Multicore Communications API (MCAPI)

MCAPI provides for communication and synchronization between processing cores in embedded systems. Get the specification at www.multicore-association.org.

- Functions that end in "_i" are non-blocking/asynchronous and return immediately.
- **[n.n]** refers to the sections in the MCAPI API 2.015 specification.
- MCAPI_IN and MCAPI_OUT distinguish between input and output parameters.
- In function prototypes, qualifiers are shown in blue, **function names are bold**, types are shown in green, and *parameters are italic*.

THE Multicore ASSOCIATION

Data Types and Endpoint Attributes

Data Types [2.16]

mcapi_domain_t	MCAPI domains
mcapi_node_t	MCAPI nodes
mcapi_port_t	MCAPI ports
mcapi_endpoint_t	Create and manage endpoints
mcapi_pktchan_send_hndl_t mcapi_pktchan_recv_hndl_t	Send/Receive packets to/from a connected packet channel
mcapi_sclchan_send_hndl_t mcapi_sclchan_recv_hndl_t	Send/Receive scalars to/from a connected scalar channel
mcapi_uint*n*_t *n* can be 64, 32, 16, or 8	Unsigned 64-, 32-, 16-, and 8-bit scalars.
mcapi_request_t	The state of a pending non-blocking MCAPI transaction [See 2.5.4]
mcapi_status_t	Error/Status code of MCAPI API call
mcapi_timeout_t	Duration **mcapi_wait_{any}** will block before reporting a timeout

Endpoint Attributes [2.16.13]

mcapi_endp_attr_max_payload_size_t Defines the maximum payload size.
mcapi_endp_attr_buffer_type_t Defines endpoint buffer type. Currently only FIFO exists.
mcapi_endp_attr_memory_type_t Defines memory's locality, whether local, shared, or remote.
mcapi_endp_attr_num_priorities_t Defines the number of endpoint priorities.
mcapi_endp_attr_priority_t Defines endpoint priority.
mcapi_endp_attr_num_send_buffers_t Number of send buffers at current endpoint priority level.
mcapi_endp_attr_num_recv_buffers_t Contains the number of receive buffers available.
mcapi_endp_attr_status_t Contains flags used to query the status of an endpoint.
mcapi_endp_attr_timeout_t Timeout value for blocking send and receive functions.

Other Data Types

Types defined in **[2.16.14], mca.h**

```
typedef int mca_int_t;
typedef char mca_int8_t;                  // 8-bit signed
typedef short mca_int16_t;                // 16-bit signed
typedef int mca_int32_t;                  // 32-bit signed
typedef long long mca_int64_t;            // 64-bit signed
typedef unsigned int mca_uint_t;
typedef unsigned char mca_uint8_t;        // 8-bit unsigned
typedef unsigned short mca_uint16_t;      // 16-bit unsigned
typedef unsigned int mca_uint32_t;        // 32-bit unsigned
typedef unsigned long long mca_uint64_t;  // 64-bit unsigned
typedef unsigned char mca_boolean_t;
typedef unsigned int mca_node_t;
typedef unsigned int mca_status_t;
typedef unsigned int mca_timeout_t;
typedef unsigned int mca_domain_t;
```

Types defined in **mcapi.h**

```
typedef mca_int_t mcapi_int_t;
typedef mca_int8_t mcapi_int8_t;          // 8-bit signed
typedef mca_int16_t mcapi_int16_t;        // 16-bit signed
typedef mca_int32_t mcapi_int32_t;        // 32-bit signed
typedef mca_int64_t mcapi_int64_t;        // 64-bit signed
typedef mca_uint_t mcapi_uint_t;
typedef mca_uint8_t mcapi_uint8_t;        // 8-bit unsigned
typedef mca_uint16_t mcapi_uint16_t;      // 16-bit unsigned
typedef mca_uint32_t mcapi_uint32_t;      // 32-bit unsigned
typedef mca_uint64_t mcapi_uint64_t;      // 64-bit unsigned
typedef mca_boolean_t mcapi_boolean_t;
typedef unsigned int mcapi_priority_t;
```

Additional types and enums are defined in header files mcapi_impl_spec.h, mca_impl_spec.h, mcapi.h, and mcapi_v1000_to_v2000.h.

