EIQMPPUG

eIQ Media Processing Pipeline User's Guide

Rev. 0 — 30 June 2022 User guide

Document information

Information	Content
Keywords	eIQ, Media, Media Processing, Processing Pipeline, Library
Abstract	This document describes the Media Processing Pipeline software library for MCUs. The library is used for constructing media-handling components graphs for Vision-specific applications.

1 MCU Media Processing Pipeline

This document describes the MCU Media Processing Pipeline API.

1.1 Features overview

The Media Processing Pipeline for MCUs is a software library for constructing graphs of media-handling components for Vision-specific applications.

This is a clean and simple API which makes it easy to build and prototype vision-based applications.

1.1.1 Concept

The concept behind the API is to create a Media Processing Pipeline (MPP) based on processing elements. The basic pipeline structure - the *mpp* in the API context - has a chain/queue structure which begins with a **source element**:

- Camera
- Static image

The pipeline continues with multiple **processing elements** having a single input and a single output:

- Image format conversion
- Labeled rectangle drawing
- Machine learning inference

The pipeline can be closed by adding a **sink element**:

- Display panel
- Null sink

Also, multiple basic *mpps* can be **joined** into a new one to which further elements can be added. An *mpp* can also be **split** when the same media stream must follow different processing paths. With these join/split operations, more complex pipelines can be constructed.

Compatibilty of elements and supplied parameters are checked at each step and only compatible elements can be added in an unequivocal way.

After the construction is complete, each *mpp* must be started for all hardware and software required to run the pipeline to initialize. Pipeline processing begins as soon as the the last start call is flagged.

At runtime the application receives events from the pipeline processing and may use these events to update ele- ments parameters. For example, in object detection when the label of a bounding box must be updated whenever a new object is detected.

Summarizing, the application controls:

- Creation of the pipeline
- Instantiation of processing elements
- Connection of elements to each other
- Reception of callbacks based on specific events
- Updation of specific elements (not all elements can be updated)

Application does not control:

- Memory management
- Data structures management

The order in which an element is added to the pipeline defines its position within this pipeline, and therefore the order is important.

1.2 Example and references

See the examples/reference documentation for practical examples using the MPP API.

2 Deployment

The eIQ Media Processing Pipeline is part of the eIQ machine learning software package, which is an optional middleware component of MCUXpresso SDK.

The eIQ component is integrated into the MCUXpresso SDK Builder delivery system available on mcuxpresso.nxp.com.

To include eIQ Media Processing Pipeline into the MCUXpresso SDK package, select both “eIQ” and “FreeRTOS” in the software component selector on the SDK Builder page.

For details, see, Figure 1.

