



# MVA™ Application Programming Interface

Version 1.3.0.307

February 19, 2021

All information contained in this document is proprietary to CSP, Inc. and may not be reproduced, distributed, or disseminated, in whole or in part, without the written permission of an authorized representative of CSP, Inc.

All specifications presented in this document are subject to change at any time, and without prior notice.

Myricom® and Myrinet® are registered trademarks of CSP, Inc. MVA™ is a trademark of CSP, Inc. Other trademarks appearing in this document are those of their respective owners.

©2011-2014, CSP, Inc.



# Contents

<b>1</b>	<b>MVA Documentation</b>	<b>1</b>
1.1	Introduction	1
1.2	Terminology	1
1.3	Overview	1
1.4	Streams	1
1.5	Memory management	2
1.6	Receiving data blocks	2
<b>2</b>	<b>Module Index</b>	<b>3</b>
2.1	API Reference	3
<b>3</b>	<b>Namespace Index</b>	<b>5</b>
3.1	Namespace List	5
<b>4</b>	<b>Data Structure Index</b>	<b>7</b>
4.1	Data Structures	7
<b>5</b>	<b>Module Documentation</b>	<b>9</b>
5.1	Initialization	9
5.1.1	Detailed Description	9
5.1.2	Macro Definition Documentation	9
5.1.2.1	MVA_VERSION_API	9
5.1.3	Enumeration Type Documentation	10
5.1.3.1	mva_link_state	10
5.1.3.2	mva_timesource_state	10
5.1.4	Function Documentation	10
5.1.4.1	mva_init	10
5.2	Streams	11
5.2.1	Detailed Description	11
5.2.2	Macro Definition Documentation	12
5.2.2.1	MVA_BOOO_ALWAYS_DROP	12
5.2.2.2	MVA_BOOO_ALWAYS_RETURN	12
5.2.2.3	MVA_BOOO_GT_ALWAYS_RETURN	12
5.2.2.4	MVA_BOOO_LT_ALWAYS_RETURN	12
5.2.2.5	MVA_OPEN_DROP_INCOMPLETE	12
5.2.2.6	MVA_OPEN_IPV6	12
5.2.2.7	MVA_OPEN_RETURN_INCOMPLETE	12
5.2.2.8	MVA_OPEN_ZEROLOSS	12
5.2.3	Typedef Documentation	12
5.2.3.1	mva_stream_t	12

5.2.4	Function Documentation	13
5.2.4.1	mva_close_stream	13
5.2.4.2	mva_get_link_state	13
5.2.4.3	mva_get_timesource_state	13
5.2.4.4	mva_open_mcast_stream	13
5.2.4.5	mva_open_stream	14
5.2.4.6	mva_open_stream_pr	15
5.3	Memory allocation and deallocation.	17
5.3.1	Detailed Description	17
5.3.2	Function Documentation	17
5.3.2.1	mva_alloc	17
5.3.2.2	mva_free	17
5.4	Enqueuing and Receiving data blocks.	19
5.4.1	Detailed Description	19
5.4.2	Macro Definition Documentation	19
5.4.2.1	MVA_BLOCK_STATUS_NO_BLOCK_RECEIVED	19
5.4.3	Function Documentation	20
5.4.3.1	mva_clear_stats	20
5.4.3.2	mva_get_stats	20
5.4.3.3	mva_info	20
5.4.3.4	mva_poll_rcv	20
5.4.3.5	mva_poll_rcv_pr	21
5.4.3.6	mva_queue_buffer	22
5.5	Manage the name of an MVA adapter.	23
5.5.1	Detailed Description	23
5.5.2	Function Documentation	23
5.5.2.1	mva_get_name	23
5.5.2.2	mva_reset_all_name	23
5.5.2.3	mva_reset_name	24
5.5.2.4	mva_set_name	24
5.6	Manage metrics.	25
5.6.1	Detailed Description	25
5.6.2	Function Documentation	25
5.6.2.1	mva_add_metrics	25
5.6.2.2	mva_create_metrics	26
5.6.2.3	mva_destroy_metrics	26
5.6.2.4	mva_disable_metrics	26
5.6.2.5	mva_enable_metrics	27
5.6.2.6	mva_remove_metrics	27
5.6.2.7	mva_start_metrics	27
5.6.2.8	mva_stop_metrics	27
5.7	manage trace	29
5.7.1	Detailed Description	29
5.7.2	Function Documentation	29
5.7.2.1	mva_add_trace	29
5.7.2.2	mva_create_trace	30
5.7.2.3	mva_destroy_trace	30
5.7.2.4	mva_disable_trace	30
5.7.2.5	mva_enable_trace	30
5.7.2.6	mva_remove_trace	31

5.7.2.7	mva_start_trace	31
5.7.2.8	mva_stop_trace	31
5.8	receive worker thread	32
5.8.1	Detailed Description	32
5.8.2	Typedef Documentation	32
5.8.2.1	mva_get_request_id_cb	32
5.8.3	Function Documentation	33
5.8.3.1	mva_block_recv_worker	33
5.8.3.2	mva_create_recv_worker	33
5.8.3.3	mva_destroy_recv_worker	33
5.8.3.4	mva_drop_packet	34
5.8.3.5	mva_get_camera_id_stream	34
5.8.3.6	mva_get_status_recv_worker	34
5.8.3.7	mva_start_recv_worker	34
5.8.3.8	mva_stop_recv_worker	35
5.8.3.9	mva_unblock_recv_worker	35
<b>6</b>	<b>Namespace Documentation</b>	<b>37</b>
6.1	mva Namespace Reference	37
6.1.1	Detailed Description	37
<b>7</b>	<b>Data Structure Documentation</b>	<b>39</b>
7.1	mva_block Struct Reference	39
7.1.1	Detailed Description	40
7.1.2	Field Documentation	40
7.1.2.1	block_id	40
7.1.2.2	crc	40
7.1.2.3	meta	40
7.1.2.4	mva_buf	40
7.1.2.5	nsecs	40
7.1.2.6	payload_data	41
7.1.2.7	payload_length	41
7.1.2.8	payload_type	41
7.1.2.9	status	41
7.1.2.10	timestamp	41
7.2	mva_block_status Struct Reference	41
7.2.1	Field Documentation	41
7.2.1.1	block_id	41
7.2.1.2	dropped_packet_id	41
7.2.1.3	next_req_id	41
7.2.1.4	resend_request	42
7.2.1.5	status	42
7.3	mva_info Struct Reference	42
7.3.1	Detailed Description	42
7.3.2	Field Documentation	42
7.3.2.1	blocks_dropped	42
7.3.2.2	blocks_received	42
7.3.2.3	port_active	42
7.3.2.4	port_link_up	42
7.4	mva_packet_resend_config Struct Reference	43
7.4.1	Field Documentation	43

7.4.1.1	block_out_of_order	43
7.4.1.2	block_timeout	43
7.4.1.3	camera_id	43
7.4.1.4	camera_ipv4	43
7.4.1.5	camera_port	43
7.4.1.6	flags	43
7.4.1.7	resend_retries	43
7.4.1.8	resend_timeout	44
7.4.1.9	stream_channel_id	44
7.5	mva_receive_stream_worker_status Struct Reference	44
7.6	mva_stats Struct Reference	44
7.6.1	Detailed Description	44
7.6.2	Field Documentation	45
7.6.2.1	block_time_outs	45
7.6.2.2	blocked_blocks_dropped	45
7.6.2.3	blocks_dropped	45
7.6.2.4	blocks_not_available	45
7.6.2.5	blocks_returned_complete	45
7.6.2.6	blocks_returned_incomplete	45
7.6.2.7	buf_allocated	45
7.6.2.8	buf_enqueued	45
7.6.2.9	buf_freed	45
7.6.2.10	callbacks_invoked	45
7.6.2.11	max_resend_retries	46
7.6.2.12	resend_requests	46
7.6.2.13	resend_requests_failed	46
7.6.2.14	resend_requests_successful	46
7.6.2.15	resend_time_outs	46
7.6.2.16	stream_blocked	46
7.6.2.17	stream_started	46
7.6.2.18	stream_stopped	46
7.6.2.19	stream_unblocked	46

# Chapter 1

## MVA Documentation

### 1.1 Introduction

The Myricom Machine Vision Accelerator MVA™ solution greatly improves the performance of machine vision applications processing data from GigE Vision devices. MVA dramatically reduces the host processor overhead while providing maximum throughput when receiving GigE Vision Stream Protocol (GVSP) content.

### 1.2 Terminology

Readers of this document should be familiar with the AIA GigE Vision Specification version 1.x and the GenICam™ Standard. See [www.machinevisiononline.org](http://www.machinevisiononline.org) and [www.genicam.org](http://www.genicam.org) respectively for more detail. "Application" in this document describes the consumer of the MVA interface, commonly GigE Vision libraries or development kits.

