Framework 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

This document describes the internal APIs (no need to be called by the user) of the Framework on which the compilation of custom operators depends and the APIs involved in the development of custom operator plug-ins. The APIs include non-class member functions and class member functions. In addition, this document describes the macros related to the APIs.

For details about how to use these APIs to develop custom operator plug-ins and compile and run code, see *TE Custom Operator Development Guide*.

You can view the API definition files in **ddk/include/inc** in the installation directory of the device development kit (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**. For details about the mapping between the definition files and the APIs, see the API description.

2 Non-Class Member Functions

2.1 custom_op_run

Function

Entry function for the commissioning of a single custom operator. This function is defined in **custom\custom_op.h**.

The return value of **custom_op_run** is of the **ErrorInfo** type, including the error code and error description. If the error code is **0**, the operation is successful. Other values indicate failure. The **ErrorInfo** type is defined as follows:

```
struct ErrorInfo
  {
     uint32_t    error_code;
     std::string error_msg;
  };
```

Syntax

ErrorInfo custom_op_run(const std::string& name, int32_t type, const std::string& bin_file,

const std::vector< std::string >& in_files,

const std::vector< std::string >& out_files,

const std::vector< uint32_t >& out_buf_sizes,

const std::vector< uint32_t >& workspace_sizes = std::vector<uint32_t>(),

const std::string& op_cfg_file = "", void *param = nullptr, int param_len = 0);

Parameter Description

Parameter	Input/ Output	Description
name	Input	Operator kernel name
type	Input	Operator type: ● 0 - TE_AICORE ● 1 -TE_AICPU ● 2 - AI CPU
bin_file	Input	Binary file path of an operator, including the file name. A relative path is supported. In this case, the binary file must be stored in the directory where the program is located, for example **./*.bin**.
in_files	Input	Path of the input data file, including the file name. A relative path is supported. In this case, the data file must be stored in the directory where the program is located.
out_files	Input	Path of the output data file, including the file name. A relative path is supported. In this case, the data file must be stored in the directory where the program is located.
out_buf_sizes	Input	Size of the output data, in bytes
workspace_sizes	Input	Size of the workspace, in bytes
op_cfg_file	Input	Input and output description of the operator specified by the user
param	Input	Address information of the parameter structure of the operator specified by the user
param_len	Input	Length of the parameter structure of the operator specified by the user, in bytes. The structure length can be obtained using **sizeof(param)**.

2.2 custom_op_compare

Function

Verification function of a single custom operator, which is used to verify data precision. This function is defined in **custom\custom_op.h**.

The return value of **custom_op_compare** is of the **ErrorInfo** type, including the error code and error description. If the error code is **0**, the operation is successful. Other values indicate failure. The **ErrorInfo** type is defined as follows:

```
struct ErrorInfo
   {
      uint32_t    error_code;
      std::string error_msg;
   };
```

Syntax

ErrorInfo custom_op_compare(const std::string& expect_file, const std::string& actual_file,

int32_t data_type, float precision_deviation,

float statistical_discrepancy, bool& compare_result);

Parameter Description

Parameter	Input/ Output	Description
expect_file	Input	Expected data file, including the file name
actual_file	Input	Actual data file, including the file name
data_type	Input	Operator type. Currently, FP32 (**0**) and FP16 (**1**) are supported.
precision_deviation	Input	Precision deviation of single data. The value range is (0, 1). A smaller deviation value indicates higher precision.
statistical_discrepancy	Input	Statistical deviation of the entire dataset. The value range is (0, 1). A smaller deviation value indicates higher precision.
compare_result	Output	Comparison result. **true**: The comparison is successful. **false**: The comparison failed.

2.3 GetContext

Function

Obtains the OMG context and returns it. This function is defined in **framework\omg\omg_types.h**.

The return value of the **GetContext** function is of the **OmgContext** type. For details, see **framework\omg\omg_types.h**.

Syntax

```
OmgContext& GetContext();
```

Parameter Description

None

2.4 AutoMappingFn

Function

Automatic mapping callback function. This function is defined in **framework\omg\register.h**.

