Thread

2.5.0.0

Generated by Doxygen 1.8.11

Contents

Chapter 1

Main Page

Introduction

Thread is a secure, wireless mesh networking protocol. The Thread stack is an open standard that is built upon a collection of existing Institute for Electrical and Electronics Engineers (IEEE) and Internet Engineering Task Force (IETF) standards, rather than a whole new standard.

For more information about Thread SDK, see UG103.11.

Product Overview

- Development hardware (see http://www.silabs.com/products/wireless/mesh-networking/thread/Pages/thread.aspx for more information):
- Mighty Gecko (EFR32MG) Mesh Networking Kit or
- EM35x Development Kit
- Required software components, as described below. A card included in your development hardware kit contains a link to a Getting Started page, which will direct you to links for the Silicon Labs software products.

Note: If you are installing an EM3x development kit, do not install Ember Desktop as documented in the Quick Start Guide. Instead, install Simplicity Studio as noted below.

Software Components

See the stack release notes for version restrictions and compatibility constraints for the stack and these components. To develop Silicon Labs Thread applications, you will need the following. Installation instructions are provided in the section Install Simplicity Studio and the Silicon Labs Thread Stack:

- IAR Embedded Workbench for ARM (IAR-EWARM) 7.80, used as a compiler in the Simplicity Studio development environment. Download the supported version from the Silicon Labs Support Portal, as described at the end of section Install Simplicity Studio and the Silicon Labs Thread Stack. Refer to the "QuickStart Installation Information" section of the IAR installer for additional information about the installation process and how to configure your license. Once IAR-EWARM is installed, the next time Simplicity Studio starts it will automatically detect and configure the IDE to use IAR-EWARM.

- The Simplicity Studio version 4 development environment, which incorporates AppBuilder. If you do not have version 4, download it here . AppBuilder is an interactive GUI tool that allows you to configure a body of Silicon Labs-supplied code to implement applications. Online help for AppBuilder and other Simplicity Studio modules is provided.
- The Silicon Labs Thread stack, an advanced implementation of a wireless protocol stack, installed through Simplicity Studio. The stack API is documented in online API reference as well as other documents available through Simplicity Studio. The stack is delivered as a collection of libraries that you can link to your applications. A description of each library is provided in the development environment. The release notes contain details on the folders installed along with their contents.
- Simplicity Commander, installed along with Simplicity Studio. A GUI with limited functionality can be accessed through Simplicity Studio's Tools menu. Most functions are accessible through a CLI invoked by opening a command prompt in the Simplicity Commander directory ().

For more information about Thread installation, see UG162.

API Sets

The Thread API Reference documentation for the EM35x and EFR32x includes the following API sets:

- Thread Stack API Reference
- Hardware Abstraction Layer (HAL) API Reference
- Application Framework API Reference

Chapter 2

Thread_API

Chapter 3

Module Index

3.1 API Reference

Here is a list of all modules:

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Module Documentation

6.1 Thread Stack API Reference

Modules

- Forming and Joining

  Forming and Joining Utilities.
- IPv6

  IPv6 Addressing Utilities.
- Commissioning

  Commissioning Utilities.
- Network Utilities

  Network Utilities.
- Device Types

  Device Types.
- Utilities

  General Utilities.
- Messaging

  Sending unicasts and multicasts.
- Command Interpreter

  The command interpreter.
- Debugging Utilities

  Debugging Utilities.
- MFGLIB

  Manufacturing Library.

6.1.1 Detailed Description

The Silicon Labs Thread stack API reference guide is Doxygen-generated documentation that is rooted at TH↩ READ HOME/documentation/Thread-Doxygen/index.html. This document contains a comprehensive list of APIs used to interface to the Thread network. These APIs concern network management, device and stack management (including management of stack tables, event scheduling, message buffers and security), messaging, fragmentation, serial communication, token access, peripheral access, bootload utilities, and so on.

6.2 Forming and Joining

Forming and Joining Utilities.

Macros

- #define EMBER_USE_DEFAULTS 0

  The following denotes which network parameters to use when forming or joining a network. Construct an uint16_t "options" flag for use in various network formation calls.
- #define EMBER_NETWORK_ID_OPTION BIT(0)
- #define EMBER_ULA_PREFIX_OPTION BIT(1)
- #define EMBER_EXTENDED_PAN_ID_OPTION BIT(2)
- #define EMBER_PAN_ID_OPTION BIT(3)
- #define EMBER_NODE_TYPE_OPTION BIT(4)
- #define EMBER_TX_POWER_OPTION BIT(5)
- #define EMBER_MASTER_KEY_OPTION BIT(6)
- #define EMBER_LEGACY_ULA_OPTION BIT(7)
- #define EMBER_JOIN_KEY_OPTION BIT(8)

Functions

- void emberConfigureNetwork (const EmberNetworkParameters *parameters, uint16_t options)

  This function configures network parameters.
- void emberFormNetwork (const EmberNetworkParameters *parameters, uint16_t options, uint32_t channelMask)

  This function forms a new network.
- void emberFormNetworkReturn (EmberStatus status)

  A callback that indicates whether a prior call to emberFormNetwork() successfully initiated the form process. The status argument is either EMBER_SUCCSS, or EMBER_INVALID_CALL if resume was called when the network status was not EMBER_NO_NETWORK, or a scan was underway.
- void emberJoinNetwork (const EmberNetworkParameters *parameters, uint16_t options, uint32_t channelMask)

  This function joins an existing network.
- void emberJoinCommissioned (int8_t radioTxPower, EmberNodeType nodeType, bool requireConnectivity)

  This function joins an already-commissioned network.
- void emberJoinNetworkReturn (EmberStatus status)

  A callback that indicates whether the join process was successfully initiated via a prior call to emberJoinNetwork() or emberJoinCommissioned(). The possible EmberStatus values are: EMBER_SUCCESS, EMBER_BAD_ARGUMENT, or EMBER_INVALID_CALL (if join was called when the network status was something other than EMBER_NO_NETWORK).
- void emberResumeNetwork (void)

  This function resumes network operation after a reboot of the Ember micro.
- void emberResumeNetworkReturn (EmberStatus status)

  A callback that indicates whether a prior call to emberResumeNetwork() successfully initiated the resume process. The status argument is either EMBER_SUCCESS, or EMBER_INVALID_CALL if resume was called when the network status was not EMBER_SAVED_NETWORK, or while a scan was underway.
- void emberAttachToNetwork (void)

  On an end device, this initiates an attach with any available router-eligible devices in the network. This call must only be made if the network materials have been pre-commissioned on this device, or if previously completed obtaining the commissioning materials from another device.
- void emberAttachToNetworkReturn (EmberStatus status)

A callback that indicates whether the attach process was successfully initiated via a prior call to emberAttachTo↩ Network(). The status argument is either EMBER_SUCCESS, or EMBER_INVALID_CALL if attach was called when the network status was not EMBER_JOINED_NETWORK_NO_PARENT, or while an attach was underway.

- void emberSetAddressHandler (const uint8_t *address)

  A callback that is generated when the host's address changes.
- void emberSetDriverAddressHandler (const uint8_t *address)

  A callback to the IP driver to tell it to change its address.
- void emberStartHostJoinClientHandler (const uint8_t *parentAddress)

  A callback to tell the host to start security commissioning.
- void emberSetNetworkKeysHandler (uint32_t sequence, const uint8_t *masterKey, uint32_t sequence2, const uint8_t *masterKey2)

  A callback to the IP driver to tell it the network keys.
- void emberSetCommProxyAppParametersHandler (const uint8_t *extendedPanId, const uint8_t *network↩ Id, const uint8_t *ulaPrefix, uint16_t panId, uint8_t channel, const EmberEui64 *eui64, const EmberEui64 *macExtendedId, EmberNetworkStatus networkStatus)

  A callback to provide the commission-proxy-app on the host with the requisite network parameters.
- void emberSetCommProxyAppSecurityHandler (const uint8_t *masterKey, uint32_t sequenceNumber)

  A callback to provide the commission-proxy-app on the host with the requisite security material.
- void emberSetCommProxyAppAddressHandler (const uint8_t *address)

  A callback to provide the commission-proxy-app on the host with our mesh local address.
- void emberSetCommProxyAppPskcHandler (const uint8_t *pskc)

  A callback to provide the commission-proxy-app on the host with the pskc.
- void emberChangeNodeType (EmberNodeType newType)

  This function changes the node type of a joined device.
- void emberChangeNodeTypeReturn (EmberStatus status)

  This function provides the result of a call to emberChangeNodeType(): either EMBER_SUCCESS, or EMBER_IN↩ VALID_CALL.

6.2.1 Detailed Description

See network-management.h for source code.

6.2.2 Macro Definition Documentation

6.2.2.1 #define EMBER_EXTENDED_PAN_ID_OPTION BIT(2)

6.2.2.2 #define EMBER_JOIN_KEY_OPTION BIT(8)

6.2.2.3 #define EMBER_LEGACY_ULA_OPTION BIT(7)

6.2.2.4 #define EMBER_MASTER_KEY_OPTION BIT(6)

6.2.2.5 #define EMBER_NETWORK_ID_OPTION BIT(0)

6.2.2.6 #define EMBER_NODE_TYPE_OPTION BIT(4)

6.2.2.7 #define EMBER_PAN_ID_OPTION BIT(3)

6.2.2.8 #define EMBER_TX_POWER_OPTION BIT(5)

6.2.2.9 #define EMBER_ULA_PREFIX_OPTION BIT(1)

6.2.2.10 #define EMBER_USE_DEFAULTS 0

6.2.3 Function Documentation

6.2.3.1 void emberAttachToNetwork (void)

The status of whether the attach process was initiated is reported to the application via emberAttachToNetwork↩ Return(). Changes to the network status resulting from the attach process are reported to the application via emberNetworkStatusHandler().

This function may only be called when the network status is EMBER_JOINED_NETWORK_NO_PARENT and an attach is not already underway.

6.2.3.2 void emberAttachToNetworkReturn (EmberStatus *status*)

6.2.3.3 void emberChangeNodeType (EmberNodeType *newType*)

The device must be joined to a network prior to making this call.

Parameters

newType	The node type to change to.

6.2.3.4 void emberChangeNodeTypeReturn (EmberStatus *status*)

This function provides the result of a call to emberChangeNodeType(): either EMBER_SUCCESS, or EMBER_I↩ NVALID_CALL.

6.2.3.5 void emberConfigureNetwork (const EmberNetworkParameters * *parameters*, uint16_t *options*)

This function assigns the network configuration values that will be used when the device forms or joins a network.

This function may only be called when the network status is EMBER_NO_NETWORK. If the node was previously part of a network, use emberResumeNetwork() to recover after a reboot. To forget the network and return to a status of EMBER_NO_NETWORK, please read cautions for emberResetNetworkState().

Parameters

parameters	Some parameters may be supplied by the caller.
options	A bitmask indicating which network parameters are being supplied by the caller. The following list enumerates the options that can be set.

- EMBER_NETWORK_ID_OPTION
- EMBER_EXTENDED_PAN_ID_OPTION
- EMBER_ULA_PREFIX_OPTION
- EMBER_MASTER_KEY_OPTION
- EMBER_PAN_ID_OPTION

6.2.3.6 void emberFormNetwork (const EmberNetworkParameters * *parameters,* uint16_t *options,* uint32_t *channelMask*)

The forming node chooses a random extended pan ID, network ULA prefix, and pan ID for the new network. It peforms an energy scan of the channels indicated by the channelMask argument and selects the one with lowest detected energy. It performs an active scan on that channel to ensure there is no pan ID conflict. emberFormNetworkReturn() indicates whether the form process was initiated. Changes to the network status resulting from the form process are reported to the application via emberNetworkStatusHandler().

This function may only be called when the network status is EMBER_NO_NETWORK, and when a scan is not underway. If the node was previously part of a network, use emberResumeNetwork() to recover after a reboot. To forget the network and return to a status of EMBER_NO_NETWORK, read cautions for emberResetNetworkState().

Parameters

parameters	Some parameters may be supplied by the caller.
options	A bitmask indicating which network parameters are being supplied by the caller. The following list enumerates the allowed options and the default value used if the option is not specified: • EMBER_NETWORK_ID_OPTION default: ember • EMBER_ULA_PREFIX_OPTION default: random • EMBER_NODE_TYPE_OPTION default: EMBER_ROUTER • EMBER_TX_POWER_OPTION default: 3
channelMask	A mask indicating the channels to be scanned. See emberStartScan() for format details.

6.2.3.7 void emberFormNetworkReturn (EmberStatus *status*)

6.2.3.8 void emberJoinCommissioned (int8_t *radioTxPower,* EmberNodeType *nodeType,* bool *requireConnectivity*)

This function assumes that commissioning data has already been cached via a call to emberConfigureNetwork().

Parameters

radioTxPower	Desired radio output power, in dBm.
nodeType	Type of device.
requireConnectivity	If commissioned join fails, specify whether this node should start a new fragment. Note: The short PAN ID MUST be commissioned if this is true.

6.2.3.9 void emberJoinNetwork (const EmberNetworkParameters * *parameters*, uint16_t *options*, uint32_t *channelMask*)

The joining node performs an active scan of the channels indicated by the channelMask argument. It looks for networks matching the criteria specified via the supplied parameters, and which currently allow joining. The status of whether the join process was initiated is reported to the application via emberResumeNetworkReturn(). Changes to the network status resulting from the join process are reported to the application via emberNetworkStatus↩ Handler().

This function may only be called when the network status is EMBER_NO_NETWORK, and when a scan is not underway. If the node was previously part of a network, use emberResumeNetwork() to recover after a reboot. To forget the network and return to a status of EMBER_NO_NETWORK, please read cautions for emberReset↩ NetworkState().

Parameters

parameters	Some parameters may be supplied by the caller.
options	A bitmask indicating which network parameters are being supplied by the caller. The following list enumerates the allowed options and the default value used if the option is not specified: • EMBER_NETWORK_ID_OPTION default: looks for any network id • EMBER_EXTENDED_PAN_ID_OPTION default: looks for any extended pan id • EMBER_PAN_ID_OPTION default: looks for any pan id • EMBER_NODE_TYPE_OPTION default: EMBER_ROUTER • EMBER_TX_POWER_OPTION default: 3 • EMBER_JOIN_KEY_OPTION default: empty
channelMask	A mask indicating the channels to be scanned. See emberStartScan() for format details.

6.2.3.10 void emberJoinNetworkReturn (EmberStatus *status*)

6.2.3.11 void emberResumeNetwork (void)

If the device was previously part of a network, it will recover its former network parameters including pan id, extended pan id, node type, etc. and resume participation in the network. The status of whether the resume process was initiated is reported to the application via emberResumeNetworkReturn(). Changes to the network status resulting from the resume process are reported to the application via emberNetworkStatusHandler().

This function may only be called when the network status is EMBER_SAVED_NETWORK and when a scan is not underway.