### 1.3 Overview

MVA leverages Myri-10G programmable 10-Gigabit Ethernet network interface cards (NICs) with custom firmware to divert GVSP data directly to user-space memory, bypassing the operating system and legacy network software stacks. MVA offloads the reassembly of GVSP data blocks from individual packets on the wire, avoiding intermediate memory copies and context switch overhead. Optionally, MVA can handle GVSP reliability in NIC firmware, requesting retransmission of lost packets in real-time without host involvement.

MVA is composed of a user library, driver, and firmware running on the embedded processor of the Myri-10G network adapter.

### 1.4 Streams

An application opens a GVSP stream by specifying a destination address and port using the GigE Vision SCDx and S-CPx registers. A matching MVA stream is created by passing the same parameters to an [mva\\_open\\_stream\(\)](#) function, including [mva\\_open\\_stream\\_pr\(\)](#). The destination address should match the Ethernet interface address of an MVA--Enabled Myri-10G NIC. Only GVSP traffic associated to MVA streams is handled by the MVA stack, all other GVSP

packets are directed to the legacy network stack in the operating system.

## 1.5 Memory management

Memory directly accessible by the network adapter DMA engine must be pinned to physical pages. The application uses `mva_alloc()` to allocate such memory. MVA buffers can be of any size, large enough to contain one or more GVSP data blocks. It is an application error to free MVA memory while it is in use.

## 1.6 Receiving data blocks

MVA delivers GVSP data blocks into buffers queued into the NIC using `mva_queue_buffer()`. Multiple buffers can be queued for a particular stream and they are used in the order they are enqueued. Buffers should be large enough to contain the corresponding data block payload. For example, this size could be the `PayloadSize` value in the GenICam Device Description file. Once a buffer is queued, its ownership is transferred to the MVA library.

The application should ensure that there is always a buffer available to receive incoming data for a given stream. If no buffer is available, all the packets related to the current data block are dropped. Queuing multiple buffers allows for consecutive data blocks to be received on a stream without host involvement.

MVA provides several modes for receiving data from GigE Vision devices. Drop mode instructs the NIC to jettison the entire GVSP data block if one or more packet is missing. In Zero-loss mode, the NIC firmware will request retransmission of missing packets according to configurable timeouts.

The application uses `mva_poll_recv()` to wait for the reception of the next GVSP data block on a given MVA stream. This function returns when a data block has been received into the next queued buffer or when the timeout has expired, whichever comes first. If successful, the address of the corresponding buffer and the actual size of the data block is returned, along with the related GVSP metadata. Once `mva_poll_recv()` indicates that a data block has been received into a buffer, the ownership of this buffer is transferred back to the application. When its content has been processed, the buffer can safely be queued again for any stream.



## Chapter 2

# Module Index

### 2.1 API Reference

Here is a list of all modules:

Initialization . . . . .	9
Streams . . . . .	11
Memory allocation and deallocation. . . . .	17
Enqueuing and Receiving data blocks. . . . .	19
Manage the name of an MVA adapter. . . . .	23
Manage metrics. . . . .	25
manage trace . . . . .	29
receive worker thread. . . . .	32



## Chapter 3

# Namespace Index

### 3.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

<a href="#">mva</a> .....	37
---------------------------	----



# Chapter 4

## Data Structure Index

### 4.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">mva_block</a>	39
<a href="#">mva_block_status</a>	41
<a href="#">mva_info</a>	42
<a href="#">mva_packet_resend_config</a>	43
<a href="#">mva_receive_stream_worker_status</a>	44
<a href="#">mva_stats</a>	44



## Chapter 5

# Module Documentation

### 5.1 Initialization

MVA Initialization function.

#### Macros

- `#define MVA_VERSION_API 0x0102`  
*MVA API version number (16 bits)*

#### Enumerations

- enum `mva_link_state` { `MVA_LINK_DOWN` = 0, `MVA_LINK_UP` = 1 }
- enum `mva_timesource_state` { `MVA_TIMESOURCE_LOCAL` = 0, `MVA_TIMESOURCE_EXT_UNSYNCED`, `MVA_TIMESOURCE_EXT_SYNCED`, `MVA_TIMESOURCE_EXT_FAILED` }

#### Functions

- `mva_init` (uint16\_t api\_version)  
*Initialize MVA library.*

#### 5.1.1 Detailed Description

MVA Initialization function.

#### 5.1.2 Macro Definition Documentation

##### 5.1.2.1 `#define MVA_VERSION_API 0x0102`

MVA API version number (16 bits)

LSB increases for minor backwards compatible changes in the API. MSB increases for incompatible changes in the API.

### 5.1.3 Enumeration Type Documentation

#### 5.1.3.1 enum mva\_link\_state

Link state enumeration, returned by [mva\\_get\\_link\\_state](#).

#### 5.1.3.2 enum mva\_timesource\_state

Timesource state (for -SYNC NICs), returned by [mva\\_get\\_timesource\\_state](#).

#### Enumerator

***MVA\_TIMESOURCE\_LOCAL*** Local timesource (no external). Returned if there is no available external timesource or if its use was explicitly disabled.

***MVA\_TIMESOURCE\_EXT\_UNSYNCED*** External Timesource: not synchronized (yet).

***MVA\_TIMESOURCE\_EXT\_SYNCED*** External Timesource: synchronized.

***MVA\_TIMESOURCE\_EXT\_FAILED*** External Timesource: NIC failure to connect to source.

### 5.1.4 Function Documentation

#### 5.1.4.1 mva\_init ( uint16\_t api\_version )

Initialize MVA library.

This function initializes the MVA library, verifies driver and linked library compatibility, and checks the license and allocates device-independent resources.

#### Parameters

in	<i>api_version</i>	Must always be <a href="#">MVA_VERSION_API</a> .
----	--------------------	--

#### Return values

<i>EINVAL</i>	Library already initialized with different API version. Incompatible library/API version.
<i>ENXIO</i>	Incompatible driver/library.
<i>ENODEV</i>	No driver found.
<i>ENOMEM</i>	Not enough available memory.

#### Remarks

This function should be invoked prior to any other MVA calls. It can be called multiple times, as long as the API version is the same.



## 5.2 Streams

Functions related to MVA Streams.

### Data Structures

- struct [mva\\_packet\\_resend\\_config](#)

### Macros

- #define [MVA\\_OPEN\\_IPV6](#) 0x1
- #define [MVA\\_OPEN\\_ZEROLOSS](#) 0x2
- #define [MVA\\_OPEN\\_DROP\\_INCOMPLETE](#) 0x4
- #define [MVA\\_OPEN\\_RETURN\\_INCOMPLETE](#) 0x8
- #define [MVA\\_BOOO\\_ALWAYS\\_DROP](#) 0x1
- #define [MVA\\_BOOO\\_ALWAYS\\_RETURN](#) 0x2
- #define [MVA\\_BOOO\\_LT\\_ALWAYS\\_RETURN](#) 0x3
- #define [MVA\\_BOOO\\_GT\\_ALWAYS\\_RETURN](#) 0x4

### Typedefs

- typedef struct mva\_stream \* [mva\\_stream\\_t](#)

### Functions

- [mva\\_open\\_stream](#) (void \*in\_addr, uint16\_t dest\_port, [mva\\_stream\\_t](#) \*mva\_stream, int flags, MVA\_OS\_HANDLE \*os\_handle)  
*Open a GigE Vision stream for MVA acceleration.*
- [mva\\_open\\_stream\\_pr](#) (void \*in\_addr, uint16\_t dest\_port, [mva\\_stream\\_t](#) \*mva\_stream, struct [mva\\_packet\\_resend\\_config](#) \*prc, MVA\_OS\_HANDLE \*os\_handle)  
*Open a GigE Vision stream for MVA acceleration with Packet Resend support.*
- [mva\\_open\\_mcast\\_stream](#) (void \*in\_addr, void \*mcast\_in\_addr, uint16\_t dest\_port, [mva\\_stream\\_t](#) \*mva\_stream, int flags, MVA\_OS\_HANDLE \*os\_handle)  
*Open a GigE Vision Multicast group stream for MVA acceleration. The Ethernet address must match the interface address of a Myri-10G network adapter (NIC) with MVA enabled.*
- [mva\\_close\\_stream](#) ([mva\\_stream\\_t](#) mva\_stream)  
*Close an MVA Stream.*
- [mva\\_get\\_link\\_state](#) ([mva\\_stream\\_t](#) strm, enum [mva\\_link\\_state](#) \*state)
- [mva\\_get\\_timesource\\_state](#) ([mva\\_stream\\_t](#) strm, enum [mva\\_timesource\\_state](#) \*state)

#### 5.2.1 Detailed Description

Functions related to MVA Streams.

## 5.2.2 Macro Definition Documentation

### 5.2.2.1 `#define MVA_B000_ALWAYS_DROP 0x1`

The packet resend config object is used by the open stream with packet resend allowed function. It's used to config how resend will work. The object will remain in effect until the stream is closed.

Always drop out-of-order blocks.

### 5.2.2.2 `#define MVA_B000_ALWAYS_RETURN 0x2`

Always return out-of-order blocks.