Syntax

```
Status AutoMappingFn(const google::protobuf::Message* op_src, ge::Operator& op);
```

Parameter Description

Parameter	Input/ Output	Description
op_src	Input	Original operator before conversion
op	Input	Mapped operator

2.5 aicpu_run_func

Function

Pointer to the function of an AI CPU custom operator. The function name can be customized. This function is defined in **cce\customize.h**.

Syntax

```
void (*aicpu_run_func)(opTensor_t **, void **, int32_t,
opTensor_t **, void **, int32_t, void *, rtStream_t);
```

Parameter Description

Parameter	Input/ Output	Description

opTensor_t **	Input	Input descriptor tensor.
		typedef struct tagOpTensor
		{
		// real dim info
		opTensorFormat_t format;
		//char align1[4];
		opDataType_t data_type;
		//char align2[4];
		int32_t dim_cnt;
		int32_t mm; //lint !e148
		int32_t dim[CC_DEVICE_DIM_MAX]; //lint !e148
		} opTensor_t;

void **	Input	Address of the input descriptor tensor
int32_t	Input	Number of input descriptors
opTensor_t **	Input	Output descriptor tensor
void **	Input	Address of the output descriptor tensor
int32_t	Input	Number of output descriptors
void *	Input	Attribute handle address
rtStream_t	Input	Stream identifier

3 Class Member Functions

3.1 Class StatusFactory

This class of functions is defined in **custom\common\fmk_error_codes.h**.

3.1.1 Instance

Function

Returns the instance object of the status factory class.

Syntax

static StatusFactory* Instance();

Parameter Description

None

3.1.2 RegisterErrorNo

Function

Registers error codes.

Syntax

void RegisterErrorNo(uint32_t err, const std::string& desc);

Parameter Description

Parameter	Input/ Output	Description
err	Input	Error code
desc	Input	Description of an error code, which is a character string

3.1.3 GetErrDesc

Function

Obtains the error code description.

Syntax

std::string GetErrDesc(uint32_t err);

Parameter Description

Parameter	Input/ Output	Description
err	Input	Error code

3.1.4 StatusFactory

Function

Constructor function

Syntax

StatusFactory();

Parameter Description

None

3.1.5 ~StatusFactory

Function

Destructor function

Syntax

~StatusFactory();

Parameter Description

None

3.2 Class ErrorNoRegisterar

This class of functions is defined in **custom\common\fmk_error_codes.h**.

3.2.1 ErrorNoRegisterar

Function

Constructor function

Syntax

ErrorNoRegisterar(uint32_t err, const std::string& desc);

Parameter Description

Parameter	Input/ Output	Description
err	Input	Error code
desc	Input	Description of an error code, which is a character string

3.2.2 ~ErrorNoRegisterar

Function

Destructor function

Syntax

~ErrorNoRegisterar();

Parameter Description

None

3.3 Class OpTypeContainer

This class of functions is defined in **custom\common\op_types.h**.

3.3.1 Instance

Function

Obtains the instance object of the **OpTypeContainer** factory class.

Syntax

static OpTypeContainer * Instance();

Parameter Description

None

3.3.2 Register

Function

Saves the operator type to a custom set.

Syntax

void Register(const std::string& op_type);

Parameter Description

Parameter	Input/ Output	Description
op_type	Input	Operator type to be saved

3.3.3 IsExisting

Function

Checks whether an operator type exists.

Syntax

bool IsExisting(const std::string& op_type);

Parameter Description

Parameter	Input/ Output	Description
op_type	Input	Operator type to be queried

3.3.4 OpTypeContainer

Function

Constructor function

Syntax

OpTypeContainer();

Parameter Description

None

3.4 Class OpTypeRegistrar

This class of functions is defined in **custom\common\op_types.h**.

3.4.1 OpTypeRegistrar

Function

Constructor function, which registers the type of an operator with the container

Syntax

OpTypeRegistrar(const std::string& op_type);

Parameter Description

Parameter	Input/ Output	Description
op_type	Input	Type of an operator to be registered

3.4.2 ~OpTypeRegistrar

Function

Destructor function

Syntax

~OpTypeRegistrar();

Parameter Description

None

3.5 Class OpRegistrationData

This class of functions is defined in **framework\omg\register.h**.