Also see the **Multicore Task Management API (MTAPI)** for task parallelism on embedded devices containing symmetric or asymmetric multicore processors, available in Q3 2012.

Learn more at www.multicore-association.org.

Concepts

Domains [2.1]
Domains are comprised of one or more MCAPI nodes in a multicore topology, and used for routing purposes.

Nodes [2.2]
Nodes can be one of a process, thread, processor, hardware accelerator, or instance of an operating system. A node ID is specified in the call to **mcapi_initialize()**.

Endpoints [2.3]
Endpoints are socket-like communication-termination points created with **mcapi_endpoint_create()** function.

Channels [2.4]
Channels provide point-to-point FIFO connections between a pair of endpoints by first establishing a connection.

Packet Channels [2.7]
Packet channels provide a method to transmit data between endpoints. Packet channels are a connected, unidirectional method to send buffered data and deliver in a FIFO manner.

Scalar Channels [2.8]
Scalar channels provide a method to transmit scalars between endpoints. Like packet channels, scalar channels are unidirectional and deliver data in a FIFO manner. The scalar functions come in 8-bit, 16-bit, 32-bit and 64-bit variants.

Messages [2.6]
Messages provide a flexible method to transmit data between endpoints without first establishing a connection. Messages may be sent with different priorities, on a per-message basis.

General Functions

Initialization and introspection functions.

Initialize MCAPI Environment [3.2.1]
```
void mcapi_initialize(
    MCAPI_IN mcapi_domain_t domain_id,
    MCAPI_IN mcapi_node_t node_id,
    MCAPI_IN mcapi_node_attributes_t
        * mcapi_node_attributes,
    MCAPI_IN mcapi_param_t * mcapi_parameters,
    MCAPI_OUT mcapi_info_t * mcapi_info,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Finalize MCAPI Environment [3.2.2]
```
void mcapi_finalize(
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Get Domain ID [3.2.3]
```
mcapi_domain_t mcapi_domain_id_get(
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Get Node ID [3.2.4]
```
mcapi_node_t mcapi_node_id_get(
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Initialize Structure Values [3.2.5]
```
void mcapi_node_init_attributes(
    MCAPI_OUT mcapi_node_attributes_t
        * mcapi_node_attributes,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Change Default Values [3.2.6]
```
void mcapi_node_set_attribute(
    MCAPI_OUT mcapi_node_attributes_t
        * mcapi_node_attributes,
    MCAPI_IN mcapi_uint_t attribute_num,
    MCAPI_IN void * attribute,
    MCAPI_IN size_t attribute_size,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Query Node Attributes [3.2.7]
```
void mcapi_node_get_attribute(
    MCAPI_IN mcapi_domain_t domain_id,
    MCAPI_IN mcapi_node_t node_id,
    MCAPI_IN mcapi_uint_t attribute_num,
    MCAPI_OUT void * attribute,
    MCAPI_IN size_t attribute_size,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Error and Status Codes [2.16.5]

Error or Status Code	Description
MCAPI_SUCCESS	Operation was successful
MCAPI_PENDING	Operation pending w/out errors
MCAPI_TIMEOUT	Operation timed out
MCAPI_ERR_PARAMETER	Incorrect parameter
MCAPI_ERR_DOMAIN_INVALID	Parameter not a valid domain
MCAPI_ERR_NODE_INVALID	Parameter not a valid node
MCAPI_ERR_NODE_INITFAILED	Node could not be initialized
MCAPI_ERR_NODE_INITIALIZED	Node already initialized
MCAPI_ERR_NODE_NOTINIT	Node not initialized
MCAPI_ERR_NODE_FINALFAILED	Node could not be finalized
MCAPI_ERR_PORT_INVALID	Parameter not a valid port
MCAPI_ERR_ENDP_INVALID	Invalid endpoint descriptor
MCAPI_ERR_ENDP_EXISTS	Endpoint is already created
MCAPI_ERR_ENDP_NOTOWNER	Endpoint can only be deleted by its creator
MCAPI_ERR_ENDP_REMOTE	Operation only allowed on the node local endpoints
MCAPI_ERR_ATTR_INCOMPATIBLE	Cannot connect endpoints with incompatible attributes
MCAPI_ERR_ATTR_SIZE	Incorrect attribute size
MCAPI_ERR_ATTR_NUM	Incorrect attribute number
MCAPI_ERR_ATTR_VALUE	Incorrect attribute value
MCAPI_ERR_ATTR_NOTSUPPORTED	Attribute not supported
MCAPI_ERR_ATTR_READONLY	Attribute is read-only
MCAPI_ERR_MSG_SIZE	Message size > max size allowed

Error or Status Code	Description
MCAPI_ERR_MSG_TRUNCATED	Message size exceeds buffer size
MCAPI_ERR_CHAN_OPEN	A channel is open, certain operations are not allowed
MCAPI_ERR_CHAN_TYPE	Attempt to open a channel on an endpoint connected with a different channel type
MCAPI_ERR_CHAN_DIRECTION	Attempt to open a send handle on a port that was connected as a receiver, or vice versa
MCAPI_ERR_CHAN_CONNECTED	A channel connection has already been established
MCAPI_ERR_CHAN_OPENPENDING	An open request is pending
MCAPI_ERR_CHAN_CLOSEPENDING	A close request is pending.
MCAPI_ERR_CHAN_NOTOPEN	Channel not open/cannot be closed
MCAPI_ERR_CHAN_INVALID	Argument is not a channel handle
MCAPI_ERR_PKT_SIZE	Packet size > max size allowed
MCAPI_ERR_TRANSMISSION	Transmission failure
MCAPI_ERR_PRIORITY	Incorrect priority level
MCAPI_ERR_BUF_INVALID	Not a valid buffer descriptor
MCAPI_ERR_MEM_LIMIT	Out of memory
MCAPI_ERR_REQUEST_INVALID	Arg. not a valid request handle
MCAPI_ERR_REQUEST_LIMIT	Out of request handles
MCAPI_ERR_REQUEST_CANCELLED	The request was already canceled
MCAPI_ERR_WAIT_PENDING	A wait is pending
MCAPI_ERR_GENERAL	Other error conditions
MCAPI_STATUSCODE_END	This should always be last

Endpoint Functions

This section describes API functions that create, delete, get, and modify endpoints.

Create Endpoint [3.3.1]