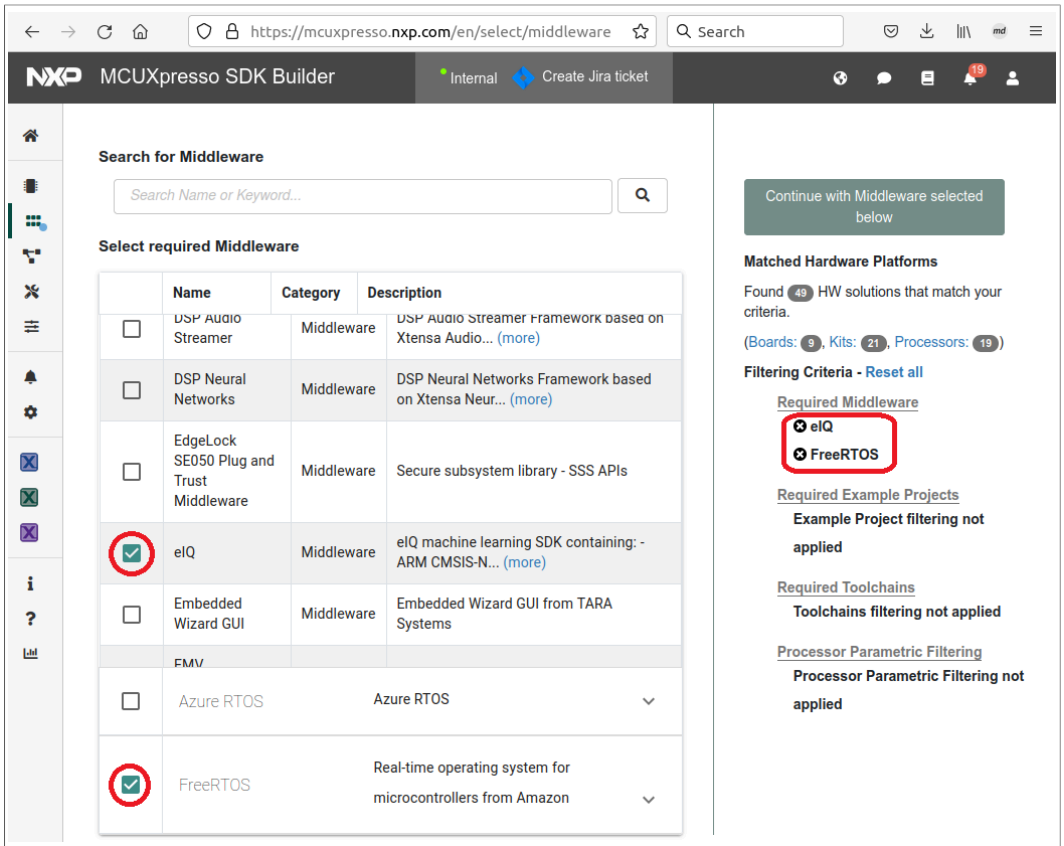

Figure 1. MCUXpresso SDK Builder software component selector

Once the MCUXpresso SDK package is downloaded, it can be extracted on a local machine or imported into the MCUXpresso IDE. For more information on the MCUXpresso SDK folder structure, see the Getting Started with MCUXpresso SDK

User's Guide (document: MCUXSDKGSUG). The package directory structure is similar to Figure 2. The eIQ Media Processing Pipeline directories are highlighted in red.

Figure 2. MCUXpresso SDK directory structure

The *boards* directory contains example application projects for supported toolchains. For the list of supported toolchains, see the *MCUXpresso SDK Release Notes*. The *middleware* directory contains the eIQ library source code and example application source code and data.

3 Example applications

The eIQ Media Processing Pipeline is provided with a set of example applications. For details, see Table 1. The applications demonstrate the usage of the API in several use cases.

Table 1. Example applications

Name	Description	Availability
`camera_view`	This basic example shows how to use the library to create two simple pipelines: • camera preview • image view	EVK-MIMXRT1170
`Camera_mobilenet_view`	This example shows how to use the library to create two image classification use-cases: • image classification using camera as source • image classification using a file as source The machine learning framework used is TensorFlow Lite Micro. The image classification model used is quantized Mobilenet convolutional neural network model that classifies the input image into one of 1000 output classes.	EVK-MIMXRT1170

For details on how to build and run the example applications with supported toolchains, see *Getting Started with MCUXpresso SDK User's Guide* (document: MCUXSDKGSUG).

When using MCUXpresso IDE, the example applications can be imported through the SDK Import Wizard as shown in Figure 3.

