GE API Reference

Issue 01

Date 2020-05-30

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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.

Contents

1 Introduction

The Graph Engine (GE) provides a collection of secure and easy-to-use APIs for image 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. This API collection consists of the following modules:

- API types
  - Operator

    The Operator module represents the basic operator type and provides APIs for the operator attribute access and input/output configuration.
  - Graph

    The Graph module represents a diagram constructed by connected operators. This module provides APIs for input/output configuration and graph verification.
  - Model

    The Model module represents the model corresponding to a graph. This module provides the model version and APIs for the model attribute access and model graph access.
- Auxiliary (data) types
  - AttrValue
    - NamedAttr
      - Attribute instance in the **key-value** format
    - AttrHolder
      - Stores attribute values of operators.
  - Tensor
    - TensorDesc: Describes the tensor information, including:
      - **Shape**: tensor shape
      - **DataType**: tensor data type
      - **Format**: tensor format

You can view the API definition files in **ddk/include/inc/graph** in the installation directory of the DDK. If you install Mind Studio and the DDK together in boot

installation mode, you can log in to the Mind Studio server as the Mind Studio installation user and view the API definition files in **~/tools/che/ddk/ddk/include/inc/graph**. For details about the mapping between the definition files and the APIs, see the API description.

2 Attribute APIs

The Attribute APIs are defined in **attr_value.h** and **detail/attributes_holder.h**.

2.1 SetAttr

Function Prototype

graphStatus 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

Parameter	Type	Description
-	graphStatus	If the operation is successful, **GRAPH_SUCCESS** is returned. Otherwise, **GRAPH_FAILED** is returned.

Exception Handling

None

Restriction

None

2.2 GetAttr

Function Prototype

graphStatus 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

Parameter	Type	Description
-	graphStatus	If the operation is successful, **GRAPH_SUCCESS** is returned. Otherwise, **GRAPH_FAILED** is returned.

Exception Handling

None

Restriction

None

2.3 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

Parameter	Type	Description
-	bool	**true**: An attribute named **name** is found. **false**: No attribute named **name** is found.

Exception Handling

None

Restriction

None

2.4 DelAttr

Function Prototype

graphStatus DelAttr(const string& name);

Function Description

Deletes an attribute.

Parameter Description

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

Return Value

Parameter	Type	Description
-	graphStatus	If the operation is successful, **GRAPH_SUCCESS** is returned. Otherwise, **GRAPH_FAILED** is returned.

Exception Handling

None

Restriction

None

2.5 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

2.6 GetName

Function Prototype

string GetName() const;

Function Description

Obtains the name of an attribute.

Parameter Description

None

Return Value

Parameter	Type	Description
-	string	Attribute name

Exception Handling

None

Restriction

None

2.7 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

Parameter	Type	Description
-	AttrValue	Obtained attribute value.

Exception Handling

None

Restriction

None

2.8 GetValueType

Function Prototype

ValueType GetValueType() const;

Function Description

Obtains the type of an attribute value.

Parameter Description

None

Return Value

Parameter	Type	Description
-	ValueType	Attribute type

Exception Handling

None

Restriction

None

2.9 IsEmpty

Function Prototype

bool IsEmpty() const;

Function Description

Checks whether the value of an attribute is set.

Parameter Description

None

Return Value

Parameter	Type	Description
-	bool	**false**: The attribute value is not set. **true**: The attribute value is set.

Exception Handling

None

Restriction

None

2.10 Copy

Function Prototype

AttrValue Copy() const;

Function Description

Copies an attribute value and returns the attribute object.

Parameter Description

None

Return Value

Parameter	Type	Description
-	AttrValue	Attribute object

Exception Handling

None

Restriction

None

2.11 MutableTensor

Function Prototype

graphStatus MutableTensor(TensorPtr& tensor);

Function Description

Obtains the tensor reference (mutable).

Parameter Description

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

Return Value

Parameter	Type	Description
-	graphStatus	If the operation is successful, **GRAPH_SUCCESS** is returned. Otherwise, **GRAPH_FAILED** is returned.

Exception Handling

None

Restriction

None

2.12 MutableListTensor

Function Prototype

graphStatus MutableListTensor(vector<TensorPtr>& list_tensor);

Function Description

Obtains the tensor list reference (mutable).

