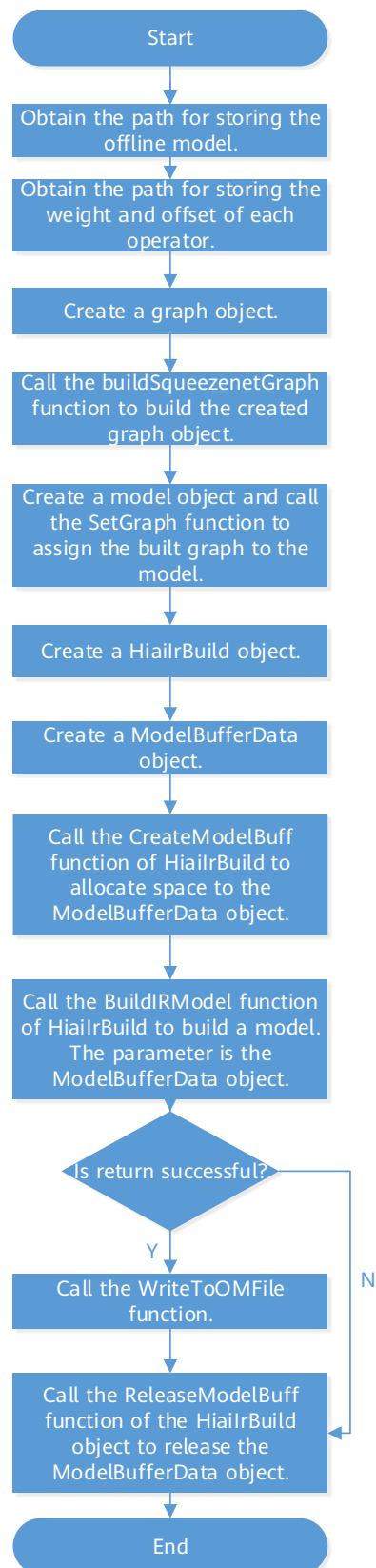
the IR model, as shown in [Figure 3-2](#). [Figure 3-3](#) describes how to invoke the buildSqueezentGraph function.

The code is implemented in

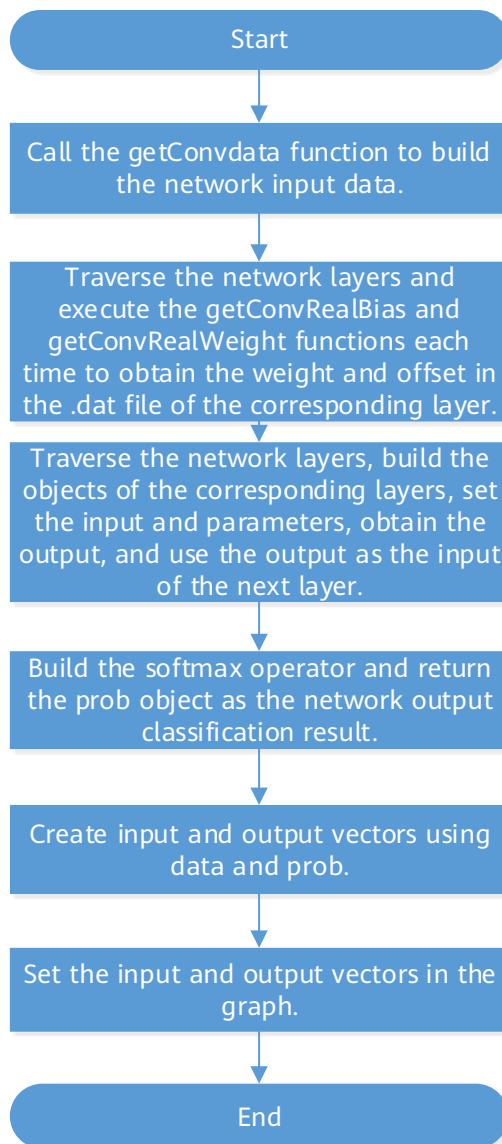
**Demo\_Soure\_Code\app\src\main\jni\createModel.cpp** of the IR\_model\_demo project in the DDK. The

Java\_com\_huawei\_hiaidemo\_utils\_ModelManager\_createOmModel function in the file is the native code of the Java method.

**Figure 3-2** Process of building the network



**Figure 3-3** Calling the buildSqueezenetGraph function



# 4 Offline Model Building

## 4.1 Model Building Demo

### 4.1.1 Dependencies

- Environment: Linux or Ubuntu
- Source code package directory:  
**ddk/tools/tools\_omg/IR\_Model\_Offline\_Demo/**  
It is an offline building demo package.  
Components:
  - **makefile**: auto compilation file
  - **squeezenet.cpp**: IR offline building demo file
  - **squeezenet**: IR offline building executable file
- Other dependencies:
  - **ddk/tools/tools\_omg/**  
It is the offline model generator (OMG), which adapts to different network files and weight files under various frameworks and generates IR OMs.  
Component: **lib64** (library file required for offline building)
  - **ddk/ddk\_ai\_ddk\_lib/include/**  
It contains the header files that the project depends on (including the IR API and operator definition header files).  
Download some of the header files that the project depends on from the following websites:  
<https://android.googlesource.com/platform/external/libcxx/+/refs/heads/pie-release/include/>  
<https://android.googlesource.com/platform/external/libcxxabi/+/refs/heads/pie-release/include/>  
For details about their storage locations, see the comments in **makefile**.

- For details about the compilation, see makefile in **ddk/tools/tools\_omg/IR\_Model\_Offline\_Demo/makefile**.

## 4.1.2 Procedure for Offline Model Building

**Step 1** Copy the source code package **IR\_Model\_Offline\_Demo**.

**Step 2** Write the IR model generation file by referring to the offline building demo **squeezenet.cpp**.

- Find all the single-operator models from the native model.
- Build single-operators: Call the operator APIs described in [5.8 Graph APIs](#) to build single-operators and cascade them according to the native model.
- Build a graph and set the inputs and outputs of the graph.
- Build a model and add the graph object to the model.
- Build the network: Call the IR building APIs described in [5.10 Model Building APIs](#) to build a model generation file.

**Step 3** In the Linux or Ubuntu environment, compile the IR model to generate an executable file, and then run the executable program to generate an OM model. Perform the following operations:

1. Link the \*.so file on which the OM building depends.

Command: **export LD\_LIBRARY\_PATH= Path** (path of the dependent dynamic link library file, for example, **ddk/tools/tools\_omg/lib64**)

2. Run the **./squeezenet** command to generate an OM.

### NOTE

- The compilation depends on the Make tool (Make 4.1 is recommended.) and g++ compiler. You need to install them by yourself.
- Compilation command: **make squeezenet**

-----End

## 4.2 Building an IR Model

### 4.2.1 Native Model

The SqueezeNet is used as an example. For details about the code, see **squeezenet.cpp**.

This model contains the following operators:

- Data
- Const
- Convolution
- Activation



- Pooling
- Concat
- Softmax

## 4.2.2 Building Single-Operators

Same as building single-operators for online models. For details, see [3.2.2 Building Single-Operators](#).

## 4.2.3 Building the Model

1. Call [5.8 Graph APIs](#) to set the input and output of graph and construct the graph object.
2. Call [5.9 Model APIs](#) to add the graph object to the model and build the model.

Build a graph object:

```
std::vector<Operator> inputs{data};
std::vector<Operator> outputs{prob};
graph.SetInputs(inputs).SetOutputs(outputs);
Build a model:
ge::Model model("model", "model_v00001");
model.SetGraph(graph);
```

## 4.2.4 Building the Network

Call the operator APIs described in [5.10 Model Building APIs](#) to generate a model.

```
domi::HiAiIrBuild ir_build;
domi::ModelBufferData om_model_buff;
ir_build.CreateModelBuff(model, om_model_buff);
bool ret = ir_build.BuildIRModel(model, om_model_buff);
if(ret){
    ret = WriteToOMFile(om_model_buff, "/squeezenet.ir.offline.om");
}
ir_build.ReleaseModelBuff(om_model_buff);
```

## 4.2.5 Generating a Model

In the Linux or Ubuntu environment, execute the compiled IR model executable file to generate an OM.

# 5 API Reference

HiAI DDK V320 provides a collection of secure and easy-to-use APIs for IR graph composition. These APIs can be called to build a network model, and set the graph in a model, operators in a graph, and attributes of the model and operators. These APIs are classified as follows.