3.5.1 OpRegistrationData

Function

Constructor function

Syntax

OpRegistrationData(const std::string& om_optype);

Parameter Description

Parameter	Input/ Output	Description
om_optype	Input	Operator type

3.5.2 ~OpRegistrationData

Function

Destructor function

Syntax

~OpRegistrationData();

Parameter Description

None

3.5.3 FrameworkType

Function

Sets the framework type.

Syntax

OpRegistrationData& FrameworkType(domi::FrameworkType fmk_type);

Parameter Description

Parameter	Input/ Output	Description
fmk_type	Input	Framework type 0: Caffe 3: TensorFlow

3.5.4 OriginOpType

Function

Sets the operator type or operator type list of the original model.

Syntax

- Setting the operator type list:

  OpRegistrationData& OriginOpType (const std::initializer_list<std::string> & ori_optype_list);
- Setting the operator type:

  OpRegistrationData& OriginOpType (const std::string& ori_optype);

Parameter Description

Parameter	Input/ Output	Description
ori_optype_list	Input	Operator type list of the original model
ori_optype	Input	Operator type of the original model

3.5.5 ParseParamsFn

Function

Parses parameters.

Syntax

OpRegistrationData& ParseParamsFn(ParseParamFunc parseParamFn);

Parameter Description

Parameter	Input/ Output	Description
parseParamFn	Input	Callback function ParseParamFunc. For details, see **Callback Function ParseParamFunc**.

Callback Function ParseParamFunc

You can customize and implement the ParseParamFunc class functions to convert the parameters and weights of the Caffe model and fill the results in the Operator class.

Syntax

Status ParseParamFunc(const Message* op_origin, ge::Operator& op_dest);

Parameter Description

Parameter	Input/ Output	Description
op_origin	Input	Data structure in protobuf format (from the prototxt file of the Caffe model), including operator parameter information
op_dest	Output	Operator data structure of offline model supported by the Ascend AI processor, which stores operator information For details about the operator class, see **Operator Class APIs** in *GE API Reference*.

3.5.6 InferShapeAndTypeFn

Function

Shape inference function

Syntax

OpRegistrationData& InferShapeAndTypeFn(InferShapeFunc inferShapeFn);

Parameter Description

Parameter	Input/ Output	Description
inferShapeFn	Input	Callback function InferShapeFunc. For details, see **Callback Function InferShapeFunc**.

Callback Function InferShapeFunc

You can customize and implement the InferShapeFunc class function to obtain the output description of an operator, including the tensor description such as the output shape information and data type.

Syntax

Status InferShapeFunc(const ge::Operator& op, vector<ge::TensorDesc>& v_output_desc);

Parameter Description

Parameter	Input/Output	Description
op	Input	Operator data structure of offline model supported by the Ascend AI processor For details about the operator class, see **Operator Class APIs** in *GE API Reference*.
v_output_desc	Output	Operator output description For details about the TensorDesc class, see **TensorDesc Class APIs** in *GE API Reference*.

3.5.7 UpdateOpDescFn

Function

Operator description update function

Syntax

OpRegistrationData& UpdateOpDescFn(UpdateOpDescFunc updateOpDescFn);

Parameter Description

Parameter	Input/ Output	Description
updateOpDescFn	Input	Callback function UpdateOpDescFunc. For details, see **Callback Function UpdateOpDescFunc**.

Callback Function UpdateOpDescFunc

You can customize and implement the UpdateOpDescFunc class function to update operator information.

Syntax

static Status UpdateOpDescFunc(ge::Operator& op);

Parameter Description

Parameter	Input/ Output	Description
op	Input/ Output	Operator data structure of offline model supported by the Ascend AI processor, which stores operator information For details about the operator class, see **Operator Class APIs** in *GE API Reference*.

3.5.8 GetWorkspaceSizeFn

Function

Obtains the size of workspace.

Syntax

OpRegistrationData& GetWorkspaceSizeFn(GetWorkspaceSizeFunc getWorkspaceSizeFn);

Parameter Description

Parameter	Input/ Output	Description
getWorkspaceSizeFn	Input	Callback function GetWorkspaceSizeFunc. For details, see **Callback Function GetWorkspaceSizeFunc**.