### 5.2.2.3 `#define MVA_B000_GT_ALWAYS_RETURN 0x4`

Only return out-of-order blocks if the next packet ID is greater than the expected one.

### 5.2.2.4 `#define MVA_B000_LT_ALWAYS_RETURN 0x3`

Only return out-of-order blocks if the next packet ID is less than the expected one.

### 5.2.2.5 `#define MVA_OPEN_DROP_INCOMPLETE 0x4`

Stream operates on "drop incomplete" mode.

### 5.2.2.6 `#define MVA_OPEN_IPV6 0x1`

Address specified to [mva\\_open\\_stream\(\)](#) is an IPv6 address (default: IPv4).

### 5.2.2.7 `#define MVA_OPEN_RETURN_INCOMPLETE 0x8`

Stream operates on "return incomplete" mode.

### 5.2.2.8 `#define MVA_OPEN_ZEROLOSS 0x2`

Stream operates in "zeroloss" modes (default: "drop" mode).

## 5.2.3 Typedef Documentation

### 5.2.3.1 `typedef struct mva_stream* mva_stream_t`

Opaque stream handle structure.

## 5.2.4 Function Documentation

### 5.2.4.1 `mva_close_stream ( mva_stream_t mva_stream )`

Close an MVA Stream.

This function closes a stream from MVA acceleration.

#### Parameters

<code>in</code>	<code>mva_stream</code>	MVA stream handle.
-----------------	-------------------------	--------------------

#### Postcondition

The MVA stream handle is no longer valid and cannot be used for any other functions. All queued buffers are released, and their ownership is transferred back to the application.

### 5.2.4.2 `mva_get_link_state ( mva_stream_t strm, enum mva_link_state * state )`

Get link status on opened handle.

#### Parameters

<code>strm</code>	Stream handle.
<code>state</code>	Returns one of <code>MVA_LINK_DOWN</code> or <code>MVA_LINK_UP</code> .

#### Remarks

The cost of retrieving the link state requires a function call that reads state kept in kernel host memory (i.e., no PCI bus reads).

### 5.2.4.3 `mva_get_timesource_state ( mva_stream_t strm, enum mva_timesource_state * state )`

Get timesource information from opened handle.

#### Parameters

<code>strm</code>	Stream handle.
<code>state</code>	Returns one of <a href="#">mva_timesource_state</a> .

#### Remarks

The cost of retrieving the timesource state requires a function call that reads state kept in kernel host memory (i.e., no PCI bus reads).

### 5.2.4.4 `mva_open_mcast_stream ( void * in_addr, void * mcast_in_addr, uint16_t dest_port, mva_stream_t * mva_stream, int flags, MVA_OS_HANDLE * os_handle )`

Open a GigE Vision Multicast group stream for MVA acceleration. The Ethernet address must match the interface address of a Myri-10G network adapter (NIC) with MVA enabled.

This function is identical to [mva\\_open\\_stream](#), except that a Multicast group address is specified as an additional parameter. The multicast address is joined to the interface address specified by the `in_addr` parameter. GVSP packets with the specified multicast address will be accepted. If the multicast address is NULL, the function behaves exactly like [mva\\_open\\_stream](#).

**Parameters**

in	<i>in_addr</i>	Pointer to an interface address structure for the address of an open GigE Vision stream (from the SCDA register): <ul style="list-style-type: none"> <li>• IPv4: (struct in_addr *)</li> <li>• IPv6: (struct in6_addr *)</li> </ul>
in	<i>mcast_addr</i>	Pointer to a multicast address structure for the address that will receive GigE Vision frames. <ul style="list-style-type: none"> <li>• IPv4: (struct in_addr *)</li> <li>• IPv6: (struct in6_addr *)</li> </ul>
in	<i>dest_port</i>	Destination port for the stream (from SCP register).
in	<i>flags</i>	Flags are single bit values; so, binary or ( ) can be used to combine them. Possible values are: <ul style="list-style-type: none"> <li>• MVA_OPEN_IPV6</li> <li>• MVA_OPEN_ZEROLOSS</li> </ul>
out	<i>mva_stream</i>	MVA stream handle.
out	<i>os_handle</i>	OS-specific file descriptor which can be passed to poll() or select() to block on receive data available. For UNIX systems, this is a file descriptor, on Windows it is a HANDLE. Specify NULL if handle is not needed.

**Return values**

<i>EINVAL</i>	The <code>dest_addr</code> was not an interface address for a Myri-10G MVA-enabled network adapter, or the multicast address was not valid.
---------------	---

**Postcondition**

The MVA stream handle is valid and can be used by other functions.

**5.2.4.5 mva\_open\_stream ( void \* in\_addr, uint16\_t dest\_port, mva\_stream\_t \* mva\_stream, int flags, MVA\_OS\_HANDLE \* os\_handle )**

Open a GigE Vision stream for MVA acceleration.

This function opens a GigE Vision stream channel for acceleration. The address must match an Ethernet interface address of a Myri-10G network adapter (NIC) with MVA enabled.

By default, streams operate in "drop" mode, where the entire data block is dropped if one related packet is lost. If the MVA\_OPEN\_ZEROLOSS flag is specified, the NIC will make reasonable attempts to retrieve any missing packets.

**Parameters**

in	<i>in_addr</i>	Pointer to an interface address structure for the address of an open GigE Vision stream (from the SCDA register): <ul style="list-style-type: none"> <li>• IPv4: (struct in_addr *)</li> <li>• IPv6: (struct in6_addr *)</li> </ul>
in	<i>dest_port</i>	Destination port for the stream (from SCP register)
in	<i>flags</i>	Flags are single bit values; so, binary or ( ) can be used to combine them. Possible values are: <ul style="list-style-type: none"> <li>• MVA_OPEN_IPV6</li> <li>• MVA_OPEN_ZEROLOSS</li> </ul>
out	<i>mva_stream</i>	MVA stream handle.
out	<i>os_handle</i>	OS-specific file descriptor which can be passed to poll() or select() to block on receive data available. For UNIX systems, this is a file descriptor; on Windows it is a HANDLE. Specify NULL if handle is not needed.

**Return values**

<i>EINVAL</i>	The <i>dest_addr</i> was not an interface address for a Myri-10G MVA-enabled network adapter.
---------------	---

**Postcondition**

The MVA stream handle is valid and can be used by other functions.

**5.2.4.6 mva\_open\_stream\_pr ( void \* *in\_addr*, uint16\_t *dest\_port*, mva\_stream\_t \* *mva\_stream*, struct mva\_packet\_resend\_config \* *prc*, MVA\_OS\_HANDLE \* *os\_handle* )**

Open a GigE Vision stream for MVA acceleration with Packet Resend support.

This function opens a GigE Vision stream channel for acceleration. The address must match an Ethernet interface address of a Myri-10G network adapter (NIC) with MVA enabled.

By default, streams operate in "drop" mode, where the entire data block is dropped if one related packet is lost. If the MVA\_OPEN\_ZEROLOSS flag is specified, the NIC will make reasonable attempts to retrieve any missing packets. MVA\_OPEN\_RETURN\_INCOMPLETE can also be set to return incomplete streams.

**Parameters**

in	<i>in_addr</i>	Pointer to an interface address structure for the address of an open GigE Vision stream (from the SCDA register): <ul style="list-style-type: none"> <li>• IPv4: (struct in_addr *)</li> <li>• IPv6: (struct in6_addr *)</li> </ul>
in	<i>dest_port</i>	Destination port for the stream (from SCP register).
in	<i>prc</i>	The packet resend config object for the stream.
out	<i>mva_stream</i>	MVA stream handle.
out	<i>os_handle</i>	OS-specific file descriptor which can be passed to poll() or select() to block on receive data available. For UNIX systems, this is a file descriptor; on Windows it is a HANDLE. Specify NULL if handle is not needed.

**Return values**

<i>EINVAL</i>	The dest_addr was not an interface address for a Myri-10G MVA-enabled network adapter.
---------------	--

**Postcondition**

The MVA stream handle is valid and can be used by other functions.

### 5.3 Memory allocation and deallocation.

Functions related to MVA Memory allocation and deallocation.

#### Functions

- [mva\\_alloc](#) ([mva\\_stream\\_t](#) mva\_stream, [size\\_t](#) size, [mva\\_buf\\_t](#) \*buf)  
*Allocate MVA memory.*
- [mva\\_free](#) ([mva\\_buf\\_t](#) buf)  
*Free MVA memory.*

#### 5.3.1 Detailed Description

Functions related to MVA Memory allocation and deallocation.

#### 5.3.2 Function Documentation

##### 5.3.2.1 [mva\\_alloc](#) ( [mva\\_stream\\_t](#) mva\_stream, [size\\_t](#) size, [mva\\_buf\\_t](#) \* buf )

Allocate MVA memory.

This function allocates an MVA zero-copy buffer of the specified size and returns a pointer to the corresponding memory. MVA buffers are pinned in physical memory to allow direct access by the DMA engine of the Myricom network adapter. After a buffer has been allocated, it can be queued to MVA. See [mva\\_queue\\_buffer\(\)](#).