6.2.3.12 void emberResumeNetworkReturn (EmberStatus *status*)

6.2.3.13 void emberSetAddressHandler (const uint8_t * *address*)

Parameters

address	IP address, 16 bytes

6.2.3.14 void emberSetCommProxyAppAddressHandler (const uint8_t * *address*)

6.2.3.15 void emberSetCommProxyAppParametersHandler (const uint8_t * *extendedPanId,* const uint8_t * *networkId,* const uint8_t * *ulaPrefix,* uint16_t *panId,* uint8_t *channel,* const **EmberEui64** * *eui64,* const **EmberEui64** * *macExtendedId,* **EmberNetworkStatus** *networkStatus*)

6.2.3.16 void emberSetCommProxyAppPskcHandler (const uint8_t * *pskc*)

6.2.3.17 void emberSetCommProxyAppSecurityHandler (const uint8_t * *masterKey,* uint32_t *sequenceNumber*)

6.2.3.18 void emberSetDriverAddressHandler (const uint8_t * *address*)

Parameters

address	IP address, 16 bytes

6.2.3.19 void emberSetNetworkKeysHandler (uint32_t *sequence,* const uint8_t * *masterKey,* uint32_t *sequence2,* const uint8_t * *masterKey2*)

Parameters

sequence	sequence number
masterKey	master key, 16 bytes
sequence2	second sequence number
masterKey2NotUsed	second key, 16 bytes

6.2.3.20 void emberStartHostJoinClientHandler (const uint8_t * *parentAddress*)

Parameters

address	parent IP address, 16 bytes

A callback to tell the host to start security commissioning.

Parameters

address	parent IP address, 16 bytes

6.3 IPv6

IPv6 Addressing Utilities.

Data Structures

- struct EmberDnsResponse

  Structure for returning information from a DNS lookup. A structure is used to make it easier to add additional values.

Macros

- #define EMBER_MAX_IPV6_ADDRESS_COUNT 10

  The maximum number of IPv6 addresses configured for the device. See emberGetLocalIpAddress.
- #define EMBER_MAX_IPV6_GLOBAL_ADDRESS_COUNT (EMBER_MAX_IPV6_ADDRESS_COUNT - 2)
- #define EMBER_MAX_IPV6_EXTERNAL_ROUTE_COUNT (EMBER_MAX_IPV6_ADDRESS_COUNT - 2)
- #define EMBER_MAX_LIFETIME_DELAY_SEC ((HALF_MAX_INT32U_VALUE - 1) / 1000)

  We enforce this limit to avoid overflow when converting lifetimes from seconds to milliseconds.
- #define EMBER_MIN_PREFERRED_LIFETIME_SEC 1800

  There should be at least half an hour of preferred lifetime remaining to advertise a DHCP server.
- #define EMBER_MIN_VALID_LIFETIME_SEC 60

  Renew when we are down to one minute of valid lifetime.
- #define EMBER_MAX_DNS_NAME_LENGTH 128

  The maximum length of a domain name that may be passed to emberDnsLookup().
- #define EMBER_MAX_DNS_QUERY_APP_DATA_LENGTH 64

  The maximum number of bytes of application data that may be passed to emberDnsLookup().

Typedefs

- typedef uint16_t EmberBorderRouterTlvFlag
- typedef uint8_t EmberDefaultRouteTlvFlag
- typedef uint8_t LocalServerFlag
- typedef void(∗ EmberDnsResponseHandler) (EmberDnsLookupStatus status, const uint8_t ∗domain↩ Name, uint8_t domainNameLength, const EmberDnsResponse ∗response, void ∗applicationData, uint16_t applicationDataLength)

  Type definition for callback handlers for DNS responses.

Enumerations

- enum EmberLocalAddressScope {
  REALM_SCOPE = 0,
  LINK_SCOPE = 1,
  GLOBAL_SCOPE = 2 }

- enum EmberBorderRouterTlvFlag_e {
  EMBER_BORDER_ROUTER_ND_DNS_FLAG = 0x0080,
  EMBER_BORDER_ROUTER_ON_MESH_FLAG = 0x0100,
  EMBER_BORDER_ROUTER_DEFAULT_ROUTE_FLAG = 0x0200,
  EMBER_BORDER_ROUTER_CONFIGURE_FLAG = 0x0400,
  EMBER_BORDER_ROUTER_DHCP_FLAG = 0x0800,
  EMBER_BORDER_ROUTER_SLAAC_FLAG = 0x1000,
  EMBER_BORDER_ROUTER_PREFERRED_FLAG = 0x2000,
  EMBER_BORDER_ROUTER_PREFERENCE_MASK = 0xC000,
  EMBER_BORDER_ROUTER_HIGH_PREFERENCE = 0x4000,
  EMBER_BORDER_ROUTER_MEDIUM_PREFERENCE = 0x0000,
  EMBER_BORDER_ROUTER_LOW_PREFERENCE = 0xC000 }

  Border router flags (see Thread spec chapter 5 for more information)
- enum EmberExternalRouteTlvFlag_e {
  EMBER_EXTERNAL_ROUTE_PREFERENCE_MASK = 0xC0,
  EMBER_EXTERNAL_ROUTE_HIGH_PREFERENCE = 0x40,
  EMBER_EXTERNAL_ROUTE_MEDIUM_PREFERENCE = 0x00,
  EMBER_EXTERNAL_ROUTE_LOW_PREFERENCE = 0xC0 }

  External route router flags (see Thread spec chapter 5 for more information)
- enum LocalServerFlag_e {
  EMBER_GLOBAL_ADDRESS_AM_GATEWAY = 0x01,
  EMBER_GLOBAL_ADDRESS_AM_DHCP_SERVER = 0x02,
  EMBER_GLOBAL_ADDRESS_AM_SLAAC_SERVER = 0x04,
  EMBER_GLOBAL_ADDRESS_DHCP = 0x08,
  EMBER_GLOBAL_ADDRESS_SLAAC = 0x10,
  EMBER_GLOBAL_ADDRESS_CONFIGURED = 0x20,
  EMBER_GLOBAL_ADDRESS_REQUEST_SENT = 0x40,
  EMBER_GLOBAL_ADDRESS_REQUEST_FAILED = 0x80,
  EMBER_LOCAL_ADDRESS = 0xFF }

  Address configuration flags. These flags denote the properties of a Thread IPv6 address.
- enum EmberDnsLookupStatus {
  EMBER_DNS_LOOKUP_SUCCESS,
  EMBER_DNS_LOOKUP_NO_BORDER_ROUTER,
  EMBER_DNS_LOOKUP_NO_BORDER_ROUTER_RESPONSE,
  EMBER_DNS_LOOKUP_BORDER_ROUTER_RESPONSE_ERROR,
  EMBER_DNS_LOOKUP_NO_DNS_SERVER,
  EMBER_DNS_LOOKUP_NO_DNS_RESPONSE,
  EMBER_DNS_LOOKUP_NO_DNS_RESPONSE_ERROR,
  EMBER_DNS_LOOKUP_NO_ENTRY_FOR_NAME,
  EMBER_DNS_LOOKUP_NO_BUFFERS }

  Status values passed to DNS response handlers.

Functions

- bool emberGetLocalIpAddress (uint8_t index, EmberIpv6Address *address)

  This function fetches one of the device IPv6 addresses into the supplied pointer. Since there may be multiple addresses, an index argument between 0 and EMBER_MAX_IPV6_ADDRESS_COUNT must be supplied.
- void emberGetRoutingLocator (void)

  This function fetches the Thread Routing Locator (RLOC).
- void emberGetRoutingLocatorReturn (const EmberIpv6Address *rloc)

  This function provides the result of a call to emberGetRoutingLocator.
- void emberSetLocalNetworkData (const uint8_t *networkData, uint16_t length)

  Sets the Network Data that describes the local node's Border Router and server capabilities. This is passed a set of Network Data TLVs that may include Prefix, Has Route, Border Router, Service and Server TLVS.
- void emberSetLocalNetworkDataReturn (EmberStatus status, uint16_t length)

Provides the result of a call to ::emberSetServerNetworkData.

- void emberConfigureGateway (EmberBorderRouterTlvFlag borderRouterFlags, bool isStable, const uint8_t *prefix, const uint8_t prefixLengthInBits, uint8_t domainId, uint32_t preferredLifetime, uint32_t validLifetime)

  This function configures the border router behavior, such as whether this device has a default route to the Internet, and whether it have a prefix that can be used by network devices to configure routable addresses.
- void emberConfigureGatewayReturn (EmberStatus status)

  This function provides the result of a call to emberConfigureGateway.
- void emberSetNdData (const uint8_t *data, uint16_t length)
- void emberSetNdDataReturn (EmberStatus status, uint16_t length)

  This function provides the result of a call to emberSetNdData.
- void emberConfigureExternalRoute (EmberDefaultRouteTlvFlag extRouteFlags, bool isStable, const uint8_t *extRoutePrefix, uint8_t extRoutePrefixLengthInBits, uint8_t extRouteDomainId)

  This function defines an external route set, a route for a Thread network IPv6 packet that must traverse a border router and be forwarded to an exterior network.
- void emberConfigureExternalRouteReturn (EmberStatus status)

  This function provides the result of a call to emberConfigureExternalRoute.
- void emberAddressConfigurationChangeHandler (const EmberIpv6Address *address, uint32_t preferred↩ Lifetime, uint32_t validLifetime, uint8_t addressFlags)

  This function is called when a new address is configured on the application.
- void emberGetGlobalPrefixes (void)

  This function returns the list of global prefixes that we know about.
- void emberGetGlobalPrefixReturn (uint8_t flags, bool isStable, const uint8_t *prefix, uint8_t prefixLengthIn↩ Bits, uint8_t domainId, uint32_t preferredLifetime, uint32_t validLifetime)

  This function provides the result of a call to ::emberGetGlobalPrefix.
- void emberDhcpServerChangeHandler (const uint8_t *prefix, uint8_t prefixLengthInBits, bool available)

  This function is called when the stack knows about a new dhcp server or if a dhcp server has become unavailable.
- void emberRequestDhcpAddress (const uint8_t *prefix, uint8_t prefixLengthInBits)

  The application can choose to request a new DHCP address when it is informed via emberDhcpServerChange↩ Handler of an available DHCP server.
- void emberRequestDhcpAddressReturn (EmberStatus status, const uint8_t *prefix, uint8_t prefixLengthIn↩ Bits)

  This function provides the result of a call to emberRequestDhcpAddress.
- void emberSlaacServerChangeHandler (const uint8_t *prefix, uint8_t prefixLengthInBits, bool available)

  This function is called when the stack knows about a new SLAAC prefix or if a SLAAC server has become unavailable.
- void emberRequestSlaacAddress (const uint8_t *prefix, uint8_t prefixLengthInBits)

  The application can choose to request a new SLAAC address when it is informed via emberSlaacServerChange↩ Handler of an available SLAAC prefix.
- void emberRequestSlaacAddressReturn (EmberStatus status, const uint8_t *prefix, uint8_t prefixLengthIn↩ Bits)

  This function provides the result of a call to emberRequestSlaacAddress.
- void emberGetGlobalAddresses (const uint8_t *prefix, uint8_t prefixLengthInBits)

  This function returns the list of global addresses configured on this device.
- void emberGetGlobalAddressReturn (const EmberIpv6Address *address, uint32_t preferredLifetime, uint32_t validLifetime, uint8_t addressFlags)

  This function provides the result of a call to emberGetGlobalAddresses.
- void emberResignGlobalAddress (const EmberIpv6Address *address)

  This function resigns this IPv6 global address from this node. If this is a DHCP address, then the server is informed about it. If it is a SLAAC address, we remove it locally.
- void emberResignGlobalAddressReturn (EmberStatus status)

  This function provides the result of a call to emberResignGlobalAddress().
- void emberExternalRouteChangeHandler (const uint8_t *prefix, uint8_t prefixLengthInBits, bool available)

  This function is called when the stack knows about a border router that has an external route to a prefix.
- void emberNetworkDataChangeHandler (const uint8_t *networkData, uint16_t length)

This function is called when the stack receives new Thread Network Data. The networkData argument may be NULL, in which case emberGetNetworkData can be used to obtain the new Thread Network Data.

- void emberGetNetworkData (uint8_t *networkDataBuffer, uint16_t bufferLength)

  This function is called to obtain the current Thread Network Data.
- void emberGetNetworkDataReturn (EmberStatus status, uint8_t *networkData, uint16_t bufferLength)

  This function provides the result of a call to emberGetNetworkData.
- EmberStatus emberDnsLookup (const uint8_t *domainName, uint8_t domainNameLength, const uint8_t *prefix64, EmberDnsResponseHandler responseHandler, uint8_t *appData, uint16_t appDataLength)

  This function initiates a DNS name lookup.

6.3.1 Detailed Description

See network-management.h for source code.

6.3.2 Macro Definition Documentation

6.3.2.1 #define EMBER_MAX_DNS_NAME_LENGTH 128

6.3.2.2 #define EMBER_MAX_DNS_QUERY_APP_DATA_LENGTH 64

6.3.2.3 #define EMBER_MAX_IPV6_ADDRESS_COUNT 10

6.3.2.4 #define EMBER_MAX_IPV6_EXTERNAL_ROUTE_COUNT (EMBER_MAX_IPV6_ADDRESS_COUNT - 2)

6.3.2.5 #define EMBER_MAX_IPV6_GLOBAL_ADDRESS_COUNT (EMBER_MAX_IPV6_ADDRESS_COUNT - 2)

6.3.2.6 #define EMBER_MAX_LIFETIME_DELAY_SEC ((HALF_MAX_INT32U_VALUE - 1) / 1000)

6.3.2.7 #define EMBER_MIN_PREFERRED_LIFETIME_SEC 1800

6.3.2.8 #define EMBER_MIN_VALID_LIFETIME_SEC 60

6.3.3 Typedef Documentation

6.3.3.1 typedef uint16_t EmberBorderRouterTlvFlag

6.3.3.2 typedef uint8_t EmberDefaultRouteTlvFlag

6.3.3.3 typedef void(* EmberDnsResponseHandler) (EmberDnsLookupStatus status, const uint8_t *domainName, uint8_t domainNameLength, const EmberDnsResponse *response, void *applicationData, uint16_t applicationDataLength)

Parameters

status	A EmberDnsLookupSatus indicating success or failure.
domainName	The name that was looked up.
domainNameLength	Length of domainName in bytes.
response	Response information, NULL if no response was received.
applicationData	Application data that was passed to emberDnsLookup.
applicationDataLength	Length of applicationData in bytes.