Parameter Description

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

Return Value

Parameter	Type	Description
-	graphStatus	If the operation is successful, **GRAPH_SUCCESS** is returned. Otherwise, **GRAPH_FAILED** is returned.

Exception Handling

None

Restriction

None

2.13 InitDefault

Function Prototype

void InitDefault();

Function Description

Initializes the default values.

Parameter Description

None

Return Value

None

Exception Handling

None

Restriction

None

3 Buffer APIs

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

3.1 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

3.2 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

Parameter	Type	Description
-	uint8_t*	Pointer to the buffered data

Exception Handling

Exception	Description
Empty buffer	Returns a null pointer.

Restriction

None

3.3 GetSize

Function Prototype

std::size_t GetSize() const;

Function Description

Obtains the size of the buffered data.

Parameter Description

None

Return Value

Parameter	Type	Description
-	size_t	Size of the buffered data

Exception Handling

Exception	Description
Empty buffer	Returns data size **0**.

Restriction

None

3.4 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

Parameter	Type	Description
-	Buffer	Buffer object for storing the copied data

Exception Handling

None

Restriction

None

4 Graph APIs

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

4.1 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

Parameter	Type	Description
-	Graph&	Caller itself

Exception Handling

None

Restriction

None

4.2 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

Parameter	Type	Description
-	Graph&	Caller itself

Exception Handling

None

Restriction

None

4.3 IsValid

Function Prototype

bool IsValid() const;

Function Description

Checks whether a graph object is valid.

Parameter Description

None

Return Value

Parameter	Type	Description
-	bool	• **true**: The constructed graph object is valid (non-null pointer). • **false**: The constructed graph object is invalid (null pointer).

Exception Handling

None

Restriction

None

4.4 AddOp

Function Prototype

graphStatus 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

Parameter	Type	Description
-	graphStatus	**GRAPH_SUCCESS**: success **GRAPH_FAILED**: failure

Exception Handling

None

Restriction

None

4.5 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

Parameter	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 **NULL** type is returned.

Exception Handling

None

Restriction

None

4.6 CheckOpByName

Function Prototype

graphStatus 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

Parameter	Type	Description
-	graphStatus	**GRAPH_SUCCESS**: The operator is found. **GRAPH_FAILED**: The operator is not found.

Exception Handling

None

Restriction

None

4.7 GetAllOpName

Function Prototype

graphStatus 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

Parameter	Type	Description
-	graphStatus	**GRAPH_SUCCESS**: success **GRAPH_FAILED**: failure

Exception Handling

None

Restriction

None

5 Model APIs

5.1 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

5.2 GetName

Function Prototype

string GetName() const;

Function Description

Obtains the name of a model.

Parameter Description

None

Return Value

Parameter	Type	Description
-	string	Model name

Exception Handling

None

Restriction

None

5.3 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

5.4 GetVersion

Function Prototype

uint32_t GetVersion() const;

Function Description

Obtains the version of an model.

Parameter Description

None

Return Value

Parameter	Type	Description
-	uint32_t	Model version

Exception Handling

None

Restriction

None

5.5 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

5.6 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

Parameter	Type	Description
-	string	Model version

Exception Handling

None

Restriction

None

5.7 GetGraph

Function Prototype

Graph GetGraph() const;

Function Description

Obtains the graph object in a model.

Parameter Description

None

Return Value

Parameter	Type	Description
-	Graph	Graph object in the model

Exception Handling

None

Restriction

None

5.8 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 Save

Function Prototype

graphStatus 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

Parameter	Type	Description
-	graphStatus	If the operation is successful, **GRAPH_SUCCESS** is returned. Otherwise, **GRAPH_FAILED** is returned.

Exception Handling

None

Restriction

None

5.10 Load

Function Prototype

static graphStatus 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

Parameter	Type	Description
-	graphStatus	If the operation is successful, **GRAPH_SUCCESS** is returned. Otherwise, **GRAPH_FAILED** is returned.

Exception Handling

None

Restriction

None

5.11 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

Parameter	Type	Description
-	bool	The value **true** indicates a valid object, and the value **false** indicates an invalid object.