#### Parameters

in	<i>mva_stream</i>	An open MVA stream on which the buffer will be queued.
in	<i>size</i>	Size of the buffer to allocate.
out	<i>buf</i>	MVA buffer handle.

#### Return values

<i>0</i>	Success.
<i>EINVAL</i>	Invalid mva_stream or size parameter.
<i>ENOMEM</i>	Out of resources.

#### Postcondition

The MVA buffer handle is valid and can be enqueued on a stream with [mva\\_queue\\_buffer\(\)](#).

##### 5.3.2.2 [mva\\_free](#) ( [mva\\_buf\\_t](#) buf )

Free MVA memory.

This function frees memory previously allocated by [mva\\_alloc\(\)](#).

**Parameters**

in	<i>buf</i>	Buffer handle.
----	------------	----------------

**Return values**

0	Success.
---	----------

**Postcondition**

The MVA buffer can no longer be used.



## 5.4 Enqueuing and Receiving data blocks.

Functions related enqueuing and receiving GVSP data blocks.

### Data Structures

- struct [mva\\_block](#)
- struct [mva\\_block\\_status](#)
- struct [mva\\_info](#)
- struct [mva\\_stats](#)

### Macros

- #define [MVA\\_BLOCK\\_STATUS\\_NO\\_BLOCK\\_RECEIVED](#) 0x0
- #define [MVA\\_BLOCK\\_STATUS\\_BLOCK\\_DROPPED](#) 0x1
- #define [MVA\\_BLOCK\\_STATUS\\_BLOCK\\_RETURNED\\_COMPLETE](#) 0x2
- #define [MVA\\_BLOCK\\_STATUS\\_BLOCK\\_RETURNED\\_INCOMPLETE](#) 0x3

### Functions

- [mva\\_queue\\_buffer](#) (mva\_buf\_t buf)  
*Queue an MVA buffer.*
- [mva\\_poll\\_recv](#) (mva\_stream\_t mva\_stream, struct [mva\\_block](#) \*block, int timeout)  
*Wait for the next GVSP data block.*
- [mva\\_poll\\_recv\\_pr](#) (mva\_stream\_t mva\_stream, struct [mva\\_block](#) \*block, uint16\_t req\_id, struct [mva\\_block\\_status](#) \*block\_status)  
*Wait for the next GVSP data block.*
- [mva\\_info](#) (mva\_stream\_t mva\_stream, struct [mva\\_info](#) \*info)  
*Get device info.*
- [mva\\_get\\_stats](#) (mva\_stream\_t mva\_stream, struct [mva\\_stats](#) \*stats)  
*Get stats.*
- [mva\\_clear\\_stats](#) (mva\_stream\_t mva\_stream)  
*Clear stats.*

#### 5.4.1 Detailed Description

Functions related enqueuing and receiving GVSP data blocks.

#### 5.4.2 Macro Definition Documentation

##### 5.4.2.1 #define MVA\_BLOCK\_STATUS\_NO\_BLOCK\_RECEIVED 0x0

[mva\\_block\\_status](#) is used by [mva\\_poll\\_recv\\_pr](#).

### 5.4.3 Function Documentation

#### 5.4.3.1 `mva_clear_stats ( mva_stream_t mva_stream )`

Clear stats.

This function clears the various information, counters, and statistics that are returned by `mva_get_stats` for a given stream.

**Parameters**

in	<i>mva_stream</i>	MVA stream for which the statistics are cleared.
----	-------------------	--

**Return values**

0	Success.
---	----------

#### 5.4.3.2 `mva_get_stats ( mva_stream_t mva_stream, struct mva_stats * stats )`

Get stats.

This function returns various information, counters, and statistics for a given MVA stream.

**Parameters**

in	<i>mva_stream</i>	MVA stream for which statistics are retrieved.
out	<i>info</i>	MVA stats structure.

**Return values**

0	Success.
---	----------

#### 5.4.3.3 `mva_info ( mva_stream_t mva_stream, struct mva_info * info )`

Get device info.

This function returns various information, counters, and statistics for a given MVA stream.

**Parameters**

in	<i>mva_stream</i>	The MVA stream for which statistics are retrieved.
out	<i>info</i>	MVA stream info structure.

**Return values**

0	Success.
---	----------

#### 5.4.3.4 `mva_poll_recv ( mva_stream_t mva_stream, struct mva_block * block, int timeout )`

Wait for the next GVSP data block.

Wait for the next GVSP data block on the specified stream. If the next data block has not been completely received, the function waits until completion or until the timeout expires. On success, the corresponding `mva_block` structure is updated with the information specific to the received GVSP payload.

**Parameters**

in	<i>mva_stream</i>	MVA stream.
in, out	<i>block</i>	MVA block structure allocated by the application and initialized with information about received block upon completion.
in	<i>timeout</i>	Timeout in msecs. If this parameter is 0, the function will return immediately. If the parameter is negative, the function will block indefinitely.

**Return values**

0	Success.
<i>EINVAL</i>	Invalid <code>mva_stream</code> .
<i>EAGAIN</i>	Timeout expired and no block was available. Fields in <code>mva_info</code> will be invalid.
<i>EINTR</i>	Signal was received. Fields in <code>mva_block</code> will be invalid.
<i>ENOMEM</i>	The queued buffer was not large enough to contain the block.

**Postcondition**

The buffer handle can be used to free or re-queue the buffer on the stream. After the buffer is freed or re-queued, the `payload_data` pointer in the block is no longer valid.

**5.4.3.5 `mva_poll_rcv_pr ( mva_stream_t mva_stream, struct mva_block * block, uint16_t req_id, struct mva_block_status * block_status )`**

Wait for the next GVSP data block.

Wait for the next GVSP data block on the specified stream. If the next data block has not been completely received, the function waits until completion or until the timeout expires. On success, the corresponding `mva_block` structure is updated with the information specific to the received GVSP payload.

**Parameters**

in	<i>mva_stream</i>	MVA stream.
in, out	<i>block</i>	MVA block structure allocated by the application and initialized with information about the received block upon completion.
in	<i>req_id</i>	The first request ID to be issued to the device if a packet resend request is issued.
out	<i>block_status</i>	The resulting status returned from the function.

**Return values**

0	Success.
<i>EINVAL</i>	Invalid <code>mva_stream</code> .
<i>EAGAIN</i>	Timeout expired and no block was available. Fields in <code>mva_info</code> will be invalid.
<i>EINTR</i>	Signal was received. Fields in <code>mva_block</code> will be invalid.
<i>ENOMEM</i>	The queued buffer was not large enough to contain the block
<i>ETIMEDOUT</i>	A packet resend timeout expired and no block was available.
<i>EIO</i>	The block was out of order and dropped based on the <code>BOOO</code> parameter.

**Postcondition**

The buffer handle can be used to free or re-queue the buffer on the stream. After the buffer is freed or re-queued, the `payload_data` pointer in the block is no longer valid.

**5.4.3.6 `mva_queue_buffer ( mva_buf_t buf )`**

Queue an MVA buffer.

Queue an MVA buffer for receiving a GVSP data block from a specific MVA stream. Once a buffer is enqueued, it should not be accessed until it is returned by `mva_poll_recv()`. Buffers are automatically dequeued when `mva_poll_recv()` returns.

**Parameters**

<code>in</code>	<i>buffer</i>	Buffer handle.
-----------------	---------------	----------------

**Return values**

<code>0</code>	Success.
<i>EINVAL</i>	Invalid buffer address.
<i>EAGAIN</i>	Max amount of queue ahead reached.

**Postcondition**

Buffer is queued to receive GVSP block data from stream from which the buffer was allocated.

**Remarks**

Once the buffer has been queued, it should not be read or written until it is returned by `mva_poll_recv()`.

## 5.5 Manage the name of an MVA adapter.

Functions related to configuring the name of an MVA adapter.

### Functions

- [mva\\_set\\_name](#) (const char \*which\_adapter, const char \*name)  
*Set the name of an adapter.*
- [mva\\_get\\_name](#) (const char \*which\_adapter, const char \*\*name)  
*Get the name of an adapter.*
- [mva\\_reset\\_name](#) (const char \*which\_adapter)  
*Reset the name of an adapter.*
- [mva\\_reset\\_all\\_name](#) (void)  
*Reset the names of all adapters.*

### 5.5.1 Detailed Description

Functions related to configuring the name of an MVA adapter.

### 5.5.2 Function Documentation

#### 5.5.2.1 `mva_get_name ( const char * which_adapter, const char ** name )`

Get the name of an adapter.

This function returns the name of an MVA adapter.

#### Parameters

in	<i>which_adapter</i>	This specifies a card by its MAC or SN based on the format S=477237 for serial numbers or M=00:60:dd:43:a3:61 for MAC addresses.
out	<i>name</i>	The name of the adapter.

