
EmberZNet 3.0 API Change Summary

February 16, 2007

EM250

EM2420

ember

Ember Corporation
343 Congress Street
Boston MA 02210
617.951.0200
www.ember.com



wireless semiconductor solutions

Copyright © 2007 by Ember Corporation

All rights reserved.

The information in this document is subject to change without notice. The statements, configurations, technical data, and recommendations in this document are believed to be accurate and reliable but are presented without express or implied warranty. Users must take full responsibility for their applications of any products specified in this document. The information in this document is the property of Ember Corporation.

Title, ownership, and all rights in copyrights, patents, trademarks, trade secrets and other intellectual property rights in the Ember Proprietary Products and any copy, portion, or modification thereof, shall not transfer to Purchaser or its customers and shall remain in Ember and its licensors.

No source code rights are granted to Purchaser or its customers with respect to all Ember Application Software. Purchaser agrees not to copy, modify, alter, translate, decompile, disassemble, or reverse engineer the Ember Hardware (including without limitation any embedded software) or attempt to disable any security devices or codes incorporated in the Ember Hardware. Purchaser shall not alter, remove, or obscure any printed or displayed legal notices contained on or in the Ember Hardware.

Ember, Ember Enabled, EmberZNet, InSight, and the Ember logo are trademarks of Ember Corporation.

All other trademarks are the property of their respective holders.

ember

Ember Corporation
343 Congress Street
Boston MA 02210
617.951.0200
www.ember.com



wireless semiconductor solutions

Introduction	4
Forming and joining networks	4
Sending and receiving messages	5
Note on Bindings	10
Aggregation	11
Disable relay	11
New storage parameters	12
Commercial Security API	12
Ember Counters	12
Removed API's	13

Introduction

This document is intended to provide an overview of changes to the EmberZNet API between EmberZNet 2.5 and EmberZNet 3.0.

Forming and joining networks

Extended PAN ID

ZigBee has added an 8 byte extended PAN id to facilitate provisioning and PAN id conflict detection. The extended PAN id is now included in the beacon payload, following the existing 3 bytes.

EmberNetworkParameters struct now contains an 8 byte extended PAN id in addition to the 2 byte PAN id. Warning: your application **must** set this value, if only to zero it out. If it doesn't, it is likely to be initialized with random garbage which will lead to unexpected behavior.

emberFormNetwork() stores the extended PAN id to the node data token. If a value of all zeroes is passed, a random value is used. In production applications it is strongly recommended that the random XPID is used. Using a fixed value (such as the EUI64 of the coordinator) can easily lead to XPID conflicts if another network running the same application is nearby or if the coordinator is used to commission two different neighboring networks.

emberJoinNetwork() now joins to a network based on the supplied extended pan id, rather than the pan id. For back-compatibility, if an extended PAN id of all zeroes is supplied, it will join based on the short PAN id, and the extended PAN id of the network will be retrieved from the beacon and stored to the node data token.

Also, the API call for emberJoinNetwork() has changed. The joinSecurely parameter has been removed. Here is the new API:

- `EmberStatus emberJoinNetwork(EmberNodeType nodeType, EmberNetworkParameters *parameters)`

There is a new API to retrieve the extended pan id:

- `void emberGetExtendedPanId(int8u *resultLocation);`

Network Rejoin

End devices that have lost contact with their parent or any node that does not have the current network key may call the new API below. This replaces emberMobileNodeHasMoved() from the EmberZNet 2.x API, and there is NOT a #define for back-compatibility. The new call uses the ZigBee network rejoin command, and takes about 150 milliseconds (mostly for the active scan) whereas the old call used MAC association and took half a second longer.

```
EmberStatus emberRejoinNetwork(boolean haveCurrentNetworkKey);
```

The `haveCurrentNetworkKey` determines if the stack performs a secure network rejoin (`haveCurrentNetworkKey = TRUE`) or an insecure network rejoin (`haveCurrentNetworkKey = FALSE`). The insecure network rejoin only works if using the Commercial Security Library. In that case the current Network Key will be sent to the rejoining node encrypted at the APS Layer with that device's Link Key.