- Auxiliary (data) types

Type	Function	Header File
Class NamedAttrs	Attribute instance in key-value format	attr_value.h, attributes_holder.h
Class AttrHolder	Stores attribute values of operators.	
Class Tensor	Sets or obtains tensor data and describe tensors.	tensor.h
Class TensorDesc	Describes tensors, including Shape, DataType, and Format.	
Class Shape	Describes the tensor shape.	
Class buffer	Serializes and deserializes model data.	buffer.h

- API types

Type	Function	Header File
Class Operator	Stores operator attributes and sets input and output.	operator.h
Class operator registration	Sets the input, output, verification attributes, and obtains dimensions during operator registration.	operator_reg.h
Class Graph	Sets the input and output of a graph and verifies the validity of the graph.	graph.h

Type	Function	Header File
Class Model	Stores the model version number, attributes, and internal graph of the model.	model.h
Class model building	Builds a model.	hiai_ir_build.h

The API definition files are stored in the **ddk/ai\_ddk\_lib/include** directory of the DDK. For details about the definition of each API, see the corresponding header file.

## 5.1 Attribute APIs

The Attribute APIs are defined in **attr\_value.h** and **attributes\_holder.h**.

### 5.1.1 SetAttr

#### Function Prototype

```
GraphErrCodeStatus SetAttr(const string& name, const AttrValue& value);
```

#### Function Description

Sets the value of an attribute.

#### Parameter Description

Parameter	Input/Output	Type	Description
name	Input	const string&	Attribute name
value	Input	const AttrValue&	Attribute value

#### Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : success <b>GRAPH_FAILED</b> : failure

## Exception Handling

None

## Restriction

None

## 5.1.2 GetAttr

### Function Prototype

```
GraphErrCodeStatus GetAttr(const string& name, AttrValue& value) const;
```

### Function Description

Obtains the value of an attribute.

### Parameter Description

Parameter	Input/Output	Type	Description
name	Input	const string&	Attribute name
value	Output	AttrValue&	Attribute value

### Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : success <b>GRAPH_FAILED</b> : failure

## Exception Handling

None

## Restriction

None

## 5.1.3 HasAttr

### Function Prototype

```
bool HasAttr(const string& name) const;
```

### Function Description

Checks whether an attribute exists by name.

### Parameter Description

Parameter	Input/Output	Type	Description
name	Input	const string&	Attribute name

### Return Value

Type	Description
bool	<b>true</b> : An attribute with <b>name</b> is found. <b>false</b> : No attribute with <b>name</b> is found.

### Exception Handling

None

### Restriction

None

## 5.1.4 DelAttr

### Function Prototype

```
GraphErrCodeStatus DelAttr(const string& name);
```

### Function Description

Deletes an attribute.

### Parameter Description

Parameter	Input/Output	Type	Description
-----------	--------------	------	-------------

Parameter	Input/Output	Type	Description
name	Input	const string&	Attribute name

## Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : success <b>GRAPH_FAILED</b> : failure

## Exception Handling

None

## Restriction

None

## 5.1.5 SetName

### Function Prototype

```
void SetName(const std::string& name);
```

### Function Description

Sets the name of an attribute.

### Parameter Description

Parameter	Input/Output	Type	Description
name	Input	const string&	Attribute name

## Return Value

None

## Exception Handling

None

## Restriction

None

## 5.1.6 GetName

### Function Prototype

```
string GetName() const;
```

### Function Description

Obtains the name of an attribute.

### Parameter Description

None

### Return Value

Type	Description
string	Attribute name

## Exception Handling

None

## Restriction

None

## 5.1.7 GetItem

### Function Prototype

```
AttrValue GetItem(const string& key) const;
```

### Function Description

Obtains the attribute value by name.

### Parameter Description

Parameter	Input/Output	Type	Description
key	Input	const string&	Attribute name

## Return Value

Type	Description
AttrValue	Obtained attribute value

## Exception Handling

None

## Restriction

None

## 5.1.8 SetValue

### Function Prototype

```
GraphErrCodeStatus SetValue(std::initializer_list<DT>&& val);
```

```
GraphErrCodeStatus SetValue(const std::vector<DT>& val);
```

```
GraphErrCodeStatus SetValue(DT&& val);
```

```
GraphErrCodeStatus SetValue(const T& t);
```

```
GraphErrCodeStatus SetValue(const vector<T>& t);
```

### Function Description

Sets the parameter values of a specified profile type.

### Parameter Description

Parameter	Input/Output	Type	Description
val	Input	<ul style="list-style-type: none"><li>std::initializer_list&lt;DT&gt;&amp;&amp;,</li><li>const std::vector&lt;DT&gt;&amp;, or</li><li>DT&amp;&amp;</li></ul>	Value to be configured
t	Input	<ul style="list-style-type: none"><li>const T&amp;, or</li><li>const vector&lt;T&gt;&amp;</li></ul>	Value to be configured



## Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : success <b>GRAPH_FAILED</b> : failure <b>GRAPH_PARAM_INVALID</b> : invalid argument

## Exception Handling

None

## Restriction

None

## 5.1.9 GetValue

### Function Prototype

```
GraphErrCodeStatus GetValue(std::vector<DT>& val) const;
```

```
GraphErrCodeStatus GetValue(DT& val) const;
```

```
GraphErrCodeStatus GetValue(T& t);
```

```
GraphErrCodeStatus GetValue(vector<T>& t);
```

### Function Description

Obtains the parameter values of a specified profile type.

### Parameter Description

Parameter	Input/Output	Type	Description
val	Input	<ul style="list-style-type: none"><li>std::vector&lt;DT&gt;&amp;, or</li><li>DT&amp;</li></ul>	Variable for saving the obtained value
t	Input	<ul style="list-style-type: none"><li>T&amp;, or</li><li>vector&lt;T&gt;&amp;</li></ul>	Variable for saving the obtained value

## Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : success <b>GRAPH_FAILED</b> : failure <b>GRAPH_PARAM_INVALID</b> : invalid obtained value

## Exception Handling

None

## Restriction

None

## 5.1.10 CreateFrom

### Function Prototype

```
static AttrValue CreateFrom(DT&& val);  
static AttrValue CreateFrom(const T& val);  
static AttrValue CreateFrom(const vector<T>& val);  
static AttrValue CreateFrom(std::initializer_list<DT>&& val);
```

### Function Description

Creates an attribute value object of a specified profile type.

### Parameter Description

Parameter	Input/Output	Type	Description
val	Input	<ul style="list-style-type: none"><li>DT&amp;&amp;,</li><li>const T&amp;,</li><li>const vector&lt;T&gt;&amp;, or</li><li>std::initializer_list&lt;DT&gt;&amp;&amp;</li></ul>	Value configured for creating an object

## Return Value

Type	Description
AttrValue	Created attribute value object

## Exception Handling

None

## Restriction

None

## 5.1.11 GetValueType

### Function Prototype

```
ValueType GetValueType() const;
```

### Function Description

Obtains the type of an attribute value.

### Parameter Description

None

## Return Value

Type	Description
ValueType	Attribute type

## Exception Handling

None

## Restriction

None

## 5.1.12 IsEmpty

### Function Prototype

```
bool IsEmpty() const;
```

### Function Description

Checks whether the value of an attribute is set.

### Parameter Description

None

### Return Value

Type	Description
bool	<b>false:</b> The attribute value is not set. <b>true:</b> The attribute value is set.

### Exception Handling

None

### Restriction

None

## 5.1.13 Copy

### Function Prototype

```
AttrValue Copy() const;
```

### Function Description

Copies an attribute value and returns the attribute object.

### Parameter Description

None

### Return Value

Type	Description
------	-------------

Type	Description
AttrValue	Attribute object

## Exception Handling

None

## Restriction

None

## 5.1.14 operator==

### Function Prototype

```
bool operator==(const AttrValue& other) const;
```

### Function Description

Reloads the "==" operator.

### Parameter Description

Parameter	Input/Output	Type	Description
other	Input	const AttrValue&	const reference to the operand of the <b>AttrValue</b> type

### Return Value

Type	Description
bool	<b>false</b> : unequal <b>true</b> : equal

## Exception Handling

None

## Restriction

None

## 5.1.15 MutableTensor

### Function Prototype

```
GraphErrCodeStatus MutableTensor(TensorPtr& tensor);
```

### Function Description

Obtains the tensor reference (mutable).

### Parameter Description

Parameter	Input/Output	Type	Description
tensor	Output	TensorPtr&	Tensor reference

### Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : success <b>GRAPH_FAILED</b> : failure

### Exception Handling

None

### Restriction

None

## 5.1.16 MutableListTensor

### Function Prototype

```
GraphErrCodeStatus MutableListTensor(vector<TensorPtr>& list_tensor);
```

### Function Description

Obtains the tensor list reference (mutable).

### Parameter Description

Parameter	Input/Output	Type	Description
-----------	--------------	------	-------------

Parameter	Input/Output	Type	Description
list_tensor	Output	vector<TensorPtr>&	Tensor list reference

## Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : success <b>GRAPH_FAILED</b> : failure

## Exception Handling

None

## Restriction

None

## 5.1.17 InitDefault

### Function Prototype

```
void InitDefault();
```

### Function Description

Initializes the default values.