#### Return values

0	Success.
---	----------

#### 5.5.2.2 `mva_reset_all_name ( void )`

Reset the names of all adapters.

This function resets the name of all MVA adapters to the default of <adapter-model>.<SN>.

#### Return values

0	Success.
---	----------

**5.5.2.3 mva\_reset\_name ( const char \* which\_adapter )**

Reset the name of an adapter.

This function resets the name of an MVA adapter to the default of <adapter-model>.<SN>.

**Parameters**

in	<i>which_adapter</i>	This specifies a card by its MAC or SN based on the format S=477237 for serial numbers or M=00:60:dd:43:a3:61 for MAC addresses.
----	----------------------	--

**Return values**

0	Success.
---	----------

**5.5.2.4 mva\_set\_name ( const char \* which\_adapter, const char \* name )**

Set the name of an adapter.

This function sets the name of the adapter.

**Parameters**

in	<i>which_adapter</i>	This specifies a card by its MAC or SN based on the format S=477237 for serial numbers or M=00:60:dd:43:a3:61 for MAC addresses.
in	<i>name</i>	The name to set for the adapter.

**Return values**

0	Success.
---	----------

## 5.6 Manage metrics.

Functions related to managing MVA metrics.

### Functions

- [mva\\_create\\_metrics](#) (mva\_metrics\_t \*metrics, uint32\_t adapter\_metrics\_period, uint32\_t stream\_metrics\_period, uint32\_t metrics\_server\_ipv4, uint16\_t metrics\_server\_port)  
*Create metrics context.*
- [mva\\_destroy\\_metrics](#) (mva\_metrics\_t metrics)  
*Destroy metrics context.*
- [mva\\_enable\\_metrics](#) (mva\_metrics\_t metrics)  
*Enable metrics.*
- [mva\\_disable\\_metrics](#) (mva\_metrics\_t metrics)  
*Disable metrics.*
- [mva\\_add\\_metrics](#) (mva\_metrics\_t metrics, mva\_stream\_t stream)  
*Add metrics to a stream.*
- [mva\\_remove\\_metrics](#) (mva\_stream\_t stream)  
*Removes metrics from a stream.*
- [mva\\_start\\_metrics](#) (mva\_metrics\_t metrics, uint64\_t cpu)  
*Starts the metrics collector.*
- [mva\\_stop\\_metrics](#) (mva\_metrics\_t metrics)  
*Stops the metrics collector.*

### 5.6.1 Detailed Description

Functions related to managing MVA metrics.

### 5.6.2 Function Documentation

#### 5.6.2.1 mva\_add\_metrics ( mva\_metrics\_t metrics, mva\_stream\_t stream )

Add metrics to a stream.

This function adds metrics to the specified stream.

#### Parameters

in	<i>metrics</i>	
in	<i>stream</i>	

#### Return values

0	Success.
---	----------

**5.6.2.2 mva\_create\_metrics ( mva\_metrics\_t \* metrics, uint32\_t adapter\_metrics\_period, uint32\_t stream\_metrics\_period, uint32\_t metrics\_server\_ipv4, uint16\_t metrics\_server\_port )**

Create metrics context.

This function creates the content for the mva metrics add-on.

**Parameters**

in	<i>adapter_metrics_period</i>	
in	<i>stream_metrics_period</i>	
in	<i>metrics_server_ipv4</i>	
in	<i>metrics_server_port</i>	
out	<i>metrics</i>	

**Return values**

	0	Success.
--	---	----------

**5.6.2.3 mva\_destroy\_metrics ( mva\_metrics\_t metrics )**

Destroy metrics context.

This function destroys the context to the MVA metrics add-on.

**Parameters**

in	<i>metrics</i>	
----	----------------	--

**Return values**

	0	Success.
--	---	----------

**5.6.2.4 mva\_disable\_metrics ( mva\_metrics\_t metrics )**

Disable metrics.

This function disables metrics for all streams to which metrics has been added.

**Parameters**

in	<i>metrics</i>	
----	----------------	--

**Return values**

	0	Success.
--	---	----------



**5.6.2.5 mva\_enable\_metrics ( mva\_metrics\_t metrics )**

Enable metrics.

This function enables metrics for all streams to which metrics has been added.

**Parameters**

in	<i>metrics</i>	
----	----------------	--

**Return values**

	0	Success.
--	---	----------

**5.6.2.6 mva\_remove\_metrics ( mva\_stream\_t stream )**

Removes metrics from a stream.

This function removes metrics from the specified stream.

**Parameters**

in	<i>stream</i>	
----	---------------	--

**Return values**

	0	Success.
--	---	----------

**5.6.2.7 mva\_start\_metrics ( mva\_metrics\_t metrics, uint64\_t cpu )**

Starts the metrics collector.

This function starts the metrics collector.

**Parameters**

in	<i>metrics</i>	
in	<i>cpu</i>	

**Return values**

	0	Success.
--	---	----------

**5.6.2.8 mva\_stop\_metrics ( mva\_metrics\_t metrics )**

Stops the metrics collector.

This function stops the metrics collector.

**Parameters**

in	<i>metrics</i>	
----	----------------	--

**Return values**

0	Success.
---	----------

## 5.7 manage trace

Functions related to managing the tracing functionality of MVA.

### Functions

- [mva\\_create\\_trace](#) (mva\_trace\_t \*trace, uint32\_t trace\_server\_ipv4, uint16\_t trace\_server\_port)  
*Create the mva\_trace addon component.*
- [mva\\_destroy\\_trace](#) (mva\_trace\_t trace)  
*Destroy the mva\_trace addon component context.*
- [mva\\_enable\\_trace](#) (mva\_trace\_t trace)  
*Enables the mva\_trace addon component.*
- [mva\\_disable\\_trace](#) (mva\_trace\_t trace)  
*Disables the mva\_trace addon component.*
- [mva\\_add\\_trace](#) (mva\_trace\_t trace, mva\_stream\_t stream)  
*Add a stream to the mva\_trace addon component.*
- [mva\\_remove\\_trace](#) (mva\_stream\_t stream)  
*Removes a trace component from a stream.*
- [mva\\_start\\_trace](#) (mva\_trace\_t trace, uint64\_t cpu)  
*Starts the trace addon component.*
- [mva\\_stop\\_trace](#) (mva\_trace\_t trace)  
*Stops the trace addon component.*

### 5.7.1 Detailed Description

Functions related to managing the tracing functionality of MVA.

### 5.7.2 Function Documentation

#### 5.7.2.1 mva\_add\_trace ( mva\_trace\_t trace, mva\_stream\_t stream )

Add a stream to the mva\_trace addon component.

This function adds a stream to the mva\_trace addon component.

#### Parameters

in	<i>trace</i>	
in	<i>stream</i>	

#### Return values

0	Success.
---	----------

**5.7.2.2 mva\_create\_trace ( mva\_trace\_t \* trace, uint32\_t trace\_server\_ipv4, uint16\_t trace\_server\_port )**

Create the mva\_trace addon component.

This function creates the mva\_trace addon component.

**Parameters**

in	<i>trace_server_ipv4</i>	
in	<i>trace_server_port</i>	
out	<i>trace</i>	

**Return values**

0	Success.
---	----------

**5.7.2.3 mva\_destroy\_trace ( mva\_trace\_t trace )**

Destroy the mva\_trace addon component context.

This function destroys the mva\_trace addon component context.

**Parameters**

in	<i>trace</i>	
----	--------------	--

**Return values**

0	Success.
---	----------

**5.7.2.4 mva\_disable\_trace ( mva\_trace\_t trace )**

Disables the mva\_trace addon component.

This function disables the mva\_trace addon component.

**Parameters**

in	<i>trace</i>	
----	--------------	--

**Return values**

0	Success.
---	----------

**5.7.2.5 mva\_enable\_trace ( mva\_trace\_t trace )**

Enables the mva\_trace addon component.

This function enables the mva\_trace addon component.

**Parameters**

in	trace	
----	-------	--

**Return values**

0	Success.
---	----------

**5.7.2.6 mva\_remove\_trace ( mva\_stream\_t stream )**

Removes a trace component from a stream.

This function removes a trace component from a stream.

**Parameters**

in	stream	
----	--------	--

**Return values**

0	Success.
---	----------

**5.7.2.7 mva\_start\_trace ( mva\_trace\_t trace, uint64\_t cpu )**

Starts the trace addon component.

This function starts the trace addon component.

**Parameters**

in	trace	
in	cpu	

**Return values**

0	Success.
---	----------

**5.7.2.8 mva\_stop\_trace ( mva\_trace\_t trace )**

Stops the trace addon component.

This function stops the trace addon component.

**Parameters**

in	trace	
----	-------	--

**Return values**

0	Success.
---	----------

## 5.8 receive worker thread.

MVA receive worker thread (RWDT) component.