Sending and receiving messages

Now that datagrams are indistinguishable from ZigBee messages we need to stop using the term 'datagram' in the API. Given the demise of sequenced messages, the meaning of 'datagram' in the API is now the opposite of what it means in TCP/IP. Their datagrams are not acked, ours are.

The main changes are: - There is now an address table, which is just an array of paired long and short addresses. This replaces the volatile (RAM) bindings. - Unicasts can be sent using either directly to an address, via an address table entry, or via a binding. - Datagrams, sequenced messages, and the SPDO will disappear.

Cluster IDs

Cluster IDs are now 16 bits long instead of 8. All uses of cluster IDs in the API have been changed from `int8u` to `int16u`. In places where an array of cluster ID is used, such as in the `EmberEndpointDescription` type, the size field is the number of cluster IDs, not the number of bytes. Over the air cluster IDs are sent least-significant byte first, as is standard in ZigBee.

APS frame

The `EmberApsFrame` struct has changes that correspond to the changes in the actual APS frame:

- The cluster ID is an `int16u` instead of an `int8u`.
- There is one-byte `sequence` field that contains the sequence number. This is only valid for incoming messages. For outgoing messages the stack ignores the value supplied by the application.
- Because the APS frame is used for multicasts and broadcasts as well as unicasts, the ZigBee standard options will change from `EMBER_UNICAST_OPTION_...` to `EMBER_APS_OPTION_...`. See below for a complete list of the options. `EmberUnicastOption` will change to `EmberApsOption` and change size from `int8u` to `int16u`.
- There is a two-byte `groupId` field. This is valid only for incoming multicasts and broadcasts. For broadcasts the field will contain the destination address.

The following are standard ZigBee APS options which can be used when sending packets and be read on the receiver.

- `EMBER_APS_OPTION_RETRY`
- `EMBER_APS_OPTION_SECURITY` (new Encrypt message using APS-level security.)

- `EMBER_APS_OPTION_SOURCE_EUI64` (new Include the source EUI64 in the network frame.)
- `EMBER_APS_OPTION_DESTINATION_EUI64` (new Include the destination EUI64 in the network frame.)

The following are Ember options that control message transmission. These are not available on the receiver.

- `EMBER_APS_OPTION_ENABLE_ROUTE_DISCOVERY`
- `EMBER_APS_OPTION_FORCE_ROUTE_DISCOVERY`
- `EMBER_APS_OPTION_ENABLE_ADDRESS_DISCOVERY` (new)
- `EMBER_APS_OPTION_POLL_RESPONSE`

ZigBee has removed indirect APS messages. The following options will no longer be available:

- `EMBER_UNICAST_OPTION_APS_INDIRECT`
- `EMBER_UNICAST_OPTION_HAVE_SOURCE`

Address table

EUI64 <-> network ID mappings will be kept in an address table. The stack will update the node IDs as new information arrives. It will not change the EUI64s.

```

EmberStatus emberSetAddressTableRemoteEui64(int8u
    addressTableIndex, EmberEui64 eui64);

void emberSetAddressTableRemoteNodeId(int8u addressTableIndex,
    EmberNodeId id);

void emberGetAddressTableRemoteEui64(int8u addressTableIndex,
    EmberEui64 eui64);

EmberNodeId emberGetAddressTableRemoteNodeId(int8u
    addressTableIndex);

```

The size of the table can be set by defining `EMBER_ADDRESS_TABLE_SIZE` before including `ember-configuration.c`.

The binding table will have its own remote ID table, just as now.

`emberGetBindingDestinationNodeId()` **will be renamed to**
`emberGetBindingRemoteNodeId()`

`emberSetBindingDestinationNodeId()` **will be renamed to**
`emberGetBindingRemoteNodeId()`

They will perform the same function as they currently do however the names have been changed to be consistent with the address calls.

