ember

26 May 2011
120-3009-000P

EM260

EM35x
EZSP

EZSP Reference Guide

For Use with EmberZNet Release 4.5.1

The EmberZNet Serial Protocol (EZSP) defined in this document is the protocol used by a host application processor to interact with the EmberZNet stack running on a Network Co-Processor (NCP). EZSP messages are sent between the host and the NCP over either a SPI or a UART interface.

Contents

Ember Corporation
25 Thomson Place
Boston, MA 02210
+1 (617) 951-0200
www.ember.com

1. What’s New

Ember has made some changes to the EZSP functions for Release 4.5.1. These functions are grouped into the following categories.

1.1 New EZSP Config IDs

These new config IDs have been added in Release 4.5.1:

- EZSP_CONFIG_BROADCAST_TABLE_SIZE: The maximum number of broadcasts during a single broadcast timeout period.
- EZSP_CONFIG_MAC_FILTER_TABLE_SIZE: The size of the MAC filter list table.

1.2 New EZSP Value IDs

This new value ID has been added in Release 4.5.1:

- EZSP_VALUE_MAC_FILTER_LIST: A list of EmberMacFilterMatchData values.

1.3 New Manufacturing Token IDs

This new manufacturing token ID has been added in Release 4.5.1:

- EZSP_STACK_CAL_FILTER: Radio channel filter calibration data.

1.4 New Counters

These new counters have been added in Release 4.5.1:

- EMBER_COUNTER_ALLOCATE_PACKET_BUFFER_FAILURE: The number of times we failed to allocate a set of linked packet buffers.
- EMBER_COUNTER_RELAYED_UNICAST: The number of relayed unicast packets.

1.5 New Join Methods

These new join methods have been added in Release 4.5.1:

- EMBER_USE_NWK_REJOIN_HAVE_NWK_KEY
- EMBER_USE_NWK_COMMISSIONING

1.6 New Functions

These new functions have been added in Release 4.5.1:

- macFilterMatchMessageHandler: A callback when a raw MAC message matched one of the configured MAC filters.
- removeDevice: Send an APS remove device command.
- unicastNwkKeyUpdate: Send a unicast transport key message containing a new NWK key.

1.7 Removed EZSP Config IDs

EZSP_CONFIG_ENABLE_DUAL_CHANNEL_SCAN was deprecated in Release 4.2.0 and has been removed in Release 4.5.1. form-and-join.c on the host should be used instead.

1.8 Removed Network Scan Types

EZSP_UNUSED_PAN_ID_SCAN and EZSP_NEXT_JOINABLE_NETWORK_SCAN were deprecated in Release 4.2.0 and have been removed in Release 4.5.1. form-and-join.c on the host should be used instead.

1.9 Removed Functions

scanErrorHandler, scanForJoinableNetwork, and unusedPanIdFoundHandler were deprecated in Release 4.2.0 and have been removed in Release 4.5.1. form-and-join.c on the host should be used instead.

2. EmberZNet Serial Protocol

All EZSP frames begin with the same 3 fields: sequence, frame control, and frame ID. The format of the rest of the frame depends on the frame ID. Chapter 3, “Protocol Format,” defines the format for all frame IDs. Most of the frames have a fixed length. A few, such as those containing application messages, are of variable length. The frame control indicates the direction of the message (command or response). For commands, the frame control also contains power management information (SPI interface only). For responses, the frame control also contains status information.

The host initiates a two-message transaction by sending a command message to the NCP. The NCP then sends a response message to the host. When connected using the SPI interface, if the NCP needs to communicate a callback to the host, it will indicate this using the interrupt line and then wait for the host to send the `callback` command. When connected using the UART interface, the NCP can send callbacks to the host asynchronously as soon as they occur.

When a command contains an application message, the host must supply a one-byte tag. This tag is used in future commands and responses to refer to the message. For example, when sending a message, the host provides both the message contents and a tag. The tag is then used to report the fate of the message in a later response from the NCP.

Ember designed EZSP to be very familiar to customers who have used the EmberZNet stack API. The majority of the commands and responses are functionally identical to those found in EmberZNet. The variations are due mainly to the timing differences of running the application on a separate processor across a serial interface.

2.1 Byte Order

All multiple octet fields are transmitted and received with the least significant octet first, also referred to as “little endian.” This is the same byte order convention specified by 802.15.4 and ZigBee. Note that EUI64 fields are treated as a 64-bit number and are therefore transmitted and received in little endian order. Each individual octet is transmitted and received by the SPI or UART interface. See the EM260 Datasheet (120-0260-000) for more information about the SPI and UART interfaces.

2.2 Conceptual Overview

This section provides an overview of the concepts that are specific to EZSP or that differ from the EmberZNet stack API. The commands and responses mentioned in this overview are described in more detail later in this document.

2.2.1 Stack configuration

To ensure that the NCP and the host agree on the protocol format, the first command sent by the host after the NCP has reset must be the `version` command. There are a number of configuration values that affect the behavior of the stack. The host can read these values at any time using the `getConfigurationValue` command. After the NCP has reset, the host can modify any of the default values using the `setConfigurationValue` command. The host must then provide information about the application endpoints using the `addEndpoint` command.

Table 1 gives the minimum, default, and maximum values for each of the configuration values. Also listed is the RAM cost—the number of bytes of additional RAM required to increase the configuration value by one. Since the total amount of RAM is fixed, the additional RAM required must be made available by reducing one of the other configuration values.

Table 1. Configuration Values

Value	Min.	Def.	Max.	Units	RAM Cost	Description
EZSP_CONFIG_PACKET_BUFFER_COUNT	5	24		packet buffers	39	The number of packet buffers available to the stack.
EZSP_CONFIG_NEIGHBOR_TABLE_SIZE	8	16	16	neighbors	18	The maximum number of router neighbors the stack can keep track of. A neighbor is a node within radio range.
EZSP_CONFIG_APS_UNICAST_MESSAGE_COUNT	0	10		messages	6	The maximum number of APS retried messages the stack can be transmitting at any time.
EZSP_CONFIG_BINDING_TABLE_SIZE	0	0	32	entries	2	The maximum number of non-volatile bindings supported by the stack.
EZSP_CONFIG_ADDRESS_TABLE_SIZE	0	8		entries	12	The maximum number of EUI64 to network address associations that the stack can maintain.
EZSP_CONFIG_MULTICAST_TABLE_SIZE	0	8		entries	4	The maximum number of multicast groups that the device may be a member of.
EZSP_CONFIG_ROUTE_TABLE_SIZE	0	16		entries	6	The maximum number of destinations to which a node can route messages. This includes both messages originating at this node and those relayed for others.
EZSP_CONFIG_DISCOVERY_TABLE_SIZE	0	8		entries	10	The number of simultaneous route discoveries that a node will support.
EZSP_CONFIG_BROADCAST_ALARM_DATA_SIZE	0	0	16	bytes	1	The size of the alarm broadcast buffer.
EZSP_CONFIG_UNICAST_ALARM_DATA_SIZE (A)	0	0	16	bytes	(C)	The size of the unicast alarm buffers allocated for end device children.
EZSP_CONFIG_STACK_PROFILE	0	0			0	Specifies the stack profile.

Value	Min.	Def.	Max.	Units	RAM Cost	Description
EZSP_CONFIG_SECURITY_LEVEL	0	5	5		0	The security level used for security at the MAC and network layers. The supported values are 0 (no security) and 5 (payload is encrypted and a four-byte MIC is used for authentication).
EZSP_CONFIG_MAX_HOPS (B)	0	10		hops	0	The maximum number of hops for a message.
EZSP_CONFIG_MAX_END_DEVICE_CHILDREN (C)	0	6	32	children	9 + (A)	The maximum number of end device children that a router will support.
EZSP_CONFIG_INDIRECT_TRANSMISSION_TIMEOUT	0	3000	30000	milli-seconds	0	The maximum amount of time that the MAC will hold a message for indirect transmission to a child.
EZSP_CONFIG_END_DEVICE_POLL_TIMEOUT	0	5	255	2^(D) seconds	0	The maximum amount of time that an end device child can wait between polls. If no poll is heard within this timeout, then the parent removes the end device from its tables.
EZSP_CONFIG_MOBILE_NODE_POLL_TIMEOUT	0	20		quarter seconds	0	The maximum amount of time that a mobile node can wait between polls. If no poll is heard within this timeout, then the parent removes the mobile node from its tables.
EZSP_CONFIG_RESERVED_MOBILE_CHILD_ENTRIES	0	0	(C)	entries	0	The number of child table entries reserved for use only by mobile nodes.
EZSP_CONFIG_TX_POWER_MODE	0	0	3		0	Enables boost power mode and/or the alternate transmitter output.
EZSP_CONFIG_DISABLE_RELAY	0	0	1		0	0: Allow this node to relay messages. 1: Prevent this node from relaying messages.
EZSP_CONFIG_TRUST_CENTER_ADDRESS_CACHE_SIZE	0	0		entries	12	The maximum number of EUI64 to network address associations that the Trust Center can maintain.
EZSP_CONFIG_SOURCE_ROUTE_TABLE_SIZE	0	0		entries	4	The size of the source route table.
EZSP_CONFIG_END_DEVICE_POLL_TIMEOUT_SHIFT (D)	0	6	10		0	The units used for timing out end devices on their parents.
EZSP_CONFIG_FRAGMENT_WINDOW_SIZE	0	0	8	blocks	0	The number of blocks of a fragmented message that can be sent in a single window.
EZSP_CONFIG_FRAGMENT_DELAY_MS	0	0		milli-seconds	0	The time the stack will wait between sending blocks of a fragmented message.

Value	Min.	Def.	Max.	Units	RAM Cost	Description
EZSP_CONFIG_KEY_TABLE_SIZE	0	0		entries	4	The size of the Key Table used for storing individual link keys (if the device is a Trust Center) or Application Link Keys (if the device is a normal node).
EZSP_CONFIG_APS_ACK_TIMEOUT		50 * (B) + 100		milli-seconds	0	The APS ACK timeout value. The stack waits this amount of time between resends of APS retried messages.
EZSP_CONFIG_ACTIVE_SCAN_DURATION	0	3	6	15.4 scan duration units	0	The duration of an active scan. This also controls the jitter used when responding to a beacon request.
EZSP_CONFIG_END_DEVICE_BIND_TIMEOUT	1	60		seconds	0	The time the coordinator will wait for a second end device bind request to arrive.
EZSP_CONFIG_PAN_ID_CONFLICT_REPORT_THRESHOLD	1	1	63	reports per minute	0	The number of PAN id conflict reports that must be received by the network manager within one minute to trigger a PAN id change.
EZSP_CONFIG_REQUEST_KEY_TIMEOUT	0	0	10	minutes	0	The timeout value in minutes for how long the Trust Center or a normal node waits for the ZigBee Request Key to complete. On the Trust Center this controls whether or not the device buffers the request, waiting for a matching pair of ZigBee Request Key. If the value is non-zero, the Trust Center buffers and waits for that amount of time. If the value is zero, the Trust Center does not buffer the request and immediately responds to the request. Zero is the most compliant behavior.
EZSP_CONFIG_CERTIFICATE_TABLE_SIZE	0	1	1		0	This value indicates the size of the runtime modifiable certificate table. Normally certificates are stored in MFG tokens but this table can be used to field upgrade devices with new Smart Energy certificates. This value cannot be set, it can only be queried.

Value	Min.	Def.	Max.	Units	RAM Cost	Description
EZSP_CONFIG_APPLICATION_ZDO_FLAGS	0	0	255		0	This is a bitmask that controls which incoming ZDO request messages are passed to the application. The bits are defined in the EmberZdoConfigurationFlags enumeration. To see if the application is required to send a ZDO response in reply to an incoming message, the application must check the APS options bitfield within the incomingMessageHandler callback to see if the EMBER_APS_OPTION_ZDO_RESPONSE_REQUIRED flag is set.
EZSP_CONFIG_BROADCAST_TABLE_SIZE	15	15	254	entries	6	The maximum number of broadcasts during a single broadcast timeout period.
EZSP_CONFIG_MAC_FILTER_TABLE_SIZE	0	0	254	entries	2	The size of the MAC filter list table.

2.2.2 Policy settings

There are some situations when the NCP must make a decision but there is not enough time to consult with the host. The host can control what decision is made by setting the policy in advance. The NCP will then make decisions according to the current policy. The host is informed via callbacks each time a decision is made, but by the time the news reaches the host, it is too late to change that decision. You can change the policies at any time by using the `setPolicy` command.

A policy is used for trust center behavior, external binding modification requests, unicast replies, generating `pollHandler` callbacks, and the contents of the `messageSent` callback.

2.2.3 Unicast replies

The policy for unicast replies allows the host to decide whether it wants to supply the NCP with a reply payload for every retried unicast received. If the host sets the policy to not supply a reply, the NCP will automatically send an empty reply (containing no payload) for every retried unicast received. If the host sets the policy to supply the reply, then the NCP will only send a reply when instructed by the host.

If the reply does not reach the sender before the APS retry timeout expires, the sender will transmit the unicast again. The host must process the incoming message and supply the reply quickly enough to avoid retransmission by the sender. Provided this timing constraint is met, multiple unicasts can be received before the first reply is supplied and the replies can be supplied in any order.

2.2.4 SPI interface callbacks

Asynchronous callbacks from the NCP are sent to the host as the response to a `callback` command. The NCP uses the interrupt line to indicate that the host should send a `callback` command. The NCP will queue multiple callbacks while it waits for the host. Each response only delivers one callback. If the NCP receives the `callback` command when there are no pending callbacks, it will reply with the `noCallbacks` response.

2.2.5 UART interface callbacks

By default, callbacks from the NCP are sent to the host asynchronously as soon as they occur and the host never needs to send the `callback` command. The host can disable asynchronous callbacks by setting `EZSP_VALUE_UART_SYNCH_CALLBACKS` to `1` using the `setValue` command. Callbacks will then only be sent to the host as the response to a callback command.

2.2.6 SPI interface power management

The NCP always idles its processor whenever possible. To further reduce power consumption when connected using the SPI interface, the NCP can be put to sleep by the host. The UART interface is designed for gateway applications and does not support power management. In power down mode, only an external interrupt will wake the NCP. In deep sleep mode, the NCP will use its internal timer to wake up for scheduled events. The NCP provides two independent timers that the host can use for any purpose, including waking up the NCP from deep sleep mode. Timers are set using the `setTimer` command and generate `timerHandler` callbacks.

The frame control byte of every command tells the NCP which sleep mode to enter after it has responded to the command. Including this information in every command (instead of having a separate power management command) allows the NCP to be put to sleep faster. If the host needs to put the NCP to sleep without also performing another action, the `nop` command can be used.

In deep sleep mode, the NCP will wake up for an internal event. If the event does not produce a callback for the host, the NCP will go back to sleep once the event has been handled. If the event does produce a callback, the NCP will signal the host and remain awake waiting for the `callback` command. If the frame control byte of the `callback` command specifies deep sleep mode, then the NCP would normally go back to sleep after responding with the callback. However, if there is a second callback pending, the NCP will remain awake waiting for another `callback` command.

To avoid disrupting the operation of the network, only put the NCP to sleep when it is not joined to a network or when it is joined as a sleeping end device. If the NCP is joined as a sleeping end device, then it must poll its parent in order to receive messages. The host controls the polling behavior using the `pollForData` command. Polls are sent periodically with the interval set by the host or a single poll can be sent. The result of every poll attempt is optionally reported using the `pollCompleteHandler` callback.

2.2.7 Tokens

Some of the non-volatile storage on the NCP is made available for use by the host. Up to 8 tokens stored in the Simulated EEPROM can be read and written using the `setToken` and `getToken` commands. Each token is 8 bytes. Tokens preserve their values between reboots. Refer to the “Simulated EEPROM” section in the EM260 Datasheet (120-0260-000) for a description of the Simulated EEPROM and write cycle estimates. The manufacturing tokens stored in the Flash Information Area can be read using the `getMfgToken` command.

2.2.8 NCP status

The frame control byte of every response sent by the NCP contains four status fields:

- The overflow bit is set if the NCP ran out of memory at any time since the previous response was sent. If this bit is set, then messages may have been lost.
- The truncated bit is set if the NCP truncated the current response. If this bit is set, the command from the host produced a response larger than the maximum EZSP frame length.
- The callback pending bit is set if the NCP has one or more callbacks that have not been delivered to the host.
- The callback type field identifies a response as either an asynchronous callback (UART interface only), a synchronous callback, or not a callback.

You can use the `nop` command to check the status of the NCP without also performing another action.

2.2.9 Random number generator

The host can obtain a random number from the NCP using the `getRandomNumber` command. The random number is generated from analog noise in the radio and can be used to seed a random number generator on the host.

3. Protocol Format

Figure 1. EZSP Frame Format

Sequence	Frame Control	Frame ID	Parameters
1 byte	1 byte	1 byte	

The first byte of all EZSP frames is a sequence number. The host should increment the sequence number each time a command is sent to the NCP. The response sent by the NCP uses the sequence number of the command, except when the response is a callback. Callback responses contain the sequence number of the last command seen at the time the callback occurred on the NCP. The second byte of all EZSP frames is the frame control byte. Table 2 describes the meaning of this byte for command and response frames. Table 3 describes the sleep modes. Table 4 describes the overflow status bit. Table 5 describes the truncated status bit. Table 6 describes the callback pending status bit. Table 7 describes the callback types. The third byte of all EZSP frames is the frame ID byte. The meaning of this byte and the associated parameters are described in the sections below.

Table 2. Frame Control Byte

Bit	Command	Response
7 (MSB)	0	1
6	0 (reserved)	0 (reserved)
5	0 (reserved)	0 (reserved)
4	0 (reserved)	callbackType[1]
3	0 (reserved)	callbackType[0]
2	0 (reserved)	callbackPending
1	sleepMode[1]	truncated
0 (LSB)	sleepMode[0]	overflow

Table 3. Sleep Modes

sleepMode[1]	sleepMode[0]	Description
1	1	Reserved.
1	0	Power down.
0	1	Deep sleep.
0	0	Idle.

Table 4. Overflow Status

overflow	Description
1	The NCP ran out of memory since the previous response.
0	No memory shortage since the previous response.

Table 5. Truncated Status

truncated	Description
1	The NCP truncated the current response to avoid exceeding the maximum EZSP frame length.
0	The current response was not truncated.

Table 6. Callback Pending Status

callbackPending	Description
1	A callback is pending on the NCP. If this response is a callback, at least one more callback is available.
0	All callbacks have been delivered to the host.

Table 7. Callback Types

callbackType[1]	callbackType[0]	Description
1	1	Reserved.
1	0	(UART interface only) This response is an asynchronous callback. It was not sent in response to a callback command.
0	1	This response is a synchronous callback. It was sent in response to a callback command.
0	0	This response is not a callback.

Section 3.1 defines all the types used by the NCP, section 3.2 defines all the structures, and section 3.3 enumerates all the named values for the different types.

The remaining sections in this chapter list all the frames supported by the NCP, specifying the frame ID, the command parameters, and the response parameters. The sections are organized by type of frames as follows.

Section No.	Type of Frames
3.4	Configuration Frames
3.5	Utilities Frames
3.6	Networking Frames
3.7	Binding Frames
3.8	Messaging Frames
3.9	Security Frames
3.10	Trust Center Frames
3.11	Certificate Based Key Exchange (CBKE)
3.12	Mfglib frames
3.13	Bootloader Frames
3.14	Alphabetical List of Frames

3.1 Type Definitions

Type	Alias	Description
boolean	int8u	True or false.
EzspConfigId	int8u	Identifies a configuration value.
EzspValueId	int8u	Identifies a value.
EmberConfigTxPowerMode	int16u	Values for EZSP_CONFIG_TX_POWER_MODE.
EzspPolicyId	int8u	Identifies a policy.
EzspDecisionId	int8u	Identifies a policy decision.
EzspMfgTokenId	int8u	Manufacturing token IDs used by ezspGetMfgToken().
EzspStatus	int8u	Status values used by EZSP.
EmberStatus	int8u	Return type for stack functions.
EmberEventUnits	int8u	Either marks an event as inactive or specifies the units for the event execution time.
EmberNodeType	int8u	The type of the node.
EmberNetworkStatus	int8u	The possible join states for a node.
EmberIncomingMessageType	int8u	Incoming message types.
EmberOutgoingMessageType	int8u	Outgoing message types.
EmberMacPassthroughType	int8u	MAC passthrough message type flags.
EmberBindingType	int8u	Binding types.
EmberApsOption	int16u	Options to use when sending a message.
EzspNetworkScanType	int8u	Network scan types.
EmberJoinDecision	int8u	Decision made by the trust center when a node attempts to join.
EmberInitialSecurityBitmask	int16u	This is the Initial Security Bitmask that controls the use of various security features.
EmberCurrentSecurityBitmask	int16u	This is the Current Security Bitmask that details the use of various security features.
EmberKeyType	int8u	Describes the type of ZigBee security key.