### Data Structures

- struct [mva\\_receive\\_stream\\_worker\\_status](#)

### Typedefs

- typedef void(\* [mva\\_get\\_request\\_id\\_cb](#) )(mva\_stream\_t mva\_stream, uint16\_t \*starting\_request\_id, uint16\_t \*max\_requests)

### Functions

- [mva\\_create\\_rcv\\_worker](#) (mva\_stream\_t mva\_stream, int cpu, [mva\\_get\\_request\\_id\\_cb](#) callback\_func)  
*Create an MVA receive worker for the specified stream.*
- [mva\\_destroy\\_rcv\\_worker](#) (mva\_stream\_t mva\_stream)  
*Destroy an MVA receive worker for the specified stream.*
- [mva\\_block\\_rcv\\_worker](#) (mva\_stream\_t mva\_stream)  
*Block the receive worker thread.*
- [mva\\_unblock\\_rcv\\_worker](#) (mva\_stream\_t mva\_stream)  
*Unblock the Receive Worker thread.*
- [mva\\_stop\\_rcv\\_worker](#) (mva\_stream\_t mva\_stream)  
*Stop the receive worker thread.*
- [mva\\_start\\_rcv\\_worker](#) (mva\_stream\_t mva\_stream)  
*Start the receive worker thread.*
- [mva\\_get\\_status\\_rcv\\_worker](#) (mva\_stream\_t mva\_stream, struct [mva\\_receive\\_stream\\_worker\\_status](#) \*rcv\_worker\_status)  
*Get the status of the receive worker thread.*
- [mva\\_get\\_camera\\_id\\_stream](#) (mva\_stream\_t mva\_stream, uint64\_t \*camera\_id)  
*Get Camera ID.*
- [mva\\_drop\\_packet](#) (mva\_stream\_t mva\_stream, uint32\_t block, uint32\_t packet)  
*Drop packet.*

### 5.8.1 Detailed Description

MVA receive worker thread (RWDT) component.

### 5.8.2 Typedef Documentation

#### 5.8.2.1 typedef void(\* [mva\\_get\\_request\\_id\\_cb](#))(mva\_stream\_t mva\_stream, uint16\_t \*starting\_request\_id, uint16\_t \*max\_requests)

This is the callback used by the calling application for the worker thread to obtain the starting request ID and maximum number of requests.

**5.8.3 Function Documentation**

**5.8.3.1 mva\_block\_recv\_worker ( mva\_stream\_t mva\_stream )**

Block the receive worker thread.

This function allows the user to block processing of the stream worker thread for a given stream. If this function is called and the worker thread is not created, it will return EINVAL.

**Parameters**

in	<i>mva_stream</i>	MVA stream to block.
----	-------------------	----------------------

**Return values**

<i>0</i>	Success. #retval EINVAL The receive worker thread has not been created.
----------	---

**5.8.3.2 mva\_create\_recv\_worker ( mva\_stream\_t mva\_stream, int cpu, mva\_get\_request\_id\_cb callback\_func )**

Create an MVA receive worker for the specified stream.

This function creates a receive worker for the specified stream. If the stream is not opened, the function will return non-zero.

**Parameters**

in	<i>mva_stream</i>	The stream for which to create a worker thread.
in	<i>cpu</i>	The CPU to which the worker thread is affinitized.
in	<i>callback_func</i>	The callback function used by the worker thread to obtain the next request ID.

**Return values**

<i>0</i>	Success.
<i>ENOENT</i>	The mva_stream is not open.
<i>EINVAL</i>	Invalid callback function.
<i>EINVAL</i>	Invalid CPU.
<i>ENOMEM</i>	Failed to allocate the receive worker.

**5.8.3.3 mva\_destroy\_recv\_worker ( mva\_stream\_t mva\_stream )**

Destroy an MVA receive worker for the specified stream.

Stop and destroy a created receive worker thread.

**Parameters**

in	<i>mva_stream</i>	The stream where the worker thread will be destroyed.
----	-------------------	---

**Return values**

<i>0</i>	Success.
----------	----------

**5.8.3.4 mva\_drop\_packet ( mva\_stream\_t mva\_stream, uint32\_t block, uint32\_t packet )**

Drop packet.

This function drops a packet on the the given stream.

**Parameters**

in	<i>mva_stream</i>	MVA stream associated with the drop.
in	<i>block</i>	Block ID containing the packet to drop.
in	<i>packet</i>	ID of the packet to drop.

**Return values**

0	Success.
---	----------

**5.8.3.5 mva\_get\_camera\_id\_stream ( mva\_stream\_t mva\_stream, uint64\_t \* camera\_id )**

Get Camera ID.

This function returns the camera ID for the given stream.

**Parameters**

in	<i>mva_stream</i>	The MVA stream from which to obtain the camera ID.
----	-------------------	--

**Return values**

0	Success.
---	----------

**5.8.3.6 mva\_get\_status\_rcv\_worker ( mva\_stream\_t mva\_stream, struct mva\_receive\_stream\_worker\_status \* rcv\_worker\_status )**

Get the status of the receive worker thread.

This function allows the user to get the state of the stream worker thread.

**Parameters**

in	<i>mva_stream</i>	MVA stream to start.
----	-------------------	----------------------

**Return values**

0	Success.
---	----------

**5.8.3.7 mva\_start\_rcv\_worker ( mva\_stream\_t mva\_stream )**

Start the receive worker thread.

This function allows the user to start processing the stream worker thread for a given stream. If this function is called and the worker thread is not created, it will return EINVAL.



**Parameters**

in	<i>mva_stream</i>	MVA stream to start.
----	-------------------	----------------------

**Return values**

0	Success. #retval EINVAL The receive worker thread has not been created.
---	---

**5.8.3.8 mva\_stop\_rcv\_worker ( mva\_stream\_t mva\_stream )**

Stop the receive worker thread.

This function allows the user to stop processing the stream worker thread for a given stream. If this function is called and the worker thread is not created, it will return EINVAL.

**Parameters**

in	<i>mva_stream</i>	MVA stream to stop.
----	-------------------	---------------------

**Return values**

0	Success. #retval EINVAL The receive worker thread has not been created.
---	---

**5.8.3.9 mva\_unblock\_rcv\_worker ( mva\_stream\_t mva\_stream )**

Unblock the Receive Worker thread.

This function allows the user to unblock processing of the stream worker thread for a given stream. If this function is called and the worker thread is not created, it will return EINVAL.

**Parameters**

in	<i>mva_stream</i>	MVA stream to unblock.
----	-------------------	------------------------

**Return values**

0	Success. #retval EINVAL The receive worker thread has not been created.
---	---



## Chapter 6

# Namespace Documentation

### 6.1 mva Namespace Reference

#### 6.1.1 Detailed Description

MachineVisionAccelerator

**Author**

Myricom, Inc.



## Chapter 7

# Data Structure Documentation

### 7.1 mva\_block Struct Reference

#### Data Fields

- `uint16_t` `payload_type`
- `uint32_t` `payload_length`
- `void *` `payload_data`
- `mva_buf_t` `mva_buf`
- `uint16_t` `status`
- `uint16_t` `block_id`
- `uint64_t` `timestamp`
- `uint64_t` `nsecs`
- `uint32_t` `crc`
- `union` {
  - `struct` {
    - `uint32_t` `pixel_type`
    - `uint32_t` `size_x`
    - `uint32_t` `size_y`
    - `uint32_t` `offset_x`
    - `uint32_t` `offset_y`
    - `uint32_t` `padding_x`
    - `uint32_t` `padding_y`
    - `uint32_t` `trailer_size_y`
  - `image`
  - `struct` {
    - `uint64_t` `payload_data_size`
  - `raw`
  - `struct` {
    - `uint64_t` `payload_data_size`
    - `char` `filename` [128]
  - `file`
  - `struct` {
    - `uint64_t` `data_payload_length`

```
    } chunk
  struct {
    uint8_t chunk_flag
    uint32_t pixel_type
    uint32_t size_x
    uint32_t size_y
    uint32_t offset_x
    uint32_t offset_y
    uint32_t padding_x
    uint32_t padding_y
    uint32_t trailer_size_y
    uint64_t data_payload_length
    uint32_t chunk_layout_id
  } extended_chunk
} meta
```

### 7.1.1 Detailed Description

GVSP block.

### 7.1.2 Field Documentation

#### 7.1.2.1 `uint16_t mva_block::block_id`

Block ID.

#### 7.1.2.2 `uint32_t mva_block::crc`

Block CRC (for internal testing).

#### 7.1.2.3 `union { ... } mva_block::meta`

Payload type specific metadata.

#### 7.1.2.4 `mva_buf_t mva_block::mva_buf`

MVA buffer to use with [mva\\_queue\\_buffer\(\)](#) or [mva\\_free\(\)](#).

#### 7.1.2.5 `uint64_t mva_block::nsecs`

If SYNC NIC in use, time since Epoch in nanoseconds of the arrival of the Leader packet; else 0.

**7.1.2.6 void\* mva\_block::payload\_data**

Pointer to block data.

**7.1.2.7 uint32\_t mva\_block::payload\_length**

Total length of block data.

**7.1.2.8 uint16\_t mva\_block::payload\_type**

Payload type.