There are 4 reserved node ID values that are used with the address and binding tables. These are described below

EMBER_TABLE_ENTRY_UNUSED_NODE_ID (0xFFFF) This value is used when setting or getting the remote node ID in the address table or getting the remote node ID from the binding table. It indicates that address or binding table entry is not in use.

EMBER_MULTICAST_NODE_ID (0xFFFE) This value is returned when getting the remote node ID from the binding table and the given binding table index refers to a multicast binding entry.

EMBER_UNKNOWN_NODE_ID (0xFFFD) This value is used when getting the remote node ID from the address or binding tables. It indicates that the address or binding table entry is currently in use but the node ID corresponding to the EUI64 in the table is currently unknown.

EMBER_DISCOVERY_ACTIVE_NODE_ID (0xFFFC) This value is used when getting the remote node ID from the address or binding tables. It indicates that the address or binding table entry is currently in use and network address discovery is underway.

There is a new function that will validate that a given node ID is valid and not one of the reserved values.

```
boolean emberIsNodeIdValid(EmberNodeId id);
```

There are also two new functions that search through all the relevant stack tables (address, binding, neighbor, child) to map a long address to a short address, or vice versa.

```
EmberNodeId emberLookupNodeIdByEui64(EmberEUI64 eui64);
EmberStatus emberLookupEui64ByNodeId(EmberNodeId nodeId,
                                     EmberEUI64 eui64Return);
```

Sending messages

The destination address for a unicast can be obtained from the address or binding tables, or passed as an argument. There is an enumeration for indicating which is to be used for a particular message. The same enumeration is used with `emberMessageSent()`; `EMBER_OUTGOING_BROADCAST` and `EMBER_OUTGOING_MULTICAST` cannot be passed to `emberSendUnicast()`.

Use of the address or binding table allows the stack to perform address discovery by setting the `EMBER_APS_OPTION_ENABLE_ADDRESS_DISCOVERY` option.

```
enum {
    EMBER_OUTGOING_DIRECT,
    EMBER_OUTGOING_VIA_ADDRESS_TABLE,
    EMBER_OUTGOING_VIA_BINDING,
    EMBER_OUTGOING_BROADCAST, // only for emberMessageSent()
    EMBER_OUTGOING_MULTICAST // only for emberMessageSent()
};
```

```

typedef int8u EmberOutgoingMessageType;
EmberStatus emberSendUnicast(EmberOutgoingMessageType type,
                             EmberApsFrame *apsFrame,
                             EmberMessageBuffer message);

```

EMBER_OUTGOING_VIA_BINDING uses only the binding's address information. The rest of the binding's information (cluster ID, endpoints, profile ID) can be retrieved using emberGetBinding() and emberGetEndpointDescription().

Below is an example of how a message might be sent using a binding. Using the options in the example duplicates the behavior of emberSendDatagram() (for a non-aggregation binding; when sending to an aggregator route and address discovery should not be enabled).

```

EmberApsFrame apsFrame = {
    profileId,
    clusterId,
    sourceEndpoint,
    destinationEndpoint,
    EMBER_APS_OPTION_RETRY
    | EMBER_APS_OPTION_ENABLE_ROUTE_DISCOVERY
    | EMBER_APS_OPTION_ENABLE_ADDRESS_DISCOVERY
    | EMBER_APS_OPTION_DESTINATION_EUI64
};

```

```

emberSendUnicast(EMBER_OUTGOING_VIA_BINDING,
                bindingIndex,
                &apsFrame,
                message);

```

emberSendReply() is unchanged. It can be used to send a reply to any retried APS messages. Replies are a nonstandard extension to ZigBee.

```

EmberStatus emberSendReply(int16u clusterId, EmberMessageBuffer
reply);