6.3.3.4 typedef uint8_t LocalServerFlag

6.3.4 Enumeration Type Documentation

6.3.4.1 enum EmberBorderRouterTlvFlag_e

Enumerator

EMBER_BORDER_ROUTER_ND_DNS_FLAG

EMBER_BORDER_ROUTER_ON_MESH_FLAG

EMBER_BORDER_ROUTER_DEFAULT_ROUTE_FLAG

EMBER_BORDER_ROUTER_CONFIGURE_FLAG

EMBER_BORDER_ROUTER_DHCP_FLAG

EMBER_BORDER_ROUTER_SLAAC_FLAG

EMBER_BORDER_ROUTER_PREFERRED_FLAG

EMBER_BORDER_ROUTER_PREFERENCE_MASK

EMBER_BORDER_ROUTER_HIGH_PREFERENCE

EMBER_BORDER_ROUTER_MEDIUM_PREFERENCE

EMBER_BORDER_ROUTER_LOW_PREFERENCE

6.3.4.2 enum EmberDnsLookupStatus

Enumerator

EMBER_DNS_LOOKUP_SUCCESS

EMBER_DNS_LOOKUP_NO_BORDER_ROUTER

EMBER_DNS_LOOKUP_NO_BORDER_ROUTER_RESPONSE

EMBER_DNS_LOOKUP_BORDER_ROUTER_RESPONSE_ERROR

EMBER_DNS_LOOKUP_NO_DNS_SERVER

EMBER_DNS_LOOKUP_NO_DNS_RESPONSE

EMBER_DNS_LOOKUP_NO_DNS_RESPONSE_ERROR

EMBER_DNS_LOOKUP_NO_ENTRY_FOR_NAME

EMBER_DNS_LOOKUP_NO_BUFFERS

6.3.4.3 enum EmberExternalRouteTlvFlag_e

Enumerator

EMBER_EXTERNAL_ROUTE_PREFERENCE_MASK

EMBER_EXTERNAL_ROUTE_HIGH_PREFERENCE

EMBER_EXTERNAL_ROUTE_MEDIUM_PREFERENCE

EMBER_EXTERNAL_ROUTE_LOW_PREFERENCE

6.3.4.4 enum EmberLocalAddressScope

Enumerator

REALM_SCOPE

LINK_SCOPE

GLOBAL_SCOPE

6.3.4.5 enum LocalServerFlag_e

The EMBER_GLOBAL_ADDRESS_AM_ flags are set for a border router that is supplying prefixes.

The rest of the EMBER_GLOBAL_ADDRESS_ flags are set for prefixes that have been administered on other devices.

EMBER_LOCAL_ADDRESS is supplied if this a Thread mesh-local or link-local IPv6 address. No other flags are set in this case.

Enumerator

EMBER_GLOBAL_ADDRESS_AM_GATEWAY

EMBER_GLOBAL_ADDRESS_AM_DHCP_SERVER

EMBER_GLOBAL_ADDRESS_AM_SLAAC_SERVER

EMBER_GLOBAL_ADDRESS_DHCP

EMBER_GLOBAL_ADDRESS_SLAAC

EMBER_GLOBAL_ADDRESS_CONFIGURED

EMBER_GLOBAL_ADDRESS_REQUEST_SENT

EMBER_GLOBAL_ADDRESS_REQUEST_FAILED

EMBER_LOCAL_ADDRESS

6.3.5 Function Documentation

6.3.5.1 void emberAddressConfigurationChangeHandler (const EmberIpv6Address * *address*, uint32_t *preferredLifetime*, uint32_t *validLifetime*, uint8_t *addressFlags*)

If addressFlags is EMBER_LOCAL_ADDRESS, it means that the address configured is a Thread-local address.

Otherwise, it means that the address assigned is a global address (DHCP or SLAAC).

In either case, if the valid lifetime is zero, then the address is no longer available.

Parameters

address	the address
preferredLifetime	the preferred lifetime of the address (in seconds)
validLifetime	the valid lifetime of the address (in seconds)
addressFlags	address configuration flags (see LocalServerFlag_e)

This function is called when a new address is configured on the application.

If addressFlags is EMBER_LOCAL_ADDRESS, it means that the address configured is a Thread-local address.

Otherwise, it means that the address assigned is a global address (DHCP or SLAAC).

In either case, if the valid lifetime is zero, then the address is no longer available.

Parameters

address	the address
preferredLifetime	the preferred lifetime of the address (in seconds)
validLifetime	the valid lifetime of the address (in seconds)
addressFlags	address configuration flags (see LocalServerFlag_e)

6.3.5.2 void emberConfigureExternalRoute (EmberDefaultRouteTlvFlag *extRouteFlags*, bool *isStable*, const uint8_t * *extRoutePrefix*, uint8_t *extRoutePrefixLengthInBits*, uint8_t *extRouteDomainId*)

Also de-configures an external route prefix (if it exists), if the extRoutePrefix and extRoutePrefixLengthInBits arguments are specified, and externalRouteFlags equals 0xFF.

Parameters

extRouteFlags	-> See EmberDefaultRouteTlvFlag -> 0xFF to de-configure the specified route.
isStable	If true, the route is expected to be available for at least MIN_STABLE_LIFETIME (168) hours.
extRoutePrefix	Prefix for the route
extRoutePrefixLengthInBits	Prefix length in bits
extRouteDomainId	Domain ID

6.3.5.3 void emberConfigureExternalRouteReturn (EmberStatus *status*)

This function provides the result of a call to emberConfigureExternalRoute.

6.3.5.4 void emberConfigureGateway (EmberBorderRouterTlvFlag *borderRouterFlags*, bool *isStable*, const uint8_t * *prefix*, const uint8_t *prefixLengthInBits*, uint8_t *domainId*, uint32_t *preferredLifetime*, uint32_t *validLifetime*)

Also de-configures a border router prefix (if it exists), if the prefix and prefixLengthInBits are specified, and borderRouterFlags equals 0x00FF.

This triggers an address configuration change on the border router, and the application is informed of this by emberAddressConfigurationChangeHandler.

Note: If the application wants to manually configure an address and not have the stack create one, then it should pass in the entire IPv6 address (in bytes) for the prefix argument, with prefixLengthInBits as 128.

Examples:

SLAAC:

To configure a valid SLAAC border router, use: EMBER_BORDER_ROUTER_SLAAC_FLAG | EMBER_BORDE↩ R_ROUTER_DEFAULT_ROUTE_FLAG

The preference of the SLAAC prefix can be set using EMBER_BORDER_ROUTER_HIGH_PREFERENCE or E↩ MBER_BORDER_ROUTER_LOW_PREFERENCE

NOTE: Preferred and valid lifetimes are ignored for SLAAC prefixes.

Configuring a SLAAC prefix will trigger emberAddressConfigurationChangeHandler on other nodes that may choose to configure a SLAAC address.

DHCP:

To configure a valid DHCP border router, use: EMBER_BORDER_ROUTER_DHCP_FLAG | EMBER_BORDER↩ _ROUTER_DEFAULT_ROUTE_FLAG)

Note that this function only informs the network that this device is a DHCP server. The application is responsible for handling all messages send to the DHCP server port.

EMBER_BORDER_ROUTER_CONFIGURE_FLAG may be set if this border router is a DHCP server that supplies other configurationd data, such as the identity of DNS servers.

Configuring a DHCP prefix will trigger emberDhcpServerChangeHandler and other devices may choose to request a DHCP address by calling emberRequestDhcpAddress. If they get an address, they are informed via ember↩ AddressConfigurationChangeHandler.

Parameters

borderRouterFlags	-> See EmberBorderRouterTlvFlag -> 0x00FF to de-configure the specified prefix.
isStable	If true, the border router that uses this prefix offers a route that is expected to be available for at least MIN_STABLE_LIFETIME (168) hours.
prefix	Prefix for the border router.
prefixLengthInBits	Prefix length in bits.
domainId	Domain ID.
preferredLifetime	Ignored; included for backward compatibility.
validLifetime	Ignored; included for backward compatibility.

6.3.5.5 void emberConfigureGatewayReturn (EmberStatus *status*)

This function provides the result of a call to emberConfigureGateway.

6.3.5.6 void emberDhcpServerChangeHandler (const uint8_t * *prefix*, uint8_t *prefixLengthInBits*, bool *available*)

"available" means the DHCP server can offer us an address if requested.

Parameters

prefix	dhcp server prefix
prefixLengthInBits	length in bits of the prefix
available	whether this dhcp server is available

This function is called when the stack knows about a new dhcp server or if a dhcp server has become unavailable.

"available" means the DHCP server can offer us an address if requested.

Parameters

prefix	dhcp server prefix
prefixLengthInBits	length in bits of the prefix
available	whether this dhcp server is available

6.3.5.7 EmberStatus emberDnsLookup (const uint8_t * *domainName,* uint8_t *domainNameLength,* const uint8_t * *prefix64,* EmberDnsResponseHandler *responseHandler,* uint8_t * *appData,* uint16_t *appDataLength*)

Parameters

domainName	The name to be looked up.
domainNameLength	Length of domainName in bytes.
prefix64	A 64-bit prefix that specifies the domain in which to perform the lookup.
responseHandler	Handler to which the response will be passed.
applicationData	Application data to be passed to responseHandler.
applicationDataLength	Length of applicationData in bytes.

6.3.5.8 void emberExternalRouteChangeHandler (const uint8_t * *prefix,* uint8_t *prefixLengthInBits,* bool *available*)

Parameters

prefix	External route prefix
prefixLengthInBits	Length in bits of the prefix
available	Whether this external route is available.

This function is called when the stack knows about a border router that has an external route to a prefix.

Parameters

prefix	external route prefix
prefixLengthInBits	length in bits of the prefix
available	whether this external route is available.

6.3.5.9 void emberGetGlobalAddresses (const uint8_t * *prefix*, uint8_t *prefixLengthInBits*)

emberGetGlobalAddressReturn callbacks contain information about these global addresses.

Once all valid entries have been returned, an extra zeroed-out entry is returned to indicate completion.

Parameters

prefix	Address prefix
prefixLengthInBits	Length in bits of the prefix

6.3.5.10 void emberGetGlobalAddressReturn (const EmberIpv6Address * *address*, uint32_t *preferredLifetime*, uint32_t *validLifetime*, uint8_t *addressFlags*)

Parameters

address	IPv6 global address
preferredLifetime	Preferred lifetime (in seconds)
validLifetime	Valid lifetime (in seconds)
addressFlags	Address configuration flags (EMBER_GLOBAL_ADDRESS_*)

This function provides the result of a call to emberGetGlobalAddresses.

Parameters

address	IPv6 global address
preferredLifetime	Preferred lifetime (in seconds)
validLifetime	Valid lifetime (in seconds)
addressFlags	Address configuration flags (EMBER_GLOBAL_ADDRESS_*)

6.3.5.11 void emberGetGlobalPrefixes (void)

emberGetGlobalPrefixReturn callbacks contain information about the border routers.

Once all valid entries have been returned, an extra zeroed-out entry is returned to indicate completion.

6.3.5.12 void emberGetGlobalPrefixReturn (uint8_t *flags*, bool *isStable*, const uint8_t * *prefix*, uint8_t *prefixLengthInBits*, uint8_t *domainId*, uint32_t *preferredLifetime*, uint32_t *validLifetime*)

Parameters

flags	Please ignore this param, it is currently unused. (returns 0)
isStable	Stable or temporary prefix
prefix	Border router prefix
prefixLengthInBits	Prefix length in bits
domainId	Provisioning domain ID
preferredLifetime	Preferred lifetime (in seconds)
validLifetime	Valid lifetime (in seconds)

This function provides the result of a call to ::emberGetGlobalPrefix.

Parameters

flags	Please ignore this param, it is currently unused. (returns 0)
isStable	Stable or temporary prefix
prefix	Border router prefix
prefixLengthInBits	Prefix length in bits
domainId	Provisioning domain ID
preferredLifetime	Preferred lifetime (in seconds)
validLifetime	Valid lifetime (in seconds)

6.3.5.13 bool emberGetLocalIpAddress (uint8_t *index*, EmberIpv6Address * *address*)

Index 0 contains the mesh-local 64 address of the node. Index 1 contains the link-local 64 address of the node. Index 2 and greater will return any global unicast addresses (GUAs) of this node.

Returns

false if no IPv6 address is stored at the given index.

6.3.5.14 void emberGetNetworkData (uint8_t * *networkDataBuffer*, uint16_t *bufferLength*)

Parameters

networkDataBuffer	Network Data will be copied to here
bufferLength	Length in bytes of the buffer

6.3.5.15 void emberGetNetworkDataReturn (EmberStatus *status*, uint8_t * *networkData*, uint16_t *bufferLength*)

The status value is one of:

- EMBER_SUCCESS
- EMBER_NETWORK_DOWN
- EMBER_BAD_ARGUMENT (the supplied buffer was too small)

Parameters

status	
networkData	Location of the Network Data
dataLength	Length in bytes of the Network Data

This function provides the result of a call to emberGetNetworkData.

The status value is one of:

- EMBER_SUCCESS
- EMBER_NETWORK_DOWN
- EMBER_BAD_ARGUMENT (the supplied buffer was too small)

Parameters

status	
networkData	location of the Network Data
dataLength	length in bytes of the Network Data

6.3.5.16 void emberGetRoutingLocator (void)

A Thread Routing Locator (RLOC) is an IPv6 address that identifies the location of a Thread interface within a Thread partition. Thread devices use RLOCs internally for communicating control traffic.

NOTE: Using RLOCs for application messaging is NOT recommended since device identifiers used to build these RLOC addresses may change at any time based on the current network state. Please note that message delivery is not guaranteed when an RLOC address is used.

It is recommended that application developers use emberGetLocalIpAddress.

6.3.5.17 void emberGetRoutingLocatorReturn (const EmberIpv6Address * *rloc*)

Parameters

rloc	The Routing Locator as a full IPv6 address.

This function provides the result of a call to emberGetRoutingLocator.

Parameters

rloc	The Routing Locator as a full IPv6 address.

6.3.5.18 void emberNetworkDataChangeHandler (const uint8_t * *networkData*, uint16_t *length*)

Parameters

networkData	Network Data
length	Length in bytes of the Network Data

This function is called when the stack receives new Thread Network Data. The networkData argument may be NULL, in which case emberGetNetworkData can be used to obtain the new Thread Network Data.

Parameters

networkData	the Network Data
length	length in bytes of the Network Data

6.3.5.19 void emberRequestDhcpAddress (const uint8_t * *prefix*, uint8_t *prefixLengthInBits*)

The application can also call emberGetGlobalPrefixes to look for DHCP servers that it can request for an address.

When the address is obtained, the application is informed of this via emberAddressConfigurationChangeHandler.

Parameters

prefix	dhcp server prefix
prefixLengthInBits	length in bits of the prefix

6.3.5.20 void emberRequestDhcpAddressReturn (EmberStatus *status*, const uint8_t * *prefix*, uint8_t *prefixLengthInBits*)

This call only indicates the status of the request (EMBER_ERR_FATAL if no DHCP server is found, and EMBER↩ _SUCCESS otherwise). The assigned IPv6 address is returned via emberAddressConfigurationChangeHandler

Parameters

status	Status of DHCP Address Request
prefix	Prefix requested in emberRequestDhcpAddress
prefixLengthInBits	Prefix length in bits requested in emberRequestDhcpAddress

This function provides the result of a call to emberRequestDhcpAddress.

This call only indicates the status of the request (EMBER_ERR_FATAL if no DHCP server is found, and EMBER↩ _SUCCESS otherwise). The assigned IPv6 address is returned via emberAddressConfigurationChangeHandler

Parameters

status	Status of DHCP Address Request
prefix	Prefix requested in emberRequestDhcpAddress
prefixLengthInBits	Prefix length in bits requested in emberRequestDhcpAddress

6.3.5.21 void emberRequestSlaacAddress (const uint8_t * *prefix*, uint8_t *prefixLengthInBits*)

The application can also call emberGetGlobalPrefixes to look for SLAAC prefixes that it can use to configure an address.