**7.1.2.9 uint16\_t mva\_block::status**

Status of block transaction.

**7.1.2.10 uint64\_t mva\_block::timestamp**

Timestamp.

## 7.2 mva\_block\_status Struct Reference

### Data Fields

- uint32\_t [status](#)
- uint64\_t [block\\_id](#)
- uint32\_t [dropped\\_packet\\_id](#)
- uint32\_t [resend\\_request](#)
- uint16\_t [next\\_req\\_id](#)

### 7.2.1 Field Documentation

**7.2.1.1 uint64\_t mva\_block\_status::block\_id**

The ID of the returned block.

**7.2.1.2 uint32\_t mva\_block\_status::dropped\_packet\_id**

If the block was incomplete, the dropped packet ID; otherwise UINT32\_MAX.

**7.2.1.3 uint16\_t mva\_block\_status::next\_req\_id**

The next request ID.

#### 7.2.1.4 `uint32_t mva_block_status::resend_request`

The number of times a packet resend request was performed.

#### 7.2.1.5 `uint32_t mva_block_status::status`

The status of the returned block.

### 7.3 `mva_info` Struct Reference

#### Data Fields

- `uint32_t port_link_up`
- `uint32_t port_active`
- `uint64_t blocks_dropped`
- `uint64_t blocks_received`

#### 7.3.1 Detailed Description

Various information related to MVA stream.

#### 7.3.2 Field Documentation

##### 7.3.2.1 `uint64_t mva_info::blocks_dropped`

Number of GVSP data blocks dropped or with a non-zero status.

##### 7.3.2.2 `uint64_t mva_info::blocks_received`

Number of GVSP data blocks received with status equal to 0.

##### 7.3.2.3 `uint32_t mva_info::port_active`

Active port on failover NIC.

##### 7.3.2.4 `uint32_t mva_info::port_link_up`

Bitmap of ports with link up.



## 7.4 mva\_packet\_resend\_config Struct Reference

### Data Fields

- int `flags`
- uint32\_t `block_timeout`
- uint32\_t `resend_timeout`
- uint32\_t `resend_retries`
- uint32\_t `block_out_of_order`
- uint64\_t `camera_id`
- uint32\_t `camera_ipv4`
- uint16\_t `camera_port`
- uint16\_t `stream_channel_id`

### 7.4.1 Field Documentation

#### 7.4.1.1 uint32\_t mva\_packet\_resend\_config::block\_out\_of\_order

The Block Out of Order (BOOO) flag.

#### 7.4.1.2 uint32\_t mva\_packet\_resend\_config::block\_timeout

The time, in milliseconds, the API waits for a resend after a first incomplete block.

#### 7.4.1.3 uint64\_t mva\_packet\_resend\_config::camera\_id

Unique camera app identification field.

#### 7.4.1.4 uint32\_t mva\_packet\_resend\_config::camera\_ipv4

The Unicast IPv4 Address to which the packet resend requests are issued.

#### 7.4.1.5 uint16\_t mva\_packet\_resend\_config::camera\_port

The port to use when issuing the packet resend requests.

#### 7.4.1.6 int mva\_packet\_resend\_config::flags

The MVA open flags.

#### 7.4.1.7 uint32\_t mva\_packet\_resend\_config::resend\_retries

The maximum number of packet resend retries allowed.

#### 7.4.1.8 `uint32_t mva_packet_resend_config::resend_timeout`

The time, in milliseconds, the API waits for a single packet resend request response.

#### 7.4.1.9 `uint16_t mva_packet_resend_config::stream_channel_id`

The stream ID for the stream.

## 7.5 `mva_receive_stream_worker_status` Struct Reference

### Data Fields

- `uint32_t thread_state`
- `uint32_t thread_mode`
- `uint32_t block_processing_mode`

## 7.6 `mva_stats` Struct Reference

### Data Fields

- `uint64_t blocks_dropped`
- `uint64_t blocked_blocks_dropped`
- `uint64_t blocks_returned_complete`
- `uint64_t blocks_returned_incomplete`
- `uint64_t blocks_not_available`
- `uint64_t resend_requests`
- `uint64_t resend_requests_successful`
- `uint64_t resend_requests_failed`
- `uint64_t block_time_outs`
- `uint64_t resend_time_outs`
- `uint64_t max_resend_retries`
- `uint64_t buf_allocated`
- `uint64_t buf_freed`
- `uint64_t buf_enqueued`
- `uint64_t stream_stopped`
- `uint64_t stream_started`
- `uint64_t stream_blocked`
- `uint64_t stream_unblocked`
- `uint64_t callbacks_invoked`

### 7.6.1 Detailed Description

MVA stats

## 7.6.2 Field Documentation

### 7.6.2.1 `uint64_t mva_stats::block_time_outs`

Number of block timeouts.

### 7.6.2.2 `uint64_t mva_stats::blocked_blocks_dropped`

Number of GVSP data blocks dropped when in the blocked stream state.

### 7.6.2.3 `uint64_t mva_stats::blocks_dropped`

Number of GVSP data blocks dropped or with a non-zero status.

### 7.6.2.4 `uint64_t mva_stats::blocks_not_available`

Number of GVSP data blocks returned as not available.

### 7.6.2.5 `uint64_t mva_stats::blocks_returned_complete`

Number of GVSP data blocks returned as completed.

### 7.6.2.6 `uint64_t mva_stats::blocks_returned_incomplete`

Number of GVSP data blocks returned as incomplete.

### 7.6.2.7 `uint64_t mva_stats::buf_allocated`

Number of buffers allocated.

### 7.6.2.8 `uint64_t mva_stats::buf_enqueued`

Number of buffers enqueued.

### 7.6.2.9 `uint64_t mva_stats::buf_freed`

Number of buffers freed.

### 7.6.2.10 `uint64_t mva_stats::callbacks_invoked`

Number of times the get starting request ID callback function was invoked.

**7.6.2.11 uint64\_t mva\_stats::max\_resend\_retries**

Number of times the resend retries counter was exceeded.

**7.6.2.12 uint64\_t mva\_stats::resend\_requests**

Number of GVCP Packet Resend requests issued.

**7.6.2.13 uint64\_t mva\_stats::resend\_requests\_failed**

Number of GVCP Packet Resend requests failed.

**7.6.2.14 uint64\_t mva\_stats::resend\_requests\_successful**

Number of GVCP Packet Resend requests successful.

**7.6.2.15 uint64\_t mva\_stats::resend\_time\_outs**

Number of resend timeouts.

**7.6.2.16 uint64\_t mva\_stats::stream\_blocked**

Number of RWDT block states entered.

**7.6.2.17 uint64\_t mva\_stats::stream\_started**

Number of RWDT start states entered.

**7.6.2.18 uint64\_t mva\_stats::stream\_stopped**

Number of RWDT stop states entered.

**7.6.2.19 uint64\_t mva\_stats::stream\_unblocked**

Number of RWDT unblock states entered.

# Index

- block\_id
  - mva\_block, 40
  - mva\_block\_status, 41
- block\_out\_of\_order
  - mva\_packet\_resend\_config, 43
- block\_time\_outs
  - mva\_stats, 45
- block\_timeout
  - mva\_packet\_resend\_config, 43
- blocked\_blocks\_dropped
  - mva\_stats, 45
- blocks\_dropped
  - mva\_info, 42
  - mva\_stats, 45
- blocks\_not\_available
  - mva\_stats, 45
- blocks\_received
  - mva\_info, 42
- blocks\_returned\_complete
  - mva\_stats, 45
- blocks\_returned\_incomplete
  - mva\_stats, 45
- buf\_allocated
  - mva\_stats, 45
- buf\_enqueued
  - mva\_stats, 45
- buf\_freed
  - mva\_stats, 45
- callbacks\_invoked
  - mva\_stats, 45
- camera\_id
  - mva\_packet\_resend\_config, 43
- camera\_ipv4
  - mva\_packet\_resend\_config, 43
- camera\_port
  - mva\_packet\_resend\_config, 43
- crc
  - mva\_block, 40
- dropped\_packet\_id
  - mva\_block\_status, 41
- Enqueuing and Receiving data blocks., 19
  - mva\_clear\_stats, 20
  - mva\_get\_stats, 20
  - mva\_info, 20
  - mva\_poll\_recv, 20
  - mva\_poll\_recv\_pr, 21
  - mva\_queue\_buffer, 22
- flags
  - mva\_packet\_resend\_config, 43
- Initialization, 9
  - MVA\_TIMESOURCE\_EXT\_FAILED, 10
  - MVA\_TIMESOURCE\_EXT\_SYNCED, 10
  - MVA\_TIMESOURCE\_EXT\_UNSYNCED, 10
  - MVA\_TIMESOURCE\_LOCAL, 10
  - MVA\_VERSION\_API, 9
  - mva\_init, 10
  - mva\_link\_state, 10
  - mva\_timesource\_state, 10
- MVA\_TIMESOURCE\_EXT\_FAILED
  - Initialization, 10
- MVA\_TIMESOURCE\_EXT\_SYNCED
  - Initialization, 10
- MVA\_TIMESOURCE\_EXT\_UNSYNCED
  - Initialization, 10
- MVA\_TIMESOURCE\_LOCAL
  - Initialization, 10
- MVA\_OPEN\_IPV6
  - Streams, 12
- MVA\_OPEN\_ZEROLOSS
  - Streams, 12
- MVA\_VERSION\_API
  - Initialization, 9
- Manage metrics., 25
  - mva\_add\_metrics, 25
  - mva\_create\_metrics, 25
  - mva\_destroy\_metrics, 26
  - mva\_disable\_metrics, 26
  - mva\_enable\_metrics, 26
  - mva\_remove\_metrics, 27