```
mcapi_endpoint_t mcapi_endpoint_create(
    MCAPI_IN mcapi_port_t port_id,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Delete Endpoint [3.3.4]

```
void mcapi_endpoint_delete(
    MCAPI_IN mcapi_endpoint_t endpoint,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Get Endpoint Identifier [3.3.2, 3.3.3]

```
void mcapi_endpoint_get_i(
    MCAPI_IN mcapi_domain_t domain_id,
    MCAPI_IN mcapi_node_t node_id,
    MCAPI_IN mcapi_port_t port_id,
    MCAPI_OUT mcapi_endpoint_t * endpoint,
    MCAPI_OUT mcapi_request_t * request,
    MCAPI_OUT mcapi_status_t * mcapi_status);

mcapi_endpoint_t mcapi_endpoint_get(
    MCAPI_IN mcapi_domain_t domain_id,
    MCAPI_IN mcapi_node_t node_id,
    MCAPI_IN mcapi_port_t port_id,
    MCAPI_IN mcapi_timeout_t timeout,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Query Endpoint Attribute [3.3.5]

```
void mcapi_endpoint_get_attribute(
    MCAPI_IN mcapi_endpoint_t endpoint,
    MCAPI_IN mcapi_uint_t attribute_num,
    MCAPI_OUT void * attribute,
    MCAPI_IN size_t attribute_size,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Assign Endpoint Attribute [3.3.6]

```
void mcapi_endpoint_set_attribute(
    MCAPI_IN mcapi_endpoint_t endpoint,
    MCAPI_IN mcapi_uint_t attribute_num,
    MCAPI_IN const void * attribute,
    MCAPI_IN size_t attribute_size,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Message Functions