If the application wants to manually configure an address and not have the stack create one, then it should pass in the entire IPv6 address (in bytes) for the prefix argument, with prefixLengthInBits as 128.

When the address is obtained, the application is informed of this via emberAddressConfigurationChangeHandler.

Parameters

prefix	SLAAC prefix
prefixLengthInBits	Length in bits of the prefix

6.3.5.22 void emberRequestSlaacAddressReturn (EmberStatus *status*, const uint8_t * *prefix*, uint8_t *prefixLengthInBits*)

This call only indicates the status of the request (EMBER_ERR_FATAL if no SLAAC server is found, and EMBE↩ R_SUCCESS otherwise). The assigned IPv6 address is returned via emberAddressConfigurationChangeHandler

Parameters

status	Status of SLAAC Address Request
prefix	Prefix requested in emberRequestSlaacAddress
prefixLengthInBits	Prefix length in bits requested in emberRequestSlaacAddress

This function provides the result of a call to emberRequestSlaacAddress.

This call only indicates the status of the request (EMBER_ERR_FATAL if no SLAAC server is found, and EMBE↩ R_SUCCESS otherwise). The assigned IPv6 address is returned via emberAddressConfigurationChangeHandler

Parameters

status	Status of SLAAC Address Request
prefix	Prefix requested in emberRequestSlaacAddress
prefixLengthInBits	Prefix length in bits requested in emberRequestSlaacAddress

6.3.5.23 void emberResignGlobalAddress (const EmberIpv6Address * *address*)

6.3.5.24 void emberResignGlobalAddressReturn (EmberStatus *status*)

This function provides the result of a call to emberResignGlobalAddress().

6.3.5.25 void emberSetLocalNetworkData (const uint8_t * *networkData*, uint16_t *length*)

The stack will set the correct local node ID into the TLVs.

This function is an alternative to the emberConfigureGateway emberConfigureExternalRoute functions that provides full access to Network Data configuration. A call to this function removes any previous configuration, including uses of emberConfigureGateway and emberConfigureExternalRoute.

Parameters

networkData	A pointer to a set of Thread Network Data TLVs that describe the local nodes Border Router and Server capabilities.
length	The number of bytes in the supplied network data.

6.3.5.26 void emberSetLocalNetworkDataReturn (EmberStatus *status,* uint16_t *length*)

Provides the result of a call to ::emberSetServerNetworkData.

6.3.5.27 void emberSetNdData (const uint8_t * *data,* uint16_t *length*)

6.3.5.28 void emberSetNdDataReturn (EmberStatus *status,* uint16_t *length*)

This function provides the result of a call to emberSetNdData.

6.3.5.29 void emberSlaacServerChangeHandler (const uint8_t * *prefix,* uint8_t *prefixLengthInBits,* bool *available*)

"available" means we can configure a SLAAC address.

Parameters

prefix	SLAAC prefix
prefixLengthInBits	length in bits of the prefix
available	whether we can configure an address

This function is called when the stack knows about a new SLAAC prefix or if a SLAAC server has become unavailable.

"available" means we can configure a SLAAC address.

Parameters

prefix	SLAAC prefix
prefixLengthInBits	length in bits of the prefix
available	whether we can configure an address

6.4 Commissioning

Commissioning Utilities.

Enumerations

- enum {
  EMBER_NO_COMMISSIONER = 0,
  EMBER_HAVE_COMMISSIONER = BIT(0),
  EMBER_AM_COMMISSIONER = BIT(1),
  EMBER_JOINING_ENABLED = BIT(2),
  EMBER_JOINING_WITH_EUI_STEERING = BIT(3) }

  Flag values for emberCommissionerStatusHandler().
- enum EmberJoiningMode {
  EMBER_NO_JOINING,
  EMBER_JOINING_ALLOW_ALL_STEERING,
  EMBER_JOINING_ALLOW_EUI_STEERING,
  EMBER_JOINING_ALLOW_SMALL_EUI_STEERING }

  Joining modes, passed to emberSetJoiningMode() on the commissioner. No change takes place until emberSend↩ SteeringData() is called. If steering is used, the EUI-64s of the joining devices should be passed to emberAdd↩ SteeringEui64() before calling emberSendSteeringData().

Functions

- void emberBecomeCommissioner (const uint8_t *deviceName, uint8_t deviceNameLength)

  This function petitions to make this device the commissioner for the network. This will succeed if there is no active commissioner and fail if there is one.
- void emberBecomeCommissionerReturn (EmberStatus status)

  Return call for emberBecomeCommissioner(). The status is EMBER_SUCCESS if a petition was sent or EMBER↩ _ERR_FATAL if some temporary resource shortage prevented doing so.
- void emberStopCommissioning (void)

  This function causes this device to cease being the active commissioner. This call always succeeds and has no return.
- void emberGetCommissioner (void)

  This function causes the stack to call emberCommissionerStatusHandler() to report the current commissioner status. This always succeeds and has no return.
- void emberAllowNativeCommissioner (bool on)

  This function causes the stack to allow a connection to a native commissioner.
- void emberAllowNativeCommissionerReturn (EmberStatus status)

  This function provides the result of a call to emberAllowNativeCommissioner(): either EMBER_SUCCESS or EMB↩ ER_INVALID_CALL.
- void emberSetCommissionerKey (const uint8_t *commissionerKey, uint8_t commissionerKeyLength)

  This function sets the key that a native commissioner must use to establish a connection to a Thread router. The commissionerKey argument is known as the "commissioning credential" in the Thread spec and must be between 6 and 255 bytes in length. Internally, it is hashed to derive the 16-byte Pre-Shared Key for the commissioner, known as the PSKc.
- void emberSetCommissionerKeyReturn (EmberStatus status)

  This function provides the result of a call to emberSetCommissionerKey(): either EMBER_SUCCESS or EMBER_↩ INVALID_CALL.
- void emberSetPskcHandler (const uint8_t *pskc)

  Handler to let application know that a PSKc TLV was successfully set.

- void emberCommissionerStatusHandler (uint16_t flags, const uint8_t *commissionerName, uint8_t commissionerNameLength)

  This function reports on the current commissioner state.
- void emberSetJoiningMode (EmberJoiningMode mode, uint8_t length)

  This function sets the joining mode, clearing the steering data if steering is to be used.
- void emberAddSteeringEui64 (const EmberEui64 *eui64)

  This function adds the given EUI64 to the steering data if this device is the active commissioner; has no effect otherwise.
- void emberSendSteeringData (void)

  This function sends the current steering data to the network, enabling joining in the process.
- void emberSendSteeringDataReturn (EmberStatus status)

  This function provides the result of a call to emberSendSteeringData().
- void emberSetJoinKey (const EmberEui64 *eui64, const uint8_t *key, uint8_t keyLength)

  This function supplies the commissioner with the key a joining node will be using.
- void emberSetJoinKeyReturn (EmberStatus status)

  This function provides the result of a call to emberSetJoinKey().
- void emberEnableHostDtlsClient (bool enable)

  This function allows DTLS implementations on the host.
- void emberCommissionNetwork (uint8_t preferredChannel, uint32_t fallbackChannelMask, const uint8_t *networkId, uint8_t networkIdLength, uint16_t panId, const uint8_t *ulaPrefix, const uint8_t *extendedPanId, const EmberKeyData *key, uint32_t keySequence)

  This function commissions the network.
- void emberCommissionNetworkReturn (EmberStatus status)

  This function provides the result of a call to emberCommissionNetwork.

6.4.1 Detailed Description

See network-management.h for source code.

6.4.2 Enumeration Type Documentation

6.4.2.1 anonymous enum

Enumerator

EMBER_NO_COMMISSIONER

EMBER_HAVE_COMMISSIONER

EMBER_AM_COMMISSIONER

EMBER_JOINING_ENABLED

EMBER_JOINING_WITH_EUI_STEERING

6.4.2.2 enum EmberJoiningMode

Enumerator

EMBER_NO_JOINING Disable joining.

EMBER_JOINING_ALLOW_ALL_STEERING Allow joining, with no steering information.

EMBER_JOINING_ALLOW_EUI_STEERING Allow joining, clearing steering data.

EMBER_JOINING_ALLOW_SMALL_EUI_STEERING Allow joining, clearing steering data. Only the low three bytes of EUI-64s will be used for steering. Note: This option is deprecated in Thread 1.1.

6.4.3 Function Documentation

6.4.3.1 void emberAddSteeringEui64 (const EmberEui64 * *eui64*)

The steering data is a Bloom filter for the EUI64s of the devices that are expected to join the network. Each added EUI64 is passed to a hash function to choose a set of bits in the filter, and those bits are set. Each potential joiner can then hash their own EUI64 and check if the resulting bits are set in the advertised filter. If so, the device is (probably) expected to join; if not, it definitely is not expected to join.

6.4.3.2 void emberAllowNativeCommissioner (bool *on*)

Note: This call must be made on the leader before forming a network.

Parameters

on	Enable / disable connections to native commissioners

6.4.3.3 void emberAllowNativeCommissionerReturn (EmberStatus *status*)

This function provides the result of a call to emberAllowNativeCommissioner(): either EMBER_SUCCESS or EM↩ BER_INVALID_CALL.

6.4.3.4 void emberBecomeCommissioner (const uint8_t * *deviceName*, uint8_t *deviceNameLength*)

Parameters

deviceName	A name for this device as a human-readable string. If this device becomes the commissioner this name is sent to any other would-be commissioners so that the user can identify the current commissioner.
deviceNameLength	The length of the name.

6.4.3.5 void emberBecomeCommissionerReturn (EmberStatus *status*)

6.4.3.6 void emberCommissionerStatusHandler (uint16_t *flags*, const uint8_t * *commissionerName*, uint8_t *commissionerNameLength*)

Parameters

flags	A combination of zero or more of the following: • EMBER_HAVE_COMMISSIONER a commissioner is active in the network • EMBER_AM_COMMISSIONER this device is the active commissioner if emberStopCommissioning is called, then this flag is not returned as we are open to commissioner petitions • EMBER_JOINING_ENABLED joining is enabled • EMBER_JOINING_WITH_EUI_STEERING steering data restricts which devices can join. if not set, no restriction, any device can join (significant only when EMBER_JOINING_ENABLED is set)
commissionerName	The name of the active commissioner, or NULL if there is none or the name is not known.
commissionerNameLength	The length of commissonerName.

This function reports on the current commissioner state.

Parameters

flags	A combination of zero or more of the following: • EMBER_HAVE_COMMISSIONER a commissioner is active in the network • EMBER_AM_COMMISSIONER this device is the active commissioner if emberStopCommissioning is called, then this flag is not returned as we are open to commissioner petitions • EMBER_JOINING_ENABLED joining is enabled • EMBER_JOINING_WITH_EUI_STEERING steering data restricts which devices can join. if not set, no restriction, any device can join (significant only when EMBER_JOINING_ENABLED is set)
commissionerName	The name of the active commissioner, or NULL if there is none or the name is not known.
commissionerNameLength	The length of commissonerName.

6.4.3.7 void emberCommissionNetwork (uint8_t *preferredChannel,* uint32_t *fallbackChannelMask,* const uint8_t * *networkId,* uint8_t *networkIdLength,* uint16_t *panId,* const uint8_t * *ulaPrefix,* const uint8_t * *extendedPanId,* const EmberKeyData * *key,* uint32_t *keySequence*)

This call must be made prior to calling emberJoinCommissioned(). It will not be successful if the node is already on a network.

All options except panId are REQUIRED. If a REQUIRED option is not provided, the callback emberJoinNetwork↩ Return will be sent to the app with an EMBER_BAD_ARGUMENT status.

Notes: If preferredChannel is 0, EMBER_ALL_802_15_4_CHANNELS_MASK is used instead of fallbackChannel↩ Mask. If preferredChannel is valid, it will automatically be added to the fallbackChannelMask.

Parameters

preferredChannel	[the preferred channel]
fallbackChannelMask	[the fallback channel mask]
networkId	[the network ID]
networkIdLength	[the string length of networkId]
panId	[the short pan ID]
ulaPrefix	[the 8-byte ULA prefix]
extendedPanId	[the 8-byte extended pan ID]
key	[the master key]
keySequence	[starting key sequence, default: 0]

6.4.3.8 void emberCommissionNetworkReturn (EmberStatus *status*)

Returns EMBER_SUCCESS if successful EMBER_BAD_ARGUMENT if any of the options are wrong EMBER_I↩ NVALID_CALL if the node is already on a network

Parameters

status	Whether the call to emberCommissionNetwork was successful

This function provides the result of a call to emberCommissionNetwork.

Returns EMBER_SUCCESS if successful EMBER_BAD_ARGUMENT if any of the options are wrong EMBER_I↩ NVALID_CALL if the node is already on a network

Parameters

status	Whether the call to emberCommissionNetwork was successful

6.4.3.9 void emberEnableHostDtlsClient (bool *enable*)

This call is made in order to force the host to interface with an external commissioner if available, or use DTLS capabilities on the host (if they exist) for Thread joining or other security handshakes.

This is enabled by default for the Thread Border Router implementation. However, if the device (Border Router or otherwise) wants to use existing DTLS capabilities on the NCP stack, such as for joining, this should be toggled to false.

Parameters

enable	If true, this call allows the host to perform DTLS.

6.4.3.10 void emberGetCommissioner (void)

6.4.3.11 void emberSendSteeringData (void)

6.4.3.12 void emberSendSteeringDataReturn (EmberStatus *status*)

This function provides the result of a call to emberSendSteeringData().

6.4.3.13 void emberSetCommissionerKey (const uint8_t * *commissionerKey,* uint8_t *commissionerKeyLength*)

Note: This call must be made on the leader before forming a network, or on an on-mesh commissioner that wants to set the PSKc in the active dataset.

Parameters

commissionerKey	the key
commissionerKeyLength	key length

6.4.3.14 void emberSetCommissionerKeyReturn (EmberStatus *status*)

This function provides the result of a call to emberSetCommissionerKey(): either EMBER_SUCCESS or EMBER↩ _INVALID_CALL.

6.4.3.15 void emberSetJoiningMode (EmberJoiningMode *mode,* uint8_t *length*)

No change takes place until emberSendSteeringData() is called. If steering is used, the EUI-64s of the joining devices should be passed to emberAddSteeringEui64() before calling emberSendSteeringData().

Parameters

mode	The joining mode
length	The length in bytes of the Bloom filter to be included in the Steering Data TLV. This field is only applicable when mode is set to EMBER_JOINING_ALLOW_EUI_STEERING. Refer to the Thread specification for details on the Bloom filter and the probability of collisions given the number of bits in the Bloom filter and the number of identifiers included.