Callback Function GetWorkspaceSizeFunc

You can customize and implement GetWorkspaceSizeFunc class function to calculate the workspace size of an operator and export the calculation result.

Syntax

Status GetWorkspaceSizeFn(domi::Status (const ge::Operator& op, std::vector<int64_t>& size));

Parameter Description

Parameter	Input/ Output	Description
op	Input	Operator object whose workspace needs to be calculated For details about the operator class, see **Operator Class APIs** in *GE API Reference*.
size	Output	Size of the workspace. The value needs to be calculated and provided by the user.

3.5.9 TEBinBuildFn

Function

Build callback function

Syntax

OpRegistrationData& TEBinBuildFn(BuildTeBinFunc buildTeBinFn);

Parameter Description

Parameter	Input/ Output	Description
buildTeBinFn	Input	Callback function BuildTeBinFunc. For details, see **Callback Function BuildTeBinFunc**.

Callback Function BuildTeBinFunc

You can customize and implement the BuildTeBinFunc class function to construct the operator binary file.

Syntax

virtual Status BuildTeBinFunc(const ge::Operator& op, TEBinInfo& teBinInfo);

Parameter Description

Parameter	Input/ Output	Description
op	Input	Operator data structure of offline model supported by the Ascend AI processor, which stores operator information For details about the operator class, see **Operator Class APIs** in *GE API Reference*.

teBinInfo	Output	Path of the binary file of a custom operator and DDK description
		struct TEBinInfo
		{
		std::string bin_file_path; // Automatically obtained from the **binFileName** field in the JSON file. To ensure compatibility with cases written by users, the field is not deleted.
		std::string json_file_path;
		std::string ddk_version;
		};

3.5.10 ImplyType

Function

Sets the operator execution mode.

Syntax

OpRegistrationData& ImplyType(domi::ImplyType imply_type);

Parameter Description

Parameter	Input/ Output	Description

imply_type	Input	Operator execution mode.
		enum class ImplyType : unsigned int
		{
		BUILDIN = 0,// Built-in operator, which is normally executed by the OME
		TVM, // Executed after being compiled into a TVM binary file
		CUSTOM, // The calculation logic is customized by the user and executed by the CPU.
		AI_CPU, // AI CPU custom operator type
		INVALID = 0xFFFFFFFF,
		};

3.5.11 Formats

Function

Sets the input/output data formats supported by an operator.

Syntax

```
OpRegistrationData& Formats(
const std::initializer_list<domi::tagDomiTensorFormat>& input_formats,
const std::initializer_list<domi::tagDomiTensorFormat>& output_formats);
OpRegistrationData& Formats(
const domi::tagDomiTensorFormat& input_format,
const domi::tagDomiTensorFormat& output_format);
```

Parameter Description

Parameter	Input/ Output	Description
input_format	Input	Data format supported by the input
output_format	Input	Data format supported by the output

3.5.12 WeightFormats

Function

Sets the data formats supported by the operator weight.

Syntax

OpRegistrationData& WeightFormats(

const std::initializer_list<domi::tagDomiTensorFormat>& weight_formats);

Parameter Description

Parameter	Input/ Output	Description
weight_formats	Input	Data formats supported by the weight: typedef enum tagDomiTensorFormat { DOMI_TENSOR_NCHW = 0, /**< NCHW */ DOMI_TENSOR_NHWC, /**< NHWC */ DOMI_TENSOR_ND, /**< Nd Tensor */ DOMI_TENSOR_NC1HWC0, /**< NC1HWC0 */ DOMI_TENSOR_FRACTAL_Z, /**< FRACTAL_Z */ DOMI_TENSOR_NC1C0HWPAD, DOMI_TENSOR_NHWC1C0, DOMI_TENSOR_FSR_NCHW, DOMI_TENSOR_FRACTAL_DECONV, DOMI_TENSOR_BN_WEIGHT, DOMI_TENSOR_CHWN, /*Android NN Depth CONV*/ DOMI_TENSOR_FILTER_HWCK, /* filter input tensor format */ DOMI_TENSOR_RESERVED } domiTensorFormat_t;

3.5.13 Finalize

Function

Creates the parser and builder functions of all supported operators. If the creation is successful, **true** is returned. Otherwise, **false** is returned.