Send Message [3.4.1, 3.4.2]

```
void mcapi_msg_send_i(
    MCAPI_IN mcapi_endpoint_t send_endpoint,
    MCAPI_IN mcapi_endpoint_t receive_endpoint,
    MCAPI_IN void * buffer,
    MCAPI_IN size_t buffer_size,
    MCAPI_IN mcapi_priority_t priority,
    MCAPI_OUT mcapi_request_t * request,
    MCAPI_OUT mcapi_status_t * mcapi_status);

void mcapi_msg_send(
    MCAPI_IN mcapi_endpoint_t send_endpoint,
    MCAPI_IN mcapi_endpoint_t receive_endpoint,
    MCAPI_IN void * buffer,
    MCAPI_IN size_t buffer_size,
    MCAPI_IN mcapi_priority_t priority,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Receive Message [3.4.3, 3.4.4]

```
void mcapi_msg_recv_i(
    MCAPI_IN mcapi_endpoint_t receive_endpoint,
    MCAPI_OUT void * buffer,
    MCAPI_IN size_t buffer_size,
    MCAPI_OUT mcapi_request_t * request,
    MCAPI_OUT mcapi_status_t * mcapi_status);

void mcapi_msg_recv(
    MCAPI_IN mcapi_endpoint_t receive_endpoint,
    MCAPI_OUT void * buffer,
    MCAPI_IN size_t buffer_size,
    MCAPI_OUT size_t * received_size,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Check Message Availability [3.4.5]

```
mcapi_uint_t mcapi_msg_available(
    MCAPI_IN mcapi_endpoint_t receive_endpoint,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Packet Channel Functions

Connect Endpoints [3.5.1]

```
void mcapi_pktchan_connect_i(
    MCAPI_IN mcapi_endpoint_t send_endpoint,
    MCAPI_IN mcapi_endpoint_t receive_endpoint,
    MCAPI_OUT mcapi_request_t * request,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Open Receive Side of Packet [3.5.2]

```
void mcapi_pktchan_recv_open_i(
    MCAPI_OUT mcapi_pktchan_recv_hndl_t * receive_handle,
    MCAPI_IN mcapi_endpoint_t receive_endpoint,
    MCAPI_OUT mcapi_request_t * request,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Open Send Side of Packet [3.5.3]

```
void mcapi_pktchan_send_open_i(
    MCAPI_OUT mcapi_pktchan_send_hndl_t * send_handle,
    MCAPI_IN mcapi_endpoint_t send_endpoint,
    MCAPI_OUT mcapi_request_t * request,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Send Packet [3.5.4, 3.5.5]

```
void mcapi_pktchan_send_i(
    MCAPI_IN mcapi_pktchan_send_hndl_t send_handle,
    MCAPI_IN void * buffer,
    MCAPI_IN size_t size,
    MCAPI_OUT mcapi_request_t * request,
    MCAPI_OUT mcapi_status_t * mcapi_status);

void mcapi_pktchan_send(
    MCAPI_IN mcapi_pktchan_send_hndl_t send_handle,
    MCAPI_IN void * buffer,
    MCAPI_IN size_t size,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Check Packet Availability [3.5.8]

```
mcapi_uint_t mcapi_pktchan_available(
    MCAPI_IN mcapi_pktchan_recv_hndl_t receive_handle,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Receive Packet [3.5.6 , 3.5.7]

```
void mcapi_pktchan_recv_i(
    MCAPI_IN mcapi_pktchan_recv_hndl_t receive_handle,
    MCAPI_OUT void ** buffer,
    MCAPI_OUT mcapi_request_t * request,
    MCAPI_OUT mcapi_status_t * mcapi_status);

void mcapi_pktchan_recv(
    MCAPI_IN mcapi_pktchan_recv_hndl_t receive_handle,
    MCAPI_OUT void ** buffer,
    MCAPI_OUT size_t * received_size,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Release Buffer [3.5.9]

```
void mcapi_pktchan_release(
    MCAPI_IN void * buffer,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Check Buffer Release [3.5.10]

```
mcapi_boolean_t mcapi_pktchan_release_test(
    MCAPI_IN void * buffer,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Close Receive Side of Packet [3.5.11]