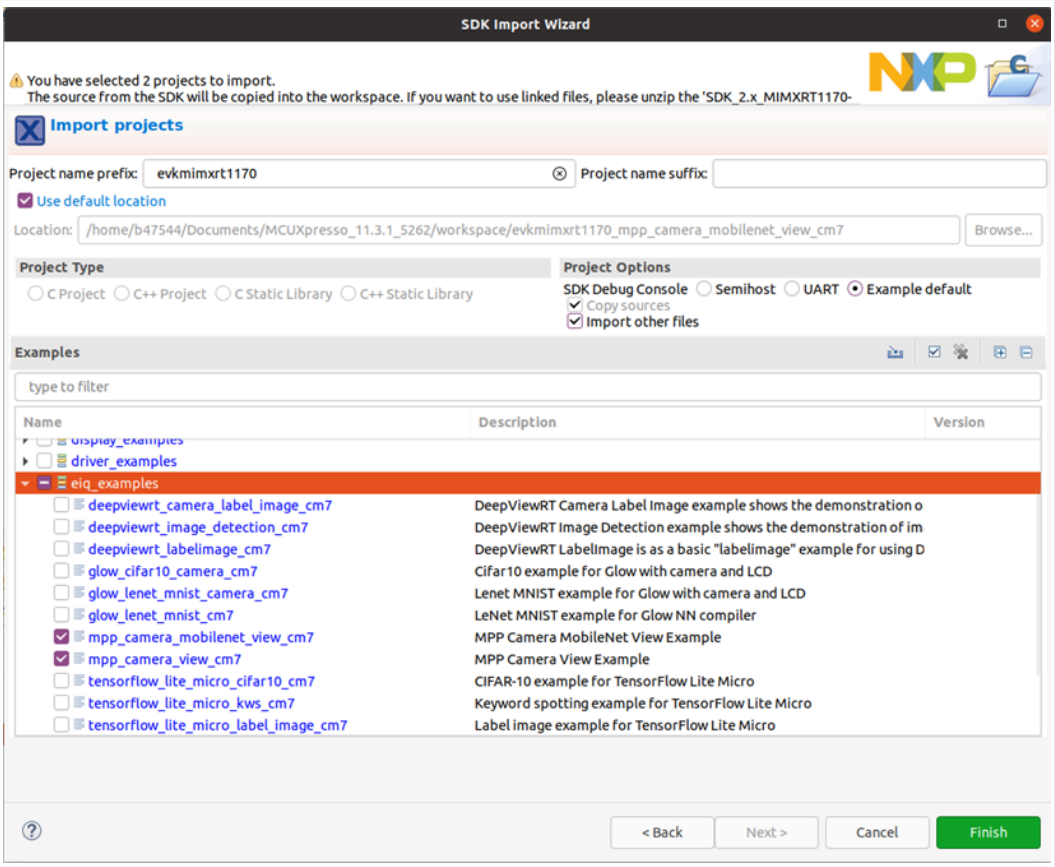

Figure 3. MCUXpresso SDK import projects wizard

After building the example application and downloading it to the target, the execution stops in the *main* function. When the execution resumes, an output message displays on the connected terminal. For example, Figure 4 shows the output of the `camera_mobilenet_view` example application printed to the MCUXpresso IDE Console window when semihosting debug console is selected in the SDK Import Wizard.

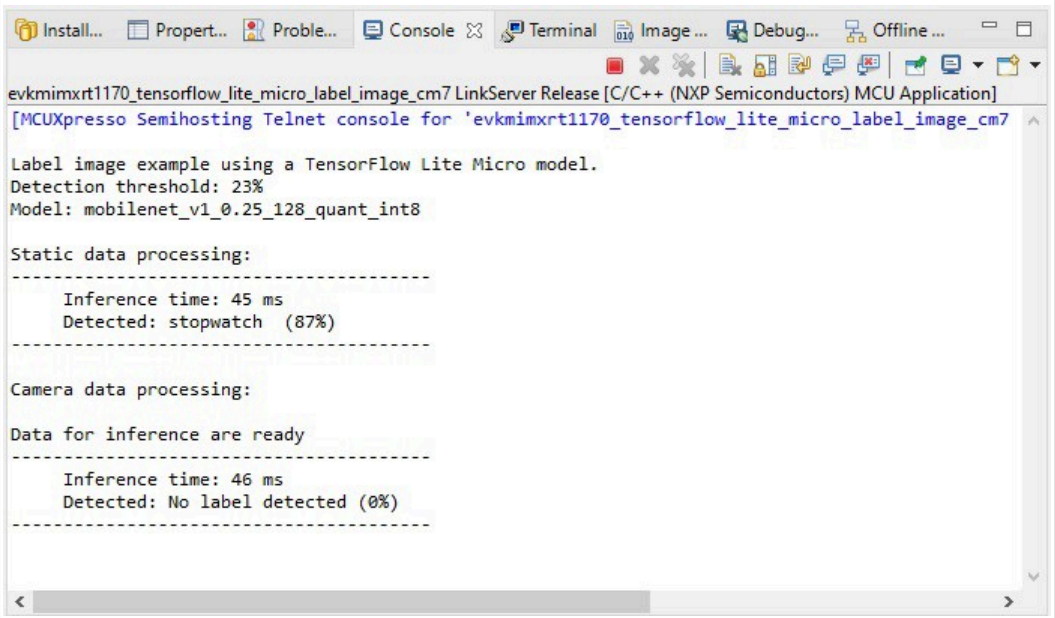

Figure 4. PuTTY console window

When building applications with armgcc, the build scripts for debug target should be edited to call `make` using more than one process.

```
make -jX
```

Where, X>1

The build scripts for armgcc are located under the directory as shown in Figure 5.

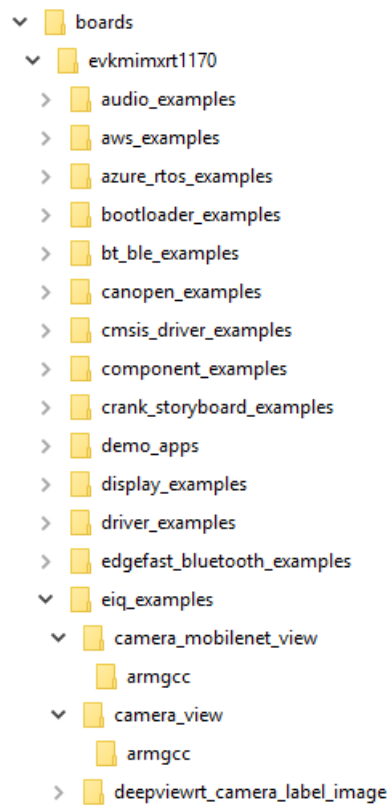

Figure 5. armgcc build scripts location

4 API references

4.1 Module documentation

This section provides information on:

- MPP API
- MPP types
- Return codes

4.1.1 MPP API

4.1.1.1 Functions

- int mpp_api_init (void)
- mpp_tmpp_create (mpp_params_t ∗params, int ∗ret)
- int mpp_camera_add (mpp_t mpp, const char ∗name, mpp_camera_params_t ∗params, _Bool defconfig)
- int mpp_static_img_add (mpp_t mpp, mpp_img_params_t ∗params, void ∗addr)
- int mpp_display_add (mpp_t mpp, const char ∗name, mpp_display_params_t ∗params)
- int mpp_nullsink_add (mpp_t mpp)

- int mpp_element_add (mpp_t mpp, mpp_element_id_t id, mpp_element_params_t ∗params, mpp_elem_handle_t

∗elem_h)

- int mpp_split (mpp_t mpp, unsigned int num, mpp_exec_flag_t ∗flag, mpp_t ∗out_list)
- int mpp_element_split (mpp_t mpp, mpp_element_id_t id, mpp_element_params_t params, mpp_t out_list, unsigned int num)
- int mpp_element_join (mpp_t ∗in_list, unsigned int num, mpp_element_id_t id, mpp_element_params_t

∗params, mpp_t out)

- int mpp_element_update (mpp_t mpp, mpp_elem_handle_t elem_h, mpp_element_params_t ∗params)
- int mpp_start (mpp_t mpp, int last)

4.1.1.2 Detailed Description

This section provides the detailed documentation for the MCU Media Processing Pipeline API.

4.1.1.3 Function Documentation

4.1.1.3.1 mpp_api_init()

```
int mpp_api_init (void)
```

Pipeline initialization.

This function initializes the library and its data structures.

It must be called before any other function of the API is called.

Returns

Return_codes

4.1.1.3.2 mpp_create()

```
mpp_t mpp_create (mpp_params_t * params, int * ret)
```

Basic pipeline creation.

This function returns a handle to the pipeline.

Parameters

in	*params*	pipeline parameters
out	*ret*	return code (0 - success, non-zero - error)

Returns

A handle to the pipeline if success. NULL, if there is an error.

4.1.1.3.3 mpp_camera_add()

```
int mpp_camera_add (mpp_t mpp, const char name, mpp_camera_params_t *
 params, _Bool defconfig)
```

Camera addition.

This function adds a camera to the pipeline.

in	*mpp*	input pipeline
in	*name*	camera driver name
in	*params*	parameters to be configured on the camera
in	*defconfig*	if set, default camera params are returned into the params structure

Returns

Return_codes

4.1.1.3.4 mpp_static_img_add()

```
int mpp_static_img_add (mpp_t mpp, mpp_img_params_t params, void addr)
```

Static image addition.

Parameters

in	*mpp*	input pipeline
in	*params*	static image parameters
in	*addr*	image buffer

Returns

Return_codes

Preconditions

Image buffer allocation/free is the responsibility of the user.

4.1.1.3.5 mpp_display_add()

```
int mpp_display_add (mpp_t mpp, const
 char name, mpp_display_params_t params)
```

Display addition.

This function adds a display to the pipeline.

in	*mpp*	input pipeline
in	*name*	display driver name
in	*params*	parameters that are configured on the display

Returns

Return_codes

4.1.1.3.6 mpp_nullsink_add()

```
int mpp_nullsink_add (mpp_t mpp)
```

Null sink addition.

This function adds a null-type sink to the pipeline.

After this call pipeline is closed and no further elements can be added. Input frames are discarded.

in	*mpp*	input pipeline

Returns

Return_codes

4.1.1.3.7 mpp_element_add()

```
int mpp_element_add
 (mpp_t mpp, mpp_element_id_t id, mpp_element_params_t params,
 mpp_elem_handle_t elem_h)
```

Add processing element (single input, single output). This function adds an element to the pipeline.

Available elements are:

- 2D image processing
- ML inference engine
- Labeled rectangle
- Compositor

in	*mpp*	input pipeline
in	*id*	element id
in	*params*	element parameters
out	elem_h	element handle in pipeline

Returns

Return_codes

4.1.1.3.8 mpp_split()

```
int mpp_split (mpp_t mpp, unsigned int num, mpp_exec_flag_t flag,
mpp_t out_list)
```

Pipeline multiplication.