6.4.3.16 void emberSetJoinKey (const EmberEui64 * *eui64,* const uint8_t * *key,* uint8_t *keyLength*)

Parameters

eui64	The EUI64 of the next node expected to join. NULL may be used if the EUI64 is not known.
key	The joining key that the device will be using.
keyLength	The length of the joining key.

6.4.3.17 void emberSetJoinKeyReturn (EmberStatus *status*)

This function provides the result of a call to emberSetJoinKey().

6.4.3.18 void emberSetPskcHandler (const uint8_t * *pskc*)

Parameters

pskc	PSKc: 16 bytes in length

6.4.3.19 void emberStopCommissioning (void)

When this call is made, emberCommissionerStatusHandler will not return the EMBER_AM_COMMISSIONER flag anymore.

6.5 Network Utilities

Network Utilities.

Data Structures

- struct EmberNetworkParameters

  An application structure to hold useful network parameters.
- struct EmberRipEntry

  Structure that holds information about a routing table entry for use on the application. See emberGetRipEntry.
- struct EmberMacBeaconData

  Structure to hold information about an 802.15.4 beacon for use on the application.
- struct EmberSecurityParameters

  Values of security parameters for use in forming or joining a network.

Macros

- #define EMBER_HIGH_PRIORITY_TASKS (EMBER_OUTGOING_MESSAGES | EMBER_INCOMING_↩ MESSAGES | EMBER_RADIO_IS_ON)

  A mask of the high priority tasks that prevent a device from sleeping. Devices should not sleep if any high priority tasks are active.
- #define ISLAND_ID_SIZE 5

  Size of the island (aka network fragment) ID.
- #define EMBER_NETWORK_KEY_OPTION BIT(0)

  Define the various options for setting network parameters. Note: Only the EMBER_NETWORK_KEY_OPTION works at this time.
- #define EMBER_PSK_JOINING_OPTION BIT(1)
- #define EMBER_ECC_JOINING_OPTION BIT(2)

Typedefs

- typedef uint8_t EmberTokenId

  Read token values stored on the Ember chip.
- typedef uint8_t EmberMfgTokenId

  Token identifier used when reading and writing manufacturing tokens.

Enumerations

- enum EmberIdleRadioState {
  IDLE_WITH_RADIO_ON,
  IDLE_WITH_POLLING,
  IDLE_WITH_RADIO_OFF }

  Required radio state while stack is idle.
- enum {
  EMBER_OUTGOING_MESSAGES = 0x01,
  EMBER_INCOMING_MESSAGES = 0x02,
  EMBER_RADIO_IS_ON = 0x04 }

  This function defines tasks that prevent the stack from sleeping.

- enum EmberResetCause {
  EMBER_RESET_UNKNOWN,
  EMBER_RESET_FIB,
  EMBER_RESET_BOOTLOADER,
  EMBER_RESET_EXTERNAL,
  EMBER_RESET_POWERON,
  EMBER_RESET_WATCHDOG,
  EMBER_RESET_SOFTWARE,
  EMBER_RESET_CRASH,
  EMBER_RESET_FLASH,
  EMBER_RESET_FATAL,
  EMBER_RESET_FAULT,
  EMBER_RESET_BROWNOUT }

  Enumerate the various chip reset causes.
- enum { EMBER_CHANNEL_CAL_DATA_TOKEN }

  Enumerate the various token values that can be retrieved by the application.
- enum {
  EMBER_CUSTOM_EUI_64_MFG_TOKEN,
  EMBER_EZSP_STORAGE_MFG_TOKEN,
  EMBER_CTUNE_MFG_TOKEN }

  Enumerate the various manufacturing token values that can be read or written by the application.

Functions

- void emberInit (void)

  This function initializes the Ember stack.
- void emberInitReturn (EmberStatus status)

  This function provides the result of a call to emberInit().
- void emberTick (void)

  A periodic tick routine that must be called in the application's main event loop.
- void emberResetNetworkState (void)

  This function erases the network state stored in nonvolatile memory after which the network status will be EMBER_↩ NO_NETWORK. This function should not be called to rejoin a former network; use emberResumeNetwork() instead. There may be difficulties joining a former network after resetting the network state, due to security considerations.
- void emberResetNetworkStateReturn (EmberStatus status)

  This function provides the result of a call to emberResetNetworkState().
- bool emberDeepSleepTick (void)

  An application handler for deep sleep on sleepy end devices. This call is ignored for non-sleepy devices. The device may or may not sleep depending on the internal state.
- void emberDeepSleep (bool sleep)

  This function turns chip deep sleep on or off for sleepy end devices. This call is ignored on non-sleepy devices. The device may or may not sleep depending on the internal state.
- void emberDeepSleepReturn (EmberStatus status)

  This function provides the result of a call to emberDeepSleep().
- void emberDeepSleepCompleteHandler (uint16_t sleepDuration)

  For a sleepy end device, report how long the chip went to deep sleep. In a NCP + host setup, the stack reports this to the host app.
- uint32_t emberStackIdleTimeMs (EmberIdleRadioState *radioStateResult)

  This function returns the time the stack will be idle, in milliseconds. Also sets radioStateResult to the required radio state while the stack is idle.
- bool emberOkToNap (void)

  This function indicates whether the stack is currently in a state where there are no high priority tasks and may sleep.
- void emberOkToNapReturn (uint8_t stateMask)

If implementing event-driven sleep on an NCP host, this method will return the bitmask indicating the stack's current tasks. (see enum above)

- void emberEventDelayUpdatedFromIsrHandler (Event *event)

  This method is called any time an event is scheduled from within an ISR context. It can be used to determine when to stop a long running sleep to see what application or stack events now need to be processed.

- void emberStackPrepareForPowerDown (void)

  This function gets the stack ready for power down, or deep sleep. Purges the MAC indirect queue, and empties the phy-to-mac and mac-to-network queues.

- bool emberStackPreparingForPowerDown (void)

  This function returns true if the stack is currently emptying any message queues or false if the MAC queue is currently not empty.

- void emberStackPowerDown (void)

  Immediately turns the radio power completely off.

- void emberStackPowerUp (void)

  This function initializes the radio. Typically called coming out of deep sleep.

- void emberStackPollForData (uint32_t pollMs)

  For sleepy hosts, use this call to have the stack manage polling for sleepy end devices. In a host/NCP setup, this means that the NCP app will take care of periodic data polling.

- void emberStackPollForDataReturn (EmberStatus status)

  This function provides the result of a call to emberStackPollForData().

- void emberPollForData (void)

  Use this call if setting up polling for sleepy end devices on the application.

- void emberPollForDataReturn (EmberStatus status)

  This function provides the result of a call to emberPollForData().

- const EmberEui64 * emberEui64 (void)

  This function returns the EUI64 of the Ember chip.

- EmberNetworkStatus emberNetworkStatus (void)

  This function returns the current status of the network. Prior to calling emberInitNetwork(), the status is EMBER_N↩ ETWORK_UNINITIALIZED.

- void emberNetworkStatusHandler (EmberNetworkStatus newNetworkStatus, EmberNetworkStatus old↩ NetworkStatus, EmberJoinFailureReason reason)

  This function reports a change to the network status. For example, the network status changes while going through the joining process, or while reattaching to the network, which can happen for a variety of reasons. In particular, after issuing a form, join, resume, or attach command, the application knows that the device is on the network and ready to communicate when this handler is called with a newNetworkStatus of EMBER_JOINED_NETWORK_ATTACHED.

- void emberGetNetworkParameters (EmberNetworkParameters *parameters)

  This function fetches the current network parameters into the supplied pointer.

- EmberPanId emberGetPanId (void)

  This function returns the pan id of the network.

- void emberGetRipEntry (uint8_t index)

  This function gets the EmberRipEntry at the specified index of the RIP table. The result is returned to the application via the emberGetRipEntryReturn() callback.

- void emberGetRipEntryReturn (uint8_t index, const EmberRipEntry *entry)

  This function provides the result of a call to emberGetRipEntry().

- void emberGetCounter (EmberCounterType type)

  This function gets the value for the specified counter. The result is returned to the application via emberGetCounter↩ Return().

- void emberGetCounterReturn (EmberCounterType type, uint16_t value)

  This function provides the result of a call to emberGetCounter().

- void emberClearCounters (void)

  This function resets all counter values to 0.

- void emberCounterHandler (EmberCounterType type, uint16_t increment)

A callback invoked to inform the application of the occurrence of an event defined by EmberCounterType, for example, transmissions and receptions at different layers of the stack.

- uint16_t emberCounterValueHandler (EmberCounterType type)

  A callback invoked to query the application for the countervalue of an event defined by EmberCounterType.
- bool emberForwardIpv6Packet (const uint8_t *packet, const uint16_t packetLength)

  This API provides a means to forward a raw IPv6 packet on the mesh.
- void emberStartScan (EmberNetworkScanType scanType, uint32_t channelMask, uint8_t duration)

  This function starts a scan. Note that while a scan can be initiated while the node is currently joined to a network, the node will generally be unable to communicate with its PAN during the scan period, so care should be taken when performing scans of any significant duration while presently joined to an existing PAN.
- void emberEnergyScanHandler (uint8_t channel, int8_t maxRssiValue)

  This function reports the maximum RSSI value measured on the channel.
- void emberActiveScanHandler (const EmberMacBeaconData *beaconData)

  This function reports an incoming beacon during an active scan.
- void emberScanReturn (EmberStatus status)

  This function provides the status upon completion of a scan.
- void emberStopScan (void)

  This function terminates a scan in progress.
- void emberResetMicro (void)

  This function resets the Ember chip.
- void emberResetMicroHandler (EmberResetCause cause)

  This function notifies the application of a reset on the Ember chip due to the indicated cause.
- void emberGetStandaloneBootloaderInfo (void)

  This function detects if the standalone bootloder is installed, and if so returns the installed version and info about the platform, micro and phy. If not version will be set to 0xffff. A returned version of 0x1234 would indicate version 1.2 build 34.
- void emberGetStandaloneBootloaderInfoReturn (uint16_t version, uint8_t platformId, uint8_t microId, uint8_t phyId)

  This function provides the result of a call to emberGetStandaloneBootloaderInfo.
- void emberLaunchStandaloneBootloader (uint8_t mode)

  This function launches the standalone bootloader (if installed). The function returns an error if the standalone bootloader is not present.
- void emberLaunchStandaloneBootloaderReturn (EmberStatus status)

  This function provides the result of a call to emberLaunchStandaloneBootloader.
- void emberInitHost (void)

  In a host/NCP setup, inform the NCP to send the network state and version information.
- void emberState (void)

  In a host/NCP setup, get the network parameters, the network status and eui64 all at once.
- void emberStateReturn (const EmberNetworkParameters *parameters, const EmberEui64 *localEui64, const EmberEui64 *macExtendedId, EmberNetworkStatus networkStatus)

  In a host/NCP setup, provides the result of a call to emberState() on the host.
- void emberHostStateHandler (const EmberNetworkParameters *parameters, const EmberEui64 *localEui64, const EmberEui64 *macExtendedId, EmberNetworkStatus networkStatus)

  In a host/NCP setup, notifies the host to changes in the network parameters.
- void emberSetRadioPower (int8_t power)

  This function sets the radio output power at which a node is to operate. 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. Note: If the requested power level is not available on a given radio, this function will use the next higher available power level.
- void emberSetRadioPowerReturn (EmberStatus status)

  This function provides the result of a call to emberSetRadioPower() on the host.

- void emberGetRadioPower (void)

  This function gets 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.
- void emberGetRadioPowerReturn (int8_t power)

  This function provides the result of a call to emberGetRadioPower() on the host.
- EmberStatus emberSetTxPowerMode (uint16_t txPowerMode)

  This function enables boost power mode and/or the alternate transmit path.
- void emberSetTxPowerModeReturn (EmberStatus status)

  This function provides the result of a call to emberSetTxPowerMode() on the host.
- void emberGetTxPowerMode (void)

  This function requests the current configuration of boost power mode and alternate transmitter output.
- void emberGetTxPowerModeReturn (uint16_t txPowerMode)

  This function provides the result of a call to emberGetTxPowerMode() on the host.
- void emberSetSecurityParameters (const EmberSecurityParameters *parameters, uint16_t options)

  This function is called before forming or joining. Fails if already formed or joined or if the arguments are inconsistent with the stack (i.e. if ECC is wanted and we have no ECC).
- void emberSetSecurityParametersReturn (EmberStatus status)

  This function provides the result of a call to emberSetSecurityParameters().
- void emberSwitchToNextNetworkKey (void)

  This function changes MAC encryption over to the next key. Fails if there is no next network key.
- void emberSwitchToNextNetworkKeyReturn (EmberStatus status)

  This function provides the result of a call to emberSwitchToNextNetworkKey().
- void emberSwitchToNextNetworkKeyHandler (EmberStatus status)

  This function can be stubbed out on the SoC and host app. It is used by the NCP to update security on the driver when it is instructed to switch the network key by an over the air update.
- void emberGetVersions (void)

  This function gets various versions: The stack version name (versionName) The management version number (managementVersionNumber, if applicable, otherwise set to 0xFFFF) The stack version number (stackVersion↩ Number) The stack build number (stackBuildNumber) The version type (versionType) The date / time of the build (buildTimestamp)
- void emberGetVersionsReturn (const uint8_t *versionName, uint16_t managementVersionNumber, uint16↩ _t stackVersionNumber, uint16_t stackBuildNumber, EmberVersionType versionType, const uint8_t *build↩ Timestamp)

  Provides the result of a call to emberGetVersions().
- void emberSetCcaThreshold (int8_t threshold)

  This function sets the CCA threshold level - the noise floor above which the channel is normally considered busy. The threshold parameter is expected to be a signed 2's complement value, in dBm.
- void emberSetCcaThresholdReturn (EmberStatus status)

  This function provides the result of a call to emberSetCcaThreshold().
- void emberGetCcaThreshold (void)

  This function gets the current CCA threshold level.
- void emberGetCcaThresholdReturn (int8_t threshold)

  This function provides the result of a call to emberGetCcaThreshold().
- void emberMacPassthroughMessageHandler (PacketHeader header)

  Application handler to intercept "passthrough" packets and handle them at the application.
- bool emberMacPassthroughFilterHandler (uint8_t *macHeader)

  Application handler to define "passthrough" packets.
- bool emberMacRssiFilterHandler (uint8_t *macHeader)

  Application handler to filter 802.15.4 packets to be observed for signal strength.
- void emberMacRssiHandler (int8_t currentRssi)

  Gets the received signal strength indication (RSSI) for the last 802.15.4 packet received by the stack.