Exception Handling

None

Restriction

None

6 Operator APIs

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

6.1 Operator Constructor

Function Prototype

Operator();

Function Description

Operator constructor

Parameter Description

None

Return Value

Parameter	Type	Description
-	Operator	Operator object

Exception Handling

None

Restriction

None

6.2 GetName

Function Prototype

string GetName() const;

Function Description

Obtains the name of an operator.

Parameter Description

None

Return Value

Parameter	Type	Description
-	string	Operator name

Exception Handling

None

Restriction

None

6.3 SetInput (srcOprt Has Only One Output)

Function Prototype

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

Function Description

Sets the input of an operator.

Parameter Description

Parameter	Input/Output	Type	Description
dstName	Input	const string&	Input name of the operator
srcOprt	Input	const Operator&	Input operator object whose input name is **dstName**.

Return Value

Parameter	Type	Description
-	Operator&	Scheduler itself

Exception Handling

None

Restriction

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

6.4 SetInput (srcOprt Has Multiple Outputs)

Function Prototype

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

Function Description

Sets the input of an operator based on the output name of **srcOprt**, when **srcOprt** has multiple outputs.

Parameter Description

Parameter	Input/Output	Type	Description
dstName	Input	const string&	Input name of the operator
srcOprt	Input	const Operator&	Input operator object whose input name is **dstName**.
name	Input	const string&	Output name of **srcOprt**

Return Value

Parameter	Type	Description
-	Operator&	Scheduler itself

Exception Handling

None

Restriction

None

6.5 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&	Input name of the operator If no operator input name is available, the default object constructed by **TensorDesc** is returned. Set **DataType** to **DT_FLOAT** (indicating the FLOAT type) and **Format** to **FORMAT_NCHW** (indicating the NCHW format).
index	Input	uint32_t	Input index of the operator If no operator input index is available, the default object constructed by **TensorDesc** is returned. Set **DataType** to **DT_FLOAT** (indicating the FLOAT type) and **Format** to **FORMAT_NCHW** (indicating the NCHW format).

Return Value

Parameter	Type	Description
TensorDesc	TensorDesc	**TensorDesc** of the operator input

Exception Handling

None

Restriction

None

6.6 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&	Input Name
tensorDesc	Output	TensorDesc&	**TensorDesc** object

Return Value

Parameter	Type	Description
-	bool	If the **TensorDesc** is obtained successfully, **true** is returned. Otherwise, **false** is returned.

Exception Handling

None

Restriction

None

6.7 UpdateInputDesc

Function Prototype

graphStatus 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&	Operator Input Name
tensorDesc	Input	const TensorDesc&	TensorDesc object

Return Value

Parameter	Type	Description
-	graphStatus	If **TensorDesc** is updated successfully, **GRAPH_SUCCESS** is returned. Otherwise, **GRAPH_FAILED** is returned.

Exception Handling

None

Restriction

None

6.8 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&	Output name of the operator

Parameter	Input/Output	Type	Description
index	Input	uint32_t	Output index of the operator If no operator output index is available, the default object constructed by **TensorDesc** is returned. Set **DataType** to **DT_FLOAT** (indicating the FLOAT type) and **Format** to **FORMAT_NCHW** (indicating the NCHW format).

Return Value

Parameter	Type	Description
-	TensorDesc	If **TensorDesc** is obtained successfully, the required **TensorDesc** object is returned. Otherwise, the default **TensorDesc** object is returned. Set **DataType** to **DT_FLOAT** (indicating the FLOAT type) and **Format** to **FORMAT_NCHW** (indicating the NCHW format).

Exception Handling

None

Restriction

None

6.9 UpdateOutputDesc

Function Prototype

graphStatus 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&	Output name of the operator
tensorDesc	Input	const TensorDesc&	TensorDesc object

Return Value

Parameter	Type	Description
-	graphStatus	If **TensorDesc** is updated successfully, **GRAPH_SUCCESS** is returned. Otherwise, **GRAPH_FAILED** is returned.