### Parameter Description

None

### Return Value

None

### Exception Handling

None

### Restriction

None

## 5.1.18 GetProtoOwner

### Function Prototype

```
inline const ProtoMsgOwner& GetProtoOwner() const;
```

### Function Description

Obtains a ProtoMsgOwner object.

### Parameter Description

None

### Return Value

Type	Description
const ProtoMsgOwner&	Obtained <b>ProtoMsgOwner</b> object

### Exception Handling

None

### Restriction

None

## 5.1.19 GetProtoMsg

### Function Prototype

```
inline ProtoType* GetProtoMsg() const;
```

### Function Description

Obtains a ProtoType object.

### Parameter Description

None

### Return Value

Type	Description
ProtoType*	Obtained <b>ProtoType</b> object



## Exception Handling

None

## Restriction

None

## 5.1.20 CopyValueFrom

### Function Prototype

```
void CopyValueFrom(const GelrProtoHelper<ProtoType>& other);
```

### Function Description

Copies an object.

### Parameter Description

Parameter	Input/Output	Type	Description
other	Input	const GelrProtoHelper<ProtoType>&	Copied <b>ProtoType</b> object

### Return Value

None

## Exception Handling

None

## Restriction

None

## 5.1.21 MoveValueFrom

### Function Prototype

```
void MoveValueFrom(GelrProtoHelper<ProtoType>&& other);
```

## Function Description

Moves an object.

## Parameter Description

Parameter	Input/Output	Type	Description
other	Input	GetIrProtoHelper<ProtoType>&&	Moved <b>ProtoType</b> object

## Return Value

None

## Exception Handling

None

## Restriction

None

# 5.2 Tensor APIs

The Tensor APIs are defined in **tensor.h**.

## 5.2.1 GetTensorDesc

### Function Prototype

```
TensorDesc GetTensorDesc() const;
```

### Function Description

Obtains the descriptor (**TensorDesc**) of a tensor.

### Parameter Description

None

### Return Value

Type	Description
TensorDesc	Tensor descriptor

## Exception Handling

None

## Restriction

None

## 5.2.2 MutableTensorDesc

### Function Prototype

TensorDesc& MutableTensorDesc();

### Function Description

Obtains the descriptor of a tensor (mutable).

### Parameter Description

None

### Return Value

Type	Description
TensorDesc&	Tensor descriptor (mutable)

## Exception Handling

None

## Restriction

None

## 5.2.3 SetTensorDesc

### Function Prototype

GraphErrCodeStatus SetTensorDesc(const TensorDesc &tensorDesc);

### Function Description

Sets the descriptor of a tensor.

## Parameter Description

Parameter	Input/Output	Type	Description
tensorDesc	Input	const TensorDesc &	Tensor descriptor to be set

## Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : success <b>GRAPH_FAILED</b> : failure

## Exception Handling

None

## Restriction

None

## 5.2.4 GetData

### Function Prototype

```
const Buffer GetData() const;
```

### Function Description

Obtains the data of a tensor.

### Parameter Description

None

### Return Value

Type	Description
const Buffer	Tensor data

## Exception Handling

None

## Restriction

None

## 5.2.5 MutableData

### Function Prototype

```
Buffer MutableData();
```

### Function Description

Obtains the data of a tensor.

### Parameter Description

None

### Return Value

Type	Description
Buffer	Tensor data

## Exception Handling

None

## Restriction

None

## 5.2.6 SetData

### Function Prototype

```
GraphErrCodeStatus SetData(std::vector<uint8_t> &&data);  
GraphErrCodeStatus SetData(const std::vector<uint8_t> &data);  
GraphErrCodeStatus SetData(const Buffer &data);  
GraphErrCodeStatus SetData(const uint8_t *data, size_t size);
```

### Function Description

Sets data to a tensor.

## Parameter Description

Parameter	Input/Output	Type	Description
data	Input	std::vector<uint8_t> && const std::vector<uint8_t> & const Buffer & or const uint8_t *	Data to be set
size	Input	size_t	Data length, in bytes

## Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : success <b>GRAPH_FAILED</b> : failure

## Exception Handling

None

## Restriction

None

## 5.2.7 Clone

### Function Prototype

Tensor Clone() const;

### Function Description

Clones a tensor.

### Parameter Description

None

### Return Value

Type	Description
Tensor	Cloned Tensor object

## Exception Handling

None

## Restriction

None

## 5.2.8 operator=

### Function Prototype

Tensor& operator= (const Tensor &other);

### Function Description

Reloads the "=" operator.

### Parameter Description

Parameter	Input/Output	Type	Description
other	Input	const Tensor&	Reference to the immutable Tensor object

### Return Value

Type	Description
Tensor&	Reference to the Tensor object

## Exception Handling

None

## Restriction

None

## 5.3 TensorDesc APIs

The TensorDesc APIs are defined in **tensor.h**.

## 5.3.1 Update

### Function Prototype

```
void Update(Shape shape, Format format = FORMAT_NCHW, DataType dt = DT_FLOAT);
```

### Function Description

Updates **shape**, **format**, and **datatype** of a **TensorDesc** object.

### Parameter Description

Parameter	Input/Output	Type	Description
shape	Input	Shape	<b>shape</b> object to be updated
format	Input	Format	<b>format</b> object to be updated. The default value is <b>FORMAT_NCHW</b> . For the definition of the <b>Format</b> data types, see <a href="#">Format</a> .
dt	Input	DataType	<b>datatype</b> object to be updated. The default value is <b>DT_FLOAT</b> . For the definition of the <b>DataType</b> data types, see <a href="#">DataType</a> .

### Return Value

None

### Exception Handling

None

### Restriction

None

### Data Type Description

- Format

```
enum Format {
    FORMAT_NCHW = 0,      /**< NCHW */
    FORMAT_NHWC,          /**< NHWC */
    FORMAT_ND,            /**< Nd Tensor*/
    ...
}
```



```
};
```

- **DataType**

```
enum DataType {  
    DT_FLOAT = 0,           // float type  
    DT_FLOAT16 = 1,        // fp16 type  
    DT_INT8 = 2,            // int8 type  
    DT_UINT8 = 4,          // uint8 type  
    DT_INT32 = 3,           //  
    DT_INT64 = 9,          // int64 type  
    DT_BOOL = 12,          // bool type  
    ...  
};
```

## 5.3.2 GetShape

### Function Prototype

```
Shape GetShape() const;
```

### Function Description

Obtains **shape** described by **TensorDesc**.

### Parameter Description

None

### Return Value

Type	Description
Shape	<b>shape</b> described by <b>TensorDesc</b>

### Exception Handling

None

### Restriction

**shape** returned is constant and is immutable.

## 5.3.3 MutableShape

### Function Prototype

```
Shape& MutableShape();
```

## Function Description

Obtains the **shape** reference (mutable) in **TensorDesc**.

## Parameter Description

None

## Return Value

Type	Description
Shape &	<b>shape</b> reference (mutable) in <b>TensorDesc</b>

## Exception Handling

None

## Restriction

None

## 5.3.4 SetShape

### Function Prototype

```
void SetShape(Shape shape);
```

### Function Description

Sets **shape** described by **TensorDesc**.

### Parameter Description

Parameter	Input/Output	Type	Description
shape	Input	Shape	<b>shape</b> object to be set to <b>TensorDesc</b>

### Return Value

None

### Exception Handling

None

## Restriction

None

## 5.3.5 GetFormat

### Function Prototype

Format GetFormat() const;

### Function Description

Obtains the **format** information of a tensor described by **TensorDesc**.

### Parameter Description

None

### Return Value

Type	Description
Format	<b>format</b> information of the tensor described by <b>TensorDesc</b>

## Exception Handling

None

## Restriction

None

## 5.3.6 SetFormat

### Function Prototype

void SetFormat(Format format);

### Function Description

Sets the **format** information of a tensor described by **TensorDesc**.

### Parameter Description

Parameter	Input/Output	Type	Description
format	Input	Format	<b>format</b> information to be set

## Return Value

None

## Exception Handling

None

## Restriction

None

## 5.3.7 GetDataType

### Function Prototype

```
DataType GetDataType() const;
```

### Function Description

Obtains the data type of a tensor described by **TensorDesc**.

### Parameter Description

None

### Return Value

Type	Description
DataType	Data type of the tensor described by <b>TensorDesc</b>

## Exception Handling

None

## Restriction

None

## 5.3.8 SetDataType

### Function Prototype

```
void SetDataType(DataType dt);
```

## Function Description

Sets the data type of a tensor described by **TensorDesc**

## Parameter Description

Parameter	Input/Output	Type	Description
dt	Input	DataType	<b>dt</b> information to be set For details, see <a href="#">DataType</a> .

