# Contents

1 MVA Documentation .............................. 1
   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

2 Module Index ............................... 3
   2.1 API Reference ................................ 3

3 Namespace Index .......................... 5
   3.1 Namespace List ................................ 5

4 Data Structure Index ...................... 7
   4.1 Data Structures .............................. 7

5 Module Documentation .................... 9
   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 ............. 11
         5.2.2.1 MVA_OPEN_IPV6 ......................... 11
         5.2.2.2 MVA_OPEN_ZEROLOSS ................. 11
      5.2.3 Typedef Documentation ..................... 12
         5.2.3.1 mva_stream_t .......................... 12
      5.2.4 Function Documentation .................... 12
         5.2.4.1 mva_close_stream ...................... 12
         5.2.4.2 mva_get_link_state ................... 12
         5.2.4.3 mva_get_timesource_state .......... 12
         5.2.4.4 mva_open_mcast_stream ............. 13
         5.2.4.5 mva_open_stream ..................... 13
5.3 Memory allocation and deallocation ........................................ 15
  5.3.1 Detailed Description .................................................. 15
  5.3.2 Function Documentation ............................................. 15
    5.3.2.1 mva_alloc .................................................. 15
    5.3.2.2 mva_free ................................................ 15

5.4 Enqueuing and Receiving data blocks .................................. 17
  5.4.1 Detailed Description ................................................ 17
  5.4.2 Function Documentation ............................................ 17
    5.4.2.1 mva_info .................................................. 17
    5.4.2.2 mva_poll_recv ............................................ 17
    5.4.2.3 mva_queue_buffer ......................................... 18

6 Namespace Documentation .................................................. 19
  6.1 mva Namespace Reference ............................................. 19
    6.1.1 Detailed Description ........................................... 19

7 Data Structure Documentation ............................................. 21
  7.1 mva_block Struct Reference ........................................... 21
    7.1.1 Detailed Description ........................................... 22
    7.1.2 Field Documentation ........................................... 22
      7.1.2.1 block_id ................................................ 22
      7.1.2.2 crc .................................................. 22
      7.1.2.3 meta .................................................. 22
      7.1.2.4 mva_buf .............................................. 22
      7.1.2.5 nsecs ................................................ 22
      7.1.2.6 payload_data .......................................... 22
      7.1.2.7 payload_length ....................................... 22
      7.1.2.8 payload_type .......................................... 22
      7.1.2.9 status ................................................. 23
      7.1.2.10 timestamp ............................................ 23
  7.2 mva_info Struct Reference ........................................... 23
    7.2.1 Detailed Description ........................................... 23
    7.2.2 Field Documentation ........................................... 23
      7.2.2.1 blocks_dropped ....................................... 23
      7.2.2.2 blocks_received ...................................... 23
      7.2.2.3 port_active ........................................... 23
      7.2.2.4 port_link_up ......................................... 23

Index ................................................................. 24
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 content.

1.2 Terminology

Readers of this document should be familiar with the AIA GigE Vision Specification version 1.0 and the GenICam™ Standard. See www.machinevisiononline.org and 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 adapters (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 SC-Px registers. A matching MVA stream is created by passing the same parameters to the mva_open_stream() function. 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 .................................... 15
- Enqueuing and Receiving data blocks ................................. 17
Chapter 3

Namespace Index

3.1 Namespace List

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

mva ............................................................... 19
Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

- mva_block .................................................. 21
- mva_info .................................................. 23
Chapter 5

Module Documentation

5.1 Initialization

MVA Initialization function.

Macros

- \#define MVA\_VERSION\_API 0x0102
  
  MVA API version number (16 bits)

Enumerations

- \texttt{enum mva\_link\_state} { MVA\_LINK\_DOWN = 0, MVA\_LINK\_UP = 1 }
- \texttt{enum mva\_timesource\_state} { MVA\_TIMESOURCE\_LOCAL = 0, MVA\_TIMESOURCE\_EXT\_UNSYNCED, MVA\_TIMESOURCE\_EXT\_SYNCED, MVA\_TIMESOURCE\_EXT\_FAILED }

Functions

- \texttt{mva\_init (uint16\_t api\_version)}

  \textit{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 time-source or if its use was explicitly disabled.
- `MVA_TIMESOURCE_EXT_UNSYNCHED`  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 api_version )

Initialize MVA library.