Type	Alias	Description
EmberKeyStructBitmask	int16u	Describes the presence of valid data within the EmberKeyStruct structure.
EmberDeviceUpdate	int8u	The status of the device update.
EmberKeyStatus	int8u	The status of the attempt to establish a key.
EmberCounterType	int8u	Defines the events reported to the application by the *readAndClearCounters* command.
EmberJoinMethod	int8u	The type of method used for joining.
EmberZdoConfigurationFlags	int8u	Flags for controlling which incoming ZDO requests are passed to the application. To see if the application is required to send a ZDO response to an incoming message, the application must check the APS options bitfield within the incomingMessageHandler callback to see if the EMBER_APS_OPTION_ZDO_RESPONSE_REQUIRED flag is set.
EmberConcentratorType	int16u	Type of concentrator.
EmberNodeId	int16u	16-bit ZigBee network address.
EmberPanId	int16u	802.15.4 PAN ID.
EmberMulticastId	int16u	16-bit ZigBee multicast group identifier.
EmberEUI64	int8u[8]	EUI 64-bit ID (an IEEE address).
EmberLibraryStatus	int8u	The presence and status of the Ember library.

3.2 Structure Definitions

Structure	Field	Description
EmberNetworkParameters		Network parameters.
	int8u[8] extendedPanId	The network's extended PAN identifier.
	int16u panId	The network's PAN identifier.
	int8s radioTxPower	A power setting, in dBm.
	int8u radioChannel	A radio channel.
	EmberJoinMethod joinMethod	The method used to initially join the network.
	EmberNodeId nwkManagerId	NWK Manager ID. The ID of the network manager in the current network. This may only be set at joining when using EMBER_USE_NWK_COMMISSIONING as the join method.
	int8u nwkUpdateId	NWK Update ID. The value of the ZigBee nwkUpdateId known by the stack. This is used to determine the newest instance of the network after a PAN ID or channel change. This may only be set at joining when using EMBER_USE_NWK_COMMISSIONING as the join method.
	int32u channels	NWK channel mask. The list of preferred channels that the NWK manager has told this device to use when searching for the network. This may only be set at joining when using EMBER_USE_NWK_COMMISSIONING as the join method.
EmberZigbeeNetwork		The parameters of a ZigBee network.
	int8u channel	The 802.15.4 channel associated with the network.
	int16u panId	The network's PAN identifier.
	int8u[8] extendedPanId	The network's extended PAN identifier.
	boolean allowingJoin	Whether the network is allowing MAC associations.
	int8u stackProfile	The Stack Profile associated with the network.
	int8u nwkUpdateId	The instance of the Network.
EmberApsFrame		ZigBee APS frame parameters.

Structure	Field	Description
	int16u profileId	The application profile ID that describes the format of the message.
	int16u clusterId	The cluster ID for this message.
	int8u sourceEndpoint	The source endpoint.
	int8u destinationEndpoint	The destination endpoint.
	EmberApsOption options	A bitmask of options.
	int16u groupId	The group ID for this message, if it is multicast mode.
	int8u sequence	The sequence number.
EmberBindingTableEntry		An entry in the binding table.
	EmberBindingType type	The type of binding.
	int8u local	The endpoint on the local node.
	int16u clusterId	A cluster ID that matches one from the local endpoint's simple descriptor. This cluster ID is set by the provisioning application to indicate which part an endpoint's functionality is bound to this particular remote node and is used to distinguish between unicast and multicast bindings. Note that a binding can be used to send messages with any cluster ID, not just that listed in the binding.
	int8u remote	The endpoint on the remote node (specified by identifier).
	EmberEUI64 identifier	A 64-bit identifier. This is either the destination EUI64 (for unicasts) or the 64-bit group address (for multicasts).
EmberMulticastTableEntry		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.
	EmberMulticastId multicastId	The multicast group ID.
	int8u endpoint	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.)

Structure	Field	Description
EmberKeyData		A 128-bit key.
	int8u[16] contents	The key data.
EmberCertificateData		The implicit certificate used in CBKE.
	int8u[48] contents	The certificate data.
EmberPublicKeyData		The public key data used in CBKE.
	int8u[22] contents	The public key data.
EmberPrivateKeyData		The private key data used in CBKE.
	int8u[21] contents	The private key data.
EmberSmacData		The Shared Message Authentication Code data used in CBKE.
	int8u[16] contents	The Shared Message Authentication Code data.
EmberSignatureData		An ECDSA signature
	int8u[42] contents	The signature data.
EmberMessageDigest		The calculated digest of a message.
	int8u[16] contents	The calculated digest of a message.
EmberAesMmoHashContext		The hash context for an ongoing hash operation.
	int8u[16] result	The result of ongoing the hash operation.
EmberNeighborTableEntry		A neighbor table entry stores information about the reliability of RF links to and from neighboring nodes.
	int16u shortId	The neighbor's two byte network id
	int8u averageLqi	An exponentially weighted moving average of the link quality values of incoming packets from this neighbor as reported by the PHY.
	int8u inCost	The incoming cost for this neighbor, computed from the average LQI. Values range from 1 for a good link to 7 for a bad link.
	int8u outCost	The outgoing cost for this neighbor, obtained from the most recently received neighbor exchange message

Structure	Field	Description
		from the neighbor. A value of zero means that a neighbor exchange message from the neighbor has not been received recently enough, or that our id was not present in the most recently received one.
	int8u age	The number of aging periods elapsed since a link status message was last received from this neighbor. The aging period is 16 seconds.
	EmberEUI64 longId	The 8 byte EUI64 of the neighbor.
EmberRouteTableEntry		A route table entry stores information about the next hop along the route to the destination.
	int16u destination	The short id of the destination. A value of 0xFFFF indicates the entry is unused.
	int16u nextHop	The short id of the next hop to this destination.
	int8u status	Indicates whether this entry is active (0), being discovered (1), unused (3), or validating (4).
	int8u age	The number of seconds since this route entry was last used to send a packet.
	int8u concentratorType	Indicates whether this destination is a High RAM Concentrator (2), a Low RAM Concentrator (1), or not a concentrator (0).
	int8u routeRecordState	For a High RAM Concentrator, indicates whether a route record is needed (2), has been sent (1), or is no long needed (0) because a source routed message from the concentrator has been received.
EmberInitialSecurityState		The security data used to set the configuration for the stack, or the retrieved configuration currently in use.
	EmberInitialSecurityBitmask bitmask	A bitmask indicating the security state used to indicate what the security configuration will be when the device forms or joins the network.
	EmberKeyData preconfiguredKey	The pre-configured Key data that should be used when forming or joining the network. The security bitmask must be set with the EMBER_HAVE_PRECONFIGURED_KEY bit to indicate that the key contains valid data.

Structure	Field	Description
	EmberKeyData networkKey	The Network Key that should be used by the Trust Center when it forms the network, or the Network Key currently in use by a joined device. The security bitmask must be set with EMBER_HAVE_NETWORK_KEY to indicate that the key contains valid data.
	int8u networkKeySequenceNumber	The sequence number associated with the network key. This is only valid if the EMBER_HAVE_NETWORK_KEY has been set in the security bitmask.
	EmberEUI64 preconfiguredTrustCenterEui64	This is the long address of the trust center on the network that will be joined. It is usually NOT set prior to joining the network and instead it is learned during the joining message exchange. This field is only examined if ::EMBER_HAVE_TRUST_CENTER_EUI64 is set in the EmberInitialSecurityState::bitmask. Most devices should clear that bit and leave this field alone. This field must be set when using commissioning mode.
EmberCurrentSecurityState		The security options and information currently used by the stack.
	EmberCurrentSecurityBitmask bitmask	A bitmask indicating the security options currently in use by a device joined in the network.
	EmberEUI64 trustCenterLongAddress	The IEEE Address of the Trust Center device.
EmberKeyStruct		A structure containing a key and its associated data.
	EmberKeyStructBitmask bitmask	A bitmask indicating the presence of data within the various fields in the structure.
	EmberKeyType type	The type of the key.
	EmberKeyData key	The actual key data.
	int32u outgoingFrameCounter	The outgoing frame counter associated with the key.
	int32u incomingFrameCounter	The frame counter of the partner device associated with the key.
	int8u sequenceNumber	The sequence number associated with the key.
	EmberEUI64 partnerEUI64	The IEEE address of the partner device also in possession of the key.

3.3 Named Values

boolean		
FALSE	0x00	An alias for zero, used for clarity.
TRUE	0x01	An alias for one, used for clarity.

EzspConfigId		
EZSP_CONFIG_PACKET_BUFFER_COUNT	0x01	The number of packet buffers available to the stack.
EZSP_CONFIG_NEIGHBOR_TABLE_SIZE	0x02	The maximum number of router neighbors the stack can keep track of. A neighbor is a node within radio range.
EZSP_CONFIG_APS_UNICAST_MESSAGE_COUNT	0x03	The maximum number of APS retried messages the stack can be transmitting at any time.
EZSP_CONFIG_BINDING_TABLE_SIZE	0x04	The maximum number of non-volatile bindings supported by the stack.
EZSP_CONFIG_ADDRESS_TABLE_SIZE	0x05	The maximum number of EUI64 to network address associations that the stack can maintain.
EZSP_CONFIG_MULTICAST_TABLE_SIZE	0x06	The maximum number of multicast groups that the device may be a member of.
EZSP_CONFIG_ROUTE_TABLE_SIZE	0x07	The maximum number of destinations to which a node can route messages. This includes both messages originating at this node and those relayed for others.
EZSP_CONFIG_DISCOVERY_TABLE_SIZE	0x08	The number of simultaneous route discoveries that a node will support.
EZSP_CONFIG_BROADCAST_ALARM_DATA_SIZE	0x09	The size of the alarm broadcast buffer.
EZSP_CONFIG_UNICAST_ALARM_DATA_SIZE	0x0A	The size of the unicast alarm buffers allocated for end device children.

EzspConfigId		
EZSP_CONFIG_STACK_PROFILE	0x0C	Specifies the stack profile.
EZSP_CONFIG_SECURITY_LEVEL	0x0D	The security level used for security at the MAC and network layers. The supported values are 0 (no security) and 5 (payload is encrypted and a four-byte MIC is used for authentication).
EZSP_CONFIG_MAX_HOPS	0x10	The maximum number of hops for a message.
EZSP_CONFIG_MAX_END_DEVICE_CHILDREN	0x11	The maximum number of end device children that a router will support.
EZSP_CONFIG_INDIRECT_TRANSMISSION_TIMEOUT	0x12	The maximum amount of time that the MAC will hold a message for indirect transmission to a child.
EZSP_CONFIG_END_DEVICE_POLL_TIMEOUT	0x13	The maximum amount of time that an end device child can wait between polls. If no poll is heard within this timeout, then the parent removes the end device from its tables.
EZSP_CONFIG_MOBILE_NODE_POLL_TIMEOUT	0x14	The maximum amount of time that a mobile node can wait between polls. If no poll is heard within this timeout, then the parent removes the mobile node from its tables.
EZSP_CONFIG_RESERVED_MOBILE_CHILD_ENTRIES	0x15	The number of child table entries reserved for use only by mobile nodes.
EZSP_CONFIG_TX_POWER_MODE	0x17	Enables boost power mode and/or the alternate transmitter output.
EZSP_CONFIG_DISABLE_RELAY	0x18	0: Allow this node to relay messages. 1: Prevent this node from relaying messages.
EZSP_CONFIG_TRUST_CENTER_ADDRESS_CACHE_SIZE	0x19	The maximum number of EUI64 to network address associations that the Trust Center can maintain.
EZSP_CONFIG_SOURCE_ROUTE_TABLE_SIZE	0x1A	The size of the source route table.
EZSP_CONFIG_END_DEVICE_POLL_TIMEOUT_SHIFT	0x1B	The units used for timing out end

EzspConfigId		
		devices on their parents.
EZSP_CONFIG_FRAGMENT_WINDOW_SIZE	0x1C	The number of blocks of a fragmented message that can be sent in a single window.
EZSP_CONFIG_FRAGMENT_DELAY_MS	0x1D	The time the stack will wait (in milliseconds) between sending blocks of a fragmented message.
EZSP_CONFIG_KEY_TABLE_SIZE	0x1E	The size of the Key Table used for storing individual link keys (if the device is a Trust Center) or Application Link Keys (if the device is a normal node).
EZSP_CONFIG_APS_ACK_TIMEOUT	0x1F	The APS ACK timeout value. The stack waits this amount of time between resends of APS retried messages.
EZSP_CONFIG_ACTIVE_SCAN_DURATION	0x20	The duration of an active scan, in the units used by the 15.4 scan parameter (((1 << duration) + 1) * 15ms). This also controls the jitter used when responding to a beacon request.
EZSP_CONFIG_END_DEVICE_BIND_TIMEOUT	0x21	The time the coordinator will wait (in seconds) for a second end device bind request to arrive.
EZSP_CONFIG_PAN_ID_CONFLICT_REPORT_THRESHOLD	0x22	The number of PAN id conflict reports that must be received by the network manager within one minute to trigger a PAN id change.
EZSP_CONFIG_REQUEST_KEY_TIMEOUT	0x24	The timeout value in minutes for how long the Trust Center or a normal node waits for the ZigBee Request Key to complete. On the Trust Center this controls whether or not the device buffers the request, waiting for a matching pair of ZigBee Request Key. If the value is non-zero, the Trust Center buffers and waits for that amount of time. If the value is zero, the Trust Center does not buffer the request and immediately responds to the request.

EzspConfigId		
		Zero is the most compliant behavior.
EZSP_CONFIG_CERTIFICATE_TABLE_SIZE	0x29	This value indicates the size of the runtime modifiable certificate table. Normally certificates are stored in MFG tokens but this table can be used to field upgrade devices with new Smart Energy certificates. This value cannot be set, it can only be queried.
EZSP_CONFIG_APPLICATION_ZDO_FLAGS	0x2A	This is a bitmask that controls which incoming ZDO request messages are passed to the application. The bits are defined in the EmberZdoConfigurationFlags enumeration. To see if the application is required to send a ZDO response in reply to an incoming message, the application must check the APS options bitfield within the incomingMessageHandler callback to see if the EMBER_APS_OPTION_ZDO_RESPONSE_REQUIRED flag is set.
EZSP_CONFIG_BROADCAST_TABLE_SIZE	0x2B	The maximum number of broadcasts during a single broadcast timeout period.
EZSP_CONFIG_MAC_FILTER_TABLE_SIZE	0x2C	The size of the MAC filter list table.

EzspValueId		
EZSP_VALUE_TOKEN_STACK_NODE_DATA	0x00	The contents of the node data stack token.
EZSP_VALUE_MAC_PASSTHROUGH_FLAGS	0x01	The types of MAC passthrough messages that the host wishes to receive.
EZSP_VALUE_EMBERNET_PASSTHROUGH_SOURCE_ADDRESS	0x02	The source address used to filter legacy EmberNet messages when the EMBER_MAC_PASSTHROUGH_EMBERNET_SOURCE flag is set in EZSP_VALUE_MAC_PASSTHROUGH_FLAGS.
EZSP_VALUE_FREE_BUFFERS	0x03	The number of available message buffers.
EZSP_VALUE_UART_SYNCH_CALLBACKS	0x04	Selects sending synchronous callbacks in ezsp-uart.
EZSP_VALUE_MAXIMUM_INCOMING_TRANSFER_SIZE	0x05	The maximum incoming transfer size for the local node.
EZSP_VALUE_MAXIMUM_OUTGOING_TRANSFER_SIZE	0x06	The maximum outgoing transfer size for the local node.
EZSP_VALUE_STACK_TOKEN_WRITING	0x07	A boolean indicating whether stack tokens are written to persistent storage as they change.
EZSP_VALUE_STACK_IS_PERFORMING_REJOIN	0x08	A read-only value indicating whether the stack is currently performing a rejoin.
EZSP_VALUE_MAC_FILTER_LIST	0x09	A list of EmberMacFilterMatchData values.

EmberConfigTxPowerMode		
EMBER_TX_POWER_MODE_DEFAULT	0x00	Normal power mode and bi-directional RF transmitter output.
EMBER_TX_POWER_MODE_BOOST	0x01	Enable boost power mode. This is a high performance radio mode which offers increased receive sensitivity and transmit power at the cost of an increase in power consumption.
EMBER_TX_POWER_MODE_ALTERNATE	0x02	Enable the alternate transmitter output. This allows for simplified connection to an external power amplifier via the RF_TX_ALT_P and RF_TX_ALT_N pins.
EMBER_TX_POWER_MODE_BOOST_AND_ALTERNATE	0x03	Enable both boost mode and the alternate transmitter output.

EzspPolicyId		
EZSP_TRUST_CENTER_POLICY	0x00	Controls trust center behavior.
EZSP_BINDING_MODIFICATION_POLICY	0x01	Controls how external binding modification requests are handled.
EZSP_UNICAST_REPLIES_POLICY	0x02	Controls whether the Host supplies unicast replies.
EZSP_POLL_HANDLER_POLICY	0x03	Controls whether pollHandler callbacks are generated.
EZSP_MESSAGE_CONTENTS_IN_CALLBACK_POLICY	0x04	Controls whether the message contents are included in the messageSentHandler callback.
EZSP_TC_KEY_REQUEST_POLICY	0x05	Controls whether the Trust Center will respond to Trust Center link key requests.
EZSP_APP_KEY_REQUEST_POLICY	0x06	Controls whether the Trust Center will respond to application link key requests.

EzspDecisionId		
EZSP_ALLOW_JOINS	0x00	Send the network key in the clear to all joining and rejoining devices.
EZSP_ALLOW_JOINS_REJOINS_HAVE_LINK_KEY	0x04	Send the network key in the clear to all joining devices. Rejoining devices are sent the network key encrypted with their trust center link key. The trust center and any rejoining device are assumed to share a link key, either preconfigured or obtained under a previous policy.
EZSP_ALLOW_PRECONFIGURED_KEY_JOINS	0x01	Send the network key encrypted with the joining or rejoining device's trust center link key. The trust center and any joining or rejoining device are assumed to share a link key, either preconfigured or obtained under a previous policy. This is the default value for the EZSP_TRUST_CENTER_POLICY.
EZSP_ALLOW_REJOINS_ONLY	0x02	Send the network key encrypted with the rejoining device's trust center link key. The trust center and any rejoining device are assumed to share a link key, either preconfigured or obtained under a previous policy. No new devices are allowed to join.
EZSP_DISALLOW_ALL_JOINS_AND_REJOINS	0x03	Reject all unsecured join and rejoin attempts.
EZSP_DISALLOW_BINDING_MODIFICATION	0x10	EZSP_BINDING_MODIFICATION_POLICY default decision. Do not allow the local binding table to be changed by remote nodes.
EZSP_ALLOW_BINDING_MODIFICATION	0x11	EZSP_BINDING_MODIFICATION_POLICY decision. Allow remote nodes to change the local binding table.
EZSP_HOST_WILL_NOT_SUPPLY_REPLY	0x20	EZSP_UNICAST_REPLIES_POLICY default decision. The NCP will automatically send an empty reply (containing no payload) for every unicast received.
EZSP_HOST_WILL_SUPPLY_REPLY	0x21	EZSP_UNICAST_REPLIES_POLICY decision. The NCP will only send a reply if it receives a sendReply command from the Host.
EZSP_POLL_HANDLER_IGNORE	0x30	EZSP_POLL_HANDLER_POLICY default decision. Do not inform the Host when a child polls.
EZSP_POLL_HANDLER_CALLBACK	0x31	EZSP_POLL_HANDLER_POLICY decision. Generate a pollHandler callback when a child polls.

EzspDecisionId		
EZSP_MESSAGE_TAG_ONLY_IN_CALLBACK	0x40	EZSP_MESSAGE_CONTENTS_IN_CALLBACK_POLICY default decision. Include only the message tag in the messageSentHandler callback.
EZSP_MESSAGE_TAG_AND_CONTENTS_IN_CALLBACK	0x41	EZSP_MESSAGE_CONTENTS_IN_CALLBACK_POLICY decision. Include both the message tag and the message contents in the messageSentHandler callback.
EZSP_DENY_TC_KEY_REQUESTS	0x50	EZSP_TC_KEY_REQUEST_POLICY decision. When the Trust Center receives a request for a Trust Center link key, it will be ignored.
EZSP_ALLOW_TC_KEY_REQUESTS	0x51	EZSP_TC_KEY_REQUEST_POLICY decision. When the Trust Center receives a request for a Trust Center link key, it will reply to it with the corresponding key.
EZSP_DENY_APP_KEY_REQUESTS	0x60	EZSP_APP_KEY_REQUEST_POLICY decision. When the Trust Center receives a request for an application link key, it will be ignored.
EZSP_ALLOW_APP_KEY_REQUESTS	0x61	EZSP_APP_KEY_REQUEST_POLICY decision. When the Trust Center receives a request for an application link key, it will randomly generate a key and send it to both partners.