## Return Value

None

## Exception Handling

None

## Restriction

None

## 5.3.9 Clone

### Function Prototype

```
TensorDesc Clone() const;
```

### Function Description

Clones **TensorDesc**.

### Parameter Description

None

### Return Value

Type	Description
TensorDesc	Cloned <b>TensorDesc</b> object

## Exception Handling

None

## Restriction

None

## 5.3.10 IsValid

### Function Prototype

```
GraphErrCodeStatus IsValid();
```

### Function Description

Checks whether a tensor object is valid.

### Parameter Description

None

### Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : valid <b>GRAPH_FAILED</b> : invalid

## Exception Handling

None

## Restriction

None

## 5.3.11 operator=

### Function Prototype

```
TensorDesc& operator=(const TensorDesc& other);  
TensorDesc& operator=(TensorDesc&& other);
```

## Function Description

Reloads the "=" operator.

## Parameter Description

Parameter	Input/Output	Type	Description
other	Input	const TensorDesc&	Reference to the immutable <b>TensorDesc</b> object
other	Input	TensorDesc&&	Rvalue reference to the <b>TensorDesc</b> object

## Return Value

Type	Description
TensorDesc&	Reference to the <b>TensorDesc</b> object

## Exception Handling

None

## Restriction

None

# 5.4 Shape APIs

The Shape APIs are defined in **tensor.h**.

## 5.4.1 GetDimNum

### Function Prototype

```
size_t GetDimNum() const;
```

### Function Description

Obtains the dimension count of **Shape**.

## Parameter Description

None

## Return Value

Type	Description
size_t	<b>Shape</b> dimension count of the tensor

## Exception Handling

None

## Restriction

None

## 5.4.2 GetDim

### Function Prototype

```
int64_t GetDim(size_t idx) const;
```

### Function Description

Obtains the length of dimension **idx** of **Shape**.

### Parameter Description

Parameter	Input/Output	Type	Description
idx	Input	size_t	Dimension index, which starts from 0

## Return Value

Type	Description
int64_t	Length of dimension <b>idx</b>

## Exception Handling

None



## Restriction

None

## 5.4.3 GetDims

### Function Prototype

```
std::vector<int64_t> GetDims() const;
```

### Function Description

Obtains the vector composed of all **Shape** dimensions.

### Parameter Description

None

### Return Value

Type	Description
std::vector<int64_t>	Vector composed of all <b>Shape</b> dimensions

## Exception Handling

None

## Restriction

None

## 5.4.4 SetDim

### Function Prototype

```
GraphErrCodeStatus SetDim(size_t idx, int64_t value);
```

### Function Description

Sets the value of dimension **idx** in **Shape**.

### Parameter Description

Parameter	Input/Output	Type	Description
idx	Input	size_t	Dimension index of <b>Shape</b> , which starts

Parameter	Input/Output	Type	Description
			from 0
value	Input	int64_t	Value to be set

## Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : success <b>GRAPH_FAILED</b> : failure

## Exception Handling

None

## Restriction

None

## 5.4.5 GetShapeSize

### Function Prototype

```
int64_t GetShapeSize() const;
```

### Function Description

Obtains the product of all dimensions in **Shape**.

### Parameter Description

None

## Return Value

Type	Description
int64_t	Product of all dimensions

## Exception Handling

None

## Restriction

None

## 5.4.6 operator=

### Function Prototype

Shape& operator=(const Shape& other);

Shape& operator=(Shape&& other);

### Function Description

Reloads the "=" operator.

### Parameter description

Parameter	Input/Output	Type	Description
other	Input	const Shape&	Reference to the immutable <b>Shape</b> object
other	Input	Shape&&	Rvalue reference to the <b>Shape</b> object

### Return Value

Type	Description
Shape&	Reference to the <b>Shape</b> object

### Exception Handling

None

## Restriction

None

## 5.5 Buffer APIs

The Buffer APIs are defined in **buffer.h**.

## 5.5.1 ClearBuffer

### Function Prototype

```
void ClearBuffer();
```

### Function Description

Clears a buffer by setting the pointer to the data buffer to null.

### Parameter Description

None

### Return Value

None

### Exception Handling

None

### Restriction

None

## 5.5.2 GetData

### Function Prototype

```
std::uint8_t* GetData();  
const std::uint8_t* GetData() const;
```

### Function Description

Obtains the pointer to the buffered data.

### Parameter Description

None

### Return Value

Type	Description
uint8_t*	Pointer to the buffered data

## Exception Handling

Exception	Description
Empty buffer	Returns a null pointer.

## Restriction

None

## 5.5.3 GetSize

### Function Prototype

```
std::size_t GetSize() const;
```

### Function Description

Obtains the size of the buffered data.

### Parameter Description

None

### Return Value

Type	Description
size_t	Size of the buffered data

## Exception Handling

Exception	Description
Empty buffer	Returns data size <b>0</b> .

## Restriction

None

## 5.5.4 CopyFrom

### Function Prototype

```
static Buffer CopyFrom(std::uint8_t* data, std::size_t bufferSize);
```

### Function Description

Copies **bufferSize** based on the address to which the pointer in data points, and saves the data address to buffer.

### Parameter Description

Parameter	Input/Output	Type	Description
data	Input	uint8_t*	Pointer to the data to be copied
bufferSize	Input	size_t	Size of the data to be copied

### Return Value

Type	Description
Buffer	Buffer object for storing the copied data

### Exception Handling

None

### Restriction

None

## 5.5.5 data

### Function Prototype

```
std::uint8_t* data();  
const std::uint8_t* data() const;
```

### Function Description

Obtains the pointer to the buffered data.

## Parameter description

None

## Return Value

Type	Description
uint8_t*	Pointer to the buffered data

## Exception Handling

Exception	Description
Empty buffer	Returns a null pointer.

## Restriction

None

## 5.5.6 size

### Function Prototype

```
std::size_t size() const;
```

### Function Description

Obtains the size of the buffered data.

## Parameter description

None

## Return Value

Type	Description
size_t	Size of the buffered data

## Exception Handling

Exception	Description
Empty buffer	Returns data size 0.

## Restriction

None

## 5.5.7 clear

### Function Prototype

```
void clear();
```

### Function Description

Clears the buffered data.

### Parameter description

None

### Return Value

Type	Description
void	Clears the buffered data.

## Exception Handling

Exception	Description
Empty buffer	No operation is required.

## Restriction

None



## 5.5.8 operator=

### Function Prototype

Buffer& operator=(const Buffer& other);

### Function Description

Reloads the "=" operator.

### Parameter description

Parameter	Input/Output	Type	Description
other	Input	const Buffer&	const reference to the operand of the <b>Buffer</b> type

### Return Value

Type	Description
Buffer&	Reference of the <b>Buffer</b> type

### Exception Handling

None

### Restriction

None

## 5.5.9 operator[]

### Function Prototype

uint8\_t operator[](size\_t index) const;

### Function Description

Reloads the "[]" operator.

## Parameter description

Parameter	Input/Output	Type	Description
index	Input	size_t	Index

## Return Value

Type	Description
uint8_t	When the buffer is not empty, the buffer address with <b>index</b> is returned.  When the buffer is empty, the default invalid value <b>0xff</b> is returned.

## Exception Handling

None

## Restriction

None

# 5.6 Operator APIs

The Operator APIs are defined in **operator.h**.

## 5.6.1 GetName

### Function Prototype

```
string GetName() const;
```

### Function Description

Obtains the name of an operator.

### Parameter Description

None

## Return Value

Type	Description
string	Operator name

## Exception Handling

None

## Restriction

None

## 5.6.2 SetInput

### Function Prototype

```
Operator& SetInput(const string& dstName, const Operator& srcOprt);
```

```
Operator& SetInput(const string& dstName, const Operator& srcOprt, const string  
&name);
```

```
Operator& SetInput(int dstIndex, const Operator& srcOprt, int srcIndex);
```

### Function Description

Sets the input of an operator.

### Parameter Description

Parameter	Input/Output	Type	Description
dstName	Input	const string&	Name of the destination node on the input edge of the current operator
srcOprt	Input	const Operator&	Reference to the source operator object where the start node on the input edge of the current operator is located
name	Input	const string&	Name of the start node on the input edge of the current operator
dstIndex	Input	int	Index of the destination node on the input edge of the current operator
srcIndex	Input	int	Index of the start node on the input edge of the current operator

## Return Value

Type	Description
Operator&	Reference to the operator object whose input is being set

## Exception Handling

None

## Restriction

**Operator& srcOprt** can have only one output.

## 5.6.3 GetInputDesc

### Function Prototype

```
TensorDesc GetInputDesc(const string& name) const;
```

```
TensorDesc GetInputDesc(uint32_t index) const;
```

### Function Description

Obtains the input **TensorDesc** of an operator based on its input name or input index.