Exception Handling

None

Restriction

None

6.10 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 input name of the operator
index	Input	const unsigned int	Dynamic input index of the operator, which starts from 1

Return Value

Parameter	Type	Description
TensorDesc	TensorDesc	If **TensorDesc** is obtained successfully, the required **TensorDesc** object is returned. Otherwise, the default **TensorDesc** object is returned. Set **DataType** to **DT_FLOAT** (indicating the FLOAT type) and **Format** to **FORMAT_NCHW** (indicating the NCHW format).

Exception Handling

None

Restriction

None

6.11 UpdateDynamicInputDesc

Function Prototype

graphStatus 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 input name of the operator
index	Input	const unsigned int	Dynamic input index of the operator, which starts from 1
tensorDesc	Input	const TensorDesc&	TensorDesc object

Return Value

Parameter	Type	Description
-	graphStatus	If the dynamic input **TensorDesc** is updated successfully, **GRAPH_SUCCESS** is returned. Otherwise, **GRAPH_FAILED** is returned.

Exception Handling

None

Restriction

None

6.12 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 output name of the operator
index	Input	const unsigned int	Dynamic output index of the operator, which starts from 1

Return Value

Parameter	Type	Description
TensorDesc	TensorDesc	If **TensorDesc** is obtained successfully, the required **TensorDesc** object is returned. Otherwise, the default **TensorDesc** object is returned. Set **DataType** to **DT_FLOAT** (indicating the FLOAT type) and **Format** to **FORMAT_NCHW** (indicating the NCHW format).

Exception Handling

None

Restriction

None

6.13 UpdateDynamicOutputDesc

Function Prototype

graphStatus 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 output name of the operator
index	Input	const unsigned int	Dynamic output index of the operator
tensorDesc	Input	const TensorDesc&	**TensorDesc** object

Return Value

Parameter	Type	Description
-	graphStatus	If the dynamic output **TensorDesc** is updated successfully, **GRAPH_SUCCESS** is returned. Otherwise, **GRAPH_FAILED** is returned.

Exception Handling

None

Restriction

None

6.14 SetAttr

Function Prototype

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

Operator& SetAttr(const string& name, const UsrQuantizeFactorParams& 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
attrValue	Input	const UsrQuantizeFactorParams&	Quantization factor For details about **UsrQuantizeFactorParams**, see **Data Type Description**.

Return Value

Parameter	Type	Description
-	Operator&	Object itself

Exception Handling

None

Restriction

None

Data Type Description

- Struct **UsrQuantizeFactorParams**

```
struct UsrQuantizeFactorParams
{
    uint32_t quantize_algo{0};
    uint32_t scale_type{0};
    UsrQuantizeFactor quantize_param;
    UsrQuantizeFactor dequantize_param;
    UsrQuantizeFactor requantize_param;
    UsrQuantizeCalcFactor quantizecalc_param;
};
```

- Struct **UsrQuantizeFactor**

```
struct UsrQuantizeFactor
{
    //QuantizeScaleMode scale_mode;
    uint32_t scale_mode{0};
    std::vector<uint8_t> scale_value;
    int64_t scale_offset{0};
    std::vector<uint8_t> offset_data_value;
    int64_t offset_data_offset{0};
    std::vector<uint8_t> offset_weight_value;
    int64_t offset_weight_offset{0};
```



```
    std::vector<uint8_t> offset_pad_value;
    int64_t offset_pad_offset{0};
};
```

- Struct **UsrQuantizeCalcFactor**

```
struct UsrQuantizeCalcFactor
{
    std::vector<uint8_t> offsetw;
    int64_t offsetw_offset{0};
    std::vector<uint8_t> offsetd;
    int64_t offsetd_offset{0};
    std::vector<uint8_t> scalereq;
    int64_t scaledreq_offset{0};
    std::vector<uint8_t> offsetdnext;
    int64_t offsetdnext_offset{0};
};
```

6.15 GetAttr

Function Prototype

graphStatus GetAttr(const string& name, AttrValue& attrValue) const;

graphStatus GetAttr(const string& name, UsrQuantizeFactorParams& 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
attrValue	Input	UsrQuantizeFactorParams&	Quantization factor For details about **UsrQuantizeFactorParams**, see **Data Type Description**.

Return Value

Parameter	Type	Description
-	graphStatus	If the attribute value is obtained successfully, **GRAPH_SUCCESS** is returned. Otherwise, **GRAPH_FAILED** is returned.