- void emberGetIndexedToken (EmberTokenId tokenId, uint8_t index)

  This function gets the indexed token stored in non-volatile memory on the Ember chip. The result is returned depending on the tokenId provided (see enum above) to the appropriate Return() API.
- void emberGetChannelCalDataTokenReturn (uint8_t lna, int8_t tempAtLna, uint8_t modDac, int8_t tempAt↩ ModDac)

  This function gets the token information for tokenId = EMBER_CHANNEL_CAL_DATA_TOKEN.
- void emberGetMfgToken (EmberMfgTokenId tokenId)

  This function gets the manufacturer token stored in non-volatile memory on the Ember chip.
- void emberGetMfgTokenReturn (EmberMfgTokenId tokenId, EmberStatus status, const uint8_t *tokenData, uint8_t tokenDataLength)

  This function provides the result of a call to emberGetMfgToken.
- void emberSetMfgToken (EmberMfgTokenId tokenId, const uint8_t *tokenData, uint8_t tokenDataLength)

  This function sets the manufacturer token stored in non-volatile memory on the Ember chip.
- void emberSetMfgTokenReturn (EmberMfgTokenId tokenId, EmberStatus status)

  This function provides the result of a call to emberSetMfgToken.
- void emberGetCtune (void)

  This function gets the CTUNE value. (Only valid on EFR32)
- void emberGetCtuneReturn (uint16_t tune, EmberStatus status)

  This function provides the result of a call to emberGetCtune.
- void emberSetCtune (uint16_t tune)

  This function changes the CTUNE value. Involves switching to HFRCO and turning off the HFXO temporarily. (Only valid on EFR32)
- void emberSetCtuneReturn (EmberStatus status)

  This function provides the result of a call to emberSetCtune.
- void emberRegisterDropIncomingMessageCallback (bool(*drop)(PacketHeader header, Ipv6Header *ip↩ Header))

  *This function registers a callback function so that the application can define rules to drop incoming packets. The callback function MUST be of the form: bool func_name(PacketHeader header, Ipv6Header *ipHeader) { ... }.*
- void emberRegisterSerialTransmitCallback (void(*serialTransmit)(uint8_t type, PacketHeader header))

  This function registers a callback function so that the application can define serial transmit logic. This should only be used for NCPs, and will have no effect for SoCs. The callback function MUST be of the form: void uartTransmit(uint8↩ _t type, Buffer b) { ... }.

6.5.1 Detailed Description

See network-management.h for source code.

6.5.2 Macro Definition Documentation

6.5.2.1 #define EMBER_ECC_JOINING_OPTION BIT(2)

6.5.2.2 #define EMBER_HIGH_PRIORITY_TASKS (EMBER_OUTGOING_MESSAGES | EMBER_INCOMING_MESSAGES | EMBER_RADIO_IS_ON)

6.5.2.3 #define EMBER_NETWORK_KEY_OPTION BIT(0)

6.5.2.4 #define EMBER_PSK_JOINING_OPTION BIT(1)

6.5.2.5 #define ISLAND_ID_SIZE 5

6.5.3 Typedef Documentation

6.5.3.1 typedef uint8_t EmberMfgTokenId

6.5.3.2 typedef uint8_t EmberTokenId

6.5.4 Enumeration Type Documentation

6.5.4.1 anonymous enum

Enumerator

EMBER_OUTGOING_MESSAGES There are messages waiting for transmission.

EMBER_INCOMING_MESSAGES One or more incoming messages are being processed.

EMBER_RADIO_IS_ON The radio is currently powered on. On sleepy devices the radio is turned off when not in use. On non-sleepy devices (EMBER_ROUTER or EMBER_END_DEVICE) the radio is always on.

6.5.4.2 anonymous enum

Enumerator

EMBER_CHANNEL_CAL_DATA_TOKEN

6.5.4.3 anonymous enum

Enumerator

EMBER_CUSTOM_EUI_64_MFG_TOKEN

EMBER_EZSP_STORAGE_MFG_TOKEN

EMBER_CTUNE_MFG_TOKEN

6.5.4.4 enum EmberIdleRadioState

Enumerator

IDLE_WITH_RADIO_ON Incoming messages are expected and the radio must be left on.

IDLE_WITH_POLLING Incoming messages are expected and must be polled for.

IDLE_WITH_RADIO_OFF No messages are expected and the radio may be left off.

6.5.4.5 enum EmberResetCause

Enumerator

EMBER_RESET_UNKNOWN

EMBER_RESET_FIB

EMBER_RESET_BOOTLOADER

EMBER_RESET_EXTERNAL

EMBER_RESET_POWERON

EMBER_RESET_WATCHDOG

EMBER_RESET_SOFTWARE

EMBER_RESET_CRASH

EMBER_RESET_FLASH

EMBER_RESET_FATAL

EMBER_RESET_FAULT

EMBER_RESET_BROWNOUT

6.5.5 Function Documentation

6.5.5.1 void emberActiveScanHandler (const EmberMacBeaconData * *beaconData*)

This function reports an incoming beacon during an active scan.

6.5.5.2 void emberClearCounters (void)

6.5.5.3 void emberCounterHandler (EmberCounterType *type,* uint16_t *increment*)

The application must define EMBER_APPLICATION_HAS_COUNTER_HANDLER in its CONFIGURATION_HE↩ ADER to use this. This function may be called in ISR context, so processing should be kept to a minimum.

Parameters

type	The type of the event.
increment	Specify the increase in the counter's tally.

6.5.5.4 uint16_t emberCounterValueHandler (EmberCounterType *type*)

The application must define EMBER_APPLICATION_HAS_COUNTER_VALUE_HANDLER in its CONFIGURAT↩ ION_HEADER to use this.

Parameters

type	The type of the event.

Returns

The counter's tally.

6.5.5.5 void emberDeepSleep (bool *sleep*)

6.5.5.6 void emberDeepSleepCompleteHandler (uint16_t *sleepDuration*)

6.5.5.7 void emberDeepSleepReturn (EmberStatus *status*)

This function provides the result of a call to emberDeepSleep().

6.5.5.8 bool emberDeepSleepTick (void)

Returns

true if going to deep sleep.

6.5.5.9 void emberEnergyScanHandler (uint8_t *channel,* int8_t *maxRssiValue*)

Parameters

channel	The 802.15.4 channel on which the RSSI value was measured.
maxRssiValue	The maximum RSSI value measured (in units of dBm).

This function reports the maximum RSSI value measured on the channel.

Parameters

channel	The 802.15.4 channel on which the RSSI value was measured.
maxRssiValue	The maximum RSSI value measured (in units of dBm).

6.5.5.10 const EmberEui64* emberEui64 (void)

6.5.5.11 void emberEventDelayUpdatedFromIsrHandler (Event * *event*)

Parameters

event	The event that was scheduled by the ISR.

6.5.5.12 bool emberForwardIpv6Packet (const uint8_t * *packet,* const uint16_t *packetLength*)

Parameters

packet	Raw bytes of the IPv6 packet
packetLength	Length of the packet

6.5.5.13 void emberGetCcaThreshold (void)

6.5.5.14 void emberGetCcaThresholdReturn (int8_t *threshold*)

This function provides the result of a call to emberGetCcaThreshold().

6.5.5.15 void emberGetChannelCalDataTokenReturn (uint8_t *lna*, int8_t *tempAtLna*, uint8_t *modDac*, int8_t *tempAtModDac*)

Parameters

lna	[msb: cal needed? | bit 0-5: lna tune value]
tempAtLna	[the temp (degC) when the LNA was calibrated]
modDac	[msb: cal needed? | bit 0-5: modulation DAC tune value]
tempAtModDac	[the temp (degC) when the mod DAC was calibrated]

This function gets the token information for tokenId = EMBER_CHANNEL_CAL_DATA_TOKEN.

Parameters

lna	[msb: cal needed? | bit 0-5: lna tune value]
tempAtLna	[the temp (degC) when the LNA was calibrated] #param modDac [msb: cal needed? | bit 0-5: modulation DAC tune value]
tempAtModDac	[the temp (degC) when the mod DAC was calibrated]

6.5.5.16 void emberGetCounter (EmberCounterType *type*)

6.5.5.17 void emberGetCounterReturn (EmberCounterType *type*, uint16_t *value*)

This function provides the result of a call to emberGetCounter().

6.5.5.18 void emberGetCtune (void)

6.5.5.19 void emberGetCtuneReturn (uint16_t *tune*, EmberStatus *status*)

Parameters

tune	The current CTUNE value.
status	An EmberStatus value indicating success or the reason for failure.

This function provides the result of a call to emberGetCtune.

Parameters

tune	The current CTUNE value.
status	An EmberStatus value indicating success or the reason for failure.

6.5.5.20 void emberGetIndexedToken (EmberTokenId *tokenId,* uint8_t *index*)

6.5.5.21 void emberGetMfgToken (EmberMfgTokenId *tokenId*)

Parameters

tokenId	Which manufacturing token to read.

6.5.5.22 void emberGetMfgTokenReturn (EmberMfgTokenId *tokenId,* EmberStatus *status,* const uint8_t * *tokenData,* uint8_t *tokenDataLength*)

Parameters

tokenId	Which manufacturing token read.
status	An EmberStatus value indicating success or the reason for failure.
tokenData	The manufacturing token data.
tokenDataLength	The length of the *tokenData* parameter in bytes.

This function provides the result of a call to emberGetMfgToken.

Parameters

tokenId	Which manufacturing token read.
status	An EmberStatus value indicating success or the
tokenData	The manufacturing token data.
tokenDataLength	The length of the *tokenData* parameter in bytes.

6.5.5.23 void emberGetNetworkParameters (EmberNetworkParameters * *parameters*)

6.5.5.24 EmberPanId emberGetPanId (void)

6.5.5.25 void emberGetRadioPower (void)

Returns

Current radio output power, in dBm.

6.5.5.26 void emberGetRadioPowerReturn (int8_t *power*)

This function provides the result of a call to emberGetRadioPower() on the host.

6.5.5.27 void emberGetRipEntry (uint8_t *index*)

The index is between 0 and 31 inclusive, but there may be fewer than 32 valid entries depending on the number of routers in the network.

The caller can pass in a 0xFF index to request all valid RIP table entries. Note that the stack will ONLY return valid entries when 0xFF is passed. Once all valid entries have been returned by this method, an extra zeroed-out entry is returned to indicate completion.

When the application requests an EmberRipEntry at a certain index, it can check for the validity of the returned EmberRipEntry by checking whether it is zeroed out. For example, the 'type' parameter should never be zero. (it should be a valid node type: EMBER_ROUTER)

6.5.5.28 void emberGetRipEntryReturn (uint8_t *index,* const **EmberRipEntry** * *entry*)

This function provides the result of a call to emberGetRipEntry().

6.5.5.29 void emberGetStandaloneBootloaderInfo (void)

6.5.5.30 void emberGetStandaloneBootloaderInfoReturn (uint16_t *version,* uint8_t *platformId,* uint8_t *microId,* uint8_t *phyId*)

Parameters

version	BOOTLOADER_INVALID_VERSION if the standalone bootloader is not present, or the version of the installed standalone bootloader.
nodePlat	The value of PLAT on the node.
nodeMicro	The value of MICRO on the node.
nodePhy	The value of PHY on the node.

This function provides the result of a call to emberGetStandaloneBootloaderInfo.

Parameters

version	BOOTLOADER_INVALID_VERSION if the standalone bootloader is not present, or the version of the installed standalone bootloader.
platform↩ Id	The value of PLAT on the node.
microId	The value of MICRO on the node.
phyId	The value of PHY on the node.

6.5.5.31 void emberGetTxPowerMode (void)

6.5.5.32 void emberGetTxPowerModeReturn (uint16_t *txPowerMode*)

Returns

the current tx power mode.

This function provides the result of a call to emberGetTxPowerMode() on the host.

Returns

the current tx power mode.

6.5.5.33 void emberGetVersions (void)

6.5.5.34 void emberGetVersionsReturn (const uint8_t * *versionName*, uint16_t *managementVersionNumber*, uint16_t *stackVersionNumber*, uint16_t *stackBuildNumber*, EmberVersionType *versionType*, const uint8_t * *buildTimestamp*)

6.5.5.35 void emberHostStateHandler (const EmberNetworkParameters * *parameters*, const EmberEui64 * *localEui64*, const EmberEui64 * *macExtendedId*, EmberNetworkStatus *networkStatus*)

Parameters

parameters	Current network parameters
localEui64	The EUI64 of the Ember chip
macExtendedId	The extended MAC ID of the Ember chip
networkStatus	The current status of the network

6.5.5.36 void emberInit (void)

6.5.5.37 void emberInitHost (void)

6.5.5.38 void emberInitReturn (EmberStatus *status*)

This function provides the result of a call to emberInit().

6.5.5.39 void emberLaunchStandaloneBootloader (uint8_t *mode*)

Parameters

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.

6.5.5.40 void emberLaunchStandaloneBootloaderReturn (EmberStatus *status*)

Parameters

status	An EmberStatus value indicating success or the reason for failure.

This function provides the result of a call to emberLaunchStandaloneBootloader.

Parameters

status	An EmberStatus value indicating success or the reason for failure.

6.5.5.41 bool emberMacPassthroughFilterHandler (uint8_t * *macHeader*)

Note

This API is for SoCs only.
The application must define EMBER_APPLICATION_HAS_MAC_PASSTHROUGH_FILTER_HANDLER

Parameters

macHeader	A pointer to the initial portion of the incoming MAC header, in the standard 802.15.4 format. The first two bytes comprise the frame control, which dictates source / destination PAN and addressing formats. (See the MAC sublayer definition in the standards definition 802.15.4e/2012)

The relevant bytes of the header are:

| **octets:** | **2** | **1** | **0/2** | **0/2/8** | **0/2** | **0/2/8** |

| | **ctl** | **seq** | **dst.pan** | **dst.addr** | **src.pan** | **src.addr** | **...**

Note that subsequent MAC fields, and the MAC payload, may not yet be present at this point.

Returns

true if the message is an application MAC passthrough message.

6.5.5.42 void emberMacPassthroughMessageHandler (PacketHeader *header*)

Note

This API is for SoCs only.
The application must define EMBER_APPLICATION_HAS_MAC_PASSTHROUGH_MESSAGE_HANDLER

Parameters

header	The message buffer pointing to the full 802.15.4 frame to be handled by the application.

6.5.5.43 bool emberMacRssiFilterHandler (uint8_t * *macHeader*)

Note

This API is for SoCs only.
The application must define EMBER_APPLICATION_HAS_RSSI_FILTER_HANDLER

Parameters

macHeader	A pointer to the initial portion of the incoming MAC header, in the standard 802.15.4 format. The first two bytes comprise the frame control, which dictates source / destination PAN and addressing formats. (See the MAC sublayer definition in the standards definition 802.15.4e/2012)

The relevant bytes of the header are:

| **octets:** | **2** | **1** | **0/2** | **0/2/8** | **0/2** | **0/2/8** |

| | **ctl** | **seq** | **dst.pan** | **dst.addr** | **src.pan** | **src.addr** | **...**

Note that subsequent MAC fields, and the MAC payload, may not yet be present at this point.

Returns

true if the application wants to peek at the RSSI for this message.

6.5.5.44 void emberMacRssiHandler (int8_t *currentRssi*)

Note

This is called on the application for all packets that match the rule defined in emberMacRssiFilterHandler()

The quantity referenced by currentRssi will contain the energy level (in units of dBm) observed during the last 802.15.4 packet received in that handler.