### Parameter Description

Parameter	Input/Output	Type	Description
name	Input	const string&	<b>Input</b> name of the operator If no operator input name is available, the default object constructed by <b>TensorDesc</b> is returned.
index	Input	uint32_t	<b>Input</b> index of the operator If no operator input index is available, the default object constructed by <b>TensorDesc</b> is returned.

## Return Value

Type	Description
TensorDesc	<b>TensorDesc</b> of the operator input

## Exception Handling

None

## Restriction

None

## 5.6.4 TryGetInputDesc

### Function Prototype

```
bool TryGetInputDesc(const string& name, TensorDesc& tensorDesc) const;
```

### Function Description

Obtains the input **TensorDesc** of an operator based on its input name.

### Parameter Description

Parameter	Input/Output	Type	Description
name	Input	const string&	<b>Input</b> name
tensorDesc	Output	TensorDesc&	Reference to the <b>TensorDesc</b> object

## Return Value

Type	Description
bool	If the <b>TensorDesc</b> is obtained successfully, <b>true</b> is returned. Otherwise, <b>false</b> is returned.

## Exception Handling

None

## Restriction

None

## 5.6.5 UpdateInputDesc

### Function Prototype

```
GraphErrCodeStatus UpdateInputDesc(const string& name, const TensorDesc&
tensorDesc);
```

### Function Description

Updates the input **TensorDesc** of an operator based on its input name.

### Parameter Description

Parameter	Input/Output	Type	Description
name	Input	const string&	<b>Input</b> name of the operator
tensorDesc	Input	const TensorDesc&	Reference to the <b>TensorDesc</b> object

### Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : success <b>GRAPH_FAILED</b> : failure

## Exception Handling

None

## Restriction

None

## 5.6.6 GetOutputDesc

### Function Prototype

```
TensorDesc GetOutputDesc(const string& name) const;
TensorDesc GetOutputDesc(uint32_t index) const;
```

## Function Description

Obtains the output **TensorDesc** of an operator based on its output name or output index.

## Parameter Description

Parameter	Input/Output	Type	Description
name	Input	const string&	<b>Output</b> name of the operator
index	Input	uint32_t	<b>Output</b> index of the operator If no operator output index is available, the default object constructed by <b>TensorDesc</b> is returned.

## Return Value

Type	Description
TensorDesc	If <b>TensorDesc</b> is obtained successfully, the required <b>TensorDesc</b> object is returned. Otherwise, the default <b>TensorDesc</b> object is returned.

## Exception Handling

None

## Restriction

None

## 5.6.7 UpdateOutputDesc

### Function Prototype

```
GraphErrCodeStatus UpdateOutputDesc (const string& name, const TensorDesc& tensorDesc);
```

### Function Description

Updates the output **TensorDesc** of an operator based on its output name.

## Parameter Description

Parameter	Input/Output	Type	Description
name	Input	const string&	<b>Output</b> name of the operator
tensorDesc	Input	const TensorDesc&	Reference to the <b>TensorDesc</b> object

## Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : success <b>GRAPH_FAILED</b> : failure

## Exception Handling

None

## Restriction

None

## 5.6.8 GetDynamicInputDesc

### Function Prototype

```
TensorDesc GetDynamicInputDesc(const string& name, const unsigned int index)  
const;
```

### Function Description

Obtains the dynamic input **TensorDesc** of an operator based on the combination of its input name and index.

## Parameter Description

Parameter	Input/Output	Type	Description
name	Input	const string&	Dynamic <b>Input</b> name of the operator
index	Input	const unsigned int	Dynamic <b>Input</b> index of the operator, which starts from 1



## Return Value

Type	Description
TensorDesc	If <b>TensorDesc</b> is obtained successfully, the required <b>TensorDesc</b> object is returned. Otherwise, the default <b>TensorDesc</b> object is returned.

## Exception Handling

None

## Restriction

None

## 5.6.9 UpdateDynamicInputDesc

### Function Prototype

```
GraphErrCodeStatus UpdateDynamicInputDesc(const string& name, const
unsigned int index, const TensorDesc& tensorDesc);
```

### Function Description

Updates the dynamic input **TensorDesc** of an operator based on the combination of its input name and index.

### Parameter Description

Parameter	Input/Output	Type	Description
name	Input	const string&	Dynamic <b>Input</b> name of the operator
index	Input	const unsigned int	Dynamic <b>Input</b> index of the operator, which starts from 1
tensorDesc	Input	const TensorDesc&	<b>TensorDesc</b> object

## Return Value

Type	Description
------	-------------

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : success <b>GRAPH_FAILED</b> : failure

## Exception Handling

None

## Restriction

None

## 5.6.10 GetDynamicOutputDesc

### Function Prototype

```
TensorDesc GetDynamicOutputDesc (const string& name, const unsigned int index)  
const;
```

### Function Description

Obtains the dynamic output **TensorDesc** of an operator based on the combination of its output name and index.

### Parameter Description

Parameter	Input/Output	Type	Description
name	Input	const string&	Dynamic <b>Output</b> name of the operator
index	Input	const unsigned int	Dynamic <b>Output</b> index of the operator, which starts from 1

### Return Value

Type	Description
TensorDesc	If <b>TensorDesc</b> is obtained successfully, the required <b>TensorDesc</b> object is returned. Otherwise, the default <b>TensorDesc</b> object is returned.

## Exception Handling

None

## Restriction

None

## 5.6.11 UpdateDynamicOutputDesc

### Function Prototype

```
GraphErrCodeStatus UpdateDynamicOutputDesc (const string& name, const  
unsigned int index, const TensorDesc& tensorDesc);
```

### Function Description

Updates the dynamic output **TensorDesc** of an operator based on the combination of its output name and index.

### Parameter Description

Parameter	Input/Output	Type	Description
name	Input	const string&	Dynamic <b>Output</b> name of the operator
index	Input	const unsigned int	Dynamic <b>Output</b> index of the operator
tensorDesc	Input	const TensorDesc&	<b>TensorDesc</b> object

### Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : success <b>GRAPH_FAILED</b> : failure

## Exception Handling

None

## Restriction

None

## 5.6.12 SetAttr

### Function Prototype

```
Operator& SetAttr(const string& name, AttrValue&& attrValue);
```

### Function Description

Sets the attribute value of an operator.

### Parameter Description

Parameter	Input/Output	Type	Description
name	Input	const string&	Attribute name
attrValue	Input	AttrValue&&	Attribute value to be set

### Return Value

Type	Description
Operator&	Operator object itself

## Exception Handling

None

## Restriction

None

## 5.6.13 GetAttr

### Function Prototype

```
GraphErrCodeStatus GetAttr(const string& name, AttrValue& attrValue) const;
```

### Function Description

Obtains the attribute value based on an attribute name.

## Parameter Description

Parameter	Input/Output	Type	Description
name	Input	const string&	Attribute name
attrValue	Input	AttrValue&	Attribute value

## Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : success <b>GRAPH_FAILED</b> : failure

## Exception Handling

None

## Restriction

None

## 5.7 Operator Registration APIs

Operator type registration starts with the **REG\_OP** API and ends with the **OP\_END** API. The input, output, and attribute information (**INPUT**, **OUTPUT**, and **ATTR**) to be registered is linked by periods (.). After an operator type is registered, a class named after the operator type is automatically generated.

Example:

```
REG_OP(FullConnection)
.INPUT(x, TensorType::ALL())
.INPUT(w, TensorType::ALL())
.INPUT(b, TensorType::ALL())
.OUTPUT(y, TensorType::ALL())
.ATTR(num_output, AttrValue::INT{0})
.OP_END()
```

The operator registration APIs are defined in **operator\_reg.h**. For details about registered operators and their header files, see [1.3 Supported Operators](#).

## 5.7.1 REG\_OP

### Function Prototype

REG\_OP(x)

### Function Description

Registers an operator type. Two constructors corresponding to the operator type are automatically generated.

For example, register an operator type **Conv2D** by calling the **REG\_OP(Conv2D)** API. Two **Conv2D** constructors are generated. The operator name needs to be specified for **Conv2D(const string& name)** for example, **Conv2D\_XX** (unique index). If the operator name is left blank, that is, **Conv2D()**, the default operator name is used.

```
class Conv2D : public Operator {  
    typedef Conv2D _THIS_TYPE;  
public:  
    explicit Conv2D(const string& name);  
    explicit Conv2D();  
}
```

### Parameter Description

Parameter	Input/Output	Type	Description
x	Input	-	Macro parameter, operator type name to be registered

### Return Value

None

### Exception Handling

None

### Restriction

The operator type name must be unique.