```
void mcapi_pktchan_recv_close_i(
    MCAPI_IN mcapi_pktchan_recv_hndl_t receive_handle,
    MCAPI_OUT mcapi_request_t * request,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Close Send Side of Packet [3.5.12]

```
void mcapi_pktchan_send_close_i(
    MCAPI_IN mcapi_pktchan_send_hndl_t send_handle,
    MCAPI_OUT mcapi_request_t * request,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Scalar Channel Functions

MCAPI scalar channels are used to transfer 8-bit, 16-bit, 32-bit and 64-bit scalars on a connected channel.

Connect Endpoints [3.6.1]

```
void mcapi_sclchan_connect_i(
    MCAPI_IN mcapi_endpoint_t send_endpoint,
    MCAPI_IN mcapi_endpoint_t receive_endpoint,
    MCAPI_OUT mcapi_request_t * request,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Open Receiver and Synchronize [3.6.2]

```
void mcapi_sclchan_recv_open_i(
    MCAPI_OUT mcapi_sclchan_recv_hndl_t * receive_handle,
    MCAPI_IN mcapi_endpoint_t receive_endpoint,
    MCAPI_OUT mcapi_request_t * request,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Open Sender and Synchronize [3.6.3]

```
void mcapi_sclchan_send_open_i(
    MCAPI_OUT mcapi_sclchan_send_hndl_t * send_handle,
    MCAPI_IN mcapi_endpoint_t send_endpoint,
    MCAPI_OUT mcapi_request_t * request,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Send Scalar [3.6.4-7]

```
void mcapi_sclchan_send_uint{64|32|16|8}(
    MCAPI_IN mcapi_sclchan_send_hndl_t send_handle,
    MCAPI_IN mcapi_uint64_t dataword,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Receive Scalar [3.6.8-11]

```
mcapi_uint{64|32|16|8}_t
    mcapi_sclchan_recv_uint{64|32|16|8}(
    MCAPI_IN mcapi_sclchan_recv_hndl_t receive_handle,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Check Scalar Availability [3.6.12]

```
mcapi_uint_t mcapi_sclchan_available(
    MCAPI_IN mcapi_sclchan_recv_hndl_t receive_handle,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Close Receiver [3.6.13]

```
void mcapi_sclchan_recv_close_i(
    MCAPI_IN mcapi_sclchan_recv_hndl_t receive_handle,
    MCAPI_OUT mcapi_request_t * request,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Close Sender [3.6.14]

```
void mcapi_sclchan_send_close_i(
    MCAPI_IN mcapi_sclchan_send_hndl_t send_handle,
    MCAPI_OUT mcapi_request_t * request,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Non-Blocking, Connectionless Message, Packet, and Channel Functions

Wait for Completion of Operation [3.7.2]

```
mcapi_boolean_t mcapi_wait(
    MCAPI_IN mcapi_request_t * request,
    MCAPI_OUT size_t * size,
    MCAPI_IN mcapi_timeout_t timeout,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Check for Non-blocking Ops Completion [3.7.1]

```
mcapi_boolean_t mcapi_test(
    MCAPI_IN mcapi_request_t * request,
    MCAPI_OUT size_t * size,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Cancel Non-blocking Ops [3.7.4]

```
void mcapi_cancel(
    MCAPI_IN mcapi_request_t * request,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Wait for Completion of Ops List [3.7.3]

```
mcapi_uint_t mcapi_wait_any(
    MCAPI_IN size_t number,
    MCAPI_IN mcapi_request_t * requests,
    MCAPI_OUT size_t * size,
    MCAPI_IN mcapi_timeout_t timeout,
    MCAPI_OUT mcapi_status_t * mcapi_status);
```

Support Function

Display MCAPI Status Message [3.8.1]

```
char* mcapi_display_status(
    MCAPI_IN mcapi_status_t mcapi_status,
    MCAPI_OUT char * status_message, MCAPI_IN size_t size);
```

THE Multicore ASSOCIATION™

The Multicore Association (MCA) is an open membership organization that includes leading companies implementing products that embrace multicore technology.