Parameters

in	*mpp*	input pipeline
in	*num*	number of output pipeline
in	*flags*	selecting the execution type

out	*out_list*	list of output pipelines

Returns

Return_codes

Preconditions

out_list array must contain at least *num* elements.

4.1.1.3.9 mpp_element_split()

```
int mpp_element_split
 (mpp_t mpp,mpp_element_id_t id, mpp_element_params_t params,
mpp_t out_list, unsigned int num)
```

Branching through an element.

Warning

NOT TESTED

Parameters

in	*mpp*	input pipeline
in	*id*	element id
in	*params*	element parameters
out	*out_list*	output pipelines
in	*num*	number of output pipelines

Returns

Return_codes

4.1.1.3.10 mpp_element_join()

```
int mpp_element_join (mpp_t in_list, unsigned
 int num, mpp_element_id_t id, mpp_element_params_t params, mpp_t out)
```

Join multiple pipelines through an element.

The element becomes a source for output pipeline.

Warning

NOT TESTED

Parameters

in	*in_list*	list of joined pipelines
in	*num*	number of pipelines in the list
in	*id*	element id
in	*params*	element params
out	*out*	output pipeline

Returns

Return_codes

4.1.1.3.11 mpp_element_update()

```
int mpp_element_update
 (mpp_t mpp, mpp_elem_handle_t elem_h, mpp_element_params_t params)
```

Update element parameters.

Parameters

in	*mpp*	input pipeline
in	*elem_h*	element handle in the pipeline
in	*params*	new element parameters

Returns

Return_codes

4.1.1.3.12 mpp_start()

```
int mpp_start (mpp_t mpp, int last)
```

Start pipeline.

When called with last=0, this function prepares the branch of the pipeline specified with mpp. When called with last!=0, this function starts the data flow of the pipeline.

Data flow should start after all the branches of the pipeline have been prepared.

Parameters

in	*mpp*	pipeline branch handle to start/prepare
in	*last*	if non-zero start pipeline processing. No further start call is possible thereafter.

Returns

Return_codes

4.1.2 MPP types

4.1.2.1 Data Structures

- struct mpp_params_t
- struct mpp_camera_params_t
- struct mpp_img_params_t
- struct mpp_display_params_t
- struct mpp_tensor_dims_t
- struct mpp_inference_cb_param_t
- union mpp_color_t
- struct mpp_color_t.rgb
- struct mpp_labeled_rect_t
- struct mpp_area_t

- union mpp_element_params_t
- struct mpp_element_params_t.compose
- struct mpp_element_params_t.labels
- struct mpp_element_params_t.convert
- struct mpp_element_params_t.resize
- struct mpp_element_params_t.color_conv
- struct mpp_element_params_t.rotate
- struct mpp_element_params_t.test
- struct mpp_element_params_t.ml_inference

4.1.2.2 Macros

- #define MPP_INVALID
- #define **MPP_EVENT_ALL**
- #define **MAX_TENSOR_DIMS**

4.1.2.3 Typedefs

- typedef void ∗ mpp_t
- typedef uintptr_t mpp_elem_handle_t
- typedef unsigned int mpp_evt_mask_t

4.1.2.4 Enumerations

- enum mpp_evt_t { MPP_EVENT_INVALID,

  MPP_EVENT_INFERENCE_OUTPUT_READY, MPP_EVENT_NUM }

- enum mpp_exec_flag_t { MPP_EXEC_INHERIT, MPP_EXEC_RC, MPP_EXEC_PREEMPT }
- enum mpp_rotate_degree_t { ROTATE_0,

  ROTATE_90, ROTATE_180, ROTATE_270 }

- enum mpp_flip_mode_t { FLIP_NONE, FLIP_HORIZONTAL, FLIP_VERTICAL, FLIP_BOTH }
- enum mpp_convert_ops_t { MPP_CONVERT_NONE, MPP_CONVERT_ROTATE, MPP_CONVERT_SCALE, MPP_CONVERT_COLOR, MPP_CONVERT_CROP }
- enum mpp_pixel_format_t { MPP_PIXEL_ARGB, MPP_PIXEL_RGB, MPP_PIXEL_RGB565, MPP_PIXEL_BGR, MPP_PIXEL_GRAY888, MPP_PIXEL_GRAY888X, MPP_PIXEL_GRAY, MPP_PIXEL_GRAY16, MPP_PIXEL_YUV1P444, MPP_PIXEL_VYUY1P422, MPP_PIXEL_UYVY1P422, MPP_PIXEL_YUYV, MPP_PIXEL_DEPTH16, MPP_PIXEL_DEPTH8, MPP_PIXEL_YUV420P, MPP_PIXEL_INVALID }
- enum mpp_element_id_t { MPP_ELEMENT_INVALID, MPP_ELEMENT_COMPOSE, MPP_ELEMENT_LABELED_RECTANGLE, MPP_ELEMENT_TEST, MPP_ELEMENT_INFERENCE, MPP_ELEMENT_CONVERT, MPP_ELEMENT_NUM }
- enum mpp_tensor_type_t { MPP_TENSOR_TYPE_FLOAT32, MPP_TENSOR_TYPE_UINT8, MPP_TENSOR_TYPE_INT8 }
- enum mpp_inference_type_t { MPP_INFERENCE_TYPE_TFLITE, MPP_INFERENCE_TYPE_DEEPVIEWRT }