EzspMfgTokenId		
EZSP_MFG_CUSTOM_VERSION	0x00	Custom version (2 bytes).
EZSP_MFG_STRING	0x01	Manufacturing string (16 bytes).
EZSP_MFG_BOARD_NAME	0x02	Board name (16 bytes).
EZSP_MFG_MANUF_ID	0x03	Manufacturing ID (2 bytes).
EZSP_MFG_PHY_CONFIG	0x04	Radio configuration (2 bytes).
EZSP_MFG_BOOTLOAD_AES_KEY	0x05	Bootload AES key (16 bytes).
EZSP_MFG_ASH_CONFIG	0x06	ASH configuration (40 bytes).
EZSP_MFG_EZSP_STORAGE	0x07	EZSP storage (8 bytes).
EZSP_STACK_CAL_DATA	0x08	Radio calibration data (64 bytes). 4 bytes are stored for each of the 16 channels. This token is not stored in the Flash Information Area. It is updated by the stack each time a calibration is performed.
EZSP_MFG_CBKE_DATA	0x09	Certificate Based Key Exchange (CBKE) data (92 bytes).
EZSP_MFG_INSTALLATION_CODE	0x0A	Installation code (20 bytes).
EZSP_STACK_CAL_FILTER	0x0B	Radio channel filter calibration data (1 byte). This token is not stored in the Flash Information Area. It is updated by the stack each time a calibration is performed.

EzspStatus		
EZSP_SUCCESS	0x00	Success.
EZSP_SPI_ERR_FATAL	0x10	Fatal error.
EZSP_SPI_ERR_NCP_RESET	0x11	The Response frame of the current transaction indicates the NCP has reset.
EZSP_SPI_ERR_OVERSIZED_EZSP_FRAME	0x12	The NCP is reporting that the Command frame of the current transaction is oversized (the length byte is too large).
EZSP_SPI_ERR_ABORTED_TRANSACTION	0x13	The Response frame of the current transaction indicates the previous transaction was aborted (nSSEL deasserted too soon).

EzspStatus		
EZSP_SPI_ERR_MISSING_FRAME_TERMINATOR	0x14	The Response frame of the current transaction indicates the frame terminator is missing from the Command frame.
EZSP_SPI_ERR_WAIT_SECTION_TIMEOUT	0x15	The NCP has not provided a Response within the time limit defined by WAIT_SECTION_TIMEOUT.
EZSP_SPI_ERR_NO_FRAME_TERMINATOR	0x16	The Response frame from the NCP is missing the frame terminator.
EZSP_SPI_ERR_EZSP_COMMAND_OVERSIZED	0x17	The Host attempted to send an oversized Command (the length byte is too large) and the AVR's spi-protocol.c blocked the transmission.
EZSP_SPI_ERR_EZSP_RESPONSE_OVERSIZED	0x18	The NCP attempted to send an oversized Response (the length byte is too large) and the AVR's spi-protocol.c blocked the reception.
EZSP_SPI_WAITING_FOR_RESPONSE	0x19	The Host has sent the Command and is still waiting for the NCP to send a Response.
EZSP_SPI_ERR_HANDSHAKE_TIMEOUT	0x1A	The NCP has not asserted nHOST_INT within the time limit defined by WAKE_HANDSHAKE_TIMEOUT.
EZSP_SPI_ERR_STARTUP_TIMEOUT	0x1B	The NCP has not asserted nHOST_INT after an EM260 reset within the time limit defined by STARTUP_TIMEOUT.
EZSP_SPI_ERR_STARTUP_FAIL	0x1C	The Host attempted to verify the SPI Protocol activity and version number, and the verification failed.
EZSP_SPI_ERR_UNSUPPORTED_SPI_COMMAND	0x1D	The Host has sent a command with a SPI Byte that is unsupported by the current mode the NCP is operating in.
EZSP_ASH_IN_PROGRESS	0x20	Operation not yet complete.
EZSP_ASH_HOST_FATAL_ERROR	0x21	Fatal error detected by host.
EZSP_ASH_NCP_FATAL_ERROR	0x22	Fatal error detected by NCP.
EZSP_ASH_DATA_FRAME_TOO_LONG	0x23	Tried to send DATA frame too long.
EZSP_ASH_DATA_FRAME_TOO_SHORT	0x24	Tried to send DATA frame too short.
EZSP_ASH_NO_TX_SPACE	0x25	No space for tx'ed DATA frame.
EZSP_ASH_NO_RX_SPACE	0x26	No space for rec'd DATA frame.
EZSP_ASH_NO_RX_DATA	0x27	No receive data available.

EzspStatus		
EZSP_ASH_NOT_CONNECTED	0x28	Not in Connected state.
EZSP_ERROR_VERSION_NOT_SET	0x30	The NCP received a command before the EZSP version had been set.
EZSP_ERROR_INVALID_FRAME_ID	0x31	The NCP received a command containing an unsupported frame ID.
EZSP_ERROR_WRONG_DIRECTION	0x32	The direction flag in the frame control field was incorrect.
EZSP_ERROR_TRUNCATED	0x33	The truncated flag in the frame control field was set, indicating there was not enough memory available to complete the response or that the response would have exceeded the maximum EZSP frame length.
EZSP_ERROR_OVERFLOW	0x34	The overflow flag in the frame control field was set, indicating one or more callbacks occurred since the previous response and there was not enough memory available to report them to the Host.
EZSP_ERROR_OUT_OF_MEMORY	0x35	Insufficient memory was available.
EZSP_ERROR_INVALID_VALUE	0x36	The value was out of bounds.
EZSP_ERROR_INVALID_ID	0x37	The configuration id was not recognized.
EZSP_ERROR_INVALID_CALL	0x38	Configuration values can no longer be modified.
EZSP_ERROR_NO_RESPONSE	0x39	The NCP failed to respond to a command.
EZSP_ERROR_COMMAND_TOO_LONG	0x40	The length of the command exceeded the maximum EZSP frame length.
EZSP_ERROR_QUEUE_FULL	0x41	The UART receive queue was full causing a callback response to be dropped.
EZSP_ASH_ERROR_VERSION	0x50	Incompatible ASH version
EZSP_ASH_ERROR_TIMEOUTS	0x51	Exceeded max ACK timeouts
EZSP_ASH_ERROR_RESET_FAIL	0x52	Timed out waiting for RSTACK
EZSP_ASH_ERROR_NCP_RESET	0x53	Unexpected ncp reset
EZSP_ASH_ERROR_SERIAL_INIT	0x54	Serial port initialization failed
EZSP_ASH_ERROR_NCP_TYPE	0x55	Invalid ncp processor type

EzspStatus		
EZSP_ASH_ERROR_RESET_METHOD	0x56	Invalid ncp reset method
EZSP_ASH_ERROR_XON_XOFF	0x57	XON/XOFF not supported by host driver
EZSP_ASH_STARTED	0x70	ASH protocol started
EZSP_ASH_CONNECTED	0x71	ASH protocol connected
EZSP_ASH_DISCONNECTED	0x72	ASH protocol disconnected
EZSP_ASH_ACK_TIMEOUT	0x73	Timer expired waiting for ack
EZSP_ASH_CANCELLED	0x74	Frame in progress cancelled
EZSP_ASH_OUT_OF_SEQUENCE	0x75	Received frame out of sequence
EZSP_ASH_BAD_CRC	0x76	Received frame with CRC error
EZSP_ASH_COMM_ERROR	0x77	Received frame with comm error
EZSP_ASH_BAD_ACKNUM	0x78	Received frame with bad ackNum
EZSP_ASH_TOO_SHORT	0x79	Received frame shorter than minimum
EZSP_ASH_TOO_LONG	0x7A	Received frame longer than maximum
EZSP_ASH_BAD_CONTROL	0x7B	Received frame with illegal control byte
EZSP_ASH_BAD_LENGTH	0x7C	Received frame with illegal length for its type
EZSP_ASH_NO_ERROR	0xFF	No reset or error

EmberStatus		
EMBER_SUCCESS	0x00	The generic 'no error' message.
EMBER_ERR_FATAL	0x01	The generic 'fatal error' message.
EMBER_BAD_ARGUMENT	0x02	An invalid value was passed as an argument to a function.
EMBER_EEPROM_MFG_STACK_VERSION_MISMATCH	0x04	The manufacturing and stack token format in non-volatile memory is different than what the stack expects (returned at initialization).

EmberStatus		
EMBER_INCOMPATIBLE_STATIC_MEMORY_DEFINITIONS	0x05	The static memory definitions in ember-static-memory.h are incompatible with this stack version.
EMBER_EEPROM_MFG_VERSION_MISMATCH	0x06	The manufacturing token format in non-volatile memory is different than what the stack expects (returned at initialization).
EMBER_EEPROM_STACK_VERSION_MISMATCH	0x07	The stack token format in non-volatile memory is different than what the stack expects (returned at initialization).
EMBER_NO_BUFFERS	0x18	There are no more buffers.
EMBER_SERIAL_INVALID_BAUD_RATE	0x20	Specified an invalid baud rate.
EMBER_SERIAL_INVALID_PORT	0x21	Specified an invalid serial port.
EMBER_SERIAL_TX_OVERFLOW	0x22	Tried to send too much data.
EMBER_SERIAL_RX_OVERFLOW	0x23	There was not enough space to store a received character and the character was dropped.
EMBER_SERIAL_RX_FRAME_ERROR	0x24	Detected a UART framing error.
EMBER_SERIAL_RX_PARITY_ERROR	0x25	Detected a UART parity error.
EMBER_SERIAL_RX_EMPTY	0x26	There is no received data to process.
EMBER_SERIAL_RX_OVERRUN_ERROR	0x27	The receive interrupt was not handled in time, and a character was dropped.
EMBER_MAC_TRANSMIT_QUEUE_FULL	0x39	The MAC transmit queue is full.
EMBER_MAC_UNKNOWN_HEADER_TYPE	0x3A	MAC header FCR error on receive.
EMBER_MAC_SCANNING	0x3D	The MAC can't complete this task because it is scanning.
EMBER_MAC_NO_DATA	0x31	No pending data exists for device doing a data poll.
EMBER_MAC_JOINED_NETWORK	0x32	Attempt to scan when we are joined to a network.
EMBER_MAC_BAD_SCAN_DURATION	0x33	Scan duration must be 0 to 14 inclusive. Attempt was made to scan with an incorrect duration value.
EMBER_MAC_INCORRECT_SCAN_TYPE	0x34	emberStartScan was called with an incorrect scan type.
EMBER_MAC_INVALID_CHANNEL_MASK	0x35	emberStartScan was called with an invalid channel

EmberStatus		
		mask.
EMBER_MAC_COMMAND_TRANSMIT_FAILURE	0x36	Failed to scan current channel because we were unable to transmit the relevant MAC command.
EMBER_MAC_NO_ACK_RECEIVED	0x40	We expected to receive an ACK following the transmission, but the MAC level ACK was never received.
EMBER_MAC_INDIRECT_TIMEOUT	0x42	Indirect data message timed out before polled.
EMBER_SIM_EEPROM_ERASE_PAGE_GREEN	0x43	The Simulated EEPROM is telling the application that there is at least one flash page to be erased. The GREEN status means the current page has not filled above the ERASE_CRITICAL_THRESHOLD. The application should call the function halSimEepromErasePage when it can to erase a page.
EMBER_SIM_EEPROM_ERASE_PAGE_RED	0x44	The Simulated EEPROM is telling the application that there is at least one flash page to be erased. The RED status means the current page has filled above the ERASE_CRITICAL_THRESHOLD. Due to the shrinking availability of write space, there is a danger of data loss. The application must call the function halSimEepromErasePage as soon as possible to erase a page.
EMBER_SIM_EEPROM_FULL	0x45	The Simulated EEPROM has run out of room to write any new data and the data trying to be set has been lost. This error code is the result of ignoring the SIM_EEPROM_ERASE_PAGE_RED error code. The application must call the function halSimEepromErasePage to make room for any further calls to set a token.
EMBER_ERR_FLASH_WRITE_INHIBITED	0x46	A fatal error has occurred while trying to write data to the Flash. The target memory attempting to be programmed is already programmed. The flash write routines were asked to flip a bit from a 0 to 1, which is physically impossible and the write was therefore inhibited. The data in the flash cannot be trusted after this error.
EMBER_ERR_FLASH_VERIFY_FAILED	0x47	A fatal error has occurred while trying to write data to the Flash and the write verification has failed. The data in the flash cannot be trusted after this error, and it is possible this error is the result of exceeding the life

EmberStatus		
		cycles of the flash.
EMBER_SIM_EEPROM_INIT_1_FAILED	0x48	Attempt 1 to initialize the Simulated EEPROM has failed. This failure means the information already stored in Flash (or a lack thereof), is fatally incompatible with the token information compiled into the code image being run.
EMBER_SIM_EEPROM_INIT_2_FAILED	0x49	Attempt 2 to initialize the Simulated EEPROM has failed. This failure means Attempt 1 failed, and the token system failed to properly reload default tokens and reset the Simulated EEPROM.
EMBER_SIM_EEPROM_INIT_3_FAILED	0x4A	Attempt 3 to initialize the Simulated EEPROM has failed. This failure means one or both of the tokens TOKEN_MFG_NVDATA_VERSION or TOKEN_STACK_NVDATA_VERSION were incorrect and the token system failed to properly reload default tokens and reset the Simulated EEPROM.
EMBER_ERR_FLASH_PROG_FAIL	0x4B	A fatal error has occurred while trying to write data to the flash, possibly due to write protection or an invalid address. The data in the flash cannot be trusted after this error, and it is possible this error is the result of exceeding the life cycles of the flash.
EMBER_ERR_FLASH_ERASE_FAIL	0x4C	A fatal error has occurred while trying to erase flash, possibly due to write protection. The data in the flash cannot be trusted after this error, and it is possible this error is the result of exceeding the life cycles of the flash.
EMBER_ERR_TOKEN_INVALID_SIZE	0x4D	An incorrect size was specified when retrieving token data.
EMBER_ERR_TOKEN_READ_ONLY	0x4E	Couldn't write token because it is marked read-only.
EMBER_ERR_BOOTLOADER_TRAP_TABLE_BAD	0x58	The bootloader received an invalid message (failed attempt to go into bootloader).
EMBER_ERR_BOOTLOADER_TRAP_UNKNOWN	0x59	Bootloader received an invalid message (failed attempt to go into bootloader).
EMBER_ERR_BOOTLOADER_NO_IMAGE	0x5A	The bootloader cannot complete the bootload operation because either an image was not found or the image exceeded memory bounds.

EmberStatus		
EMBER_DELIVERY_FAILED	0x66	The APS layer attempted to send or deliver a message, but it failed.
EMBER_BINDING_INDEX_OUT_OF_RANGE	0x69	This binding index is out of range of the current binding table.
EMBER_ADDRESS_TABLE_INDEX_OUT_OF_RANGE	0x6A	This address table index is out of range for the current address table.
EMBER_INVALID_BINDING_INDEX	0x6C	An invalid binding table index was given to a function.
EMBER_INVALID_CALL	0x70	The API call is not allowed given the current state of the stack.
EMBER_COST_NOT_KNOWN	0x71	The link cost to a node is not known.
EMBER_MAX_MESSAGE_LIMIT_REACHED	0x72	The maximum number of in-flight messages (i.e. EMBER_APS_UNICAST_MESSAGE_COUNT) has been reached.
EMBER_MESSAGE_TOO_LONG	0x74	The message to be transmitted is too big to fit into a single over-the-air packet.
EMBER_BINDING_IS_ACTIVE	0x75	The application is trying to delete or overwrite a binding that is in use.
EMBER_ADDRESS_TABLE_ENTRY_IS_ACTIVE	0x76	The application is trying to overwrite an address table entry that is in use.
EMBER_ADC_CONVERSION_DONE	0x80	Conversion is complete.
EMBER_ADC_CONVERSION_BUSY	0x81	Conversion cannot be done because a request is being processed.
EMBER_ADC_CONVERSION_DEFERRED	0x82	Conversion is deferred until the current request has been processed.
EMBER_ADC_NO_CONVERSION_PENDING	0x84	No results are pending.
EMBER_SLEEP_INTERRUPTED	0x85	Sleeping (for a duration) has been abnormally interrupted and exited prematurely.
EMBER_PHY_TX_UNDERFLOW	0x88	The transmit hardware buffer underflowed.
EMBER_PHY_TX_INCOMPLETE	0x89	The transmit hardware did not finish transmitting a packet.

EmberStatus		
EMBER_PHY_INVALID_CHANNEL	0x8A	An unsupported channel setting was specified.
EMBER_PHY_INVALID_POWER	0x8B	An unsupported power setting was specified.
EMBER_PHY_TX_BUSY	0x8C	The packet cannot be transmitted because the physical MAC layer is currently transmitting a packet. (This is used for the MAC backoff algorithm.)
EMBER_PHY_TX_CCA_FAIL	0x8D	The transmit attempt failed because all CCA attempts indicated that the channel was busy.
EMBER_PHY_OSCILLATOR_CHECK_FAILED	0x8E	The software installed on the hardware doesn't recognize the hardware radio type.
EMBER_PHY_ACK_RECEIVED	0x8F	The expected ACK was received after the last transmission.
EMBER_NETWORK_UP	0x90	The stack software has completed initialization and is ready to send and receive packets over the air.
EMBER_NETWORK_DOWN	0x91	The network is not operating.
EMBER_JOIN_FAILED	0x94	An attempt to join a network failed.
EMBER_MOVE_FAILED	0x96	After moving, a mobile node's attempt to re-establish contact with the network failed.
EMBER_CANNOT_JOIN_AS_ROUTER	0x98	An attempt to join as a router failed due to a ZigBee versus ZigBee Pro incompatibility. ZigBee devices joining ZigBee Pro networks (or vice versa) must join as End Devices, not Routers.
EMBER_NODE_ID_CHANGED	0x99	The local node ID has changed. The application can obtain the new node ID by calling emberGetNodeId().
EMBER_PAN_ID_CHANGED	0x9A	The local PAN ID has changed. The application can obtain the new PAN ID by calling emberGetPanId().
EMBER_NO_BEACONS	0xAB	An attempt to join or rejoin the network failed because no router beacons could be heard by the joining node.
EMBER_RECEIVED_KEY_IN_THE_CLEAR	0xAC	An attempt was made to join a Secured Network using a pre-configured key, but the Trust Center sent back a Network Key in-the-clear when an encrypted Network Key was required.
EMBER_NO_NETWORK_KEY_RECEIVED	0xAD	An attempt was made to join a Secured Network, but

EmberStatus		
		the device did not receive a Network Key.
EMBER_NO_LINK_KEY_RECEIVED	0xAE	After a device joined a Secured Network, a Link Key was requested but no response was ever received.
EMBER_PRECONFIGURED_KEY_REQUIRED	0xAF	An attempt was made to join a Secured Network without a pre-configured key, but the Trust Center sent encrypted data using a pre-configured key.
EMBER_NOT_JOINED	0x93	The node has not joined a network.
EMBER_INVALID_SECURITY_LEVEL	0x95	The chosen security level (the value of EMBER_SECURITY_LEVEL) is not supported by the stack.
EMBER_NETWORK_BUSY	0xA1	A message cannot be sent because the network is currently overloaded.
EMBER_INVALID_ENDPOINT	0xA3	The application tried to send a message using an endpoint that it has not defined.
EMBER_BINDING_HAS_CHANGED	0xA4	The application tried to use a binding that has been remotely modified and the change has not yet been reported to the application.
EMBER_INSUFFICIENT_RANDOM_DATA	0xA5	An attempt to generate random bytes failed because of insufficient random data from the radio.
EMBER_APS_ENCRYPTION_ERROR	0xA6	There was an error in trying to encrypt at the APS Level. This could result from either an inability to determine the long address of the recipient from the short address (no entry in the binding table) or there is no link key entry in the table associated with the destination, or there was a failure to load the correct key into the encryption core.
EMBER_TRUST_CENTER_MASTER_KEY_NOT_SET	0xA7	There was an attempt to form a network using commercial security without setting the Trust Center master key first.
EMBER_SECURITY_STATE_NOT_SET	0xA8	There was an attempt to form or join a network using Standard or Commercial security without calling emberSetSecurity() first.
EMBER_KEY_TABLE_INVALID_ADDRESS	0xB3	There was an attempt to set an entry in the key table using an invalid long address. An entry cannot be set using either the local device's or Trust Center's IEEE address. Or an entry already exists in the table with the same IEEE address. An Address of all zeros or all F's are