Syntax

bool Finalize();

Parameter Description

None

3.6 Class OpRegistry

This class of functions is defined in **framework\omg\register.h**.

3.6.1 Instance

Function

Obtains the instance of an object of the **OpRegistry** class.

Syntax

static OpRegistry* Instance();

Parameter Description

None

3.6.2 Register

Function

Registers **OpRegistrationData**. If the registration is successful, **true** is returned. Otherwise, **false** is returned.

Syntax

bool Register(const OpRegistrationData& reg_data);

Parameter Description

Parameter	Input/ Output	Description
reg_data	Input	Operator registration information

3.6.3 GetImplyType

Function

Obtains the operator execution type.

Syntax

domi::ImplyType GetImplyType(const std::string& op_type);

Parameter Description

Parameter	Input/ Output	Description
op_type	Input	Operator type to be queried

3.6.4 GetOpTypeByImplyType

Function

Queries the names of all operators of a specified execution type.

Syntax

void GetOpTypeByImplyType(std::vector<std::string>& vec_op_type,const domi::ImplyType& imply_type);

Parameter Description

Parameter	Input/ Output	Description
vec_op_type	Output	Operator name information to be saved and queried
imply_type	Input	Operator execution type

3.6.5 GetFormats

Function

Queries the input and output data types supported by the **op_type** operator.

Syntax

void GetFormats(const std::string& op_type, std::vector<domi::tagDomiTensorFormat>& input_format_vector,std::vector<domi::tagDomiTensorFormat>& output_format_vector);

Parameter Description

Parameter	Input/ Output	Description
op_type	Input	Name of an operator type
input_format_vector	Output	Supported input data types
output_format_vector	Output	Supported output data types

3.6.6 GetWeightFormats

Function

Queries the weight data type supported by the **op_type** operator.

Syntax

void GetWeightFormats(const std::string& op_type,

std::vector<domi::tagDomiTensorFormat>& format_vector);

Parameter Description

Parameter	Input/ Output	Description
op_type	Input	Name of an operator type
format_vector	Output	Supported weight data types

3.6.7 GetParseParamFunc

Function

Obtains the callback function for parsing parameters.

Syntax

domi::ParseParamFunc GetParseParamFunc(const std::string& op_type);

Parameter Description

Parameter	Input/ Output	Description
op_type	Input	Operator type

3.6.8 GetInferShapeFunc

Function

Obtains the inference shape callback function.

Syntax

domi::InferShapeFunc GetInferShapeFunc(const std::string& op_type);

Parameter Description

Parameter	Input/ Output	Description
op_type	Input	Operator type

3.6.9 GetGetWorkspaceSizeFunc

Function

Obtains the callback function for the workspace size.

Syntax

domi::GetWorkspaceSizeFunc GetGetWorkspaceSizeFunc(const std::string& op_type);

Parameter Description

Parameter	Input/ Output	Description
op_type	Input	Operator type

3.6.10 GetUpdateOpDescFunc

Function

Obtains the callback function for updating the operator description.

Syntax

domi::UpdateOpDescFunc GetUpdateOpDescFunc(const std::string& op_type);

Parameter Description

Parameter	Input/ Output	Description
op_type	Input	Operator type

3.6.11 GetBuildTeBinFunc

Function

Obtains the **build** callback function.

Syntax

domi::BuildTeBinFunc GetBuildTeBinFunc(const std::string& op_type);

Parameter Description

Parameter	Input/ Output	Description
op_type	Input	Operator type

3.6.12 GetTransWeightFunc

Function

Obtains the callback function for transferring the weight.

Syntax

domi::TransWeightFunc GetTransWeightFunc(const std::string& op_type);

Parameter Description

Parameter	Input/ Output	Description
op_type	Input	Operator type

3.7 Class OpReceiver

This class of functions is defined in **framework\omg\register.h**.