Note

This API is for SoCs only.
The application must define EMBER_APPLICATION_HAS_RSSI_FILTER_HANDLER
This functionality is not available for packets such as 802.15.4 data requests or acknowledgements. Data requests must be handled quickly due to strict 15.4 timing requirements, and so the RSSI information is not recorded. Similarly, 802.15.4 ACKs are handled by the hardware and the information does not make it up to the stack.

Parameters

currentRssi	The RSSI for the last incoming message processed.

6.5.5.45 EmberNetworkStatus emberNetworkStatus (void)

6.5.5.46 void emberNetworkStatusHandler (EmberNetworkStatus *newNetworkStatus,* EmberNetworkStatus *oldNetworkStatus,* EmberJoinFailureReason *reason*)

If the status handler is reporting a join failure, then the newNetworkStatus argument will have a value of EMBER↩_NO_NETWORK and the reason argument will contain an appropriate value. For other network status reports, the reason argument does not apply and is set to EMBER_JOIN_FAILURE_REASON_NONE.

This function reports a change to the network status. For example, the network status changes while going through the joining process, or while reattaching to the network, which can happen for a variety of reasons. In particular, after issuing a form, join, resume, or attach command, the application knows that the device is on the network and ready to communicate when this handler is called with a newNetworkStatus of EMBER_JOINED_NETWORK_A↩TTACHED.

If the status handler is reporting a join failure, then the newNetworkStatus argument will have a value of EMBER↩_NO_NETWORK and the reason argument will contain an appropriate value. For other network status reports, the reason argument does not apply and is set to EMBER_JOIN_FAILURE_REASON_NONE.

6.5.5.47 bool emberOkToNap (void)

There may be tasks expecting incoming messages, in which case the device should periodically wake up and call emberPollForData() in order to receive messages. This function can only be called for sleepy end devices.

6.5.5.48 void emberOkToNapReturn (uint8_t *stateMask*)

The mask EMBER_HIGH_PRIORITY_TASKS defines which tasks are high priority. Devices should not sleep if any high priority tasks are active. Active tasks that are not high priority are waiting for messages to arrive from other devices. If there are active tasks, but no high priority ones, the device may sleep but should periodically wake up and call emberPollForData() in order to receive messages. Parents will hold messages for EMBER_INDIRECT_↩TRANSMISSION_TIMEOUT (in quarter seconds) before discarding them.

Returns

A bitmask of the stack's active tasks.

6.5.5.49 void emberPollForData (void)

This function allows a sleepy end device to query its parent for any pending data.

Sleepy end devices must call this function periodically to maintain contact with their parent. The parent will remove a sleepy end device from its child table if it has not received a poll from it within the last EMBER_SLEEPY_CHIL↩D_POLL_TIMEOUT seconds.

If the sleepy end device has lost contact with its parent, it re-joins then network using another router.

The default values for the timeouts are set in config/ember-configuration-defaults.h, and can be overridden in the application's configuration header.

6.5.5.50 void emberPollForDataReturn (EmberStatus *status*)

Parameters

An	EmberStatus value: • EMBER_SUCCESS - The poll message has been submitted for transmission • EMBER_INVALID_CALL - The node is not a sleepy end device. • EMBER_NOT_JOINED - The node is not part of a network.

This function provides the result of a call to emberPollForData().

Parameters

An	EmberStatus value: • EMBER_SUCCESS - The poll message has been submitted for transmission • EMBER_INVALID_CALL - The node is not a sleepy end device. • EMBER_NOT_JOINED - The node is not part of a network.

6.5.5.51 void emberRegisterDropIncomingMessageCallback (bool(∗)(PacketHeader header, Ipv6Header ∗ipHeader) *drop*)

6.5.5.52 void emberRegisterSerialTransmitCallback (void(∗)(uint8_t type, PacketHeader header) *serialTransmit*)

6.5.5.53 void emberResetMicro (void)

6.5.5.54 void emberResetMicroHandler (EmberResetCause *cause*)

This function notifies the application of a reset on the Ember chip due to the indicated cause.

6.5.5.55 void emberResetNetworkState (void)

6.5.5.56 void emberResetNetworkStateReturn (EmberStatus *status*)

This function provides the result of a call to emberResetNetworkState().

6.5.5.57 void emberScanReturn (EmberStatus *status*)

This function provides the status upon completion of a scan.

6.5.5.58 void emberSetCcaThreshold (int8_t *threshold*)

6.5.5.59 void emberSetCcaThresholdReturn (EmberStatus *status*)

This function provides the result of a call to emberSetCcaThreshold().

6.5.5.60 void emberSetCtune (uint16_t *tune*)

Parameters

tune	Value to set CTUNE to.

6.5.5.61 void emberSetCtuneReturn (EmberStatus *status*)

Parameters

status	An EmberStatus value indicating success or the reason for failure.

This function provides the result of a call to emberSetCtune.

Parameters

status	An EmberStatus value indicating success or the reason for failure.

6.5.5.62 void emberSetMfgToken (EmberMfgTokenId *tokenId*, const uint8_t * *tokenData*, uint8_t *tokenDataLength*)

Parameters

tokenId	Which manufacturing token to set.
tokenData	The manufacturing token data.
tokenDataLength	The length of the *tokenData* parameter in bytes.

6.5.5.63 void emberSetMfgTokenReturn (EmberMfgTokenId *tokenId*, EmberStatus *status*)

Parameters

tokenId	Which manufacturing token set.
status	An EmberStatus value indicating success or the reason for failure.

This function provides the result of a call to emberSetMfgToken.

Parameters

tokenId	Which manufacturing token set.
status	An EmberStatus value indicating success or the reason for failure.

6.5.5.64 void emberSetRadioPower (int8_t *power*)

Parameters

power	Desired radio output power, in dBm.

Returns

An EmberStatus value indicating the success or failure of the command. Failure indicates that the requested power level is out of range.

6.5.5.65 void emberSetRadioPowerReturn (EmberStatus *status*)

This function provides the result of a call to emberSetRadioPower() on the host.

6.5.5.66 void emberSetSecurityParameters (const EmberSecurityParameters * *parameters*, uint16_t *options*)

*** Only the EMBER_NETWORK_KEY_OPTION works at this time. ***

6.5.5.67 void emberSetSecurityParametersReturn (EmberStatus *status*)

This function provides the result of a call to emberSetSecurityParameters().

6.5.5.68 EmberStatus emberSetTxPowerMode (uint16_t *txPowerMode*)

Boost power mode is a high-performance radio mode which offers increased transmit power and receive sensitivity at the cost of an increase in power consumption. The alternate transmit output path allows for simplified connection to an external power amplifier via the RF_TX_ALT_P and RF_TX_ALT_N pins on the em250. emberInit() calls this function using the power mode and transmitter output settings as specified in the MFG_PHY_CONFIG token (with each bit inverted so that the default token value of 0xffff corresponds to normal power mode and bi-directional RF transmitter output). The application only needs to call emberSetTxPowerMode() if it wishes to use a power mode or transmitter output setting different from that specified in the MFG_PHY_CONFIG token. After this initial call to emberSetTxPowerMode(), the stack will automatically maintain the specified power mode configuration across sleep/wake cycles.

Note

This function does not alter the MFG_PHY_CONFIG token. The MFG_PHY_CONFIG token must be properly configured to ensure optimal radio performance when the standalone bootloader runs in recovery mode. The MFG_PHY_CONFIG can only be set using external tools. IF YOUR PRODUCT USES BOOST MODE OR THE ALTERNATE TRANSMITTER OUTPUT AND THE STANDALONE BOOTLOADER YOU MUST SET THE PHY_CONFIG TOKEN INSTEAD OF USING THIS FUNCTION. Contact support@ember.com for instructions on how to set the MFG_PHY_CONFIG token appropriately.

Parameters

txPowerMode	Specifies which of the transmit power mode options are to be activated. This parameter should be set to one of the literal values described in stack/include/ember-types.h. Any power option not specified in the txPowerMode parameter will be deactivated.

Returns

EMBER_SUCCESS if successful; an error code otherwise.

6.5.5.69 void emberSetTxPowerModeReturn (EmberStatus *status*)

This function provides the result of a call to emberSetTxPowerMode() on the host.

6.5.5.70 uint32_t emberStackIdleTimeMs (EmberIdleRadioState * *radioStateResult*)

This function returns directly, rather than having a ...Return() callback, because it is only available on SOCs.

Parameters

radioStateResult	Used to return the required radio state while the stack is idle.

Returns

The number of milliseonds for which the stack will be idle.

6.5.5.71 void emberStackPollForData (uint32_t *pollMs*)

6.5.5.72 void emberStackPollForDataReturn (EmberStatus *status*)

This function provides the result of a call to emberStackPollForData().

6.5.5.73 void emberStackPowerDown (void)

After calling this function, you must not call any other stack function except emberStackPowerUp(). This is because all other stack functions require that the radio is powered on for their proper operation.

Referenced by usbSuspendDsr().

6.5.5.74 void emberStackPowerUp (void)

For non-sleepy devices, also turns the radio on and leaves it in rx mode.

6.5.5.75 void emberStackPrepareForPowerDown (void)

6.5.5.76 bool emberStackPreparingForPowerDown (void)

6.5.5.77 void emberStartScan (EmberNetworkScanType *scanType,* uint32_t *channelMask,* uint8_t *duration*)

Upon completion of the scan, a status is returned via emberScanReturn(). Possible EmberStatus values and their meanings:

- EMBER_SUCCESS, the scan completed succesffully.
- EMBER_MAC_SCANNING, we are already scanning.
- 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 on the current platform.

Parameters

scanType	Indicates the type of scan to be performed. Possible values: EMBER_ENERGY_SCAN, EMBER_ACTIVE_SCAN.
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 0x07 FF F8 00. As a convenience, a channelMask of 0 is reinterpreted as the mask for the current channel.
duration	Sets the exponent of the number of scan periods, where a scan period is 960 symbols, and a symbol is 16 microseconds. The scan will occur for ((2^duration) + 1) scan periods. The value of duration must be less than 15. The time corresponding to the first few values are as follows: 0 = 31 msec, 1 = 46 msec, 2 = 77 msec, 3 = 138 msec, 4 = 261 msec, 5 = 507 msec, 6 = 998 msec.

6.5.5.78 void emberState (void)

6.5.5.79 void emberStateReturn (const EmberNetworkParameters * *parameters*, const EmberEui64 * *localEui64*, const EmberEui64 * *macExtendedId*, EmberNetworkStatus *networkStatus*)

Parameters

parameters	Current network parameters
localEui64	The EUI64 of the Ember chip
macExtendedId	The extended MAC ID of the Ember chip
networkStatus	The current status of the network

6.5.5.80 void emberStopScan (void)

6.5.5.81 void emberSwitchToNextNetworkKey (void)

6.5.5.82 void emberSwitchToNextNetworkKeyHandler (EmberStatus *status*)

This function can be stubbed out on the SoC and host app. It is used by the NCP to update security on the driver when it is instructed to switch the network key by an over the air update.

6.5.5.83 void emberSwitchToNextNetworkKeyReturn (EmberStatus *status*)

This function provides the result of a call to emberSwitchToNextNetworkKey().

6.5.5.84 void emberTick (void)

6.6 Device Types

Device Types.

Enumerations

- enum EmberNodeType {
  EMBER_UNKNOWN_DEVICE = 0,
  EMBER_ROUTER = 2,
  EMBER_END_DEVICE = 3,
  EMBER_SLEEPY_END_DEVICE = 4,
  EMBER_MINIMAL_END_DEVICE = 5,
  EMBER_COMMISSIONER = 7 }

6.6.1 Detailed Description

Defines the possible types of nodes and the roles that a node might play in a network.

6.6.2 Enumeration Type Documentation

6.6.2.1 enum EmberNodeType

Enumerator

EMBER_UNKNOWN_DEVICE Device is not joined.

EMBER_ROUTER Will relay messages and can act as a parent to other nodes.

EMBER_END_DEVICE Communicates only with its parent and will not relay messages.

EMBER_SLEEPY_END_DEVICE An end device whose radio can be turned off to save power. The application must call emberPollForData() to receive messages.

EMBER_MINIMAL_END_DEVICE An always-on end device like EMBER_END_DEVICE, but IP address discovery is performed by the parent on its behalf to help it conserve resources.

EMBER_COMMISSIONER Authentication server for new Thread devices and the authorizer for providing the network credentials they require joining the network.

6.7 Utilities

General Utilities.

Modules

- AES crypto routines
- Device Types

  Device Types.

Data Structures

- struct EmberEui64

  EUI 64-bit ID (an IEEE address).
- struct EmberIpv6Prefix

  An IPv6 Prefix structure.
- struct EmberIpv6Address

  An IPv6 Address structure.
- struct EmberKeyData

  This data structure contains the key data that is passed into various other functions.
- struct EmberVersion

  For use when declaring data that holds the Ember software version type.
- struct Ipv6Header

  A structure that holds an IPv6 header. All values are in their local byte order (as opposed to network byte order, which might be different).
- struct TlsSessionState

  Defines a TLS session state.
- struct Bytes8

  Defines a data type of size 8 bytes.
- struct Bytes16

  Defines a data type of size 16 bytes.
- struct CertificateAuthority

  Defines a certificate authority structure.
- struct DeviceCertificate

  Defines a device certificate structure.
- struct EventActions_s

  The static part of an event. Each event can be used with only one event queue.
- struct Event_s
- struct EventQueue_s

  An event queue is currently a list of events ordered by execution time.
- struct EmberEventControl

  Control structure for events.
- struct EmberTaskControl

  Control structure for tasks.

Macros

- #define EMBER_VERSION_NAME "Thread"

  If the application defined a configuration file, include it.
- #define EMBER_HEAP_SIZE 6000

  The minimum heap size allocated for an application.
- #define EMBER_MALLOC_HEAP_SIZE_BYTES 32768

  The default amount of heap allocated for the mbedtls malloc library, if in use.
- #define EMBER_ASSERT_SERIAL_PORT 1

  Settings to control if and where assert information will be printed.
- #define EMBER_INDIRECT_TRANSMISSION_TIMEOUT 30

  The maximum amount of time (in quarter seconds) that the MAC will hold a message for indirect transmission to a child.
- #define EMBER_CHILD_TABLE_SIZE 16

  The size of the child table. This include sleepy and powered end device children, as well as router eligible end devices.
- #define EMBER_RETRY_QUEUE_SIZE 8
- #define EMBER_SECURITY_LEVEL 5

  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).
- #define EMBER_SECURITY_TO_HOST false
- #define EMBER_TASK_COUNT (3)

  The number of event tasks that can be tracked for the purpose of processor idling. The Thread stack requires 1, an application and associated libraries may use additional tasks, though typically no more than 3 are needed for most applications.
- #define EMBER_SLEEPY_CHILD_POLL_TIMEOUT 240

  The number of seconds after which the parent will time an EMBER_SLEEPY_END_DEVICE out of its table if it has not heard a data poll from it.
- #define EMBER_END_DEVICE_POLL_TIMEOUT 240

  The maximum amount of time that an EMBER_END_DEVICE can wait between polls.
- #define EMBER_MFG_RX_NCP_TO_HOST_INTERVAL 50

  The number of packets received by an NCP before it decides to send aggregated packet information to the host when running an mfg send test.
- #define EMBER_USE_DIRECT_IP_CALLBACK false
- #define RIP_MAX_LURKERS 0
- #define INT16U_MAX ((uint16_t)(~(uint16_t)0))

  Defines the maximum value of an unsigned short data type.
- #define DEFAULT_SCAN_DURATION 5

  Default scan duration for an energy or active scan.
- #define EMBER_COUNTER_STRINGS

  Defines the CLI enumerations for the EmberCounterType enum.