```

Multicasts get an APS frame plus radii. The groupId is specified in the APS frame. The nonmemberRadius specifies how many hops the message should be forwarded by devices that are not members of the group. A value of 7 or greater is treated as infinite. There is no longer a separate limited multicast API call.

```

EmberStatus emberSendMulticast(EmberApsFrame *apsFrame,
                               int8u radius,
                               int8u nonMemberRadius,
                               EmberMessageBuffer message);

```

There is a new multicast table. The size is EMBER_MULTICAST_TABLE_SIZE and defaults to 8. Multicast table entries should be created and modified manually by the application; just index into the array.

```

EmberMulticastTableEntry *emberMulticastTable;

```

```

/** @brief Defines an entry in the multicast table.
 * @description A multicast table entry indicates that a particular
 * endpoint is a member of a particular multicast group. Only
 * devices with an endpoint in a multicast group will receive
 * messages sent to that multicast group.
 */

typedef struct {
    /** The multicast group ID. */
    EmberMulticastId multicastId;
    /** The endpoint that is a member, or 0 if this entry is not in use
     * (the ZDO is not a member of any multicast groups).
     */
    int8u endpoint;
} EmberMulticastTableEntry;

```

Now that ZigBee has three different broadcast addresses (everyone, rx-on-when-idle only, routers only) broadcasting requires specifying an address. On the receiver this can be read out of the `groupId` field in the `apsFrame`.

```

#define EMBER_BROADCAST_ADDRESS 0xFFFC
#define EMBER_RX_ON_WHEN_IDLE_BROADCAST_ADDRESS 0xFFFD
#define EMBER_SLEEPY_BROADCAST_ADDRESS 0xFFFF

EmberStatus emberSendBroadcast(EmberNodeId destination,
                               EmberApsFrame *apsFrame,
                               int8u radius,
                               EmberMessageBuffer message);

```

Message status

`emberMessageSent()` will be called for all outgoing messages, just to be consistent. The type of message is given by the enumeration of outgoing message types.

`emberMessageSent()` will be called:

- for retried unicasts, when an ACK arrives or the message times out.
- for non-retried unicasts, when the MAC receives an ACK from the next hop or the MAC retries are exhausted.
- for broadcasts and multicasts, when the message is removed from the retry queue (not the broadcast table).

The `indexOrAddress` argument is the destination address for direct unicasts, broadcasts, and multicasts. For unicasts sent via the address or binding tables it is the index into the relevant table.

The `status` argument for broadcasts and multicasts is:

- EMBER_DELIVERY_FAILED if the message was never transmitted.
- EMBER_DELIVERY_FAILED if radius is greater than 1 and we don't hear at least one neighbor relay the message.
- EMBER_SUCCESS otherwise.

```
void emberMessageSent(EmberOutgoingMessageType type,
                     int16u indexOrDestination,
                     EmberApsFrame *apsFrame,
                     EmberMessageBuffer message,
                     EmberStatus status);
```

Incoming messages

emberIncomingMessageHandler() is unchanged. The possible message types are now:

```
EMBER_INCOMING_UNICAST
EMBER_INCOMING_UNICAST_REPLY
EMBER_INCOMING_BROADCAST
EMBER_INCOMING_BROADCAST_LOOPBACK
EMBER_INCOMING_MULTICAST
EMBER_INCOMING_MULTICAST_LOOPBACK
```

For incoming broadcasts, the destination broadcast ID will be in the groupId field of the APS struct.