## 5.7.2 ATTR

### Function Prototype

ATTR(x, default\_value)

### Function Description

Registers an operator attribute. The default value must be specified, so that the default value can be used if the attribute value of an operator object is not set.

After the operator attributes are successfully registered, three external APIs (for obtaining the attribute name, obtaining the attribute value, and setting the attribute value, respectively) are automatically generated.

The following describes the operator attribute APIs generated in `int64_t` and `int64_t` list scenarios:

- Register the attribute **mode** by calling **ATTR(mode, AttrValue::INT{1})**. The attribute type is **int64\_t** and the default value is **1**.

After the attribute is successfully registered, the following APIs are automatically generated:

```
static const string name_attr_mode(); // Returns the attribute name, that is, mode.  
int64_t get_attr_mode() const;       // Returns the value of the mode attribute.  
_THIS_TYPE& set_attr_mode(int64_t v); // Sets the value of the mode attribute. The operator object  
is returned.
```

- Register the **pad** attribute by calling **ATTR(pad, AttrValue::LIST\_INT{0, 0, 0, 0})**. The attribute type is **int64\_t list**. The default value is **{0,0,0,0}**.

After the attribute is successfully registered, the following APIs are automatically generated:

```
static const string name_attr_pad(); // Returns the attribute name, that is, pad.  
vector<int64_t> get_attr_pad() const; // Returns the value of the pad attribute.  
_THIS_TYPE& set_attr_pad(vector<int64_t> v); // Sets the value of the pad attribute. The operator  
object is returned.
```

### Parameter Description

Parameter	Input/Output	Type	Description
x	Input	-	Macro parameter, attribute name of the operator
default_value	Input	-	Value of an operator attribute. The default value varies depending on the attribute type. The following attribute types are supported: <ul style="list-style-type: none"><li><b>AttrValue::INT</b>: The attribute type is <code>int64_t</code>.</li><li><b>AttrValue::FLOAT</b>: The attribute type</li></ul>

Parameter	Input/Output	Type	Description
			<p>is float.</p> <ul style="list-style-type: none"> <li>• <b>AttrValue::STR</b>: The attribute type is string.</li> <li>• <b>AttrValue::BOOL</b>: The attribute type is bool.</li> <li>• <b>AttrValue::TENSOR</b>: The attribute type is tensor.</li> <li>• <b>AttrValue::LIST_INT</b>: The attribute type is vector&lt;int64_t&gt; (int64_t list).</li> <li>• <b>AttrValue::LIST_FLOAT</b>: The attribute type is vector&lt;float&gt; (float list).</li> <li>• <b>AttrValue::LIST_STR</b>: The attribute type is vector&lt;string&gt; (string list).</li> <li>• <b>AttrValue::LIST_BOOL</b>: The attribute type is vector&lt;bool&gt; (bool list).</li> <li>• <b>AttrValue::LIST_TENSOR</b>: The attribute type is vector&lt;Tensor&gt; (tensor list).</li> </ul>

## Return Value

None

## Exception Handling

None

## Restriction

For an operator, the registered attribute name must be unique.

## 5.7.3 REQUIRED\_ATTR

### Function Prototype

REQUIRED\_ATTR (x, type)

### Function Description

Registers an operator attribute. The default value must be specified.



After the operator attributes are successfully registered, three external APIs (for obtaining the attribute name, obtaining the attribute value, and setting the attribute value, respectively) are automatically generated.

For example, register the attribute **mode** of the `int64_t` type by calling the **REQUIRED\_ATTR (mode, Int)** API. After the operator attribute is successfully registered, the following APIs are automatically generated:

```
static const string name_attr_mode();           // Returns the attribute name, that is, mode.  
OpInt get_attr_mode() const;                  // Returns the value of the mode attribute. OpInt indicates  
int64_t.  
_THIS_TYPE& set_attr_mode(const OpInt& v);    // Sets the value of the mode attribute. A this object is  
returned.
```

## Parameter Description

Parameter	Input/Output	Type	Description
x	Input	-	Macro parameter, attribute name of the operator
type	Input	-	The following attribute types are supported: <ul style="list-style-type: none"><li>• <b>AttrValue::INT</b>: The attribute type is <code>int64_t</code>.</li><li>• <b>AttrValue::FLOAT</b>: The attribute type is <code>float</code>.</li><li>• <b>AttrValue::STR</b>: The attribute type is <code>string</code>.</li><li>• <b>AttrValue::BOOL</b>: The attribute type is <code>bool</code>.</li><li>• <b>AttrValue::TENSOR</b>: The attribute type is <code>tensor</code>.</li><li>• <b>AttrValue::LIST_INT</b>: The attribute type is <code>vector&lt;int64_t&gt;</code> (<code>int64_t</code> list).</li><li>• <b>AttrValue::LIST_FLOAT</b>: The attribute type is <code>vector&lt;float&gt;</code> (<code>float</code> list).</li><li>• <b>AttrValue::LIST_STR</b>: The attribute type is <code>vector&lt;string&gt;</code> (<code>string</code> list).</li><li>• <b>AttrValue::LIST_BOOL</b>: The attribute type is <code>vector&lt;bool&gt;</code> (<code>bool</code> list).</li><li>• <b>AttrValue::LIST_TENSOR</b>: The attribute type is <code>vector&lt;Tensor&gt;</code> (<code>tensor</code> list).</li></ul>

## Return Value

None

## Exception Handling

None

## Restriction

For an operator, the registered attribute name must be unique.

## 5.7.4 INPUT

### Function Prototype

INPUT (x, t)

### Function Description

Registers the input information of an operator.

After the operator input information is successfully registered, APIs (for obtaining the operator input name and setting the operator input description) are generated automatically.

For example, if the operator input is **x** and the data type supported by the operator input is **TensorType{DT\_FLOAT}**, call the **INPUT(x, TensorType{DT\_FLOAT})** API. After the operator input is successfully registered, the following APIs are automatically generated:

```
static const string name_in_x();           // Returns the input name, that is, x.
_THIS_TYPE& set_input_x(Operator& v, const string& srcName);
                                           // Specifies the link between input x and output srcName
of the operator object v. The operator object itself is returned.
_THIS_TYPE& set_input_x(Operator& v);     // Specifies the link between input x and output 0 of the
operator object v. The operator object is returned.
TensorDesc get_input_desc_x();           // Returns the description of input x.
GraphErrCodeStatus update_input_desc_x(const TensorDesc& tensorDesc);
                                           // Sets the description of input x, including Shape,
DataType, and Format. GraphErrCodeStatus indicates the uint32_t type. If a non-zero value is returned, an
error occurs.
```

### Parameter Description

Parameter	Input/Output	Type	Description
x	Input	-	Macro parameter, input name of the operator
t	Input	-	Data type supported by the operator input. One or more data types defined by <b>TensorType</b> are supported. Separate data types with commas (,). For example: <b>TensorType{DT_FLOAT}</b>

Parameter	Input/Output	Type	Description
			TensorType({DT_FLOAT, DT_INT8}) For details about class <b>TensorType</b> , see <a href="#">Description of Class TensorType</a> .

## Return Value

None

## Exception Handling

None

## Restriction

For an operator, the registered input name must be unique.

## Description of Class TensorType

Class TensorType defines the data types supported by the input or output. The following APIs are provided:

- **TensorType(DataType dt)**: Specifies that only one data type is supported.
- **TensorType(std::initializer\_list<DataType> types)**: Specifies that multiple data types are supported.
- **static TensorType ALL()**: Specifies that all data types are supported.
- **static TensorType FLOAT()**: Specifies that the DT\_FLOAT and DT\_FLOAT16 data types are supported.

## 5.7.5 OPTIONAL\_INPUT

### Function Prototype

OPTIONAL\_INPUT(x, t)

### Function Description

Registers the optional input information of an operator.

After the optional input information is successfully registered, APIs (for obtaining the operator input name and setting the operator input description) are generated automatically.

For example, if the operator input is **b** and the data type supported by the operator input is **TensorType{DT\_FLOAT}**, call the **OPTIONAL\_INPUT(b,**

**TensorType{DT\_FLOAT}** API. After the operator input is successfully registered, the following APIs are automatically generated:

```
static const string name_in_b();    // Return the input name, that is, b.
_THIS_TYPE& set_input_b(Operator& v, const string& srcName); // Specifies the link between input b and
output srcName of the operator object v. The operator object itself is returned.
_THIS_TYPE& set_input_b(Operator& v);    // Specifies the link between input x and output 0 of the
operator object v. The operator object is returned.
TensorDesc get_input_desc_b();    // Returns the description of input b.
GraphErrCodeStatus update_input_desc_b(const TensorDesc& tensorDesc);
// Sets the description of input b, including Shape,
DataType, and Format.
```

## Parameter Description

Parameter	Input/ Output	Type	Description
x	Input	-	Macro parameter, input name of the operator
t	Input	-	Data type supported by the operator input. One or more data types defined by <b>TensorType</b> are supported. Separate data types with commas (.). For example: TensorType{DT_FLOAT} TensorType({DT_FLOAT, DT_INT8}) For details about class <b>TensorType</b> , see <a href="#">Description of Class TensorType</a> .