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

**Parameters**

| in | api_version | Must always be MVA_VERSION_API. |

**Return values**

| EINV | Library already initialized with different API version. Incompatible library/API version. |
| ENXIO | Incompatible driver/library. |
| ENODEV | No driver found. |
| ENOMEM | 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.

Macros

- #define MVA_OPEN_IPV6 0x1
- #define MVA_OPEN_ZEROLOSS 0x2

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_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 the 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_OPEN_IPV6 0x1

Address specified to mva_open_stream() is an IPv6 address (default: IPv4).

5.2.2.2 #define MVA_OPEN_ZEROLOSS 0x2

Stream operates in "zero loss" 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

| in       | mva_stream | 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 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

| strm  | Stream handle |
| state | Returns one of MVA_LINK_DOWN or MVA_LINK_UP |

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

| strm  | Stream handle |
| state | Returns one of mva_timesource_state |

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 the 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   | in_addr | Pointer to an interface address structure for the address of an open GigE Vision stream (from the SCDA register):
|      |        | • IPv4: (struct in_addr *)
|      |        | • IPv6: (struct in6_addr *)
| in   | mcast_addr | Pointer to a multicast address structure for the address to receive GigE Vision frames on.
|      |        | • IPv4: (struct in_addr *)
|      |        | • IPv6: (struct in6_addr *)
| in   | dest_port | Destination port for the stream (from SCP register)
| in   | flags | Flags are single bit values so binary or() can be used to combine them. Possible values:
|      |        | • MVA_OPEN_IPV6.
|      |        | • MVA_OPEN_ZEROLOSS.
| out  | mva_stream | MVA stream handle.
| out  | os_handle | 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

| EINVAL | dest_addr 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 the 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

<table>
<thead>
<tr>
<th>in</th>
<th>in_addr</th>
<th>Pointer to an interface address structure for the address of an open GigE Vision stream (from the SCDA register):</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• IPv4: (struct in_addr *)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• IPv6: (struct in6_addr *)</td>
</tr>
<tr>
<td>in</td>
<td>dest_port</td>
<td>Destination port for the stream (from SCP register)</td>
</tr>
<tr>
<td>in</td>
<td>flags</td>
<td>Flags are single bit values so binary or() can be used to combine them. Possible values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MVA_OPEN_IPV6.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MVA_OPEN_ZEROLOSS.</td>
</tr>
<tr>
<td>out</td>
<td>mva_stream</td>
<td>MVA stream handle.</td>
</tr>
<tr>
<td>out</td>
<td>os_handle</td>
<td>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.</td>
</tr>
</tbody>
</table>

Return values

| EINVAL | 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 a 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 | mva_stream | An open MVA stream on which the buffer will be queued. |
| in | size       | Size of the buffer to allocate. |
| out | buf       | MVA buffer handle. |

Return values

<table>
<thead>
<tr>
<th></th>
<th>Success.</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINVAL</td>
<td>Invalid mva_stream or size parameter.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>Out of resources</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>in</th>
<th>buf</th>
<th>Buffer handle.</th>
</tr>
</thead>
</table>

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_info

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_info (mva_stream_t mva_stream, struct mva_info *info)
  Get device info.

5.4.1 Detailed Description

Functions related enqueuing and receiving GVSP data blocks.