Exception Handling

None

Restriction

None

7 Shape APIs

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

7.1 GetDimNum

Function Prototype

size_t GetDimNum() const;

Function Description

Obtains the dimension count of **Shape**.

Parameter Description

None

Return Value

Parameter	Type	Description
-	size_t	**Shape** dimension count of the tensor

Exception Handling

None

Restriction

None

7.2 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

Parameter	Type	Description
-	int64_t	Length of dimension **idx**

Exception Handling

None

Restriction

None

7.3 GetDims

Function Prototype

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

Function Description

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

Parameter Description

None

Return Value

Parameter	Type	Description
-	std::vector<int64_t>	Vector composed of all **Shape** dimensions

Exception Handling

None

Restriction

None

7.4 SetDim

Function Prototype

graphStatus 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 **Shape**, which starts from 0
value	Input	int64_t	Value to be set

Return Value

Parameter	Type	Description
-	graphStatus	If the operation is successful, **GRAPH_SUCCESS** is returned. Otherwise, **GRAPH_FAILED** is returned.

Exception Handling

None

Restriction

None

7.5 GetShapeSize

Function Prototype

int64_t GetShapeSize() const;

Function Description

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

Parameter Description

None

Return Value

Parameter	Type	Description
-	int64_t	Product of all dimensions

Exception Handling

None

Restriction

None

8 Tensor APIs

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

8.1 GetTensorDesc

Function Prototype

TensorDesc GetTensorDesc() const;

Function Description

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

Parameter Description

None

Return Value

Parameter	Type	Description
-	TensorDesc	Tensor descriptor

Exception Handling

None

Restriction

None

8.2 MutableTensorDesc

Function Prototype

TensorDesc& MutableTensorDesc();

Function Description

Obtains the descriptor of a tensor (mutable).

Parameter Description

None

Return Value

Parameter	Type	Description
-	TensorDesc&	Tensor descriptor (mutable)

Exception Handling

None

Restriction

None

8.3 SetTensorDesc

Function Prototype

graphStatus 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

Parameter	Type	Description
-	graphStatus	If the operation is successful, **GRAPH_SUCCESS** is returned. Otherwise, **GRAPH_FAILED** is returned.

Exception Handling

None

Restriction

None

8.4 GetData

Function Prototype

const Buffer GetData() const;

Function Description

Obtains the data of a tensor.

Parameter Description

None

Return Value

Parameter	Type	Description
-	const Buffer	Tensor data

Exception Handling

None

Restriction

None

8.5 MutableData

Function Prototype

Buffer MutableData();

Function Description

Obtains the data of a tensor.

Parameter Description

None

Return Value

Parameter	Type	Description
-	Buffer	Tensor data

Exception Handling

None

Restriction

None

8.6 SetData

Function Prototype

graphStatus SetData(std::vector<uint8_t> &&data);

graphStatus SetData(const std::vector<uint8_t> &data);

graphStatus SetData(const Buffer &data);

graphStatus 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

Parameter	Type	Description
-	graphStatus	If the operation is successful, **GRAPH_SUCCESS** is returned. Otherwise, **GRAPH_FAILED** is returned.

Exception Handling

None

Restriction

None

8.7 Clone

Function Prototype

Tensor Clone() const;

Function Description

Clones a tensor.

Parameter Description

None

Return Value

Parameter	Type	Description
-	Tensor	Cloned tensor object

Exception Handling

None

Restriction

None

9 TensorDesc APIs

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

9.1 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	**shape** object to be updated
format	Input	Format	**format** object to be updated. The default value is **FORMAT_NCHW**. For the definition of the Format data types, see **Format**.
dt	Input	DataType	**datatype** object to be updated. The default value is **DT_FLOAT**. For the definition of the **DataType** data types, see **DataType**.