4.1.2.5 Detailed Description

This section provides the detailed documentation for the MCU Media Processing Pipeline types.

4.1.2.6 Data Structure Documentation

4.1.2.6.1 struct mpp_params_t

Pipeline creation parameters.

Data Fields

- int(∗ **evt_callback_f**)(mpp_t mpp, mpp_evt_t evt, void evt_data, void ∗user_data)
- mpp_evt_mask_t**mask**
- mpp_exec_flag_t**exec_flag**
- void **cb_userdata**

4.1.2.6.2 struct mpp_camera_params_t

Camera parameters.

Data Fields

int	height	buffer height
int	width	buffer width
mpp_pixel_format_t	format	pixel format
int	fps	frames per second

4.1.2.6.3 struct mpp_img_params_t

Static image parameters.

Data Fields

int	height	
int	width	
mpp_pixel_format_t	format	

4.1.2.6.4 struct mpp_display_params_t

Display parameters.

Data Fields

int	height	buffer resolution: setting to 0 will default to panel physical resolution
int	width	buffer resolution: setting to 0 will default to panel physical resolution

int	pitch	buffer resolution: setting to 0 will default to panel physical resolution
int	left	active rect: setting to 0 will default to fullscreen
int	top	active rect: setting to 0 will default to fullscreen
int	right	active rect: setting to 0 will default to fullscreen
int	bottom	active rect: setting to 0 will default to fullscreen
mpp_rotate_degree_t	rotate	rotate degree
mpp_pixel_format_t	format	pixel format

4.1.2.6.5 struct mpp_tensor_dims_t

Inference tensor dimensions.

Data Fields

uint32_t	size
uint32_t	data[MAX_TENSOR_DIMS]

4.1.2.6.6 struct mpp_inference_cb_param_t

Inference callback parameters.

Data Fields

void *	user_data	callback will pass this pointer
const uint8_t *	tensor	output tensor data
mpp_tensor_dims_t	dims	tensor data dimensions
mpp_tensor_type_t	type	tensor data type
int	inference_time_ms	inference run time measurement - output to user

4.1.2.6.7 union mpp_color_t

MPP color encoding.

Data Fields

uint32_t	raw	Raw color.
struct mpp_color_t	rgb	rgb color values RGB color

4.1.2.6.8 struct mpp_color_t.rgb

RGB color values.

Data Fields

uint8_t	R	Red byte.
uint8_t	G	Green byte.
uint8_t	B	Blue byte.
uint8_t	pad	padding byte

4.1.2.6.9 struct mpp_labeled_rect_t

MPP labeled rectangle element structure.

Data Fields

uint8_t	label[64]	label to print
uint16_t	clear	clear rectangle
uint16_t	line_width	rectangle line thickness
mpp_color_t	line_color	rectangle line color
uint16_t	top	rectangle top position
uint16_t	left	rectangle left position
uint16_t	bottom	rectangle bottom position
uint16_t	right	rectangle right position
uint16_t	tag	labeled rectangle tag
uint16_t	reserved	pad for 32 bits alignment

4.1.2.6.10 struct mpp_area_t

Image area coordinates.

Data Fields

int	top	
int	left	
int	bottom	
int	right	

4.1.2.6.11 union mpp_element_params_t

Processing element parameters.

Data Fields

struct mpp_element_params_t	compose	Compose element's parameters - NOT IMPLEMENTED YET.
struct mpp_element_params_t	labels	Labeled Rectangle element's parameters.
struct mpp_element_params_t	convert	Convert element's parameters.
struct mpp_element_params_t	resize	Resize element's parameters.

struct mpp_element_params_t	color_conv	Color convert element's parameters.
struct mpp_element_params_t	rotate	Rotate element's parameters.
struct mpp_element_params_t	test	Test element's parameters.
struct mpp_element_params_t	ml_inference	ML inference element's parameters.

4.1.2.6.12 struct mpp_element_params_t.compose

Compose element's parameters. NOT IMPLEMENTED YET.