## Return Value

None

## Exception Handling

None

## Restriction

For an operator, the registered input name must be unique.

## 5.7.6 DYNAMIC\_INPUT

### Function Prototype

DYNAMIC\_INPUT (x, t)

### Function Description

Registers the dynamic input information of an operator.

After the dynamic input information is successfully registered, APIs (for creating dynamic input and setting the input description) are generated automatically.

For example, if the dynamic input is **d** and the data type supported by the operator input is **TensorType{DT\_FLOAT}**, call the **DYNAMIC\_INPUT(d, TensorType{DT\_FLOAT})** API. After the dynamic input is successfully registered, the following APIs are automatically generated:

```
_THIS_TYPE& create_dynamic_input_d(unsigned int num);    // Creates dynamic input d, including a  
number of num inputs.  
TensorDesc get_dynamic_input_desc_d(unsigned int index); // Returns description indexth of dynamic input  
d, including Shape, DataType, and Format.  
GraphErrCodeStatus update_dynamic_input_desc_d(unsigned int index, const TensorDesc& tensorDesc);  
// Updates description indexth of dynamic input d.  
_THIS_TYPE& set_dynamic_input_d(unsigned int dstIndex, Operator &v);    // Specifies the link between  
input dstIndex of d and index 0 of the operator object v. The operator object itself is returned.  
_THIS_TYPE& set_dynamic_input_d(unsigned int dstIndex, Operator &v, const string &srcName);    //  
Specifies the link between input dstIndex of d and output srcName of the operator object v. The operator  
object itself is returned.
```

### Parameter Description

Parameter	Input/Output	Type	Description
x	Input	-	Macro parameter, input name of the operator
t	Input	-	Data type supported by the operator input. One or more data types defined by <b>TensorType</b> are supported. Separate data types with commas (.). For example: TensorType{DT_FLOAT} TensorType({DT_FLOAT, DT_INT8} For details about class <b>TensorType</b> , see <a href="#">Description of Class TensorType</a> .

### Return Value

None

## Exception Handling

None

## Restriction

For an operator, the registered input name must be unique.

## 5.7.7 OUTPUT

### Function Prototype

OUTPUT (x, t)

### Function Description

Registers the output information of an operator.

After the operator output information is successfully registered, APIs (for obtaining the operator output name and setting the operator output description) are generated automatically.

For example, if the operator output is **y** and the data type supported by the operator input is **TensorType{DT\_FLOAT}**, call the **OUTPUT(y, TensorType{DT\_FLOAT})** API. After the operator output is successfully registered, the following APIs are automatically generated:

```
static const string name_out_y();           // Returns the output name, that is, y.
TensorDesc get_output_desc_y();           // Returns the description of output y.
GraphErrCodeStatus update_output_desc_y(const TensorDesc& tensorDesc); // Sets the description of output y, including Shape,
DataType, and Format.
```

### Parameter Description

Parameter	Input/Output	Type	Description
x	Input	-	Macro parameter, output name of the operator
t	Input	-	Data type supported by the operator output. One or more data types defined by <b>TensorType</b> are supported. Separate data types with commas (,). For example: TensorType{DT_FLOAT} TensorType({DT_FLOAT, DT_INT8}) For details about class <b>TensorType</b> , see <a href="#">Description of Class TensorType</a> .

## Return Value

None

## Exception Handling

None

## Restriction

For an operator, the registered output name must be unique.

## 5.7.8 DYNAMIC\_OUTPUT

### Function Prototype

DYNAMIC\_OUTPUT (x, t)

### Function Description

Registers the dynamic output information of an operator.

After the dynamic output information is successfully registered, APIs (for creating dynamic output and setting the output description) are generated automatically.

For example, if the dynamic output is **d** and the data type supported by the operator output is **TensorType{DT\_FLOAT}**, call the **DYNAMIC\_OUTPUT (d, TensorType{DT\_FLOAT})** API. After the dynamic output is successfully registered, the following APIs are automatically generated:

```
_THIS_TYPE& create_dynamic_output_d(unsigned int num);    // Creates dynamic output d, including a  
number of num inputs.  
TensorDesc get_dynamic_output_desc_d(unsigned int index); // Returns description indexth of dynamic  
output d, including Shape, DataType, and Format.  
GraphErrCodeStatus update_dynamic_output_desc_d(unsigned int index, const TensorDesc& tensorDesc);  
// Updates description indexth of dynamic output d.
```

### Parameter Description

Parameter	Input/Output	Type	Description
x	Input	-	Macro parameter, output name of the operator
t	Input	-	Data type supported by the operator output. One or more data types defined by <b>TensorType</b> are supported. Separate data types with commas (,). For example: <b>TensorType{DT_FLOAT}</b> <b>TensorType{{DT_FLOAT, DT_INT8}}</b>

Parameter	Input/Output	Type	Description
			For details about class <b>TensorType</b> , see <a href="#">Description of Class TensorType</a> .

## Return Value

None

## Exception Handling

None

## Restriction

For an operator, the registered output name must be unique.

## 5.7.9 OP\_END

### Function Prototype

OP\_END ()

### Function Description

Ends operator registration.

### Parameter Description

None

## Return Value

None

## Exception Handling

None

## Restriction

None



## 5.7.10 GET\_INPUT\_SHAPE

### Function Prototype

GET\_INPUT\_SHAPE(op, name)

### Function Description

Obtains the input shape of an operator.

### Parameter Description

Parameter	Input/Output	Type	Description
op	Input	-	Operator object
name	Input	-	<b>Input</b> name of the operator

### Return Value

Type	Description
-	<b>Input</b> shape of the operator

### Exception Handling

None

### Restriction

None

## 5.7.11 GET\_DYNAMIC\_INPUT\_SHAPE

### Function Prototype

GET\_DYNAMIC\_INPUT\_SHAPE(op, name, index)

### Function Description

Obtains the dynamic input shape of an operator.

### Parameter Description

Parameter	Input/Output	Type	Description
-----------	--------------	------	-------------

Parameter	Input/Output	Type	Description
op	Input	-	Operator object
name	Input	-	Dynamic <b>Input</b> name of the operator
index	Input	-	Dynamic <b>Input</b> index of the operator

## Return Value

Type	Description
-	Dynamic <b>Input</b> shape of the operator

## Exception Handling

None

## 5.7.12 SET\_OUTPUT\_SHAPE

### Function Prototype

SET\_OUTPUT\_SHAPE(op, name, shape)

### Function Description

Sets the shape of an output operator.

### Parameter Description

Parameter	Input/Output	Type	Description
op	Input	-	Operator object
name	Input	-	Name of the output operator
shape	Input	-	Shape of the output operator to be set

## Return Value

None

## Exception Handling

None

## 5.7.13 SET\_DYNAMIC\_OUTPUT\_SHAPE

### Function Prototype

SET\_DYNAMIC\_OUTPUT\_SHAPE(op, name, index, shape)

### Function Description

Sets the shape of the dynamic output operator.

### Parameter Description

Parameter	Input/Output	Type	Description
op	Input	-	Operator object
name	Input	-	Name of the dynamic output operator
index	Input	-	Index of the dynamic output operator to be set
shape	Input	-	Shape of the dynamic output operator to be set

### Return Value

None

### Exception Handling

None

## 5.7.14 GET\_ATTR

### Function Prototype

GET\_ATTR(op, name, type, val)

### Function Description

Obtains the attribute value of an operator.

### Parameter Description

Parameter	Input/Output	Type	Description
op	Input	-	Operator object

Parameter	Input/Output	Type	Description
name	Input	-	Operator attribute name
type	Input	-	Type of the operator attribute value to be obtained
val	Input	-	Operator attribute value to be obtained

## Return Value

Type	Description
-	Operator attribute value

## Exception Handling

None

# 5.8 Graph APIs

The Graph APIs are defined in **graph.h**.

## 5.8.1 SetInputs

### Function Prototype

```
Graph& SetInputs(std::vector<Operator>& inputs);
```

### Function Description

Sets the input operator in a graph.

### Parameter Description

Parameter	Input/Output	Type	Description
inputs	Input	std::vector<Operator>&	Input operator in the Graph

## Return Value

Type	Description
Graph&	Caller itself

## Exception Handling

None

## Restriction

None

## 5.8.2 SetOutputs

### Function Prototype

```
Graph& SetOutputs(std::vector<Operator>& outputs);
```

### Function Description

Sets the output operator linked with a graph.