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 */
    FORMAT_NC1HWC0,          /**< NC1HWC0 */
    FORMAT_FRACTAL_Z,        /**< FRACTAL_Z */
    FORMAT_NC1C0HWPAD,
    FORMAT_NHWC1C0,
    FORMAT_FSR_NCHW,
    FORMAT_FRACTAL_DECONV,
    FORMAT_C1HWNC0,
    FORMAT_FRACTAL_DECONV_TRANSPOSE,
    FORMAT_FRACTAL_DECONV_SP_STRIDE_TRANS,
    FORMAT_NC1HWC0_C04,  /**< NC1HWC0, C0 =4*/
    FORMAT_FRACTAL_Z_C04,/**< FRACZ, C0 = 4 */
    FORMAT_CHWN,
    FORMAT_FRACTAL_DECONV_SP_STRIDE8_TRANS,
    FORMAT_HWCN,
    FORMAT_NC1KHKWHWC0, /** < KH,KW kernel h& kernel w maxpooling max output format*/
    FORMAT_BN_WEIGHT,
    FORMAT_FILTER_HWCK,     /* filter input tensor format */
    FORMAT_HASHTABLE_LOOKUP_LOOKUPS=20,
    FORMAT_HASHTABLE_LOOKUP_KEYS,
    FORMAT_HASHTABLE_LOOKUP_VALUE,
```



```
    FORMAT_HASHTABLE_LOOKUP_OUTPUT,
    FORMAT_HASHTABLE_LOOKUP_HITS=24,
    FORMAT_RESERVED

};
```

- DataType

```
enum DataType {
    DT_UNDEFINED = 16,   // Used to indicate a DataType field has not been set.
    DT_FLOAT = 0,           // float type
    DT_FLOAT16 = 1,         // fp16 type
    DT_INT8 = 2,            // int8 type
    DT_INT16 = 6,     // int16 type
    DT_UINT16 = 7,      // uint16 type
    DT_UINT8 = 4,           // uint8 type
    DT_INT32 = 3,           //
    DT_INT64 = 9,           // int64 type
    DT_UINT32 = 8,          // unsigned int32
    DT_UINT64 = 10,         // unsigned int64
    DT_BOOL = 12,           // bool type
    DT_DOUBLE = 11,         // double type
    DT_DUAL = 13,           /**< dual output type */
    DT_DUAL_SUB_INT8 = 14,    /**< dual output int8 type */
    DT_DUAL_SUB_UINT8 = 15,    /**< dual output uint8 type */
};
```

9.2 GetShape

Function Prototype

Shape GetShape() const;

Function Description

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

Parameter Description

None

Return Value

Parameter	Type	Description
-	Shape	**shape** described by **TensorDesc**

Exception Handling

None

Restriction

shape returned is constant and is immutable.

9.3 MutableShape

Function Prototype

Shape& MutableShape();

Function Description

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

Parameter Description

None

Return Value

Parameter	Type	Description
-	Shape &	**shape** reference (mutable) in **TensorDesc**

Exception Handling

None

Restriction

None

9.4 SetShape

Function Prototype

void SetShape(Shape shape);

Function Description

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

Parameter Description

Parameter	Input/Output	Type	Description
shape	Input	Shape	**shape** object to be set to **TensorDesc**

Return Value

None

Exception Handling

None

Restriction

None

9.5 GetFormat

Function Prototype

Format GetFormat() const;

Function Description

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

Parameter Description

None

Return Value

Parameter	Type	Description
-	Format	**format** information of the tensor described by **TensorDesc**

Exception Handling

None

Restriction

None

9.6 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	**format** information to be set

Return Value

None

Exception Handling

None

Restriction

None

9.7 GetDataType

Function Prototype

DataType GetDataType() const;

Function Description

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

Parameter Description

None

Return Value

Parameter	Type	Description
-	DataType	Data type of the tensor described by **TensorDesc**

Exception Handling

None

Restriction

None

9.8 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	**dt** information to be set For details, see **DataType**.

Return Value

None

Exception Handling

None

Restriction

None

9.9 Clone

Function Prototype

TensorDesc Clone() const;

Function Description

Clones **TensorDesc**.

Parameter Description

None

Return Value

Parameter	Type	Description
-	TensorDesc	Cloned **TensorDesc** object

Exception Handling

None

Restriction

None

9.10 IsValid

Function Prototype

graphStatus IsValid();

Function Description

Checks whether a tensor object is valid.

Parameter Description

None

Return Value

Parameter	Type	Description
-	graphStatus	If the tensor object is valid, **GRAPH_SUCCESS** is returned. Otherwise, **GRAPH_FAILED** is returned.

Exception Handling

None

Restriction

None

10 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.

Instance