Data Fields

float	a	
float	b	

4.1.2.6.13 struct mpp_element_params_t.labels

Labeled rectangle element's parameters.

Data Fields

uint32_t	max_count	maximum number of rectangles
uint32_t	detected_count	detected rectangles
mpp_labeled_rect_t *	rectangles	array of rectangle data

4.1.2.6.14 struct mpp_element_params_t.convert

Convert element's parameters.

Data Fields

unsigned int	width	output image width
unsigned int	height	output image height
mpp_pixel_format_t	pixel_format	new pixel format
mpp_rotate_degree_t	angle	rotation angle
mpp_area_t	crop	input crop area
mpp_area_t	out_area	output window area
mpp_convert_ops_t	ops	operation selector mask

4.1.2.6.15 struct mpp_element_params_t.resize

Resize element's parameters.

Data Fields

unsigned int	width	
unsigned int	height	

4.1.2.6.16 struct mpp_element_params_t.color_conv

Color convert element's parameters.

Data Fields

mpp_pixel_format_t	pixel_format	

4.1.2.6.17 struct mpp_element_params_t.rotate

Rotate element's parameters.

Data Fields

mpp_rotate_degree_t	angle	

4.1.2.6.18 struct mpp_element_params_t.test

Test element's parameters.

Data Fields

_Bool	inp	
unsigned int	width	
unsigned int	height	
mpp_pixel_format_t	format	

4.1.2.6.19 struct mpp_element_params_t.ml_inference

ML inference element's parameters.

Data Fields

const void *	model_data	pointer to model binary
mpp_inference_type_t	type	inference type
int	model_size	model binary size
float	model_input_mean	model 'mean' of input values, used for normalization
float	model_input_std	model 'standard deviation' of input values, used for normalization

4.1.2.7 Macro Definition Documentation

4.1.2.7.1 MPP_INVALID

```
#define MPP_INVALID
```

Invalid pipeline handle.

4.1.2.8 Typedef Documentation

4.1.2.8.1 mpp_t

```
typedef void mpp_t
```

Pipeline handle type.

4.1.2.8.2 mpp_elem_handle_t

```
typedef uintptr_t mpp_elem_handle_t
```

Element handle type.

4.1.2.8.3 mpp_evt_mask_t

```
typedef unsigned int mpp_evt_mask_t
```

Event mask for pipeline creation.

4.1.2.9 Enumeration Type Documentation

4.1.2.9.1 mpp_evt_t

```
enum mpp_evt_t
```

Pipeline generated events.

Enumerator

MPP_EVENT_INVALID	invalid event
MPP_EVENT_INFERENCE_OUTPUT_READY	inference out is ready
MPP_EVENT_NUM	DO NOT USE.

4.1.2.9.2 mpp_exec_flag_t

```
enum mpp_exec_flag_t
```

Execution parameters.

These parameters control the execution of the elements of an mpp.

The "mpps" created using the flag MPP_EXEC_RC are guaranteed to run up to the completion of all processing elements, while not being preempted by other "mpps".

The "mpps" created using the flag MPP_EXEC_PREEMPT are preempted after a given time interval by "mpps" that will run-to-completion again.

The "mpps" created with the MPP_EXEC_INHERIT flag inherit the same execution flag as the parent(s) in case of split/join operation.

Note: *It is not possible to request run-to-completion execution when spliting/joining preemptable-execution "mpps".*

Enumerator

MPP_EXEC_INHERIT	inherit from parent(s)
MPP_EXEC_RC	run-to-completion
MPP_EXEC_PREEMPT	preemptable

4.1.2.9.3 mpp_rotate_degree_t

`enum mpp_rotate_degree_t`

Rotation value.

Enumerator

ROTATE_0	0 degree
ROTATE_90	90 degrees
ROTATE_180	180 degrees
ROTATE_270	270 degrees

4.1.2.9.4 mpp_flip_mode_t

`enum mpp_flip_mode_t`

Flip type.

Enumerator

FLIP_NONE	no flip
FLIP_HORIZONTAL	horizontal flip
FLIP_VERTICAL	vertical flip
FLIP_BOTH	vertical and horizontal flip

4.1.2.9.5 mpp_convert_ops_t

`enum mpp_convert_ops_t`

The convert operations selector flags.

Enumerator

MPP_CONVERT_NONE	no frame conversion
MPP_CONVERT_ROTATE	frame rotation
MPP_CONVERT_SCALE	frame scaling
MPP_CONVERT_COLOR	frame color conversion
MPP_CONVERT_CROP	frame crop

4.1.2.9.6 mpp_pixel_format_t

```
enum mpp_pixel_format_t
```

Pixel format.