EmberStatus		
		not valid addresses in 802.15.4.
EMBER_SECURITY_CONFIGURATION_INVALID	0xB7	There was an attempt to set a security configuration that is not valid given the other security settings.
EMBER_TOO_SOON_FOR_SWITCH_KEY	0xB8	There was an attempt to broadcast a key switch too quickly after broadcasting the next network key. The Trust Center must wait at least a period equal to the broadcast timeout so that all routers have a chance to receive the broadcast of the new network key.
EMBER_KEY_NOT_AUTHORIZED	0xBB	The message could not be sent because the link key corresponding to the destination is not authorized for use in APS data messages. APS Commands (sent by the stack) are allowed. To use it for encryption of APS data messages it must be authorized using a key agreement protocol (such as CBKE).
EMBER_SECURITY_DATA_INVALID	0xBD	The security data provided was not valid, or an integrity check failed.
EMBER_SOURCE_ROUTE_FAILURE	0xA9	A ZigBee route error command frame was received indicating that a source routed message from this node failed en route.
EMBER_MANY_TO_ONE_ROUTE_FAILURE	0xAA	A ZigBee route error command frame was received indicating that a message sent to this node along a many-to-one route failed en route. The route error frame was delivered by an ad-hoc search for a functioning route.
EMBER_STACK_AND_HARDWARE_MISMATCH	0xB0	A critical and fatal error indicating that the version of the stack trying to run does not match with the chip it is running on. The software (stack) on the chip must be replaced with software that is compatible with the chip.
EMBER_INDEX_OUT_OF_RANGE	0xB1	An index was passed into the function that was larger than the valid range.
EMBER_TABLE_FULL	0xB4	There are no empty entries left in the table.
EMBER_TABLE_ENTRY_ERASED	0xB6	The requested table entry has been erased and contains no valid data.
EMBER_LIBRARY_NOT_PRESENT	0xB5	The requested function cannot be executed because the library that contains the necessary functionality is not

EmberStatus		
		present.
EMBER_OPERATION_IN_PROGRESS	0xBA	The stack accepted the command and is currently processing the request. The results will be returned via an appropriate handler.
EMBER_APPLICATION_ERROR_0	0xF0	This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
EMBER_APPLICATION_ERROR_1	0xF1	This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
EMBER_APPLICATION_ERROR_2	0xF2	This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
EMBER_APPLICATION_ERROR_3	0xF3	This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
EMBER_APPLICATION_ERROR_4	0xF4	This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
EMBER_APPLICATION_ERROR_5	0xF5	This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
EMBER_APPLICATION_ERROR_6	0xF6	This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
EMBER_APPLICATION_ERROR_7	0xF7	This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
EMBER_APPLICATION_ERROR_8	0xF8	This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
EMBER_APPLICATION_ERROR_9	0xF9	This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
EMBER_APPLICATION_ERROR_10	0xFA	This error is reserved for customer application use. This will never be returned from any portion of the network

EmberStatus		
		stack or HAL.
EMBER_APPLICATION_ERROR_11	0xFB	This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
EMBER_APPLICATION_ERROR_12	0xFC	This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
EMBER_APPLICATION_ERROR_13	0xFD	This error is reserved for customer application use. This will never be returned from any portion of the network `stack or HAL.
EMBER_APPLICATION_ERROR_14	0xFE	This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
EMBER_APPLICATION_ERROR_15	0xFF	This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.

EmberEventUnits		
EMBER_EVENT_INACTIVE	0x00	The event is not scheduled to run.
EMBER_EVENT_MS_TIME	0x01	The execution time is in approximate milliseconds.
EMBER_EVENT_QS_TIME	0x02	The execution time is in 'binary' quarter seconds (256 approximate milliseconds each).
EMBER_EVENT_MINUTE_TIME	0x03	The execution time is in 'binary' minutes (65536 approximate milliseconds each).

EmberNodeType		
EMBER_UNKNOWN_DEVICE	0x00	Device is not joined.
EMBER_COORDINATOR	0x01	Will relay messages and can act as a parent to other nodes.
EMBER_ROUTER	0x02	Will relay messages and can act as a parent to other nodes.
EMBER_END_DEVICE	0x03	Communicates only with its parent and will not relay messages.
EMBER_SLEEPY_END_DEVICE	0x04	An end device whose radio can be turned off to save power. The application must poll to receive messages.
EMBER_MOBILE_END_DEVICE	0x05	A sleepy end device that can move through the network.

EmberNetworkStatus		
EMBER_NO_NETWORK	0x00	The node is not associated with a network in any way.
EMBER_JOINING_NETWORK	0x01	The node is currently attempting to join a network.
EMBER_JOINED_NETWORK	0x02	The node is joined to a network.
EMBER_JOINED_NETWORK_NO_PARENT	0x03	The node is an end device joined to a network but its parent is not responding.
EMBER_LEAVING_NETWORK	0x04	The node is in the process of leaving its current network.

EmberIncomingMessageType		
EMBER_INCOMING_UNICAST	0x00	Unicast.
EMBER_INCOMING_UNICAST_REPLY	0x01	Unicast reply.
EMBER_INCOMING_MULTICAST	0x02	Multicast.
EMBER_INCOMING_MULTICAST_LOOPBACK	0x03	Multicast sent by the local device.
EMBER_INCOMING_BROADCAST	0x04	Broadcast.
EMBER_INCOMING_BROADCAST_LOOPBACK	0x05	Broadcast sent by the local device.
EMBER_INCOMING_MANY_TO_ONE_ROUTE_REQUEST	0x06	Many to one route request.

EmberOutgoingMessageType		
EMBER_OUTGOING_DIRECT	0x00	Unicast sent directly to an EmberNodeId.
EMBER_OUTGOING_VIA_ADDRESS_TABLE	0x01	Unicast sent using an entry in the address table.
EMBER_OUTGOING_VIA_BINDING	0x02	Unicast sent using an entry in the binding table.
EMBER_OUTGOING_MULTICAST	0x03	Multicast message. This value is passed to emberMessageSentHandler() only. It may not be passed to emberSendUnicast().
EMBER_OUTGOING_BROADCAST	0x04	Broadcast message. This value is passed to emberMessageSentHandler() only. It may not be passed to emberSendUnicast().

EmberMacPassthroughType		
EMBER_MAC_PASSTHROUGH_NONE	0x00	No MAC passthrough messages.
EMBER_MAC_PASSTHROUGH_SE_INTERPAN	0x01	SE InterPAN messages.
EMBER_MAC_PASSTHROUGH_EMBERNET	0x02	Legacy EmberNet messages.
EMBER_MAC_PASSTHROUGH_EMBERNET_SOURCE	0x04	Legacy EmberNet messages filtered by their source address.

EmberBindingType		
EMBER_UNUSED_BINDING	0x00	A binding that is currently not in use.
EMBER_UNICAST_BINDING	0x01	A unicast binding whose 64-bit identifier is the destination EUI64.
EMBER_MANY_TO_ONE_BINDING	0x02	A unicast binding whose 64-bit identifier is the aggregator EUI64.
EMBER_MULTICAST_BINDING	0x03	A multicast binding whose 64-bit identifier is the group address. A multicast binding can be used to send messages to the group and to receive messages sent to the group.

EmberApsOption		
EMBER_APS_OPTION_NONE	0x0000	No options.
EMBER_APS_OPTION_ENCRYPTION	0x0020	Send the message using APS Encryption, using the Link Key shared with the destination node to encrypt the data at the APS Level.
EMBER_APS_OPTION_RETRY	0x0040	Resend the message using the APS retry mechanism.
EMBER_APS_OPTION_ENABLE_ROUTE_DISCOVERY	0x0100	Causes a route discovery to be initiated if no route to the destination is known.
EMBER_APS_OPTION_FORCE_ROUTE_DISCOVERY	0x0200	Causes a route discovery to be initiated even if one is known.
EMBER_APS_OPTION_SOURCE_EUI64	0x0400	Include the source EUI64 in the network frame.
EMBER_APS_OPTION_DESTINATION_EUI64	0x0800	Include the destination EUI64 in the network frame.
EMBER_APS_OPTION_ENABLE_ADDRESS_DISCOVERY	0x1000	Send a ZDO request to discover the node ID of the destination, if it is not already know.
EMBER_APS_OPTION_POLL_RESPONSE	0x2000	Reserved.
EMBER_APS_OPTION_ZDO_RESPONSE_REQUIRED	0x4000	This incoming message is a ZDO request not handled by the EmberZNet stack, and the application is responsible for sending a ZDO response. This flag is used only when the ZDO is configured to have requests handled by the application. See the EZSP_CONFIG_APPLICATION_ZDO_FLAGS configuration parameter for more information.
EMBER_APS_OPTION_FRAGMENT	0x8000	This message is part of a fragmented message. This option may only be set for unicasts. The groupId field gives the index of this fragment in the low-order byte. If the low-order byte is zero this is the first fragment and the high-order byte contains the number of fragments in the message.

EzspNetworkScanType		
EZSP_ENERGY_SCAN	0x00	An energy scan scans each channel for its RSSI value.
EZSP_ACTIVE_SCAN	0x01	An active scan scans each channel for available networks.

EmberJoinDecision		
EMBER_USE_PRECONFIGURED_KEY	0x00	Allow the node to join. The joining node should have a pre-configured key. The security data sent to it will be encrypted with that key.
EMBER_SEND_KEY_IN_THE_CLEAR	0x01	Allow the node to join. Send the necessary key (the Network Key in Standard Security mode, the Trust Center Master in High Security mode) in-the-clear to the joining device.
EMBER_DENY_JOIN	0x02	Deny join.
EMBER_NO_ACTION	0x03	Take no action.

EmberInitialSecurityBitmask		
EMBER_STANDARD_SECURITY_MODE	0x0000	This enables ZigBee Standard Security on the node.
EMBER_HIGH_SECURITY_MODE	0x0001	This enables ZigBee High Security on the node.
EMBER_DISTRIBUTED_TRUST_CENTER_MODE	0x0002	This enables Distributed Trust Center Mode for the device forming the network. (Previously known as EMBER_NO_TRUST_CENTER_MODE)
EMBER_GLOBAL_LINK_KEY	0x0004	This enables a Global Link Key for the Trust Center. All nodes will share the same Trust Center Link Key.
EMBER_PRECONFIGURED_NETWORK_KEY_MODE	0x0008	This enables devices that perform MAC Association with a pre-configured Network Key to join the network. It is only set on the Trust Center.
EMBER_TRUST_CENTER_USES_HASHED_LINK_KEY	0x0084	This denotes that the preconfiguredKey is not the actual Link Key but a Secret Key known only to the Trust Center. It is hashed with the IEEE Address of the destination device in order to create the actual Link Key used in encryption. This is bit is only used by the Trust Center. The joining device need not set this.
EMBER_HAVE_PRECONFIGURED_KEY	0x0100	This denotes that the preconfiguredKey element has valid data that should be used to configure the initial security state.
EMBER_HAVE_NETWORK_KEY	0x0200	This denotes that the networkKey element has valid data that should be used to configure the initial security state.
EMBER_GET_LINK_KEY_WHEN_JOINING	0x0400	This denotes to a joining node that it should attempt to acquire a Trust Center Link Key during joining. This is only necessary if the device does not have a pre-configured key.
EMBER_REQUIRE_ENCRYPTED_KEY	0x0800	This denotes that a joining device should only accept an encrypted network key from the Trust Center (using its pre-configured key). A key sent in-the-clear by the Trust Center will be rejected and the join will fail. This option is only valid when utilizing a pre-configured key.
EMBER_NO_FRAME_COUNTER_RESET	0x1000	This denotes whether the device should NOT reset its outgoing frame counters (both NWK and APS) when ::emberSetInitialSecurityState() is called. Normally it is advised to reset the frame counter before joining a new network. However in cases where a device is joining to the same network a again (but not using ::emberRejoinNetwork()) it should keep the NWK and APS frame counters stored in its

EmberInitialSecurityBitmask		
		tokens.
EMBER_GET_PRECONFIGURED_KEY_FROM_INSTALL_CODE	0x2000	This denotes that the device should obtain its preconfigured key from an installation code stored in the manufacturing token. The token contains a value that will be hashed to obtain the actual preconfigured key. If that token is not valid, then the call to emberSetInitialSecurityState() will fail.
EMBER_HAVE_TRUST_CENTER_EUI64	0x0040	This denotes that the ::EmberInitialSecurityState::preconfiguredTrustCenterEui64 has a value in it containing the trust center EUI64. The device will only join a network and accept commands from a trust center with that EUI64. Normally this bit is NOT set, and the EUI64 of the trust center is learned during the join process. When commissioning a device to join onto an existing network, which is using a trust center, and without sending any messages, this bit must be set and the field ::EmberInitialSecurityState::preconfiguredTrustCenterEui64 must be populated with the appropriate EUI64.

EmberCurrentSecurityBitmask		
EMBER_STANDARD_SECURITY_MODE	0x0000	This denotes that the device is running in a network with ZigBee Standard Security.
EMBER_HIGH_SECURITY_MODE	0x0001	This denotes that the device is running in a network with ZigBee High Security.
EMBER_DISTRIBUTED_TRUST_CENTER_MODE	0x0002	This denotes that the device is running in a network without a centralized Trust Center.
EMBER_GLOBAL_LINK_KEY	0x0004	This denotes that the device has a Global Link Key. The Trust Center Link Key is the same across multiple nodes.
EMBER_HAVE_TRUST_CENTER_LINK_KEY	0x0010	This denotes that the node has a Trust Center Link Key.
EMBER_TRUST_CENTER_USES_HASHED_LINK_KEY	0x0084	This denotes that the Trust Center is using a Hashed Link Key.

EmberKeyType		
EMBER_TRUST_CENTER_LINK_KEY	0x01	A shared key between the Trust Center and a device.
EMBER_TRUST_CENTER_MASTER_KEY	0x02	A shared secret used for deriving keys between the Trust Center and a device
EMBER_CURRENT_NETWORK_KEY	0x03	The current active Network Key used by all devices in the network.
EMBER_NEXT_NETWORK_KEY	0x04	The alternate Network Key that was previously in use, or the newer key that will be switched to.
EMBER_APPLICATION_LINK_KEY	0x05	An Application Link Key shared with another (non-Trust Center) device.
EMBER_APPLICATION_MASTER_KEY	0x06	An Application Master Key shared secret used to derive an Application Link Key.

EmberKeyStructBitmask		
EMBER_KEY_HAS_SEQUENCE_NUMBER	0x0001	The key has a sequence number associated with it.
EMBER_KEY_HAS_OUTGOING_FRAME_COUNTER	0x0002	The key has an outgoing frame counter associated with it.
EMBER_KEY_HAS_INCOMING_FRAME_COUNTER	0x0004	The key has an incoming frame counter associated with it.
EMBER_KEY_HAS_PARTNER_EUI64	0x0008	The key has a Partner IEEE address associated with it.

EmberDeviceUpdate	
EMBER_STANDARD_SECURITY_SECURED_REJOIN	0x0
EMBER_STANDARD_SECURITY_UNSECURED_JOIN	0x1
EMBER_DEVICE_LEFT	0x2
EMBER_STANDARD_SECURITY_UNSECURED_REJOIN	0x3
EMBER_HIGH_SECURITY_SECURED_REJOIN	0x4
EMBER_HIGH_SECURITY_UNSECURED_JOIN	0x5
EMBER_HIGH_SECURITY_UNSECURED_REJOIN	0x7

EmberKeyStatus	
EMBER_APP_LINK_KEY_ESTABLISHED	0x01
EMBER_APP_MASTER_KEY_ESTABLISHED	0x02
EMBER_TRUST_CENTER_LINK_KEY_ESTABLISHED	0x03
EMBER_KEY_ESTABLISHMENT_TIMEOUT	0x04
EMBER_KEY_TABLE_FULL	0x05
EMBER_TC_RESPONDED_TO_KEY_REQUEST	0x06
EMBER_TC_APP_KEY_SENT_TO_REQUESTER	0x07
EMBER_TC_RESPONSE_TO_KEY_REQUEST_FAILED	0x08
EMBER_TC_REQUEST_KEY_TYPE_NOT_SUPPORTED	0x09
EMBER_TC_NO_LINK_KEY_FOR_REQUESTER	0x0A
EMBER_TC_REQUESTER_EUI64_UNKNOWN	0x0B
EMBER_TC_RECEIVED_FIRST_APP_KEY_REQUEST	0x0C
EMBER_TC_TIMEOUT_WAITING_FOR_SECOND_APP_KEY_REQUEST	0x0D
EMBER_TC_NON_MATCHING_APP_KEY_REQUEST_RECEIVED	0x0E
EMBER_TC_FAILED_TO_SEND_APP_KEYS	0x0F
EMBER_TC_FAILED_TO_STORE_APP_KEY_REQUEST	0x10
EMBER_TC_REJECTED_APP_KEY_REQUEST	0x11

EmberCounterType		
EMBER_COUNTER_MAC_RX_BROADCAST	0	The MAC received a broadcast.
EMBER_COUNTER_MAC_TX_BROADCAST	1	The MAC transmitted a broadcast.
EMBER_COUNTER_MAC_RX_UNICAST	2	The MAC received a unicast.
EMBER_COUNTER_MAC_TX_UNICAST_SUCCESS	3	The MAC successfully transmitted a unicast.
EMBER_COUNTER_MAC_TX_UNICAST_RETRY	4	The MAC retried a unicast.
EMBER_COUNTER_MAC_TX_UNICAST_FAILED	5	The MAC unsuccessfully transmitted a unicast.
EMBER_COUNTER_APS_DATA_RX_BROADCAST	6	The APS layer received a data broadcast.
EMBER_COUNTER_APS_DATA_TX_BROADCAST	7	The APS layer transmitted a data broadcast.
EMBER_COUNTER_APS_DATA_RX_UNICAST	8	The APS layer received a data unicast.
EMBER_COUNTER_APS_DATA_TX_UNICAST_SUCCESS	9	The APS layer successfully transmitted a data unicast.
EMBER_COUNTER_APS_DATA_TX_UNICAST_RETRY	10	The APS layer retried a data unicast.
EMBER_COUNTER_APS_DATA_TX_UNICAST_FAILED	11	The APS layer unsuccessfully transmitted a data unicast.
EMBER_COUNTER_ROUTE_DISCOVERY_INITIATED	12	The network layer successfully submitted a new route discovery to the MAC.
EMBER_COUNTER_NEIGHBOR_ADDED	13	An entry was added to the neighbor table.
EMBER_COUNTER_NEIGHBOR_REMOVED	14	An entry was removed from the neighbor table.
EMBER_COUNTER_NEIGHBOR_STALE	15	A neighbor table entry became stale because it had not been heard from.
EMBER_COUNTER_JOIN_INDICATION	16	A node joined or rejoined to the network via this node.
EMBER_COUNTER_CHILD_REMOVED	17	An entry was removed from the child table.
EMBER_COUNTER_ASH_OVERFLOW_ERROR	18	EZSP-UART only. An overflow error occurred in the UART.
EMBER_COUNTER_ASH_FRAMING_ERROR	19	EZSP-UART only. A framing error occurred in the UART.
EMBER_COUNTER_ASH_OVERRUN_ERROR	20	EZSP-UART only. An overrun error occurred in the UART.

EmberCounterType		
EMBER_COUNTER_NWK_FRAME_COUNTER_FAILURE	21	A message was dropped at the network layer because the NWK frame counter was not higher than the last message seen from that source.
EMBER_COUNTER_APS_FRAME_COUNTER_FAILURE	22	A message was dropped at the APS layer because the APS frame counter was not higher than the last message seen from that source.
EMBER_COUNTER_UTILITY	23	Utility counter for general debugging use.
EMBER_COUNTER_APS_LINK_KEY_NOT_AUTHORIZED	24	A message was dropped at the APS layer because it had APS encryption but the key associated with the sender has not been authenticated, and thus the key is not authorized for use in APS data messages.
EMBER_COUNTER_NWK_DECRYPTION_FAILURE	25	A NWK encrypted message was received but dropped because decryption failed.
EMBER_COUNTER_APS_DECRYPTION_FAILURE	26	An APS encrypted message was received but dropped because decryption failed.
EMBER_COUNTER_ALLOCATE_PACKET_BUFFER_FAILURE	27	The number of times we failed to allocate a set of linked packet buffers. This doesn't necessarily mean that the packet buffer count was 0 at the time, but that the number requested was greater than the number free.
EMBER_COUNTER_RELAYED_UNICAST	28	The number of relayed unicast packets.
EMBER_COUNTER_TYPE_COUNT	29	A placeholder giving the number of Ember counter types.