```
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})
.INFER_SHAPE_AND_TYPE(FullConnectionInfer)
.ATTR_ALL_VERIFY(FullConnectionVerify)
.OP_END()
```

The operator registration APIs are defined in **operator_reg.h**. For details about registered operators and their header files, see **10.12 List of Built-in Operators**.

10.1 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 *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.

10.2 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

Parameter	Input/Output	Type	Description
default_value	Input	-	Value of an operator attribute. The default value varies depending on the attribute type. The following attribute types are supported: • **AttrValue::INT**: The attribute type is int64_t. • **AttrValue::FLOAT**, The attribute type is float. • **AttrValue::STR**: The attribute type is string. • **AttrValue::BOOL**: The attribute type is bool. • **AttrValue::TENSOR**: The attribute type is tensor. • **AttrValue::LIST_INT**: The attribute type is vector<int64_t> (int64_t list). • **AttrValue::LIST_FLOAT**: The attribute type is vector<float> (float list). • **AttrValue::LIST_STR**: The attribute type is vector<string> (string list). • **AttrValue::LIST_BOOL**: The attribute type is vector<bool> (bool list). • **AttrValue::LIST_TENSOR**: The attribute type is vector<Tensor> (tensor list).

Return Value

None

Exception Handling

None

Restriction

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

10.3 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

Parameter	Input/Output	Type	Description
type	Input	-	The following attribute types are supported: • **AttrValue::INT**: The attribute type is int64_t. • **AttrValue::FLOAT**, The attribute type is float. • **AttrValue::STR**: The attribute type is string. • **AttrValue::BOOL**: The attribute type is bool. • **AttrValue::TENSOR**: The attribute type is tensor. • **AttrValue::LIST_INT**: The attribute type is vector<int64_t> (int64_t list). • **AttrValue::LIST_FLOAT**: The attribute type is vector<float> (float list). • **AttrValue::LIST_STR**: The attribute type is vector<string> (string list). • **AttrValue::LIST_BOOL**: The attribute type is vector<bool> (bool list). • **AttrValue::LIST_TENSOR**: The attribute type is vector<Tensor> (tensor list).

Return Value

None

Exception Handling

None

Restriction

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

10.4 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.
graphStatus update_input_desc_x(const TensorDesc& tensorDesc);// Sets the description of input x, including Shape, DataType, and Format. graphStatus 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 **TensorType** are supported. Separate data types with commas (,). For example: TensorType{DT_FLOAT} TensorType({DT_FLOAT, DT_INT8} For details about class **TensorType**, see **Description of Class TensorType** .

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.

10.5 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.
graphStatus 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 **TensorType** are supported. Separate data types with commas (,). For example: TensorType{DT_FLOAT} TensorType({DT_FLOAT, DT_INT8}

Return Value

None

Exception Handling

None

Restriction

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

10.6 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.
```


graphStatus 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 **TensorType** are supported. Separate data types with commas (,). For example: TensorType{DT_FLOAT} TensorType({DT_FLOAT, DT_INT8}

Return Value

None

Exception Handling

None

Restriction

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

10.7 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.
graphStatus 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 **TensorType** are supported. Separate data types with commas (,). For example: TensorType{DT_FLOAT} TensorType({DT_FLOAT, DT_INT8}

Return Value

None

Exception Handling

None

Restriction

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

10.8 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.
graphStatus 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 **TensorType** are supported. Separate data types with commas (,). For example: TensorType{DT_FLOAT} TensorType({DT_FLOAT, DT_INT8}

Return Value

None

Exception Handling

None

Restriction

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

10.9 INFER_SHAPE_AND_TYPE

Function Prototype

INFER_SHAPE_AND_TYPE (x)

Function Description

Registers the **Shape** and **DataType** inference function.

Parameter Description

Parameter	Input/Output	Type	Description
x	Input	-	Macro parameter, **Shape** and **DataType** inference function For example, **INFER_SHAPE_AND_TYPE(FullConnectionInfer)** is used to register the **FullConnectionInfer** function for inferring operator **Shape** and **DataType**. **FullConnectionInfer** is declared in **DECLARE_INFERFUNC** and defined in **IMPLEMT_INFERFUNC**. For details, see **Description of Macros DECLARE_INFERFUNC and IMPLEMT_INFERFUNC**.