### Parameter Description

Parameter	Input/Output	Type	Description
outputs	Input	std::vector<Operator>&	Output operator linked with the graph

## Return Value

Type	Description
Graph&	Caller itself

## Exception Handling

None

## Restriction

None

## 5.8.3 IsValid

### Function Prototype

```
bool IsValid() const;
```

### Function Description

Checks whether a graph object is valid.

### Parameter Description

None

### Return Value

Type	Description
bool	<b>true</b> : valid (non-null pointer) <b>false</b> : invalid (null pointer)

### Exception Handling

None

### Restriction

None

## 5.8.4 AddOp

### Function Prototype

```
GraphErrCodeStatus AddOp(ge::Operator& op);
```

### Function Description

Adds an operator to a graph.

### Parameter Description

Parameter	Input/Output	Type	Description
op	Input	ge::Operator&	Operator to be added

## Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : success <b>GRAPH_FAILED</b> : failure

## Exception Handling

None

## Restriction

None

## 5.8.5 FindOpByName

### Function Prototype

```
ge::Operator FindOpByName(const string& name) const;
```

### Function Description

Returns the operator instance in a graph based on the operator name.

### Parameter Description

Parameter	Input/Output	Type	Description
name	Input	const string&	Operator name

## Return Value

Type	Description
ge::Operator	If the operator of the corresponding name is found in the graph, the operator in the graph is returned. Otherwise, an operator of the <b>NULL</b> type is returned.

## Exception Handling

None

## Restriction

None

## 5.8.6 CheckOpByName

### Function Prototype

```
GraphErrCodeStatus CheckOpByName(const string& name) const;
```

### Function Description

Checks whether an operator exists in a graph based on the operator name.

### Parameter Description

Parameter	Input/Output	Type	Description
name	Input	const string&	Operator name

### Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : The operator is found. <b>GRAPH_FAILED</b> : The operator is not found.

## Exception Handling

None

## Restriction

None

## 5.8.7 GetAllOpName

### Function Prototype

```
GraphErrCodeStatus GetAllOpName(std::vector<string>& opName) const ;
```

### Function Description

Returns the names of all operators in a graph.



## Parameter Description

Parameter	Input/Output	Type	Description
opName	Output	std::vector<string>&	Returns the names of all operators in a graph.

## Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : success <b>GRAPH_FAILED</b> : failure

## Exception Handling

None

## Restriction

None

# 5.9 Model APIs

The Model APIs are defined in **model.h**.

## 5.9.1 SetName

### Function Prototype

```
void SetName(const string& name);
```

### Function Description

Sets the name of a model.

### Parameter Description

Parameter	Input/Output	Type	Description
name	Input	const string&	Model name

## Return Value

None

## Exception Handling

None

## Restriction

None

## 5.9.2 GetName

### Function Prototype

```
string GetName() const;
```

### Function Description

Obtains the name of a model.

### Parameter Description

None

## Return Value

Type	Description
string	Model name

## Exception Handling

None

## Restriction

None

## 5.9.3 SetVersion

### Function Prototype

```
void SetVersion(uint32_t version)
```

## Function Description

Sets the version of a model.

## Parameter Description

Parameter	Type	Description
version	uint32_t	Model version

## Return Value

None

## Exception Handling

None

## Restriction

None

## 5.9.4 GetVersion

### Function Prototype

```
uint32_t GetVersion() const;
```

## Function Description

Obtains the version of a model.

## Parameter Description

None

## Return Value

Type	Description
uint32_t	Model version

## Exception Handling

None

## Restriction

None

## 5.9.5 SetPlatformVersion

### Function Prototype

```
void SetPlatformVersion(string version)
```

### Function Description

Sets the version of a user-defined model.

### Parameter Description

Parameter	Data Type	Description
version	string	Version of the user-defined model

### Return Value

None

### Exception Handling

None

## Restriction

None

## 5.9.6 GetPlatformVersion

### Function Prototype

```
std::string GetPlatformVersion() const;
```

### Function Description

Obtains the version of a user-defined model. The version value is a constant.

### Parameter Description

None

## Return Value

Type	Description
string	Model version

## Exception Handling

None

## Restriction

None

## 5.9.7 GetGraph

### Function Prototype

Graph GetGraph() const;

### Function Description

Obtains the graph object in a model.

### Parameter Description

None

## Return Value

Type	Description
Graph	Graph object in the model

## Exception Handling

None

## Restriction

None

## 5.9.8 SetGraph

### Function Prototype

```
void SetGraph(const Graph& graph);
```

### Function Description

Sets the graph object of a model.

### Parameter Description

Parameter	Input/Output	Type	Description
graph	Input	const Graph&	Graph object

### Return Value

None

### Exception Handling

None

### Restriction

None

## 5.9.9 Save

### Function Prototype

```
GraphErrCodeStatus Save(Buffer& buffer) const;
```

### Function Description

Serializes a model object.

### Parameter Description

Parameter	Input/Output	Type	Description
buffer	Input	Buffer&	Reference to the serialized output object

## Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : success <b>GRAPH_FAILED</b> : failure

## Exception Handling

None

## Restriction

None

## 5.9.10 Load

### Function Prototype

```
static GraphErrCodeStatus Load(const uint8_t* data, size_t len, Model& model);
```

### Function Description

Loads serialized data and deserializes a model object.

### Parameter Description

Parameter	Input/Output	Type	Description
data	Input	const uint8_t *	Pointer to serialized data
len	Input	size_t	Length of serialized data
model	Output	Model &	Deserialized model object

## Return Value

Type	Description
GraphErrCodeStatus	<b>GRAPH_SUCCESS</b> : success <b>GRAPH_FAILED</b> : failure

## Exception Handling

None

## Restriction

None

## 5.9.11 IsValid

### Function Prototype

```
bool IsValid() const;
```

### Function Description

Checks whether a graph object in a model is valid. A null pointer indicates an invalid graph object.

### Parameter Description

None

### Return Value

Type	Description
bool	<b>true:</b> valid <b>false:</b> invalid

## Exception Handling

None

## Restriction

None

## 5.10 Model Building APIs

The model building APIs are defined in **hiai\_ir\_build.h**.

The API calling sequence during model building is as follows:

CreateModelBuff > BuildIRModel > ReleaseModelBuff



## 5.10.1 CreateModelBuff

### Function Prototype

```
bool CreateModelBuff(ge::Model& irModel,ModelBufferData& output);
```

### Function Description

Creates a model buffer.

### Parameter Description

Parameter	Input/Output	Type	Description
irModel	Input	ge::Model&	Model object
output	Output	ModelBufferData&	Offline model struct object struct ModelBufferData { void* data; uint32_t length; };

### Return Value

Type	Description
bool	<b>true:</b> The model buffer is successfully created. <b>false:</b> The model buffer fails to be created.

### Exception Handling

None

### Restriction

None

## 5.10.2 CreateModelBuff

### Function Prototype

```
bool CreateModelBuff(ge::Model& irModel, ModelBufferData& output, uint32_t  
customSize);
```

### Function Description

Creates model buffer by size.

### Parameter Description

Parameter	Input/Output	Type	Description
irModel	Input	ge::Model&	Model object
output	Output	ModelBufferData&	Offline model struct object struct ModelBufferData { void* data; uint32_t length; };
customSize	Input	uint32_t	Buffer size specified by the user (in bytes). The value must be a non-negative integer. Keep the value within 200 MB. When <b>customSize</b> is 0, the API automatically calculates the appropriate buffer size based on <b>irModel</b> .

### Return Value

Type	Description
bool	<b>true</b> : The model buffer is successfully created. <b>false</b> : The model buffer fails to be created.

## Exception Handling

None

## Restriction

None

## 5.10.3 BuildIRModel

### Function Prototype

```
bool BuildIRModel(ge::Model& irModel, ModelBufferData& output);
```

### Function Description

Builds an offline model, with a model object as the input and an offline model as the output.

### Parameter Description

Parameter	Input/Output	Type	Description
irModel	Input	ge::Model&	Model object
output	Output	ModelBufferData&	Offline model struct object struct ModelBufferData { void* data; uint32_t length; };

### Return Value

Type	Description
bool	<b>true:</b> The model is successfully created. <b>false:</b> The model fails to be created.

## Exception Handling

None

## Restriction

None

## 5.10.4 ReleaseModelBuff

### Function Prototype

```
void ReleaseModelBuff(ModelBufferData& output);
```

### Function Description

Releases a model buffer.

### Parameter Description

Parameter	Input/Output	Type	Description
output	Output	ModelBufferData&	Offline model struct object struct ModelBufferData { void* data; uint32_t length; };

### Return Value

None

### Exception Handling

None

## Restriction

None