EmberJoinMethod		
EMBER_USE_MAC_ASSOCIATION	0x0	
EMBER_USE_NWK_REJOIN	0x1	
EMBER_USE_NWK_REJOIN_HAVE_NWK_KEY	0x2	
EMBER_USE_NWK_COMMISSIONING	0x3	

EmberZdoConfigurationFlags		
EMBER_APP_RECEIVES_SUPPORTED_ZDO_REQUESTS	0x01	Set this flag in order to receive supported ZDO request messages via the incomingMessageHandler callback. A supported ZDO request is one that is handled by the EmberZNet stack. The stack will continue to handle the request and send the appropriate ZDO response even if this configuration option is enabled.
EMBER_APP_HANDLES_UNSUPPORTED_ZDO_REQUESTS	0x02	Set this flag in order to receive unsupported ZDO request messages via the incomingMessageHandler callback. An unsupported ZDO request is one that is not handled by the EmberZNet stack, other than to send a 'not supported' ZDO response. If this configuration option is enabled, the stack will no longer send any ZDO response, and it is the application's responsibility to do so.
EMBER_APP_HANDLES_ZDO_ENDPOINT_REQUESTS	0x04	Set this flag in order to receive the following ZDO request messages via the incomingMessageHandler callback: SIMPLE_DESCRIPTOR_REQUEST, MATCH_DESCRIPTORS_REQUEST, and ACTIVE_ENDPOINTS_REQUEST. If this configuration option is enabled, the stack will no longer send any ZDO response for these requests, and it is the application's responsibility to do so.
EMBER_APP_HANDLES_ZDO_BINDING_REQUESTS	0x08	Set this flag in order to receive the following ZDO request messages via the incomingMessageHandler callback: BINDING_TABLE_REQUEST, BIND_REQUEST, and UNBIND_REQUEST. If this configuration option is enabled, the stack will no longer send any ZDO response for these requests, and it is the application's responsibility to do so.

EmberConcentratorType		
EMBER_LOW_RAM_CONCENTRATOR	0xFFF8	A concentrator with insufficient memory to store source routes for the entire network. Route records are sent to the concentrator prior to every inbound APS unicast.
EMBER_HIGH_RAM_CONCENTRATOR	0xFFF9	A concentrator with sufficient memory to store source routes for the entire network. Remote nodes stop sending route records once the concentrator has successfully received one.

3.4 Configuration Frames

Name: version	**ID:** 0x00
Description: The command allows the Host to specify the desired EZSP version and must be sent before any other command. This document describes EZSP version 4 and stack type 2 (mesh). The response provides information about the firmware running on the NCP.	
Command Parameters:	
int8u desiredProtocolVersion	The EZSP version the Host wishes to use. To successfully set the version and allow other commands, this must be 4.
Response Parameters:	
int8u protocolVersion	The EZSP version the NCP is using (4).
int8u stackType	The type of stack running on the NCP (2).
int16u stackVersion	The version number of the stack.

Name: getConfigurationValue	**ID:** 0x52
Description: Reads a configuration value from the NCP.	
Command Parameters:	
EzspConfigId configId	Identifies which configuration value to read.
Response Parameters:	
EzspStatus status	EZSP_SUCCESS if the value was read successfully, EZSP_ERROR_INVALID_ID if the NCP does not recognize *configId*.
int16u value	The configuration value.

Name: setConfigurationValue	**ID:** 0x53
Description: Writes a configuration value to the NCP. Configuration values can be modified by the Host after the NCP has reset. Once the status of the stack changes to EMBER_NETWORK_UP, configuration values can no longer be modified and this command will respond with EZSP_ERROR_INVALID_CALL.	
Command Parameters:	
EzspConfigId configId	Identifies which configuration value to change.
int16u value	The new configuration value.
Response Parameters:	
EzspStatus status	EZSP_SUCCESS if the configuration value was changed, EZSP_ERROR_OUT_OF_MEMORY if the new value exceeded the available memory, EZSP_ERROR_INVALID_VALUE if the new value was out of bounds, EZSP_ERROR_INVALID_ID if the NCP does not recognize *configId*, EZSP_ERROR_INVALID_CALL if configuration values can no longer be modified.

Name: addEndpoint	**ID:** 0x02
Description: Configures endpoint information on the NCP. The NCP does not remember these settings after a reset. Endpoints can be added by the Host after the NCP has reset. Once the status of the stack changes to EMBER_NETWORK_UP, endpoints can no longer be added and this command will respond with EZSP_ERROR_INVALID_CALL.	
Command Parameters:	
int8u endpoint	The application endpoint to be added.
int16u profileId	The endpoint's application profile.
int16u deviceId	The endpoint's device ID within the application profile.
int8u appFlags	The device version and flags indicating description availability.
int8u inputClusterCount	The number of cluster IDs in *inputClusterList*.
int8u outputClusterCount	The number of cluster IDs in *outputClusterList*.
int16u[] inputClusterList	Input cluster IDs the endpoint will accept.
int16u[] outputClusterList	Output cluster IDs the endpoint may send.
Response Parameters:	
EzspStatus status	EZSP_SUCCESS if the endpoint was added, EZSP_ERROR_OUT_OF_MEMORY if there is not enough memory available to add the endpoint, EZSP_ERROR_INVALID_VALUE if the endpoint already exists, EZSP_ERROR_INVALID_CALL if endpoints can no longer be added.

Name: setPolicy	**ID:** 0x55
Description: Allows the Host to change the policies used by the NCP to make fast decisions.	
Command Parameters:	
EzspPolicyId policyId	Identifies which policy to modify.
EzspDecisionId decisionId	The new decision for the specified policy.
Response Parameters:	
EzspStatus status	EZSP_SUCCESS if the policy was changed, EZSP_ERROR_INVALID_ID if the NCP does not recognize *policyId*.

Name: getPolicy	**ID:** 0x56
Description: Allows the Host to read the policies used by the NCP to make fast decisions.	
Command Parameters:	
EzspPolicyId policyId	Identifies which policy to read.
Response Parameters:	
EzspStatus status	EZSP_SUCCESS if the policy was read successfully, EZSP_ERROR_INVALID_ID if the NCP does not recognize *policyId*.
EzspDecisionId decisionId	The current decision for the specified policy.

Name: getValue	**ID:** 0xAA
Description: Reads a value from the NCP.	
Command Parameters:	
EzspValueId valueId	Identifies which value to read.
Response Parameters:	
EzspStatus status	EZSP_SUCCESS if the value was read successfully, EZSP_ERROR_INVALID_ID if the NCP does not recognize *valueId*.
int8u valueLength	The length of the *value* parameter in bytes.
int8u[] value	The value.

Name: setValue	**ID:** 0xAB
Description: Writes a value to the NCP.	
Command Parameters:	
EzspValueId valueId	Identifies which value to change.
int8u valueLength	The length of the *value* parameter in bytes.
int8u[] value	The new value.
Response Parameters:	
EzspStatus status	EZSP_SUCCESS if the value was changed, EZSP_ERROR_INVALID_VALUE if the new value was out of bounds, EZSP_ERROR_INVALID_ID if the NCP does not recognize *valueId*, EZSP_ERROR_INVALID_CALL if the value could not be modified.

3.5 Utilities Frames

Name: nop	**ID:** 0x05
Description: A command which does nothing. The Host can use this to set the sleep mode or to check the status of the NCP.	
Command Parameters: None	
Response Parameters: None	

Name: echo	**ID:** 0x81
Description: Variable length data from the Host is echoed back by the NCP. This command has no other effects and is designed for testing the link between the Host and NCP.	
Command Parameters:	
int8u dataLength	The length of the *data* parameter in bytes.
int8u[] data	The data to be echoed back.
Response Parameters:	
int8u echoLength	The length of the *echo* parameter in bytes.
int8u[] echo	The echo of the data.

Name: invalidCommand	**ID:** 0x58
Description: Indicates that the NCP received an invalid command.	
This frame is a response to an invalid command.	
Response Parameters:	
EzspStatus reason	The reason why the command was invalid.

Name: callback	**ID:** 0x06
Description: Allows the NCP to respond with a pending callback.	
Command Parameters: None	
The response to this command can be any of the callback responses.	

Name: noCallbacks	**ID:** 0x07
Description: Indicates that there are currently no pending callbacks.	
This frame is a response to the *callback* command.	
Response Parameters: None	

Name: setToken	**ID:** 0x09
Description: Sets a token (8 bytes of non-volatile storage) in the Simulated EEPROM of the NCP.	
Command Parameters:	
int8u tokenId	Which token to set (0 to 7).
int8u[8] tokenData	The data to write to the token.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: getToken	**ID:** 0x0A
Description: Retrieves a token (8 bytes of non-volatile storage) from the Simulated EEPROM of the NCP.	
Command Parameters:	
int8u tokenId	Which token to read (0 to 7).
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.
int8u[8] tokenData	The contents of the token.

Name: getMfgToken	**ID:** 0x0B
Description: Retrieves a manufacturing token from the Flash Information Area of the NCP (except for EZSP_STACK_CAL_DATA which is managed by the stack).	
Command Parameters:	
EzspMfgTokenId tokenId	Which manufacturing token to read.
Response Parameters:	
int8u tokenDataLength	The length of the *tokenData* parameter in bytes.
int8u[] tokenData	The manufacturing token data.

Name: getRandomNumber	**ID:** 0x49
Description: Returns a pseudorandom number.	
Command Parameters: None	
Response Parameters:	
EmberStatus status	Always returns EMBER_SUCCESS.
int16u value	A pseudorandom number.

Name: setTimer	**ID:** 0x0E
Description: Sets a timer on the NCP. There are 2 independent timers available for use by the Host. A timer can be cancelled by setting *time* to 0 or *units* to EMBER_EVENT_INACTIVE.	
Command Parameters:	
int8u timerId	Which timer to set (0 or 1).
int16u time	The delay before the *timerHandler* callback will be generated. Note that the timer clock is free running and is not synchronized with this command. This means that the actual delay will be between *time* and (*time* - 1). The maximum delay is 32767.
EmberEventUnits units	The units for *time*.
boolean repeat	If true, a *timerHandler* callback will be generated repeatedly. If false, only a single *timerHandler* callback will be generated.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: getTimer	**ID:** 0x4E
Description: Gets information about a timer. The Host can use this command to find out how much longer it will be before a previously set timer will generate a callback.	
Command Parameters:	
int8u timerId	Which timer to get information about (0 or 1).
Response Parameters:	
int16u time	The delay before the *timerHandler* callback will be generated.
EmberEventUnits units	The units for *time*.
boolean repeat	True if a *timerHandler* callback will be generated repeatedly. False if only a single *timerHandler* callback will be generated.

Name: timerHandler	**ID:** 0x0F
Description: A callback from the timer.	
This frame is a response to the *callback* command.	
Response Parameters:	
int8u timerId	Which timer generated the callback (0 or 1).

Name: debugWrite	**ID:** 0x12
Description: Sends a debug message from the Host to the InSight debug system via the NCP.	
Command Parameters:	
boolean binaryMessage	TRUE if the message should be interpreted as binary data, FALSE if the message should be interpreted as ASCII text.
int8u messageLength	The length of the *messageContents* parameter in bytes.
int8u[] messageContents	The binary message.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: readAndClearCounters	**ID:** 0x65
Description: Retrieves and clears Ember counters. See the EmberCounterType enumeration for the counter types.	
Command Parameters: None	
Response Parameters:	
int16u[EMBER_COUNTER_TYPE_COUNT] values	A list of all counter values ordered according to the EmberCounterType enumeration.

Name: delayTest	**ID:** 0x9D
Description: Used to test that UART flow control is working correctly.	
Command Parameters:	
int16u delay	Data will not be read from the host for this many milliseconds.
Response Parameters: None	

Name: getLibraryStatus	**ID:** 0x01
Description: This retrieves the status of the passed library ID to determine if it is compiled into the stack.	
Command Parameters:	
int8u libraryId	The ID of the library being queried.
Response Parameters:	
EmberLibraryStatus status	The status of the library being queried.

3.6 Networking Frames

Name: setManufacturerCode	**ID:** 0x15
Description: Sets the manufacturer code to the specified value. The manufacturer code is one of the fields of the node descriptor.	
Command Parameters:	
int16u code	The manufacturer code for the local node.
Response Parameters: None	

Name: setPowerDescriptor	**ID:** 0x16
Description: Sets the power descriptor to the specified value. The power descriptor is a dynamic value, therefore you should call this function whenever the value changes.	
Command Parameters:	
int16u descriptor	The new power descriptor for the local node.
Response Parameters: None	

Name: networkInit	**ID:** 0x17
Description: Resume network operation after a reboot. The node retains its original type. This should be called on startup whether or not the node was previously part of a network. EMBER_NOT_JOINED is returned if the node is not part of a network.	
Command Parameters: None	
Response Parameters:	
EmberStatus status	An EmberStatus value that indicates one of the following: successful initialization, EMBER_NOT_JOINED if the node is not part of a network, or the reason for failure.

Name: networkState	**ID:** 0x18
Description: Returns a value indicating whether the node is joining, joined to, or leaving a network.	
Command Parameters: None	
Response Parameters:	
EmberNetworkStatus status	An EmberNetworkStatus value indicating the current join status.

Name: stackStatusHandler	**ID:** 0x19
Description: A callback invoked when the status of the stack changes. If the status parameter equals EMBER_NETWORK_UP, then the *getNetworkParameters* command can be called to obtain the new network parameters. If any of the parameters are being stored in nonvolatile memory by the Host, the stored values should be updated.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberStatus status	Stack status. One of the following: EMBER_NETWORK_UP, EMBER_NETWORK_DOWN, EMBER_JOIN_FAILED, EMBER_MOVE_FAILED

Name: startScan	**ID:** 0x1A
Description: This function will start a scan.	
Command Parameters:	
EzspNetworkScanType scanType	Indicates the type of scan to be performed. Possible values are: EZSP_ENERGY_SCAN and EZSP_ACTIVE_SCAN. For each type, the respective callback for reporting results is: energyScanResultHandler and networkFoundHandler. The energy scan and active scan report errors and completion via the scanCompleteHandler.
int32u channelMask	Bits set as 1 indicate that this particular channel should be scanned. Bits set to 0 indicate that this particular channel should not be scanned. For example, a channelMask value of 0x00000001 would indicate that only channel 0 should be scanned. Valid channels range from 11 to 26 inclusive. This translates to a channel mask value of 0x07FFF800. As a convenience, a value of 0 is reinterpreted as the mask for the current channel.
int8u duration	Sets the exponent of the number of scan periods, where a scan period is 960 symbols. The scan will occur for ((2^duration) + 1) scan periods.
Response Parameters:	
EmberStatus status	EMBER_SUCCESS signals that the scan successfully started. Possible error responses and their meanings: EMBER_MAC_SCANNING, we are already scanning; EMBER_MAC_JOINED_NETWORK, we are currently joined to a network and cannot begin a scan; EMBER_MAC_BAD_SCAN_DURATION, we have set a duration value that is not 0..14 inclusive; EMBER_MAC_INCORRECT_SCAN_TYPE, we have requested an undefined scanning type; EMBER_MAC_INVALID_CHANNEL_MASK, our channel mask did not specify any valid channels.

Name: energyScanResultHandler	**ID:** 0x48
Description: Reports the result of an energy scan for a single channel. The scan is not complete until the *scanCompleteHandler* callback is called.	
This frame is a response to the *callback* command.	
Response Parameters:	
int8u channel	The 802.15.4 channel number that was scanned.
int8s maxRssiValue	The maximum RSSI value found on the channel.

Name: networkFoundHandler	**ID:** 0x1B
Description: Reports that a network was found as a result of a prior call to *startScan*. Gives the network parameters useful for deciding which network to join.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberZigbeeNetwork networkFound	The parameters associated with the network found.
int8u lastHopLqi	The link quality from the node that generated this beacon.
int8s lastHopRssi	The energy level (in units of dBm) observed during the reception.

Name: scanCompleteHandler	**ID:** 0x1C
Description: Returns the status of the current scan of type EZSP_ENERGY_SCAN or EZSP_ACTIVE_SCAN. EMBER_SUCCESS signals that the scan has completed. Other error conditions signify a failure to scan on the channel specified.	
This frame is a response to the *callback* command.	
Response Parameters:	
int8u channel	The channel on which the current error occurred. Undefined for the case of EMBER_SUCCESS.
EmberStatus status	The error condition that occurred on the current channel. Value will be EMBER_SUCCESS when the scan has completed.

Name: stopScan	**ID:** 0x1D
Description: Terminates a scan in progress.	
Command Parameters: None	
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: formNetwork	**ID:** 0x1E
Description: Forms a new network by becoming the coordinator.	
Command Parameters:	
EmberNetworkParameters parameters	Specification of the new network.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: joinNetwork	**ID:** 0x1F
Description: Causes the stack to associate with the network using the specified network parameters. It can take several seconds for the stack to associate with the local network. Do not send messages until the *stackStatusHandler* callback informs you that the stack is up.	
Command Parameters:	
EmberNodeType nodeType	Specification of the role that this node will have in the network. This role must not be EMBER_COORDINATOR. To be a coordinator, use the *formNetwork* command.
EmberNetworkParameters parameters	Specification of the network with which the node should associate.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: leaveNetwork	**ID:** 0x20
Description: Causes the stack to leave the current network. This generates a *stackStatusHandler* callback to indicate that the network is down. The radio will not be used until after sending a *formNetwork* or *joinNetwork* command.	
Command Parameters: None	
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: findAndRejoinNetwork	**ID:** 0x21
Description: The application may call this function when contact with the network has been lost. The most common usage case is when an end device can no longer communicate with its parent and wishes to find a new one. Another case is when a device has missed a Network Key update and no longer has the current Network Key. The stack will call *ezspStackStatusHandler* to indicate that the network is down, then try to re-establish contact with the network by performing an active scan, choosing a network with matching extended pan id, and sending a ZigBee network rejoin request. A second call to the *ezspStackStatusHandler* callback indicates either the success or the failure of the attempt. The process takes approximately 150 milliseconds per channel to complete. This call replaces the *emberMobileNodeHasMoved* API from EmberZNet 2.x, which used MAC association and consequently took half a second longer to complete.	
Command Parameters:	
boolean haveCurrentNetworkKey	This parameter tells the stack whether to try to use the current network key. If it has the current network key it will perform a secure rejoin (encrypted). If this fails the device should try an unsecure rejoin. If the Trust Center allows the rejoin then the current Network Key will be sent encrypted using the device's Link Key. The unsecured rejoin is only supported in the Commercial Security Library.
int32u channelMask	A mask indicating the channels to be scanned. See *emberStartScan* for format details. A value of 0 is reinterpreted as the mask for the current channel.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: permitJoining	**ID:** 0x22
Description: Tells the stack to allow other nodes to join the network with this node as their parent. Joining is initially disabled by default.	
Command Parameters:	
int8u duration	A value of 0x00 disables joining. A value of 0xFF enables joining. Any other value enables joining for that number of seconds.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: childJoinHandler	**ID:** 0x23
Description: Indicates that a child has joined or left.	
This frame is a response to the *callback* command.	
Response Parameters:	
int8u index	The index of the child of interest.
boolean joining	True if the child is joining. False the child is leaving.
EmberNodeId childId	The node ID of the child.
EmberEUI64 childEui64	The EUI64 of the child.
EmberNodeType childType	The node type of the child.

Name: energyScanRequest	**ID:** 0x9C
Description: Sends a ZDO energy scan request. This request may only be sent by the current network manager and must be unicast, not broadcast. See ezsp-utils.h for related macros emberSetNetworkManagerRequest() and emberChangeChannelRequest().	
Command Parameters:	
EmberNodeId target	The network address of the node to perform the scan.
int32u scanChannels	A mask of the channels to be scanned.
int8u scanDuration	How long to scan on each channel. Allowed values are 0..5, with the scan times as specified by 802.15.4 (0 = 31ms, 1 = 46ms, 2 = 77ms, 3 = 138ms, 4 = 261ms, 5 = 507ms).
int16u scanCount	The number of scans to be performed on each channel (1..8).
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: getEui64	**ID:** 0x26
Description: Returns the EUI64 ID of the local node.	
Command Parameters: None	
Response Parameters:	
EmberEUI64 eui64	The 64-bit ID.