Return Value

None

Exception Handling

None

Restriction

None

Description of Macros DECLARE_INFERFUNC and IMPLEMT_INFERFUNC

Before registering the function for **Shape** and **DataType** inference, you need to declare the function in the **DECLARE_INFERFUNC** macro and define the function in the **IMPLEMT_INFERFUNC** macro.

- Declaring the function
  ```
  DECLARE_INFERFUNC(FullConnection, FullConnectionInfer)
  ```
  The expanded implementation of the **DECLARE_INFERFUNC** macro is as follows:
  ```
  namespace op {
  class FullConnection;
  }
  static graphStatus FullConnectionInfer(op::FullConnection& op);
  ```
- Defining the function
  ```
  IMPLEMT_INFERFUNC(FullConnection, FullConnectionInfer) {
  // Implementation details
  }
  ```


The expanded implementation of the **IMPLEMT_INFERFUNC** macro is as follows:

```
static graphStatus FullConnectionInfer(op::FullConnection& op){
// Implementation details
}
```

10.10 ATTR_ALL_VERIFY

Function Prototype

ATTR_ALL_VERIFY (x)

Function Description

Registers the operator verification function.

Parameter Description

Parameter	Input/Output	Type	Description
x	Input	-	Macro parameter, operator verification function For example, call **ATTR_ALL_VERIFY(FullConnectionVerify)** to register the operator verification function **FullConnectionVerify**. **FullConnectionVerify** is declared in **DECLARE_VERIFIER** and defined in **IMPLEMT_VERIFIER**. For details, see **Description of Macros DECLARE_VERIFIER and IMPLEMT_VERIFIER**.

Return Value

None

Exception Handling

None

Restriction

None

Description of Macros DECLARE_VERIFIER and IMPLEMT_VERIFIER

Before registering the operator verification function, you need to declare the function in the DECLARE_VERIFIER macro and define the function in the IMPLEMT_VERIFIER macro.

- Declaring the function
  ```
  DECLARE_VERIFIER(FullConnection, FullConnectionVerify)
  ```
  The expanded implementation of the **DECLARE_VERIFIER** macro is as follows:
  ```
  namespace op {
  class FullConnection;
  }
  static graphStatus FullConnectionVerify(op::FullConnection op);
  ```
- Defining the function
  ```
  IMPLEMT_VERIFIER(FullConnection, FullConnectionVerify) {
  // Implementation details
  }
  ```
  The expanded implementation of the **IMPLEMT_VERIFIER** macro is as follows:
  ```
  static graphStatus FullConnectionVerify(op::FullConnection op){
  // Implementation details
  }
  ```

10.11 OP_END

Function Prototype

OP_END ()

Function Description

Ends operator registration.

Parameter Description

None

Return Value

None

Exception Handling

None

Restriction

None

10.12 List of Built-in Operators

You can check the built-in operators in the header files for operator registration, and set the operator input, output, and attribute during operator registration.

Table 10-1 Operator list

Operator Name	Header File
Data	array_defs.h
Concat	
Flatten	
Reshape	
Split	
Const	const_defs.h
Permute	detection_defs.h
Add	math_defs.h
Mul	
Activation	nn_defs.h
BatchNorm	
Convolution	
Eltwise	
LRN	
ConvolutionDepthwise	
FullConnection	
Pooling	
Scale	
ShuffleChannel	
Softmax	

11 Model Building APIs

The API calling sequence during model building is as follows:

CreateModelBuff > BuildIRModel > ReleaseModelBuff

11.1 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

Parameter	Type	Description
-	bool	• **true**: The model buffer is successfully created. • **false**: The model buffer fails to be created.

Exception Handling

None

Restriction

None

11.2 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

Parameter	Type	Description
-	bool	• **true**: The model is successfully created. • **false**: The model fails to be created.

Exception Handling

None

Restriction

None

11.3 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

12 Appendix

12.1 Change History

Release Date	Description
2020-05-30	This issue is the first official release.