- mva\_start\_metrics, [27](#)
- mva\_stop\_metrics, [27](#)
- Manage the name of an MVA adapter., [23](#)
  - mva\_get\_name, [23](#)
  - mva\_reset\_all\_name, [23](#)
  - mva\_reset\_name, [23](#)
  - mva\_set\_name, [24](#)
- manage trace, [29](#)
  - mva\_add\_trace, [29](#)
  - mva\_create\_trace, [29](#)
  - mva\_destroy\_trace, [30](#)
  - mva\_disable\_trace, [30](#)
  - mva\_enable\_trace, [30](#)
  - mva\_remove\_trace, [31](#)
  - mva\_start\_trace, [31](#)
  - mva\_stop\_trace, [31](#)
- max\_resend\_retries
  - mva\_stats, [45](#)
- Memory allocation and deallocation., [17](#)
  - mva\_alloc, [17](#)
  - mva\_free, [17](#)
- meta
  - mva\_block, [40](#)
- mva, [37](#)
- mva\_add\_metrics
  - Manage metrics., [25](#)
- mva\_add\_trace
  - manage trace, [29](#)
- mva\_alloc
  - Memory allocation and deallocation., [17](#)
- mva\_block, [39](#)
  - block\_id, [40](#)
  - crc, [40](#)
  - meta, [40](#)
  - mva\_buf, [40](#)
  - nsecs, [40](#)
  - payload\_data, [40](#)
  - payload\_length, [41](#)
  - payload\_type, [41](#)
  - status, [41](#)
  - timestamp, [41](#)
- mva\_block\_recv\_worker
  - receive worker thread., [33](#)
- mva\_block\_status, [41](#)
  - block\_id, [41](#)
  - dropped\_packet\_id, [41](#)
  - next\_req\_id, [41](#)
  - resend\_request, [41](#)
  - status, [42](#)
- mva\_buf
  - mva\_block, [40](#)
- mva\_clear\_stats
  - Enqueuing and Receiving data blocks., [20](#)
- mva\_close\_stream
  - Streams, [13](#)
- mva\_create\_metrics
  - Manage metrics., [25](#)
- mva\_create\_recv\_worker
  - receive worker thread., [33](#)
- mva\_create\_trace
  - manage trace, [29](#)
- mva\_destroy\_metrics
  - Manage metrics., [26](#)
- mva\_destroy\_recv\_worker
  - receive worker thread., [33](#)
- mva\_destroy\_trace
  - manage trace, [30](#)
- mva\_disable\_metrics
  - Manage metrics., [26](#)
- mva\_disable\_trace
  - manage trace, [30](#)
- mva\_drop\_packet
  - receive worker thread., [33](#)
- mva\_enable\_metrics
  - Manage metrics., [26](#)
- mva\_enable\_trace
  - manage trace, [30](#)
- mva\_free
  - Memory allocation and deallocation., [17](#)
- mva\_get\_camera\_id\_stream
  - receive worker thread., [34](#)
- mva\_get\_link\_state
  - Streams, [13](#)
- mva\_get\_name
  - Manage the name of an MVA adapter., [23](#)
- mva\_get\_request\_id\_cb
  - receive worker thread., [32](#)
- mva\_get\_stats
  - Enqueuing and Receiving data blocks., [20](#)
- mva\_get\_status\_recv\_worker
  - receive worker thread., [34](#)
- mva\_get\_timesource\_state
  - Streams, [13](#)
- mva\_info, [42](#)
  - blocks\_dropped, [42](#)
  - blocks\_received, [42](#)
  - Enqueuing and Receiving data blocks., [20](#)
  - port\_active, [42](#)
  - port\_link\_up, [42](#)
- mva\_init

- Initialization, 10
- mva\_link\_state
  - Initialization, 10
- mva\_open\_mcast\_stream
  - Streams, 13
- mva\_open\_stream
  - Streams, 14
- mva\_open\_stream\_pr
  - Streams, 15
- mva\_packet\_resend\_config, 43
  - block\_out\_of\_order, 43
  - block\_timeout, 43
  - camera\_id, 43
  - camera\_ipv4, 43
  - camera\_port, 43
  - flags, 43
  - resend\_retries, 43
  - resend\_timeout, 43
  - stream\_channel\_id, 44
- mva\_poll\_recv
  - Enqueuing and Receiving data blocks., 20
- mva\_poll\_recv\_pr
  - Enqueuing and Receiving data blocks., 21
- mva\_queue\_buffer
  - Enqueuing and Receiving data blocks., 22
- mva\_receive\_stream\_worker\_status, 44
- mva\_remove\_metrics
  - Manage metrics., 27
- mva\_remove\_trace
  - manage trace, 31
- mva\_reset\_all\_name
  - Manage the name of an MVA adapter., 23
- mva\_reset\_name
  - Manage the name of an MVA adapter., 23
- mva\_set\_name
  - Manage the name of an MVA adapter., 24
- mva\_start\_metrics
  - Manage metrics., 27
- mva\_start\_recv\_worker
  - receive worker thread., 34
- mva\_start\_trace
  - manage trace, 31
- mva\_stats, 44
  - block\_time\_outs, 45
  - blocked\_blocks\_dropped, 45
  - blocks\_dropped, 45
  - blocks\_not\_available, 45
  - blocks\_returned\_complete, 45
  - blocks\_returned\_incomplete, 45
  - buf\_allocated, 45
  - buf\_enqueued, 45
  - buf\_freed, 45
  - callbacks\_invoked, 45
  - max\_resend\_retries, 45
  - resend\_requests, 46
  - resend\_requests\_failed, 46
  - resend\_requests\_successful, 46
  - resend\_time\_outs, 46
  - stream\_blocked, 46
  - stream\_started, 46
  - stream\_stopped, 46
  - stream\_unblocked, 46
- mva\_stop\_metrics
  - Manage metrics., 27
- mva\_stop\_recv\_worker
  - receive worker thread., 35
- mva\_stop\_trace
  - manage trace, 31
- mva\_stream\_t
  - Streams, 12
- mva\_timesource\_state
  - Initialization, 10
- mva\_unblock\_recv\_worker
  - receive worker thread., 35
- next\_req\_id
  - mva\_block\_status, 41
- nsecs
  - mva\_block, 40
- payload\_data
  - mva\_block, 40
- payload\_length
  - mva\_block, 41
- payload\_type
  - mva\_block, 41
- port\_active
  - mva\_info, 42
- port\_link\_up
  - mva\_info, 42
- receive worker thread., 32
  - mva\_block\_recv\_worker, 33
  - mva\_create\_recv\_worker, 33
  - mva\_destroy\_recv\_worker, 33
  - mva\_drop\_packet, 33
  - mva\_get\_camera\_id\_stream, 34
  - mva\_get\_request\_id\_cb, 32
  - mva\_get\_status\_recv\_worker, 34
  - mva\_start\_recv\_worker, 34
  - mva\_stop\_recv\_worker, 35

- mva\_unblock\_rcv\_worker, 35
- resend\_request
  - mva\_block\_status, 41
- resend\_requests
  - mva\_stats, 46
- resend\_requests\_failed
  - mva\_stats, 46
- resend\_requests\_successful
  - mva\_stats, 46
- resend\_retries
  - mva\_packet\_resend\_config, 43
- resend\_time\_outs
  - mva\_stats, 46
- resend\_timeout
  - mva\_packet\_resend\_config, 43
  
- status
  - mva\_block, 41
  - mva\_block\_status, 42
- stream\_blocked
  - mva\_stats, 46
- stream\_channel\_id
  - mva\_packet\_resend\_config, 44
- stream\_started
  - mva\_stats, 46
- stream\_stopped
  - mva\_stats, 46
- stream\_unblocked
  - mva\_stats, 46
- Streams, 11
  - MVA\_OPEN\_IPV6, 12
  - MVA\_OPEN\_ZEROLOSS, 12
  - mva\_close\_stream, 13
  - mva\_get\_link\_state, 13
  - mva\_get\_timesource\_state, 13
  - mva\_open\_mcast\_stream, 13
  - mva\_open\_stream, 14
  - mva\_open\_stream\_pr, 15
  - mva\_stream\_t, 12
  
- timestamp
  - mva\_block, 41