EMBER_INCOMING_SHARED_MULTICAST, which indicated that there were multiple bindings whose group matched an incoming multicast, will no longer be used. Instead, the following procedure can be used to walk through the matching bindings. This examines the bindings starting from startIndex and returns the index of the first multicast binding for the specified group. 0xFF is returned if there are no matching bindings.

```
int8u emberNextMatchingMulticastBinding(int16u groupId,
                                       int8u startIndex,
                                       EmberBindingTableEntry
                                       *binding);
```

There is one additional function that can only be called from within emberIncomingMessageHandler(). This extracts the source EUI64 from the network frame of the message (only present when EMBER_APS_OPTION_SOURCE_EUI64 is set).

```
EmberStatus emberGetSenderEui64(EmberEUI64 senderEui64);
```

Note on Bindings

In ZigBee, the binding table is now used only in support of provisioning. The idea is that a provisioning tool uses the ZDO to discover the services available on a device and to add entries to the device's binding table. Other

than the ZDO, no use is made of the binding table within ZigBee. It is up to the application to make use of the information in the table how it sees fit.

Up to now, our API has used the binding table as a combined binding and address table and as a way to simplify the API. Rather than passing a lot of arguments when sending a message, you just gave a binding index and were done with it. This dual use of the binding table has led to some confusion in the API, including the splitting of the binding table into volatile and non-volatile portions. In addition, our increasing ZigBee focus has reduced the overlap between what is in a binding and what needs to be specified when sending a message.

As a result of all this, it seems like the time has come to relegate the binding table for its original use, provisioning, and use a simple address table for most messaging. This has the side benefit of allowing us to put the binding table into a library, freeing up flash for those applications that do not need it.

Aggregation

The old `emberCreateAggregationRoutes()` is replaced with the new

```
EmberStatus emberSendManyToOneRouteRequest(int16u concentratorType,  
int8u radius, EmberMessageBuffer message);
```

Where `concentratorType` is `EMBER_HIGH_RAM_CONCENTRATOR` or `EMBER_LOW_RAM_CONCENTRATOR`.

The following two callbacks must be implemented by the application if `EMBER_APPLICATION_USES_SOURCE_ROUTING` is defined in the configuration header:

```
boolean emberAppendSourceRoute(EmberNodeId destination,  
                                EmberMessageBuffer header);  
  
void emberIncomingRouteRecordHandler(EmberNodeId source,  
                                     int8u relayCount,  
                                     EmberMessageBuffer header,  
                                     int8u relayListIndex);
```

A ready-to-use implementation of these two callbacks is provided in `app/util/source-route.[c|h]`.

Disable relay

If `EMBER_DISABLE_RELAY` is defined in the app configuration header, the node will not relay unicasts, route requests, or route replies. It can still generate route requests and replies, so that it can be the source or destination of network messages. This was requested by a customer for use in a gateway.

New storage parameters

- `EMBER_ADDRESS_TABLE_SIZE` The size of the address table.
 - `EMBER_APS_UNICAST_MESSAGE_COUNT` The maximum number of concurrent unicasts.
-

Commercial Security API

This section intentionally removed while Ember finishes testing.