Name: getNodeId	**ID:** 0x27
Description: Returns the 16-bit node ID of the local node.	
Command Parameters: None	
Response Parameters:	
EmberNodeId nodeId	The 16-bit ID.

Name: getNetworkParameters	**ID:** 0x28
Description: Returns the current network parameters.	
Command Parameters: None	
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.
EmberNodeType nodeType	An EmberNodeType value indicating the current node type.
EmberNetworkParameters parameters	The current network parameters.

Name: getParentChildParameters	**ID:** 0x29
Description: Returns information about the children of the local node and the parent of the local node.	
Command Parameters: None	
Response Parameters:	
int8u childCount	The number of children the node currently has.
EmberEUI64 parentEui64	The parent's EUI64. The value is undefined for nodes without parents (coordinators and nodes that are not joined to a network).
EmberNodeId parentNodeId	The parent's node ID. The value is undefined for nodes without parents (coordinators and nodes that are not joined to a network).

Name: getChildData	**ID:** 0x4A
Description: Returns information about a child of the local node.	
Command Parameters:	
int8u index	The index of the child of interest in the child table. Possible indexes range from zero to EMBER_CHILD_TABLE_SIZE.
Response Parameters:	
EmberStatus status	EMBER_SUCCESS if there is a child at *index*. EMBER_NOT_JOINED if there is no child at *index*.
EmberNodeId childId	The node ID of the child.
EmberEUI64 childEui64	The EUI64 of the child.
EmberNodeType childType	The EmberNodeType value for the child.

Name: getNeighbor	**ID:** 0x79
Description: Returns the neighbor table entry at the given index. The number of active neighbors can be obtained using the neighborCount command.	
Command Parameters:	
int8u index	The index of the neighbor of interest. Neighbors are stored in ascending order by node id, with all unused entries at the end of the table.
Response Parameters:	
EmberStatus status	EMBER_ERR_FATAL if the index is greater or equal to the number of active neighbors, or if the device is an end device. Returns EMBER_SUCCESS otherwise.
EmberNeighborTableEntry value	The contents of the neighbor table entry.

Name: neighborCount	**ID:** 0x7A
Description: Returns the number of active entries in the neighbor table.	
Command Parameters: None	
Response Parameters:	
int8u value	The number of active entries in the neighbor table.

Name: getRouteTableEntry	**ID:** 0x7B
Description: Returns the route table entry at the given index. The route table size can be obtained using the getConfigurationValue command.	
Command Parameters:	
int8u index	The index of the route table entry of interest.
Response Parameters:	
EmberStatus status	EMBER_ERR_FATAL if the index is out of range or the device is an end device, and EMBER_SUCCESS otherwise.
EmberRouteTableEntry value	The contents of the route table entry.

Name: setRadioPower	**ID:** 0x99
Description: Sets the radio output power at which a node is operating. Ember radios have discrete power settings. For a list of available power settings, see the technical specification for the RF communication module in your Developer Kit. Note: Care should be taken when using this API on a running network, as it will directly impact the established link qualities neighboring nodes have with the node on which it is called. This can lead to disruption of existing routes and erratic network behavior.	
Command Parameters:	
int8s power	Desired radio output power, in dBm.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating the success or failure of the command.

Name: setRadioChannel	**ID:** 0x9A
Description: Sets the channel to use for sending and receiving messages. For a list of available radio channels, see the technical specification for the RF communication module in your Developer Kit. Note: Care should be taken when using this API, as all devices on a network must use the same channel.	
Command Parameters:	
int8u channel	Desired radio channel.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating the success or failure of the command.

3.7 Binding Frames

Name: clearBindingTable	**ID:** 0x2A
Description: Deletes all binding table entries.	
Command Parameters: None	
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: setBinding	**ID:** 0x2B
Description: Sets an entry in the binding table.	
Command Parameters:	
int8u index	The index of a binding table entry.
EmberBindingTableEntry value	The contents of the binding entry.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: getBinding	**ID:** 0x2C
Description: Gets an entry from the binding table.	
Command Parameters:	
int8u index	The index of a binding table entry.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.
EmberBindingTableEntry value	The contents of the binding entry.

Name: deleteBinding	**ID:** 0x2D
Description: Deletes a binding table entry.	
Command Parameters:	
int8u index	The index of a binding table entry.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: bindingIsActive	**ID:** 0x2E
Description: Indicates whether any messages are currently being sent using this binding table entry. Note that this command does not indicate whether a binding is clear. To determine whether a binding is clear, check whether the type field of the EmberBindingTableEntry has the value EMBER_UNUSED_BINDING.	
Command Parameters:	
int8u index	The index of a binding table entry.
Response Parameters:	
boolean active	True if the binding table entry is active, false otherwise.

Name: getBindingRemoteNodeId	**ID:** 0x2F
Description: Returns the node ID for the binding's destination, if the ID is known. If a message is sent using the binding and the destination's ID is not known, the stack will discover the ID by broadcasting a ZDO address request. The application can avoid the need for this discovery by using *setBindingRemoteNodeId* when it knows the correct ID via some other means. The destination's node ID is forgotten when the binding is changed, when the local node reboots or, much more rarely, when the destination node changes its ID in response to an ID conflict.	
Command Parameters:	
int8u index	The index of a binding table entry.
Response Parameters:	
EmberNodeId nodeId	The short ID of the destination node or EMBER_NULL_NODE_ID if no destination is known.

Name: setBindingRemoteNodeId	**ID:** 0x30
Description: Set the node ID for the binding's destination. See *getBindingRemoteNodeId* for a description.	
Command Parameters:	
int8u index	The index of a binding table entry.
EmberNodeId nodeId	The short ID of the destination node.
Response Parameters: None	

Name: remoteSetBindingHandler	**ID:** 0x31
Description: The NCP used the external binding modification policy to decide how to handle a remote set binding request. The Host cannot change the current decision, but it can change the policy for future decisions using the *setPolicy* command.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberBindingTableEntry entry	The requested binding.
int8u index	The index at which the binding was added.
EmberStatus policyDecision	EMBER_SUCCESS if the binding was added to the table and any other status if not.

Name: remoteDeleteBindingHandler	**ID:** 0x32
Description: The NCP used the external binding modification policy to decide how to handle a remote delete binding request. The Host cannot change the current decision, but it can change the policy for future decisions using the *setPolicy* command.	
This frame is a response to the *callback* command.	
Response Parameters:	
int8u index	The index of the binding whose deletion was requested.
EmberStatus policyDecision	EMBER_SUCCESS if the binding was removed from the table and any other status if not.

3.8 Messaging Frames

Name: maximumPayloadLength	**ID:** 0x33
Description: Returns the maximum size of the payload. The size depends on the security level in use.	
Command Parameters: None	
Response Parameters:	
int8u apsLength	The maximum APS payload length.

Name: sendUnicast	**ID:** 0x34
Description: Sends a unicast message as per the ZigBee specification. The message will arrive at its destination only if there is a known route to the destination node. Setting the ENABLE_ROUTE_DISCOVERY option will cause a route to be discovered if none is known. Setting the FORCE_ROUTE_DISCOVERY option will force route discovery. Routes to end-device children of the local node are always known. Setting the APS_RETRY option will cause the message to be retransmitted until either a matching acknowledgement is received or three transmissions have been made. **Note:** Using the FORCE_ROUTE_DISCOVERY option will cause the first transmission to be consumed by a route request as part of discovery, so the application payload of this packet will not reach its destination on the first attempt. If you want the packet to reach its destination, the APS_RETRY option must be set so that another attempt is made to transmit the message with its application payload after the route has been constructed. **Note:** When sending fragmented messages, the stack will only assign a new APS sequence number for the first fragment of the message (i.e., EMBER_APS_OPTION_FRAGMENT is set and the low-order byte of the groupId field in the APS frame is zero). For all subsequent fragments of the same message, the application must set the sequence number field in the APS frame to the sequence number assigned by the stack to the first fragment.	
Command Parameters:	
EmberOutgoingMessageType type	Specifies the outgoing message type. Must be one of EMBER_OUTGOING_DIRECT, EMBER_OUTGOING_VIA_ADDRESS_TABLE, or EMBER_OUTGOING_VIA_BINDING.
EmberNodeId indexOrDestination	Depending on the type of addressing used, this is either the EmberNodeId of the destination, an index into the address table, or an index into the binding table.
EmberApsFrame apsFrame	The APS frame which is to be added to the message.
int8u messageTag	A value chosen by the Host. This value is used in the *ezspMessageSentHandler* response to refer to this message.
int8u messageLength	The length of the *messageContents* parameter in bytes.
int8u[] messageContents	Content of the message.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.
int8u sequence	The sequence number that will be used when this message is transmitted.

Name: sendBroadcast	**ID:** 0x36
Description: Sends a broadcast message as per the ZigBee specification.	
Command Parameters:	
EmberNodeId destination	The destination to which to send the broadcast. This must be one of the three ZigBee broadcast addresses.
EmberApsFrame apsFrame	The APS frame for the message.
int8u radius	The message will be delivered to all nodes within *radius* hops of the sender. A radius of zero is converted to EMBER_MAX_HOPS.
int8u messageTag	A value chosen by the Host. This value is used in the *ezspMessageSentHandler* response to refer to this message.
int8u messageLength	The length of the *messageContents* parameter in bytes.
int8u[] messageContents	The broadcast message.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.
int8u sequence	The sequence number that will be used when this message is transmitted.

Name: sendMulticast	**ID:** 0x38
Description: Sends a multicast message to all endpoints that share a specific multicast ID and are within a specified number of hops of the sender.	
Command Parameters:	
EmberApsFrame apsFrame	The APS frame for the message. The multicast will be sent to the groupId in this frame.
int8u hops	The message will be delivered to all nodes within this number of hops of the sender. A value of zero is converted to EMBER_MAX_HOPS.
int8u nonmemberRadius	The number of hops that the message will be forwarded by devices that are not members of the group. A value of 7 or greater is treated as infinite.
int8u messageTag	A value chosen by the Host. This value is used in the *ezspMessageSentHandler* response to refer to this message.
int8u messageLength	The length of the *messageContents* parameter in bytes.
int8u[] messageContents	The multicast message.
Response Parameters:	
EmberStatus status	An EmberStatus value. For any result other than EMBER_SUCCESS, the message will not be sent. EMBER_SUCCESS - The message has been submitted for transmission. EMBER_INVALID_BINDING_INDEX - The bindingTableIndex refers to a non-multicast binding. EMBER_NETWORK_DOWN - The node is not part of a network. EMBER_MESSAGE_TOO_LONG - The message is too large to fit in a MAC layer frame. EMBER_NO_BUFFERS - The free packet buffer pool is empty. EMBER_NETWORK_BUSY - Insufficient resources available in Network or MAC layers to send message.
int8u sequence	The sequence number that will be used when this message is transmitted.

Name: sendReply	**ID:** 0x39
Description: Sends a reply to a received unicast message. The *incomingMessageHandler* callback for the unicast being replied to supplies the values for all the parameters except the reply itself.	
Command Parameters:	
EmberNodeId sender	Value supplied by incoming unicast.
EmberApsFrame apsFrame	Value supplied by incoming unicast.
int8u messageLength	The length of the *messageContents* parameter in bytes.
int8u[] messageContents	The reply message.
Response Parameters:	
EmberStatus status	An EmberStatus value. EMBER_INVALID_CALL - The EZSP_UNICAST_REPLIES_POLICY is set to EZSP_HOST_WILL_NOT_SUPPLY_REPLY. This means the NCP will automatically send an empty reply. The Host must change the policy to EZSP_HOST_WILL_SUPPLY_REPLY before it can supply the reply. There is one exception to this rule: In the case of responses to message fragments, the host must call sendReply when a message fragment is received. In this case, the policy set on the NCP does not matter. The NCP expects a sendReply call from the Host for message fragments regardless of the current policy settings. EMBER_NO_BUFFERS - Not enough memory was available to send the reply. EMBER_NETWORK_BUSY - Either no route or insufficient resources available. EMBER_SUCCESS - The reply was successfully queued for transmission.

Name: messageSentHandler	**ID:** 0x3F
Description: A callback indicating the stack has completed sending a message.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberOutgoingMessageType type	The type of message sent.
int16u indexOrDestination	The destination to which the message was sent, for direct unicasts, or the address table or binding index for other unicasts. The value is unspecified for multicasts and broadcasts.
EmberApsFrame apsFrame	The APS frame for the message.
int8u messageTag	The value supplied by the Host in the *ezspSendUnicast*, *ezspSendBroadcast* or *ezspSendMulticast* command.
EmberStatus status	An EmberStatus value of EMBER_SUCCESS if an ACK was received from the destination or EMBER_DELIVERY_FAILED if no ACK was received.
int8u messageLength	The length of the *messageContents* parameter in bytes.
int8u[] messageContents	The unicast message supplied by the Host. The message contents are only included here if the decision for the messageContentsInCallback policy is messageTagAndContentsInCallback.

Name: sendManyToOneRouteRequest **ID:** 0x41

Description: Sends a route request packet that creates routes from every node in the network back to this node. This function should be called by an application that wishes to communicate with many nodes, for example, a gateway, central monitor, or controller. A device using this function was referred to as an 'aggregator' in EmberZNet 2.x and earlier, and is referred to as a 'concentrator' in the ZigBee specification and EmberZNet 3.

This function enables large scale networks, because the other devices do not have to individually perform bandwidth-intensive route discoveries. Instead, when a remote node sends an APS unicast to a concentrator, its network layer automatically delivers a special route record packet first, which lists the network ids of all the intermediate relays. The concentrator can then use source routing to send outbound APS unicasts. (A source routed message is one in which the entire route is listed in the network layer header.) This allows the concentrator to communicate with thousands of devices without requiring large route tables on neighboring nodes.

This function is only available in ZigBee Pro (stack profile 2), and cannot be called on end devices. Any router can be a concentrator (not just the coordinator), and there can be multiple concentrators on a network.

Note that a concentrator does not automatically obtain routes to all network nodes after calling this function. Remote applications must first initiate an inbound APS unicast.

Many-to-one routes are not repaired automatically. Instead, the concentrator application must call this function to rediscover the routes as necessary, for example, upon failure of a retried APS message. The reason for this is that there is no scalable one-size-fits-all route repair strategy. A common and recommended strategy is for the concentrator application to refresh the routes by calling this function periodically.

Command Parameters:

int16u concentratorType	Must be either EMBER_HIGH_RAM_CONCENTRATOR or EMBER_LOW_RAM_CONCENTRATOR. The former is used when the caller has enough memory to store source routes for the whole network. In that case, remote nodes stop sending route records once the concentrator has successfully received one. The latter is used when the concentrator has insufficient RAM to store all outbound source routes. In that case, route records are sent to the concentrator prior to every inbound APS unicast.
int8u radius	The maximum number of hops the route request will be relayed. A radius of zero is converted to EMBER_MAX_HOPS.

Response Parameters:

EmberStatus status	EMBER_SUCCESS if the route request was successfully submitted to the transmit queue, and EMBER_ERR_FATAL otherwise.

Name: pollForData	**ID:** 0x42
Description: Periodically request any pending data from our parent. Setting *interval* to 0 or *units* to EMBER_EVENT_INACTIVE will generate a single poll.	
Command Parameters:	
int16u interval	The time between polls. Note that the timer clock is free running and is not synchronized with this command. This means that the time will be between *interval* and (*interval* - 1). The maximum interval is 32767.
EmberEventUnits units	The units for *interval*.
int8u failureLimit	The number of poll failures that will be tolerated before a *pollCompleteHandler* callback is generated. A value of zero will result in a callback for every poll. Any status value apart from EMBER_SUCCESS and EMBER_MAC_NO_DATA is counted as a failure.
Response Parameters:	
EmberStatus status	The result of sending the first poll.

Name: pollCompleteHandler	**ID:** 0x43
Description: Indicates the result of a data poll to the parent of the local node.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberStatus status	An EmberStatus value: EMBER_SUCCESS - Data was received in response to the poll. EMBER_MAC_NO_DATA - No data was pending. EMBER_DELIVERY_FAILED - The poll message could not be sent. EMBER_MAC_NO_ACK_RECEIVED - The poll message was sent but not acknowledged by the parent.

Name: pollHandler	**ID:** 0x44
Description: Indicates that the local node received a data poll from a child.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberNodeId childId	The node ID of the child that is requesting data.

Name: incomingSenderEui64Handler	**ID:** 0x62
Description: A callback indicating a message has been received containing the EUI64 of the sender. This callback is called immediately before the *incomingMessageHandler* callback. It is not called if the incoming message did not contain the EUI64 of the sender.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberEUI64 senderEui64	The EUI64 of the sender

Name: incomingMessageHandler	**ID:** 0x45
Description: A callback indicating a message has been received.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberIncomingMessageType type	The type of the incoming message. One of the following: EMBER_INCOMING_UNICAST, EMBER_INCOMING_UNICAST_REPLY, EMBER_INCOMING_MULTICAST, EMBER_INCOMING_MULTICAST_LOOPBACK, EMBER_INCOMING_BROADCAST, EMBER_INCOMING_BROADCAST_LOOPBACK
EmberApsFrame apsFrame	The APS frame from the incoming message.
int8u lastHopLqi	The link quality from the node that last relayed the message.
int8s lastHopRssi	The energy level (in units of dBm) observed during the reception.
EmberNodeId sender	The sender of the message.
int8u bindingIndex	The index of a binding that matches the message or 0xFF if there is no matching binding.
int8u addressIndex	The index of the entry in the address table that matches the sender of the message or 0xFF if there is no matching entry.
int8u messageLength	The length of the *messageContents* parameter in bytes.
int8u[] messageContents	The incoming message.

Name: incomingRouteRecordHandler	**ID:** 0x59
Description: Reports the arrival of a route record command frame.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberNodeId source	The source of the route record.
EmberEUI64 sourceEui	The EUI64 of the source.
int8u lastHopLqi	The link quality from the node that last relayed the route record.
int8s lastHopRssi	The energy level (in units of dBm) observed during the reception.
int8u relayCount	The number of relays in *relayList*.
int8u[] relayList	The route record. Each relay in the list is an int16u node ID. The list is passed as int8u * to avoid alignment problems.

Name: setSourceRoute	**ID:** 0x5A
Description: Supply a source route for the next outgoing message.	
Command Parameters:	
EmberNodeId destination	The destination of the source route.
int8u relayCount	The number of relays in *relayList*.
int16u[] relayList	The source route.
Response Parameters:	
EmberStatus status	EMBER_SUCCESS if the source route was successfully stored, and EMBER_NO_BUFFERS otherwise.

Name: incomingManyToOneRouteRequestHandler	**ID:** 0x7D
Description: A callback indicating that a many-to-one route to the concentrator with the given short and long id is available for use.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberNodeId source	The short id of the concentrator.
EmberEUI64 longId	The EUI64 of the concentrator.
int8u cost	The path cost to the concentrator. The cost may decrease as additional route request packets for this discovery arrive, but the callback is made only once.

Name: incomingRouteErrorHandler	**ID:** 0x80
Description: A callback invoked when a route error message is received. The error indicates that a problem routing to or from the target node was encountered.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberStatus status	EMBER_SOURCE_ROUTE_FAILURE or EMBER_MANY_TO_ONE_ROUTE_FAILURE.
EmberNodeId target	The short id of the remote node.

Name: addressTableEntryIsActive	**ID:** 0x5B
Description: Indicates whether any messages are currently being sent using this address table entry. Note that this function does not indicate whether the address table entry is unused. To determine whether an address table entry is unused, check the remote node ID. The remote node ID will have the value EMBER_TABLE_ENTRY_UNUSED_NODE_ID when the address table entry is not in use.	
Command Parameters:	
int8u addressTableIndex	The index of an address table entry.
Response Parameters:	
boolean active	True if the address table entry is active, false otherwise.

Name: setAddressTableRemoteEui64	**ID:** 0x5C
Description: Sets the EUI64 of an address table entry. This function will also check other address table entries, the child table and the neighbor table to see if the node ID for the given EUI64 is already known. If known then this function will also set node ID. If not known it will set the node ID to EMBER_UNKNOWN_NODE_ID.	
Command Parameters:	
int8u addressTableIndex	The index of an address table entry.
EmberEUI64 eui64	The EUI64 to use for the address table entry.
Response Parameters:	
EmberStatus status	EMBER_SUCCESS if the EUI64 was successfully set, and EMBER_ADDRESS_TABLE_ENTRY_IS_ACTIVE otherwise.