5.4.2 Function Documentation

5.4.2.1 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

<table>
<thead>
<tr>
<th>in</th>
<th>mva_stream</th>
<th>MVA stream to get statistics for.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>info</td>
<td>MVA stream info structure.</td>
</tr>
</tbody>
</table>

Return values

| 0  | Success. |

5.4.2.2 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 wait 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

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

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>Invalid mva_stream</td>
</tr>
<tr>
<td>EAGAIN</td>
<td>Timeout expired and no block was available. Fields in mva_info will be invalid.</td>
</tr>
<tr>
<td>EINTR</td>
<td>Signal was received. Fields in mva_block will be invalid.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>The queued buffer was not large enough to contain the block</td>
</tr>
</tbody>
</table>

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 block is no longer valid.

5.4.2.3 mva_queue_buffer ( mva_buff_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

| in | buffer | Buffer handle. |

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>Invalid buffer address</td>
</tr>
<tr>
<td>EAGAIN</td>
<td>Max amount of queue ahead reached</td>
</tr>
</tbody>
</table>

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().
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
  } 21
7.1.1 Detailed Description

GVSP block

7.1.2 Field Documentation

7.1.2.1 uint16 mva_block::block_id

Block id

7.1.2.2 uint32 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 mva_block::nsecs

If SYNC NIC in use, time since Epoc 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 mva_block::payload_length

Total length of block data

7.1.2.8 uint16 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_info Struct Reference

#### Data Fields

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

#### 7.2.1 Detailed Description

Various information related to MVA stream.

#### 7.2.2 Field Documentation

7.2.2.1 `uint64_t mva_info::blocks_dropped`

Number of GVSP Data Blocks dropped or having non-zero status

7.2.2.2 `uint64_t mva_info::blocks_received`

Number of GVSP Data Blocks received with status equal 0

7.2.2.3 `uint32_t mva_info::port_active`

Active port on failover NIC

7.2.2.4 `uint32_t mva_info::port_link_up`

Bitmap of ports with link up
Index

block_id
  mva_block, 22
blocks_dropped
  mva_info, 23
blocks_received
  mva_info, 23
crc
  mva_block, 22

Enqueuing and Receiving data blocks, 17
  mva_info, 17
  mva_poll_recv, 17
  mva_queue_buffer, 18

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, 11
MVA_OPEN_ZEROLOSS
  Streams, 11
MVA_VERSION_API
  Initialization, 9
Memory allocation and deallocation, 15
  mva_alloc, 15
  mva_free, 15
meta
  mva_block, 22
mva, 19
mva_alloc
  Memory allocation and deallocation, 15
  mva_block, 21
  block_id, 22
  crc, 22
  meta, 22
  mva_buf, 22
  nsecs, 22
  payload_data, 22
  payload_length, 22
  payload_type, 22
  status, 22
  timestamp, 23
mva_buf
  mva_block, 22
mva_close_stream
  Streams, 12
mva_free
  Memory allocation and deallocation, 15
mva_get_link_state
  Streams, 12
mva_get_timesource_state
  Streams, 12
mva_info, 23
  blocks_dropped, 23
  blocks_received, 23
  Enqueuing and Receiving data blocks, 17
  port_active, 23
  port_link_up, 23
mva_init
  Initialization, 10
mva_link_state
  Initialization, 10
mva_open_mcast_stream
  Streams, 13
mva_open_stream
  Streams, 13
mva_poll_recv
Enqueuing and Receiving data blocks, 17

mva_queue_buffer
    Enqueuing and Receiving data blocks, 18

mva_stream_t
    Streams, 12

mva_timesource_state
    Initialization, 10

nsecs
    mva_block, 22

payload_data
    mva_block, 22

payload_length
    mva_block, 22

payload_type
    mva_block, 22

port_active
    mva_info, 23

port_link_up
    mva_info, 23

status
    mva_block, 22

Streams, 11
    MVA_OPEN_IPV6, 11
    MVA_OPEN_ZEROLOSS, 11
    mva_close_stream, 12
    mva_get_link_state, 12
    mva_get_timesource_state, 12
    mva_open_mcast_stream, 13
    mva_open_stream, 13
    mva_stream_t, 12

timestamp
    mva_block, 23