Ember Counters

There is a new callback for counting MAC and APS transmission and reception events. The application must define `EMBER_APPLICATION_HAS_COUNTER_HANDLER` in order to implement it. A default implementation is provided in `app/util/counter.[c|h]`. There is a new EZSP frame `readAndClearCounters` which allows the host to retrieve all the counters at once and clear them. This API was provided at the request of Tendril. The counter types are enumerated in `ember-types.h`.

```
/** @description A callback invoked to inform the application of the
 * occurrence of an event defined by \c EmberCounterType, for example,
 * transmissions and receptions at different layers of the stack.
 * The application must define EMBER_APPLICATION_HAS_COUNTERS_HANDLER
 * in its CONFIGURATION_HEADER to use this.
 *
 * @param type: The type of the event.
 * @param data: For transmission events, the number of retries used.
 * For other events, this parameter is unused and is set to zero.
 */
void emberCounterHandler(EmberCounterType type, int8u data);

/**
 * @description Defines the events reported to the application
 * by the \c emberCounterHandler().
 */
enum {
    /** The MAC received a broadcast. */
    EMBER_COUNTER_MAC_RX_BROADCAST = 0,
    /** The MAC transmitted a broadcast. */
    EMBER_COUNTER_MAC_TX_BROADCAST = 1,
    /** The MAC received a unicast. */
    EMBER_COUNTER_MAC_RX_UNICAST = 2,
    /** The MAC successfully transmitted a unicast. */
    EMBER_COUNTER_MAC_TX_UNICAST_SUCCESS = 3,
    /** The MAC retried a unicast. This is a placeholder
     * and is not used by the emberCounterHandler() callback. Instead
```

```

* the number of APS retries are returned in the data parameter
* of the callback for the EMBER_COUNTER_MAC_TX_UNICAST
* and EMBER_COUNTER_MAC_TX_UNICAST_FAILED types. */
EMBER_COUNTER_MAC_TX_UNICAST_RETRY = 4,
/** The MAC unsuccessfully transmitted a unicast. */
EMBER_COUNTER_MAC_TX_UNICAST_FAILED = 5,
/** The APS layer received a data broadcast. */
EMBER_COUNTER_APS_DATA_RX_BROADCAST = 6,
/** The APS layer transmitted a data broadcast. */
EMBER_COUNTER_APS_DATA_TX_BROADCAST = 7,
/** The APS layer received a data unicast. */
EMBER_COUNTER_APS_DATA_RX_UNICAST = 8,
/** The APS layer successfully transmitted a data unicast. */
EMBER_COUNTER_APS_DATA_TX_UNICAST_SUCCESS = 9,
/** The APS layer retried a data unicast. This is a placeholder
* and is not used by the emberCounterHandler() callback. Instead
* the number of APS retries are returned in the data parameter
* of the callback for the EMBER_COUNTER_APS_DATA_TX_UNICAST_SUCCESS
* and EMBER_COUNTER_APS_DATA_TX_UNICAST_FAILED types. */
EMBER_COUNTER_APS_DATA_TX_UNICAST_RETRY = 10,
/** The APS layer unsuccessfully transmitted a data unicast. */
EMBER_COUNTER_APS_DATA_TX_UNICAST_FAILED = 11,
/** The network layer successfully submitted a new route discovery
* to the MAC. */
EMBER_COUNTER_ROUTE_DISCOVERY_INITIATED = 12,
/** A placeholder giving the number of Ember counter types. */
EMBER_COUNTER_TYPE_COUNT = 13

```

Removed API's

The following will no longer be available for the reasons given.

All bindings will now be nonvolatile.

- emberTemporaryBindingEntries
- emberNonvolatileBindingEntries
- emberBindingIsNonvolatile()
- emberBindingIsTemporary()
- EMBER_TEMPORARY_BINDING_ENTRIES
- EMBER_APS_INDIRECT_BINDING_TABLE_SIZE
- EMBER_APS_INDIRECT_BINDING_TABLE_TOKEN_SIZE

Datagrams have been merged with APS retried messages.

- emberMaximumTransportPayloadLength()
- emberSendDatagram()

- `emberUnicastSent()` (superceded by `emberMessageSent()`)

`EMBER_TRANSPORT_PACKET_COUNT` will remain as an alias for `EMBER_APS_UNICAST_MESSAGE_COUNT`.

There will be no SPDO. Its primary purpose was in conjunction with JIT messages, which we cannot support on the 260. Other uses can be replaced by making use of reserved ZDO cluster IDs (there are 32000 of them, so we should be safe using one or two).

- `emberSendSpdoDatagramToParent()`
- `emberSpdoUnicastSent()`
- `emberIncomingSpdoMessageHandler()`

Sequenced messages have been removed. ZigBee fragmented messages, which bear some resemblance to our sequenced messages, are currently scheduled to appear in 3.1.

- `emberSendSequenced()`
- `emberOpenConnection()`
- `emberConnectionStatus()`
- `emberCloseConnection()`
- `emberConnectionStatusHandler()`
- `EMBER_TRANSPORT_CONNECTION_COUNT`