Name: setAddressTableRemoteNodeId	**ID:** 0x5D
Description: Sets the short ID of an address table entry. Usually the application will not need to set the short ID in the address table. Once the remote EUI64 is set the stack is capable of figuring out the short ID on its own. However, in cases where the application does set the short ID, the application must set the remote EUI64 prior to setting the short ID.	
Command Parameters:	
int8u addressTableIndex	The index of an address table entry.
EmberNodeId id	The short ID corresponding to the remote node whose EUI64 is stored in the address table at the given index or EMBER_TABLE_ENTRY_UNUSED_NODE_ID which indicates that the entry stored in the address table at the given index is not in use.
Response Parameters: None	

Name: getAddressTableRemoteEui64	**ID:** 0x5E
Description: Gets the EUI64 of an address table entry.	
Command Parameters:	
int8u addressTableIndex	The index of an address table entry.
Response Parameters:	
EmberEUI64 eui64	The EUI64 of the address table entry is copied to this location.

Name: getAddressTableRemoteNodeId	**ID:** 0x5F
Description: Gets the short ID of an address table entry.	
Command Parameters:	
int8u addressTableIndex	The index of an address table entry.
Response Parameters:	
EmberNodeId nodeId	One of the following: The short ID corresponding to the remote node whose EUI64 is stored in the address table at the given index. EMBER_UNKNOWN_NODE_ID - Indicates that the EUI64 stored in the address table at the given index is valid but the short ID is currently unknown. EMBER_DISCOVERY_ACTIVE_NODE_ID - Indicates that the EUI64 stored in the address table at the given location is valid and network address discovery is underway. EMBER_TABLE_ENTRY_UNUSED_NODE_ID - Indicates that the entry stored in the address table at the given index is not in use.

Name: setExtendedTimeout	**ID:** 0x7E
Description: Tells the stack whether or not the normal interval between retransmissions of a retried unicast message should be increased by EMBER_INDIRECT_TRANSMISSION_TIMEOUT. The interval needs to be increased when sending to a sleepy node so that the message is not retransmitted until the destination has had time to wake up and poll its parent. The stack will automatically extend the timeout: - For our own sleepy children. - When an address response is received from a parent on behalf of its child. - When an indirect transaction expiry route error is received. - When an end device announcement is received from a sleepy node.	
Command Parameters:	
EmberEUI64 remoteEui64	The address of the node for which the timeout is to be set.
boolean extendedTimeout	TRUE if the retry interval should be increased by EMBER_INDIRECT_TRANSMISSION_TIMEOUT. FALSE if the normal retry interval should be used.
Response Parameters: None	

Name: getExtendedTimeout	**ID:** 0x7F
Description: Indicates whether or not the stack will extend the normal interval between retransmissions of a retried unicast message by EMBER_INDIRECT_TRANSMISSION_TIMEOUT.	
Command Parameters:	
EmberEUI64 remoteEui64	The address of the node for which the timeout is to be returned.
Response Parameters:	
boolean extendedTimeout	TRUE if the retry interval will be increased by EMBER_INDIRECT_TRANSMISSION_TIMEOUT and FALSE if the normal retry interval will be used.

Name: replaceAddressTableEntry	**ID:** 0x82
Description: Replaces the EUI64, short ID and extended timeout setting of an address table entry. The previous EUI64, short ID and extended timeout setting are returned.	
Command Parameters:	
int8u addressTableIndex	The index of the address table entry that will be modified.
EmberEUI64 newEui64	The EUI64 to be written to the address table entry.
EmberNodeId newId	One of the following: The short ID corresponding to the new EUI64. EMBER_UNKNOWN_NODE_ID if the new EUI64 is valid but the short ID is unknown and should be discovered by the stack. EMBER_TABLE_ENTRY_UNUSED_NODE_ID if the address table entry is now unused.
boolean newExtendedTimeout	TRUE if the retry interval should be increased by EMBER_INDIRECT_TRANSMISSION_TIMEOUT. FALSE if the normal retry interval should be used.
Response Parameters:	
EmberStatus status	EMBER_SUCCESS if the EUI64, short ID and extended timeout setting were successfully modified, and EMBER_ADDRESS_TABLE_ENTRY_IS_ACTIVE otherwise.
EmberEUI64 oldEui64	The EUI64 of the address table entry before it was modified.
EmberNodeId oldId	One of the following: The short ID corresponding to the EUI64 before it was modified. EMBER_UNKNOWN_NODE_ID if the short ID was unknown. EMBER_DISCOVERY_ACTIVE_NODE_ID if discovery of the short ID was underway. EMBER_TABLE_ENTRY_UNUSED_NODE_ID if the address table entry was unused.
boolean oldExtendedTimeout	TRUE if the retry interval was being increased by EMBER_INDIRECT_TRANSMISSION_TIMEOUT. FALSE if the normal retry interval was being used.

Name: lookupNodeIdByEui64	**ID:** 0x60
Description: Returns the node ID that corresponds to the specified EUI64. The node ID is found by searching through all stack tables for the specified EUI64.	
Command Parameters:	
EmberEUI64 eui64	The EUI64 of the node to look up.
Response Parameters:	
EmberNodeId nodeId	The short ID of the node or EMBER_NULL_NODE_ID if the short ID is not known.

Name: lookupEui64ByNodeId	**ID:** 0x61
Description: Returns the EUI64 that corresponds to the specified node ID. The EUI64 is found by searching through all stack tables for the specified node ID.	
Command Parameters:	
EmberNodeId nodeId	The short ID of the node to look up.
Response Parameters:	
EmberStatus status	EMBER_SUCCESS if the EUI64 was found, EMBER_ERR_FATAL if the EUI64 is not known.
EmberEUI64 eui64	The EUI64 of the node.

Name: getMulticastTableEntry	**ID:** 0x63
Description: Gets an entry from the multicast table.	
Command Parameters:	
int8u index	The index of a multicast table entry.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.
EmberMulticastTableEntry value	The contents of the multicast entry.

Name: setMulticastTableEntry	**ID:** 0x64
Description: Sets an entry in the multicast table.	
Command Parameters:	
int8u index	The index of a multicast table entry
EmberMulticastTableEntry value	The contents of the multicast entry.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: idConflictHandler	**ID:** 0x7C
Description: A callback invoked by the EmberZNet stack when an id conflict is discovered, that is, two different nodes in the network were found to be using the same short id. The stack automatically removes the conflicting short id from its internal tables (address, binding, route, neighbor, and child tables). The application should discontinue any other use of the id.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberNodeId id	The short id for which a conflict was detected

Name: sendRawMessage	**ID:** 0x96
Description: Transmits the given message without modification. The MAC header is assumed to be configured in the message at the time this function is called.	
Command Parameters:	
int8u messageLength	The length of the *messageContents* parameter in bytes.
int8u[] messageContents	The raw message.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: macPassthroughMessageHandler	**ID:** 0x97
Description: A callback invoked by the EmberZNet stack when a MAC passthrough message is received.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberMacPassthroughType messageType	The type of MAC passthrough message received.
int8u lastHopLqi	The link quality from the node that last relayed the message.
int8s lastHopRssi	The energy level (in units of dBm) observed during reception.
int8u messageLength	The length of the *messageContents* parameter in bytes.
int8u[] messageContents	The raw message that was received.

Name: macFilterMatchMessageHandler	**ID:** 0x46
Description: A callback invoked by the EmberZNet stack when a raw MAC message that has matched one of the application's configured MAC filters.	
This frame is a response to the *callback* command.	
Response Parameters:	
int8u filterIndexMatch	The index of the filter that was matched.
EmberMacPassthroughType legacyPassthroughType	The type of MAC passthrough message received.
int8u lastHopLqi	The link quality from the node that last relayed the message.
int8s lastHopRssi	The energy level (in units of dBm) observed during reception.
int8u messageLength	The length of the *messageContents* parameter in bytes.
int8u[] messageContents	The raw message that was received.

Name: rawTransmitCompleteHandler	**ID:** 0x98
Description: A callback invoked by the EmberZNet stack when the MAC has finished transmitting a raw message.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberStatus status	EMBER_SUCCESS if the transmission was successful, or EMBER_DELIVERY_FAILED if not

3.9 Security Frames

Name: setInitialSecurityState	**ID:** 0x68
Description: Sets the security state that will be used by the device when it forms or joins the network.	
Command Parameters:	
EmberInitialSecurityState state	The security configuration to be set.
Response Parameters:	
EmberStatus success	The success or failure code of the operation.

Name: getCurrentSecurityState	**ID:** 0x69
Description: Gets the current security state that is being used by a device that is joined in the network.	
Command Parameters: None	
Response Parameters:	
EmberStatus status	The success or failure code of the operation.
EmberCurrentSecurityState state	The security configuration in use by the stack.

Name: getKey	**ID:** 0x6a
Description: Gets a Security Key based on the passed key type.	
Command Parameters:	
EmberKeyType keyType	
Response Parameters:	
EmberStatus status	The success or failure code of the operation.
EmberKeyStruct keyStruct	The structure containing the key and its associated data.

Name: switchNetworkKeyHandler	**ID:** 0x6e
Description: A callback to inform the application that the Network Key has been updated and the node has been switched over to use the new key. The actual key being used is not passed up, but the sequence number is.	
This frame is a response to the *callback* command.	
Response Parameters:	
int8u sequenceNumber	The sequence number of the new network key.

Name: getKeyTableEntry	**ID:** 0x71
Description: This function retrieves the key table entry at the specified index. If the index is invalid or the key table entry is empty, then FALSE is returned.	
Command Parameters:	
int8u index	The index of the entry in the table to retrieve.
Response Parameters:	
EmberStatus status	The success or failure error code of the operation.
EmberKeyStruct keyStruct	A structure containing the data to be written by the stack.

Name: setKeyTableEntry	**ID:** 0x72
Description: This function sets the key table entry at the specified index. If the index is invalid, then FALSE is returned.	
Command Parameters:	
int8u index	The index of the entry in the table to set.
EmberEUI64 address	The address of the partner device that shares the key
boolean linkKey	This boolean indicates whether the key is a Link or a Master Key
EmberKeyData keyData	The actual key data associated with the table entry.
Response Parameters:	
EmberStatus status	The success or failure error code of the operation.

Name: findKeyTableEntry	**ID:** 0x75
Description: This function searches through the Key Table and tries to find the entry that matches the passed search criteria.	
Command Parameters:	
EmberEUI64 address	The address to search for. Alternatively, all zeros may be passed in to search for the first empty entry.
boolean linkKey	This indicates whether to search for an entry that contains a link key or a master key. TRUE means to search for an entry with a Link Key.
Response Parameters:	
int8u index	This indicates the index of the entry that matches the search criteria. A value of 0xFF is returned if not matching entry is found.

Name: addOrUpdateKeyTableEntry	**ID:** 0x66
Description: This function updates an existing entry in the key table or adds a new one. It first searches the table for an existing entry that matches the passed EUI64 address. If no entry is found, it searches for the first free entry. If successful, it updates the key data and resets the associated incoming frame counter. If it fails to find an existing entry and no free one exists, it returns a failure.	
Command Parameters:	
EmberEUI64 address	The address of the partner device associated with the Key.
boolean linkKey	An indication of whether this is a Link Key (TRUE) or Master Key (FALSE)
EmberKeyData keyData	The actual key data associated with the entry.
Response Parameters:	
EmberStatus status	The success or failure error code of the operation.

Name: eraseKeyTableEntry	**ID:** 0x76
Description: This function erases the data in the key table entry at the specified index. If the index is invalid, FALSE is returned.	
Command Parameters:	
int8u index	This indicates the index of entry to erase.
Response Parameters:	
EmberStatus status	The success or failure of the operation.

Name: requestLinkKey	**ID:** 0x14
Description: A function to request a Link Key from the Trust Center with another device on the Network (which could be the Trust Center). A Link Key with the Trust Center is possible but the requesting device cannot be the Trust Center. Link Keys are optional in ZigBee Standard Security and thus the stack cannot know whether the other device supports them. If EMBER_REQUEST_KEY_TIMEOUT is non-zero on the Trust Center and the partner device is not the Trust Center, both devices must request keys with their partner device within the time period. The Trust Center only supports one outstanding key request at a time and therefore will ignore other requests. If the timeout is zero then the Trust Center will immediately respond and not wait for the second request. The Trust Center will always immediately respond to requests for a Link Key with it. Sleepy devices should poll at a higher rate until a response is received or the request times out. The success or failure of the request is returned via ezspZigbeeKeyEstablishmentHandler(...).	
Command Parameters:	
EmberEUI64 partner	This is the IEEE address of the partner device that will share the link key.
Response Parameters:	
EmberStatus status	The success or failure of sending the request. This is not the final result of the attempt. ezspZigbeeKeyEstablishmentHandler(...) will return that.

Name: zigbeeKeyEstablishmentHandler	**ID:** 0x9B
Description: This is a callback that indicates the success or failure of an attempt to establish a key with a partner device.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberEUI64 partner	This is the IEEE address of the partner that the device successfully established a key with. This value is all zeros on a failure.
EmberKeyStatus status	This is the status indicating what was established or why the key establishment failed.

3.10 Trust Center Frames

Name: trustCenterJoinHandler	**ID:** 0x24
Description: The NCP used the trust center behavior policy to decide whether to allow a new node to join the network. The Host cannot change the current decision, but it can change the policy for future decisions using the *setPolicy* command.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberNodeId newNodeId	The Node Id of the node whose status changed
EmberEUI64 newNodeEui64	The EUI64 of the node whose status changed.
EmberDeviceUpdate status	The status of the node: Secure Join/Rejoin, Unsecure Join/Rejoin, Device left.
EmberJoinDecision policyDecision	An EmberJoinDecision reflecting the decision made.
EmberNodeId parentOfNewNodeId	The parent of the node whose status has changed.

Name: broadcastNextNetworkKey	**ID:** 0x73
Description: This function broadcasts a new encryption key, but does not tell the nodes in the network to start using it. To tell nodes to switch to the new key, use emberSendNetworkKeySwitch(). This is only valid for the Trust Center/Coordinator. It is up to the application to determine how quickly to send the Switch Key after sending the alternate encryption key.	
Command Parameters:	
EmberKeyData key	An optional pointer to a 16-byte encryption key (EMBER_ENCRYPTION_KEY_SIZE). An all zero key may be passed in, which will cause the stack to randomly generate a new key.
Response Parameters:	
EmberStatus status	EmberStatus value that indicates the success or failure of the command.

Name: broadcastNetworkKeySwitch	**ID:** 0x74
Description: This function broadcasts a switch key message to tell all nodes to change to the sequence number of the previously sent Alternate Encryption Key.	
Command Parameters: None	
Response Parameters:	
EmberStatus status	EmberStatus value that indicates the success or failure of the command.

Name: becomeTrustCenter	**ID:** 0x77
Description: This function causes a coordinator to become the Trust Center when it is operating in a network that is not using one. It will send out an updated Network Key to all devices that will indicate a transition of the network to now use a Trust Center. The Trust Center should also switch all devices to using this new network key with the appropriate API.	
Command Parameters:	
EmberKeyData newNetworkKey	The key data for the Updated Network Key.
Response Parameters:	
EmberStatus status	

Name: aesMmoHash	**ID:** 0x6F
Description: This routine processes the passed chunk of data and updates the hash context based on it. If the 'finalize' parameter is not set, then the length of the data passed in must be a multiple of 16. If the 'finalize' parameter is set then the length can be any value up 1-16, and the final hash value will be calculated.	
Command Parameters:	
EmberAesMmoHashContext context	The hash context to update.
boolean finalize	This indicates whether the final hash value should be calculated
int8u length	The length of the data to hash.
int8u[] data	The data to hash.
Response Parameters:	
EmberStatus status	The result of the operation
EmberAesMmoHashContext returnContext	The updated hash context.

Name: removeDevice	**ID:** 0xA8
Description: This command sends an APS remove device using APS encryption to the destination indicating either to remove itself from the network, or one of its children.	
Command Parameters:	
EmberNodeId destShort	The node ID of the device that will receive the message
EmberEUI64 destLong	The long address (EUI64) of the device that will receive the message.
int8u[16] targetLong	The long address (EUI64) of the device to be removed.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success, or the reason for failure

3.11 Certificate Based Key Exchange (CBKE)

Name: generateCbkeKeys	**ID:** 0xA4
Description: This call starts the generation of the ECC Ephemeral Public/Private key pair. When complete it stores the private key. The results are returned via ezspGenerateCbkeKeysHandler().	
Command Parameters: None	
Response Parameters:	
EmberStatus status	

Name: generateCbkeKeysHandler	**ID:** 0x9E
Description: A callback by the Crypto Engine indicating that a new ephemeral public/private key pair has been generated. The public/private key pair is stored on the NCP, but only the associated public key is returned to the host. The node's associated certificate is also returned.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberStatus status	The result of the CBKE operation.
EmberPublicKeyData ephemeralPublicKey	The generated ephemeral public key.

Name: calculateSmacs	**ID:** 0x9F
Description: Calculates the SMAC verification keys for both the initiator and responder roles of CBKE using the passed parameters and the stored public/private key pair previously generated with ezspGenerateKeysRetrieveCert(). It also stores the unverified link key data in temporary storage on the NCP until the key establishment is complete.	
Command Parameters:	
boolean amInitiator	The role of this device in the Key Establishment protocol.
EmberCertificateData partnerCertificate	The key establishment partner's implicit certificate.
EmberPublicKeyData partnerEphemeralPublicKey	The key establishment partner's ephemeral public key
Response Parameters:	
EmberStatus status	

Name: calculateSmacsHandler	**ID:** 0xA0
Description: A callback to indicate that the NCP has finished calculating the Secure Message Authentication Codes (SMAC) for both the initiator and responder. The associated link key is kept in temporary storage until the host tells the NCP to store or discard the key via emberClearTemporaryDataMaybeStoreLinkKey().	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberStatus status	The Result of the CBKE operation.
EmberSmacData initiatorSmac	The calculated value of the initiator's SMAC
EmberSmacData responderSmac	The calculated value of the responder's SMAC

Name: clearTemporaryDataMaybeStoreLinkKey	**ID:** 0xA1
Description: Clears the temporary data associated with CBKE and the key establishment, most notably the ephemeral public/private key pair. If storeLinKey is TRUE it moves the unverfied link key stored in temporary storage into the link key table. Otherwise it discards the key.	
Command Parameters:	
boolean storeLinkKey	A boolean indicating whether to store (TRUE) or discard (FALSE) the unverified link key derived when ezspCalculateSmacs() was previously called.
Response Parameters:	
EmberStatus status	

Name: getCertificate	**ID:** 0xA5
Description: Retrieves the certificate installed on the NCP.	
Command Parameters: None	
Response Parameters:	
EmberStatus status	
EmberCertificateData localCert	The locally installed certificate.

Name: dsaSign	**ID:** 0xA6
Description: LEGACY FUNCTION: This functionality has been replaced by a single bit in the EmberApsFrame, EMBER_APS_OPTION_DSA_SIGN. Devices wishing to send signed messages should use that as it requires fewer function calls and message buffering. The dsaSignHandler response is still called when EMBER_APS_OPTION_DSA_SIGN is used. However, this function is still supported. This function begins the process of signing the passed message contained within the messageContents array. If no other ECC operation is going on, it will immediately return with EMBER_OPERATION_IN_PROGRESS to indicate the start of ECC operation. It will delay a period of time to let APS retries take place, but then it will shutdown the radio and consume the CPU processing until the signing is complete. This may take up to 1 second. The signed message will be returned in the dsaSignHandler response. Note that the last byte of the messageContents passed to this function has special significance. As the typical use case for DSA signing is to sign the ZCL payload of a DRLC Report Event Status message in SE 1.0, there is often both a signed portion (ZCL payload) and an unsigned portion (ZCL header). The last byte in the content of messageToSign is therefore used as a special indicator to signify how many bytes of leading data in the array should be excluded from consideration during the signing process. If the signature needs to cover the entire array (all bytes except last one), the caller should ensure that the last byte of messageContents is 0x00. When the signature operation is complete, this final byte will be replaced by the signature type indicator (0x01 for ECDSA signatures), and the actual signature will be appended to the original contents after this byte.	
Command Parameters:	
int8u messageLength	The length of the *messageContents* parameter in bytes.
int8u[] messageContents	The message contents for which to create a signature. Per above notes, this may include a leading portion of data not included in the signature, in which case the last byte of this array should be set to the index of the first byte to be considered for signing. Otherwise, the last byte of messageContents should be 0x00 to indicate that a signature should occur across the entire contents.
Response Parameters:	
EmberStatus status	EMBER_OPERATION_IN_PROGRESS if the stack has queued up the operation for execution. EMBER_INVALID_CALL if the operation can't be performed in this context, possibly because another ECC operation is pending.