3.7.1 OpReceiver

Function

Constructor function

Syntax

OpReceiver(OpRegistrationData& reg_data);

Parameter Description

Parameter	Input/ Output	Description
reg_data	Input	Operator information to be registered

3.7.2 ~OpReceiver

Function

Destructor function

Syntax

~OpReceiver();

Parameter Description

None

4 Macro Definitions

4.1 Macro for Generating Error Codes

Function

Generates the error codes of a specified module by calling **DEF_ERRORNO**.

Syntax

DEF_ERRORNO(sysid, modid, name, value, desc)

Parameter Description

Parameter	Input/ Output	Description
sysid	Input	System ID
modid	Input	Module ID

Parameter	Input/ Output	Description
name	Input	Name of an error code
value	Input	Actual value of an error code
desc	Input	Description of an error code, which is a character string

4.2 Macro for Generating Error Codes of the Common Module

Function

Generates the error codes of the common module by calling **DEF_ERRORNO_COMMON**.

Syntax

DEF_ERRORNO_COMMON(name, value, desc)

Parameter Description

Parameter	Input/ Output	Description
name	Input	Name of an error code
value	Input	Actual value of an error code
desc	Input	Description of an error code, which is a character string

4.3 Macro for Generating Error Codes of the OMG Module

Function

Generates the error codes of the OMG module by calling **DEF_ERRORNO_OMG**.

Syntax

DEF_ERRORNO_OMG(name, value,desc)

Parameter Description

Parameter	Input/ Output	Description
name	Input	Name of an error code
value	Input	Actual value of an error code
desc	Input	Description of an error code, which is a character string

4.4 Macro for Generating Error Codes of the OME Module

Function

Generates the error codes of the OME module by calling **DEF_ERRORNO_OME**.

Syntax

DEF_ERRORNO_OME(name, value,desc)

Parameter Description

Parameter	Input/ Output	Description
name	Input	Name of an error code
value	Input	Actual value of an error code
desc	Input	Description of an error code, which is a character string

4.5 Macro for Generating Error Codes of the CALIBRATION Module

Function

Generates the error codes of the CALIBRATION module by calling **DEF_ERRORNO_CALIBRATION**.

Syntax

DEF_ERRORNO_CALIBRATION(name, value, desc)

Parameter Description

Parameter	Input/ Output	Description
name	Input	Name of an error code
value	Input	Actual value of an error code
desc	Input	Description of an error code, which is a character string

4.6 Macro for Obtaining Error Code Description.

Function

Obtains the error code description by calling **GET_ERRORNO_STR**.

Syntax

GET_ERRORNO_STR(value)

Parameter Description

Parameter	Input/ Output	Description
value	Input	Value of an error code

4.7 Macro for Registering the Operator Type

Function

Register an operator type, mainly to declare a constant string, which is the name of the operator type, and then save the string to a set.

Syntax

REGISTER_OPTYPE_DECLARE(var_name, str_name)

REGISTER_OPTYPE_DEFINE(var_name, str_name)

Parameter Description

Parameter	Input/ Output	Description
var_name	Input	Name of a constant string
str_name	Input	Name of an operator type

4.8 Macro Indicating Whether an Operator Type Exists

Function

Checks whether an operator type exists.

Syntax

IS_OPTYPE_EXISTING(str_name)

Parameter Description

Parameter	Input/ Output	Description
str_name	Input	Name of an operator type

4.9 Macro for Registering an Operator

4.9.1 REGISTER_CUSTOM_OP

Function

Registers an operator with a specified name.

Syntax

REGISTER_CUSTOM_OP(name)

Parameter Description

Parameter	Input/ Output	Description
name	Input	Operator name

4.10 Macros for Marking External APIs

4.10.1 FMK_FUNC_HOST_VISIBILITY

Function

Marks APIs exposed to external use on the host, so that the APIs are visible and can be called in the dynamic link library.

Syntax

FMK_FUNC_HOST_VISIBILITY

Parameter Description

None

4.10.2 FMK_FUNC_DEV_VISIBILITY

Function

Marks APIs exposed to external use on the device, so that the APIs are visible and can be called in the dynamic link library.

Syntax

FMK_FUNC_DEV_VISIBILITY

Parameter Description

None

5 Appendix

5.1 Change History

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