Enumerator

MPP_PIXEL_ARGB	ARGB 32 bits.
MPP_PIXEL_RGB	RGB 24 bits.
MPP_PIXEL_RGB565	RGB 16 bits.

MPP_PIXEL_BGR	BGR 24 bits.
MPP_PIXEL_GRAY888	gray 3x8 bits
MPP_PIXEL_GRAY888X	gray 3x8 bits +8 unused bits
MPP_PIXEL_GRAY	gray 8 bits
MPP_PIXEL_GRAY16	gray 16 bits
MPP_PIXEL_YUV1P444	YUVX interleaved 4:4:4.
MPP_PIXEL_VYUY1P422	VYUY interleaved 4:2:2.
MPP_PIXEL_UYVY1P422	UYVY interleaved 4:2:2.
MPP_PIXEL_YUYV	YUYV interleaved 4:2:2.
MPP_PIXEL_DEPTH16	depth 16 bits
MPP_PIXEL_DEPTH8	depth 8 bits
MPP_PIXEL_YUV420P	YUV planar 4:2:0.
MPP_PIXEL_INVALID	invalid pixel format

4.1.2.9.7 mpp_element_id_t

```
enum mpp_element_id_t
```

Processing element ids.

Enumerator

MPP_ELEMENT_INVALID	Invalid element.
MPP_ELEMENT_COMPOSE	Image composition - NOT IMPLEMENTED YET.
MPP_ELEMENT_LABELED_RECTANGLE	Labeled rectangle - bounding box.
MPP_ELEMENT_TEST	Test inplace element - NOT FOR USE.
MPP_ELEMENT_INFERENCE	Inference engine.
MPP_ELEMENT_CONVERT	Image conversion: resolution, orientation, color format.
MPP_ELEMENT_NUM	DO NOT USE.

4.1.2.9.8 mpp_tensor_type_t

```
enum mpp_tensor_type_t
```

Inference tensor type.

Enumerator

MPP_TENSOR_TYPE_FLOAT32	floating point 32 bits
MPP_TENSOR_TYPE_UINT8	unsigned integer 8 bits
MPP_TENSOR_TYPE_INT8	signed integer 8 bits

4.1.2.9.9 mpp_inference_type_t

```
enum mpp_inference_type_t
```

Inference type.

Enumerator

MPP_INFERENCE_TYPE_TFLITE	TensorFlow-Lite.
MPP_INFERENCE_TYPE_DEEPVIEWRT	DeepView RT.

4.1.3 Return codes

4.1.3.1 Macros

- #define MPP_SUCCESS
- #define MPP_ERROR
- #define MPP_INVALID_ELEM
- #define MPP_INVALID_PARAM
- #define MPP_ERR_ALLOC_MUTEX
- #define MPP_INVALID_MUTEX
- #define MPP_MUTEX_TIMEOUT
- #define MPP_MUTEX_ERROR
- #define MPP_MALLOC_ERROR

4.1.3.2 Detailed Description

MPP APIs return status definitions.

4.1.3.3 Macro Definition Documentation

4.1.3.3.1 MPP_SUCCESS

```
#define MPP_SUCCESS
```

Success return code.

4.1.3.3.2 MPP_ERROR

```
#define MPP_ERROR
```

A generic error occured.

4.1.3.3.3 MPP_INVALID_ELEM

```
#define MPP_INVALID_ELEM
```

Invalid element provided.

4.1.3.3.4 MPP_INVALID_PARAM

```
#define MPP_INVALID_PARAM
```

Invalid parameter provided.

4.1.3.3.5 MPP_ERR_ALLOC_MUTEX

```
#define MPP_ERR_ALLOC_MUTEX
```

Error occured while allocating mutex.

4.1.3.3.6 MPP_INVALID_MUTEX

```
#define MPP_INVALID_MUTEX
```

Invalid mutex provided.

4.1.3.3.7 MPP_MUTEX_TIMEOUT

```
#define MPP_MUTEX_TIMEOUT
```

Mutex timeout occured.

4.1.3.3.8 MPP_MUTEX_ERROR

```
#define MPP_MUTEX_ERROR
```

Mutex error occured.

4.1.3.3.9 MPP_MALLOC_ERROR

```
#define MPP_MALLOC_ERROR
```

Memory allocation error occured.

5 Revision history

Table 2 summarizes the changes done to this document since the initial release.

Table 2. Revision history

Revision number	Date	Substantive changes
0	30 June 2022	Initial release

6 Legal information

6.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

6.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

6.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 30 June 2022
Document identifier: eIQ Media Processing Pipeline User's Guide