Name: dsaSignHandler	**ID:** 0xA7
Description: The handler that returns the results of the signing operation. On success, the signature will be appended to the original message including the signature type indicator that replaced the startIndex field for the signing) and both are returned via this callback.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberStatus status	The result of the DSA signing operation.
int8u messageLength	The length of the *messageContents* parameter in bytes.
int8u[] messageContents	The message and attached which includes the original message and the appended signature.

Name: dsaVerify	**ID:** 0xA3
Description: Verify that signature of the associated message digest was signed by the private key of the associated certificate.	
Command Parameters:	
EmberMessageDigest digest	The AES-MMO message digest of the signed data. If dsaSign command was used to generate the signature for this data, the final byte (replaced by signature type of 0x01) in the messageContents array passed to dsaSign is included in the has context used for the digest calculation.
EmberCertificateData signerCertificate	The certificate of the signer. Note that the signer's certificate and the verifier's certificate must both be issued by the same Certificate Authority, so they should share the same CA Public Key.
EmberSignatureData receivedSig	The signature of the signed data.
Response Parameters:	
EmberStatus status	

Name: dsaVerifyHandler	**ID:** 0x78
Description: This callback is executed by the stack when the DSA verification has completed and has a result. If the result is EMBER_SUCCESS, the signature is valid. If the result is EMBER_SIGNATURE_VERIFY_FAILURE then the signature is invalid. If the result is anything else then the signature verify operation failed and the validity is unknown.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberStatus status	The result of the DSA verification operation.

Name: setPreinstalledCbkeData	**ID:** 0xA2
Description: Sets the device's CA public key, local certificate, and static private key on the NCP associated with this node.	
Command Parameters:	
EmberPublicKeyData caPublic	The Certificate Authority's public key.
EmberCertificateData myCert	The node's new certificate signed by the CA.
EmberPrivateKeyData myKey	The node's new static private key.
Response Parameters:	
EmberStatus status	

3.12 Mfglib frames

Name: mfglibStart	**ID:** 0x83
Description: Activate use of mfglib test routines and enables the radio receiver to report packets it receives to the mfgLibRxHandler() callback. These packets will not be passed up with a CRC failure. All other mfglib functions will return an error until the mfglibStart() has been called	
Command Parameters:	
boolean rxCallback	TRUE to generate a mfglibRxHandler callback when a packet is received.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: mfglibEnd	**ID:** 0x84
Description: Deactivate use of mfglib test routines; restores the hardware to the state it was in prior to mfglibStart() and stops receiving packets started by mfglibStart() at the same time.	
Command Parameters: None	
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: mfglibStartTone	**ID:** 0x85
Description: Starts transmitting an unmodulated tone on the currently set channel and power level. Upon successful return, the tone will be transmitting. To stop transmitting tone, application must call mfglibStopTone(), allowing it the flexibility to determine its own criteria for tone duration (time, event, etc.)	
Command Parameters: None	
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: mfglibStopTone	**ID:** 0x86
Description: Stops transmitting tone started by mfglibStartTone().	
Command Parameters: None	
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: mfglibStartStream	**ID:** 0x87
Description: Starts transmitting a random stream of characters. This is so that the radio modulation can be measured.	
Command Parameters: None	
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: mfglibStopStream	**ID:** 0x88
Description: Stops transmitting a random stream of characters started by mfglibStartStream().	
Command Parameters: None	
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: mfglibSendPacket	**ID:** 0x89
Description: Sends a single packet consisting of the following bytes: packetLength, packetContents[0], ... , packetContents[packetLength - 3], CRC[0], CRC[1]. The total number of bytes sent is packetLength + 1. The radio replaces the last two bytes of packetContents[] with the 16-bit CRC for the packet.	
Command Parameters:	
int8u packetLength	The length of the packetContents parameter in bytes. Must be greater than 3 and less than 123.
int8u[] packetContents	The packet to send. The last two bytes will be replaced with the 16-bit CRC.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: mfglibSetChannel	**ID:** 0x8a
Description: Sets the radio channel. Calibration occurs if this is the first time the channel has been used.	
Command Parameters:	
int8u channel	The channel to switch to. Valid values are 11 to 26.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: mfglibGetChannel	**ID:** 0x8b
Description: Returns the current radio channel, as previously set via mfglibSetChannel().	
Command Parameters: None	
Response Parameters:	
int8u channel	The current channel.

Name: mfglibSetPower	**ID:** 0x8c
Description: First select the transmit power mode, and then include a method for selecting the radio transmit power. The valid power settings depend upon the specific radio in use. Ember radios have discrete power settings, and then requested power is rounded to a valid power setting; the actual power output is available to the caller via mfglibGetPower().	
Command Parameters:	
int16u txPowerMode	Power mode. Refer to txPowerModes in stack/include/ember-types.h for possible values.
int8s power	Power in units of dBm. Refer to radio datasheet for valid range.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: mfglibGetPower	**ID:** 0x8d
Description: Returns the current radio power setting, as previously set via mfglibSetPower().	
Command Parameters: None	
Response Parameters:	
int8s power	Power in units of dBm. Refer to radio datasheet for valid range.

Name: mfglibRxHandler	**ID:** 0x8e
Description: A callback indicating a packet with a valid CRC has been received.	
This frame is a response to the *callback* command.	
Response Parameters:	
int8u linkQuality	The link quality observed during the reception
int8s rssi	The energy level (in units of dBm) observed during the reception.
int8u packetLength	The length of the packetContents parameter in bytes. Will be greater than 3 and less than 123.
int8u[] packetContents	The received packet. The last two bytes are the 16-bit CRC.

3.13 Bootloader Frames

Name: launchStandaloneBootloader	**ID:** 0x8f
Description: Quits the current application and launches the standalone bootloader (if installed) The function returns an error if the standalone bootloader is not present	
Command Parameters:	
int8u mode	Controls the mode in which the standalone bootloader will run. See the app. note for full details. Options are: STANDALONE_BOOTLOADER_NORMAL_MODE: Will listen for an over-the-air image transfer on the current channel with current power settings. STANDALONE_BOOTLOADER_RECOVERY_MODE: Will listen for an over-the-air image transfer on the default channel with default power settings. Both modes also allow an image transfer to begin with XMODEM over the serial protocol's Bootloader Frame.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: sendBootloadMessage	**ID:** 0x90
Description: Transmits the given bootload message to a neighboring node using a specific 802.15.4 header that allows the EmberZNet stack as well as the bootloader to recognize the message, but will not interfere with other ZigBee stacks.	
Command Parameters:	
boolean broadcast	If TRUE, the destination address and pan id are both set to the broadcast address.
EmberEUI64 destEui64	The EUI64 of the target node. Ignored if the broadcast field is set to TRUE.
int8u messageLength	The length of the *messageContents* parameter in bytes.
int8u[] messageContents	The multicast message.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

Name: getStandaloneBootloaderVersionPlatMicroPhy	**ID:** 0x91
Description: Detects if the standalone bootloader is installed, and if so returns the installed version. If not return 0xffff. A returned version of 0x1234 would indicate version 1.2 build 34. Also return the node's version of PLAT, MICRO and PHY.	
Command Parameters: None	
Response Parameters:	
int16u bootloader_version	BOOTLOADER_INVALID_VERSION if the standalone bootloader is not present, or the version of the installed standalone bootloader.
int8u nodePlat	The value of PLAT on the node
int8u nodeMicro	The value of MICRO on the node
int8u nodePhy	The value of PHY on the node

Name: incomingBootloadMessageHandler	**ID:** 0x92
Description: A callback invoked by the EmberZNet stack when a bootload message is received.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberEUI64 longId	The EUI64 of the sending node.
int8u lastHopLqi	The link quality from the node that last relayed the message.
int8s lastHopRssi	The energy level (in units of dBm) observed during the reception.
int8u messageLength	The length of the *messageContents* parameter in bytes.
int8u[] messageContents	The bootload message that was sent.

Name: bootloadTransmitCompleteHandler	**ID:** 0x93
Description: A callback invoked by the EmberZNet stack when the MAC has finished transmitting a bootload message.	
This frame is a response to the *callback* command.	
Response Parameters:	
EmberStatus status	An EmberStatus value of EMBER_SUCCESS if an ACK was received from the destination or EMBER_DELIVERY_FAILED if no ACK was received.
int8u messageLength	The length of the *messageContents* parameter in bytes.
int8u[] messageContents	The message that was sent.

Name: aesEncrypt	**ID:** 0x94
Description: Perform AES encryption on plaintext using key.	
Command Parameters:	
int8u[16] plaintext	16 bytes of plaintext.
int8u[16] key	The 16 byte encryption key to use.
Response Parameters:	
int8u[16] ciphertext	16 bytes of ciphertext.

Name: overrideCurrentChannel	**ID:** 0x95
Description: A bootloader method for selecting the radio channel. This routine only works for sending and receiving bootload packets. Does not correctly do ZigBee stack changes.	
Command Parameters:	
int8u channel	The channel to switch to. Valid values are 11 to 26.
Response Parameters:	
EmberStatus status	An EmberStatus value indicating success or the reason for failure.

3.14 Alphabetical List of Frames

Name	ID
addEndpoint	0x02
addOrUpdateKeyTableEntry	0x66
addressTableEntryIsActive	0x5B
aesEncrypt	0x94
aesMmoHash	0x6F
becomeTrustCenter	0x77
bindingIsActive	0x2E
bootloadTransmitCompleteHandler	0x93
broadcastNetworkKeySwitch	0x74
broadcastNextNetworkKey	0x73
calculateSmacs	0x9F
calculateSmacsHandler	0xA0
callback	0x06
childJoinHandler	0x23
clearBindingTable	0x2A
clearTemporaryDataMaybeStoreLinkKey	0xA1
debugWrite	0x12
delayTest	0x9D
deleteBinding	0x2D
dsaSign	0xA6
dsaSignHandler	0xA7
dsaVerify	0xA3

Name	ID
dsaVerifyHandler	0x78
echo	0x81
energyScanRequest	0x9C
energyScanResultHandler	0x48
eraseKeyTableEntry	0x76
findAndRejoinNetwork	0x21
findKeyTableEntry	0x75
formNetwork	0x1E
generateCbkeKeys	0xA4
generateCbkeKeysHandler	0x9E
getAddressTableRemoteEui64	0x5E
getAddressTableRemoteNodeId	0x5F
getBinding	0x2C
getBindingRemoteNodeId	0x2F
getCertificate	0xA5
getChildData	0x4A
getConfigurationValue	0x52
getCurrentSecurityState	0x69
getEui64	0x26
getExtendedTimeout	0x7F
getKey	0x6a
getKeyTableEntry	0x71
getLibraryStatus	0x01
getMfgToken	0x0B

Name	ID
getMulticastTableEntry	0x63
getNeighbor	0x79
getNetworkParameters	0x28
getNodeId	0x27
getParentChildParameters	0x29
getPolicy	0x56
getRandomNumber	0x49
getRouteTableEntry	0x7B
getStandaloneBootloaderVersionPlatMicroPhy	0x91
getTimer	0x4E
getToken	0x0A
getValue	0xAA
idConflictHandler	0x7C
incomingBootloadMessageHandler	0x92
incomingManyToOneRouteRequestHandler	0x7D
incomingMessageHandler	0x45
incomingRouteErrorHandler	0x80
incomingRouteRecordHandler	0x59
incomingSenderEui64Handler	0x62
invalidCommand	0x58
joinNetwork	0x1F
launchStandaloneBootloader	0x8f
leaveNetwork	0x20
lookupEui64ByNodeId	0x61

Name	ID
lookupNodeIdByEui64	0x60
macFilterMatchMessageHandler	0x46
macPassthroughMessageHandler	0x97
maximumPayloadLength	0x33
messageSentHandler	0x3F
mfglibEnd	0x84
mfglibGetChannel	0x8b
mfglibGetPower	0x8d
mfglibRxHandler	0x8e
mfglibSendPacket	0x89
mfglibSetChannel	0x8a
mfglibSetPower	0x8c
mfglibStart	0x83
mfglibStartStream	0x87
mfglibStartTone	0x85
mfglibStopStream	0x88
mfglibStopTone	0x86
neighborCount	0x7A
networkFoundHandler	0x1B
networkInit	0x17
networkState	0x18
noCallbacks	0x07
nop	0x05
overrideCurrentChannel	0x95

Name	ID
permitJoining	0x22
pollCompleteHandler	0x43
pollForData	0x42
pollHandler	0x44
rawTransmitCompleteHandler	0x98
readAndClearCounters	0x65
remoteDeleteBindingHandler	0x32
remoteSetBindingHandler	0x31
removeDevice	0xA8
replaceAddressTableEntry	0x82
requestLinkKey	0x14
scanCompleteHandler	0x1C
sendBootloadMessage	0x90
sendBroadcast	0x36
sendManyToOneRouteRequest	0x41
sendMulticast	0x38
sendRawMessage	0x96
sendReply	0x39
sendUnicast	0x34
setAddressTableRemoteEui64	0x5C
setAddressTableRemoteNodeId	0x5D
setBinding	0x2B
setBindingRemoteNodeId	0x30
setConfigurationValue	0x53

Name	ID
setExtendedTimeout	0x7E
setInitialSecurityState	0x68
setKeyTableEntry	0x72
setManufacturerCode	0x15
setMulticastTableEntry	0x64
setPolicy	0x55
setPowerDescriptor	0x16
setPreinstalledCbkeData	0xA2
setRadioChannel	0x9A
setRadioPower	0x99
setSourceRoute	0x5A
setTimer	0x0E
setToken	0x09
setValue	0xAB
stackStatusHandler	0x19
startScan	0x1A
stopScan	0x1D
switchNetworkKeyHandler	0x6e
timerHandler	0x0F
trustCenterJoinHandler	0x24
unicastNwkKeyUpdate	0xA9
version	0x00
zigbeeKeyEstablishmentHandler	0x9B

4. Sample Transactions

The following sections illustrate the following sample transactions:

- Join
- Set Address Table
- Send
- Receive

4.1 Join

```
1)  sequence            = 0x00
    frame control       = 0x00 (command frame, don't sleep)
    joinNetwork command = 0x1F
    nodeType            = 0x02 (EMBER ROUTER)
    extendedPanId       = 0x1122334455667788
    panId               = 0x1234
    radioTxPower        = 0xFF (-1)
    radioChannel        = 0x0B (11)
    joinMethod          = 0x00 (EMBER USE MAC ASSOCIATION)
    nwkManagerId        = 0x0000     (unused)
    nwkUpdateId         = 0x00       (unused)
    channels            = 0x00000000 (unused)

    HOST -> NCP: | 00 | 00 | 1F | 02 | 88 | 77 | 66 | 55 | 44 | 33 | 22
                 | 11 | 34 | 12 | FF | 0B | 00 | 00 | 00 | 00 | 00 | 00
                 | 00 | 00 |

    sequence             = 0x00
    frame control        = 0x80 (response frame, not a callback,
                                 no callbacks pending, no overflow, not truncated)
    joinNetwork response = 0x1F
    status               = 0x00 (EMBER SUCCESS)

    NCP -> HOST: | 00 | 80 | 1F | 00 |


2)  Host waits for callback signal while NCP tries to join the network.

3)  sequence         = 0x01
    frame control    = 0x00 (command frame, don't sleep)
    callback command = 0x06

    HOST -> NCP: | 01 | 00 | 06 |

    sequence                    = 0x00
    frame control               = 0x88 (response frame, synchronous callback,
                                        no callbacks pending, no overflow,
                                        not truncated)
    stackStatusHandler response = 0x19
    status                      = 0x90 (EMBER NETWORK UP)

    NCP -> HOST: | 00 | 88 | 19 | 90 |
```

4.2 Set Address Table

```
1)  sequence                           = 0x02
    frame control                      = 0x00 (command frame, don't sleep)
    setAddressTableRemoteEui64 command = 0x5C
    index                              = 0x00
    eui64                              = 0x1122334455667788

    HOST -> NCP: | 02 | 00 | 5C | 00 | 88 | 77 | 66 | 55 | 44 | 33 | 22
                 | 11 |

    sequence                = 0x02
    frame control           = 0x80 (response frame, not a callback,
                                    no callbacks pending, no overflow, not truncated)
    setRemoteEui64 response = 0x5C
    status                  = 0x00 (EMBER SUCCESS)

    NCP -> HOST: | 02 | 80 | 5C | 00 |
```

4.3 Send

```
1)   sequence            = 0x03
     frame control       = 0x00 (command frame, don't sleep)
     sendUnicast command = 0x34
     type                = 0x01 (EMBER_OUTGOING_VIA_ADDRESS_TABLE)
     indexOrDestination  = 0x0000
     profileId           = 0xABCD
     clusterId           = 0x0055
     sourceEndpoint      = 0x11
     destinationEndpoint = 0x12
     options             = 0x1140 (address discovery, route discovery, retries)
     groupId             = 0x0000
     sequence            = 0x00
     messageTag          = 0x01
     messageLength       = 0x03
     messageContents     = 0xE1, 0xE2, 0xE3

     HOST -> NCP: | 03 | 00 | 34 | 01 | 00 | 00 | CD | AB | 55 | 00 | 11
                  | 12 | 40 | 11 | 00 | 00 | 00 | 01 | 03 | E1 | E2 | E3 |

     sequence             = 0x03
     frame control        = 0x80 (response frame, not a callback,
                                  no callbacks pending, no overflow, not truncated)
     sendUnicast response = 0x34
     status               = 0x00 (EMBER SUCCESS)

     NCP -> HOST: | 03 | 80 | 34 | 00 |

2)   Host waits for callback signal while NCP tries to send the message.

3)   sequence         = 0x04
     frame control    = 0x00 (command frame, don't sleep)
     callback command = 0x06

     HOST -> NCP: | 04 | 00 | 06 |

     sequence                    = 0x03
     frame control               = 0x88 (response frame, synchronous callback,
                                         no callbacks pending, no overflow,
                                         not truncated)
     messageSentHandler response = 0x3F
     type                        = 0x01 (EMBER OUTGOING VIA ADDRESS TABLE)
     indexOrDestination          = 0x0000
     profileId                   = 0xABCD
     clusterId                   = 0x0055
     sourceEndpoint              = 0x11
     destinationEndpoint         = 0x12
     options                     = 0x1140 (address discovery, route discovery, retries)
     groupId                     = 0x0000
     sequence                    = 0x00
     messageTag                  = 0x01
     status                      = 0x00 (EMBER SUCCESS)
     messageLength               = 0x00

     NCP -> HOST: | 03 | 88 | 3F | 01 | 00 | 00 | CD | AB | 55 | 00 | 11
                  | 12 | 40 | 11 | 00 | 00 | 00 | 01 | 00 | 00 |
```

4.4 Receive

```
1)  Host waits for callback signal after a message is received by the NCP.

2)  sequence         = 0x05
    frame control    = 0x00 (command frame, don't sleep)
    callback command = 0x06

    HOST -> NCP: | 05 | 00 | 06 |

    sequence                        = 0x04
    frame control                   = 0x88 (response frame, synchronous callback,
                                            no callbacks pending, no overflow,
                                            not truncated)
    incomingMessageHandler response = 0x45
    type                            = 0x00 (EMBER INCOMING UNICAST)
    profileId                       = 0xABCD
    clusterId                       = 0x0055
    sourceEndpoint                  = 0x11
    destinationEndpoint             = 0x12
    options                         = 0x0000
    groupId                         = 0x0000
    sequence                        = 0x01
    lastHopLqi                      = 0xF0
    lastHopRssi                     = 0xC4 (-60)
    sender                          = 0x0001
    bindingIndex                    = 0xFF
    addressIndex                    = 0xFF
    messageLength                   = 0x03
    messageContents                 = 0xE1, 0xE2, 0xE3

    NCP -> HOST: | 04 | 88 | 45 | 00 | CD | AB | 55 | 00 | 11 | 12 | 00
                 | 00 | 00 | 00 | 01 | F0 | C4 | 01 | 00 | FF | FF | 03
                 | E1 | E2 | E3 |
```

After reading this document

If you have questions or require assistance with the procedures described in this document, contact Ember Customer Support. The Ember Customer Support portal provides a wide array of hardware and software documentation such as FAQ's, reference designs, user guides, application notes, and the latest software available to download. To obtain support on all Ember products and to gain access to the Ember Customer Support portal, visit http://www.ember.com/support_index.html.

ember

The information in this document is subject to change without notice. This document is believed to be accurate and reliable, but the statements contained herein are presented without express or implied warranty.

Ember Corporation
25 Thomson Place
Boston, MA 02210
+1 (617) 951-0200
www.ember.com

wireless semiconductor solutions