Typedefs

- typedef uint8_t EmberTaskId
- typedef const struct EventActions_s EventActions

  The static part of an event. Each event can be used with only one event queue.
- typedef struct Event_s Event
- typedef struct EventQueue_s EventQueue

  An event queue is currently a list of events ordered by execution time.
- struct {
  EmberEventControl * control
  void(* handler)(void)
  } EmberEventData

  Complete events with a control and a handler procedure.

Enumerations

- enum EmberNetworkStatus {
  EMBER_NO_NETWORK,
  EMBER_SAVED_NETWORK,
  EMBER_JOINING_NETWORK,
  EMBER_JOINED_NETWORK_ATTACHED,
  EMBER_JOINED_NETWORK_NO_PARENT,
  EMBER_JOINED_NETWORK_ATTACHING }

  Defines the possible join states for a node.

- enum EmberJoinFailureReason {
  EMBER_JOIN_FAILURE_REASON_NONE,
  EMBER_JOIN_FAILURE_REASON_FORM_SCAN,
  EMBER_JOIN_FAILURE_REASON_ACTIVE_SCAN,
  EMBER_JOIN_FAILURE_REASON_COMMISSIONING,
  EMBER_JOIN_FAILURE_REASON_SECURITY }

  Defines the reason why a network status change occurred.

- enum EmberNetworkScanType {
  EMBER_ENERGY_SCAN,
  EMBER_ACTIVE_SCAN }

  Type for a network scan.

- enum EmberEventUnits {
  EMBER_EVENT_INACTIVE = 0,
  EMBER_EVENT_MS_TIME,
  EMBER_EVENT_QS_TIME,
  EMBER_EVENT_MINUTE_TIME,
  EMBER_EVENT_ZERO_DELAY }

  Either marks an event as inactive or specifies the units for the event execution time.

- enum EmberCounterType {

EMBER_COUNTER_PHY_IN_PACKETS,
EMBER_COUNTER_PHY_OUT_PACKETS,
EMBER_COUNTER_PHY_IN_OCTETS,
EMBER_COUNTER_PHY_OUT_OCTETS,
EMBER_COUNTER_MAC_IN_UNICAST,
EMBER_COUNTER_MAC_IN_BROADCAST,
EMBER_COUNTER_MAC_OUT_UNICAST_SUCCESS,
EMBER_COUNTER_MAC_OUT_UNICAST_ACK_FAIL,
EMBER_COUNTER_MAC_OUT_UNICAST_CCA_FAIL,
EMBER_COUNTER_MAC_OUT_UNICAST_EXT_FAIL,
EMBER_COUNTER_MAC_OUT_UNICAST_RETRY,
EMBER_COUNTER_MAC_OUT_BROADCAST,
EMBER_COUNTER_MAC_OUT_BROADCAST_CCA_FAIL,
EMBER_COUNTER_MAC_DROP_IN_MEMORY,
EMBER_COUNTER_MAC_DROP_IN_NO_EUI,
EMBER_COUNTER_MAC_DROP_IN_FRAME_COUNTER,
EMBER_COUNTER_MAC_DROP_IN_DECRYPT,
EMBER_COUNTER_MAC_DROP_IN_DUPLICATE,
EMBER_COUNTER_IP_IN_UNICAST,
EMBER_COUNTER_IP_OUT_UNICAST,
EMBER_COUNTER_IP_IN_MULTICAST,
EMBER_COUNTER_IP_OUT_MULTICAST,
EMBER_COUNTER_UDP_IN,
EMBER_COUNTER_UDP_OUT,
EMBER_COUNTER_UART_IN_DATA,
EMBER_COUNTER_UART_IN_MANAGEMENT,
EMBER_COUNTER_UART_IN_FAIL,
EMBER_COUNTER_UART_OUT_DATA,
EMBER_COUNTER_UART_OUT_MANAGEMENT,
EMBER_COUNTER_UART_OUT_FAIL,
EMBER_COUNTER_ROUTE_2_HOP_LOOP,
EMBER_COUNTER_BUFFER_ALLOCATION_FAIL,
EMBER_ASH_V3_ACK_SENT,
EMBER_ASH_V3_ACK_RECEIVED,
EMBER_ASH_V3_NACK_SENT,
EMBER_ASH_V3_NACK_RECEIVED,
EMBER_ASH_V3_RESEND,
EMBER_ASH_V3_BYTES_SENT,
EMBER_ASH_V3_TOTAL_BYTES_RECEIVED,
EMBER_ASH_V3_VALID_BYTES_RECEIVED,
EMBER_ASH_V3_PAYLOAD_BYTES_SENT,
EMBER_COUNTER_PTA_LO_PRI_REQUESTED,
EMBER_COUNTER_PTA_HI_PRI_REQUESTED,
EMBER_COUNTER_PTA_LO_PRI_DENIED,
EMBER_COUNTER_PTA_HI_PRI_DENIED,
EMBER_COUNTER_PTA_LO_PRI_TX_ABORTED,
EMBER_COUNTER_PTA_HI_PRI_TX_ABORTED,
EMBER_COUNTER_TYPE_COUNT,
EMBER_COUNTER_ALL = 0xFF }

Defines the events reported to the application by the emberCounterHandler().

Functions

- EmberStatus emberSetRadioChannel (uint8_t channel)

  This function sets the channel 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.
- uint8_t emberGetRadioChannel (void)

This function gets the radio channel to which a node is set. The possible return values depend on the radio in use. For a list of available radio channels, see the technical specification for the RF communication module in your Developer Kit.

Variables

- uint8_t EmberEui64::bytes [EUI64_SIZE]
- uint8_t EmberIpv6Prefix::bytes [8]
- uint8_t EmberIpv6Address::bytes [16]
- uint8_t EmberKeyData::contents [EMBER_ENCRYPTION_KEY_SIZE]
- uint8_t EmberVersion::major
- uint8_t EmberVersion::minor
- uint8_t EmberVersion::patch
- EmberVersionType EmberVersion::type
- uint16_t EmberVersion::build
- uint32_t EmberVersion::change
- uint16_t Ipv6Header::ipPayloadLength
- uint32_t Ipv6Header::flowLabel
- uint8_t Ipv6Header::trafficClass
- uint8_t Ipv6Header::nextHeader
- uint8_t Ipv6Header::hopLimit
- uint8_t Ipv6Header::source [16]
- uint8_t Ipv6Header::destination [16]
- uint8_t * Ipv6Header::ipPayload
- uint8_t Ipv6Header::transportProtocol
- uint8_t * Ipv6Header::transportHeader
- uint16_t Ipv6Header::transportHeaderLength
- uint8_t * Ipv6Header::transportPayload
- uint16_t Ipv6Header::transportPayloadLength
- uint16_t Ipv6Header::sourcePort
- uint16_t Ipv6Header::destinationPort
- uint8_t Ipv6Header::icmpType
- uint8_t Ipv6Header::icmpCode
- uint8_t TlsSessionState::idLength
- uint16_t TlsSessionState::id [(TLS_SESSION_ID_SIZE+1)/2]
- uint8_t TlsSessionState::master [TLS_MASTER_SECRET_SIZE]
- uint8_t Bytes8::contents [8]
- uint8_t Bytes16::contents [16]
- const uint8_t * CertificateAuthority::name
- uint16_t CertificateAuthority::nameLength
- uint8_t * CertificateAuthority::publicKey
- uint8_t CertificateAuthority::maxPathLength
- const uint8_t * DeviceCertificate::privateKey
- const uint8_t * DeviceCertificate::certificate
- const uint16_t DeviceCertificate::certificateSize
- struct EventQueue_s * EventActions_s::queue
- void(* EventActions_s::handler)(struct Event_s *)
- void(* EventActions_s::marker)(struct Event_s *)
- const char * EventActions_s::name
- EventActions * Event_s::actions
- struct Event_s * Event_s::next
- uint32_t Event_s::timeToExecute
- Event * EventQueue_s::isrEvents
- Event * EventQueue_s::events

- uint32_t EventQueue_s::runTime
- bool EventQueue_s::running
- EmberEventUnits EmberEventControl::status
- EmberTaskId EmberEventControl::taskid
- uint32_t EmberEventControl::timeToExecute
- EmberEventControl * control
- void(* handler)(void)
- uint32_t EmberTaskControl::nextEventTime
- EmberEventData * EmberTaskControl::events
- bool EmberTaskControl::busy

Miscellaneous Ember Types

- enum EmberVersionType {
  EMBER_VERSION_TYPE_INTERNAL = 0,
  EMBER_VERSION_TYPE_ALPHA = 1,
  EMBER_VERSION_TYPE_BETA = 2,
  EMBER_VERSION_TYPE_GA = 3,
  EMBER_VERSION_TYPE_SPECIAL = 4,
  EMBER_VERSION_TYPE_LEGACY = 5 }

  Type of Ember software version.
- enum EmberIcmpType {
  ICMP_DESTINATION_UNREACHABLE = 1,
  ICMP_PACKET_TOO_BIG = 2,
  ICMP_TIME_EXCEEDED = 3,
  ICMP_PARAMETER_PROBLEM = 4,
  ICMP_PRIVATE_EXPERIMENTATION_0 = 100,
  ICMP_ECHO_REQUEST = 128,
  ICMP_ECHO_REPLY = 129,
  ICMP_ROUTER_SOLICITATION = 133,
  ICMP_ROUTER_ADVERTISEMENT = 134,
  ICMP_NEIGHBOR_SOLICITATION = 135,
  ICMP_NEIGHBOR_ADVERTISEMENT = 136,
  ICMP_RPL = 155,
  ICMP_DUPLICATE_ADDRESS_REQUEST = 157,
  ICMP_DUPLICATE_ADDRESS_CONFIRM = 158 }

  Definitions for ICMP message types.
- enum EmberIcmpCode {
  ICMP_CODE_NO_ROUTE_TO_DESTINATION = 0,
  ICMP_CODE_ERROR_IN_SOURCE_ROUTING_HEADER = 7 }

  Definitions for ICMP message codes.
- enum EmberIpv6NextHeader {
  IPV6_NEXT_HEADER_ICMP = 1,
  IPV6_NEXT_HEADER_TCP = 6,
  IPV6_NEXT_HEADER_UDP = 17,
  IPV6_NEXT_HEADER_IPV6 = 41,
  IPV6_NEXT_HEADER_ICMPV6 = 58,
  IPV6_NEXT_HEADER_NO_NEXT = 59,
  IPV6_NEXT_HEADER_MOBILITY = 137,
  IPV6_NEXT_HEADER_HOP_BY_HOP = 0,
  IPV6_NEXT_HEADER_DESTINATION = 60,
  IPV6_NEXT_HEADER_ROUTING = 43,
  IPV6_NEXT_HEADER_FRAGMENT = 44,
  IPV6_NEXT_HEADER_UNKNOWN = 0xFF }

  Structure to hold an IPv6 "Next Header" See http://www.iana.org/assignments/protocol-numbers.

- typedef uint8_t EmberStatus

  Size of EUI64 (an IEEE address) in bytes (8).
- typedef uint8_t EmberEUI64[EUI64_SIZE]

  Obsolete version of EUI64 structure used by some platform-dependent applications. Use EmberEui64.
- typedef uint16_t EmberNodeId

  16-bit 802.15.4 network address.
- typedef uint16_t EmberPanId

  802.15.4 PAN ID.
- typedef uint16_t Buffer

  For use when declaring a Buffer.
- typedef uint16_t EmberMessageBuffer

  For use when declaring a buffer to hold a message.
- typedef Buffer PacketHeader

  For use when declaring a buffer to hold a packet header.
- typedef uint16_t ChildStatusFlags

  For use when declaring data that holds child status flags.
- #define EUI64_SIZE 8

  Size of EUI64 (an IEEE address) in bytes (8).
- #define EMBER_ENCRYPTION_KEY_SIZE 16

  Size of an encryption key in bytes (16).
- #define EXTENDED_PAN_ID_SIZE 8

  Size of an extended PAN identifier in bytes (8).
- #define LEADER_SIZE EUI64_SIZE

  Size of a leader EUI64 in bytes (8).
- #define EMBER_NETWORK_ID_SIZE 16

  Size of a network ID in bytes (16).
- #define EMBER_JOIN_KEY_MAX_SIZE 32

  Maximum size of a device join key (PSKd) in bytes (32).
- #define __EMBERSTATUS_TYPE__

  Return type for Ember functions.
- #define EMBER_MAX_802_15_4_CHANNEL_NUMBER 26

  The maximum 802.15.4 channel number is 26.
- #define EMBER_MIN_802_15_4_CHANNEL_NUMBER 11

  The minimum 802.15.4 channel number is 11.
- #define EMBER_NUM_802_15_4_CHANNELS (EMBER_MAX_802_15_4_CHANNEL_NUMBER - EMBE↩ R_MIN_802_15_4_CHANNEL_NUMBER + 1)

  There are sixteen 802.15.4 channels.
- #define EMBER_ALL_802_15_4_CHANNELS_MASK 0x07FFF800UL

  Bitmask to scan all 802.15.4 channels.
- #define EMBER_ZIGBEE_COORDINATOR_ADDRESS 0x0000

  The network ID of the coordinator in a ZigBee network is 0x0000.
- #define EMBER_NULL_NODE_ID 0xFFFF

  A distinguished network ID that will never be assigned to any node. Used to indicate the absence of a node ID.
- #define EMBER_VERSION_TYPE_MAX EMBER_VERSION_TYPE_LEGACY

  Size of EUI64 (an IEEE address) in bytes (8).
- #define EMBER_VERSION_TYPE_NAMES

  Size of EUI64 (an IEEE address) in bytes (8).
- #define NULL_BUFFER 0x0000

  Denotes a null buffer.
- #define TLS_SESSION_ID_SIZE 32

  Size of EUI64 (an IEEE address) in bytes (8).
- #define TLS_MASTER_SECRET_SIZE 48

  Size of EUI64 (an IEEE address) in bytes (8).

Broadcast Addresses

Broadcasts are normally sent only to routers. Broadcasts can also be forwarded to end devices, either all of them or only those that do not sleep. Broadcasting to end devices is both significantly more resource-intensive and significantly less reliable than broadcasting to routers.

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

txPowerModes for emberSetTxPowerMode and mfglibSetPower

- #define EMBER_TX_POWER_MODE_DEFAULT 0x0000

  The application should call emberSetTxPowerMode() with the txPowerMode parameter set to this value to disable all power mode options resulting in normal power mode and bi-directional RF transmitter output.
- #define EMBER_TX_POWER_MODE_BOOST 0x0001

  The application should call emberSetTxPowerMode() with the txPowerMode parameter set to this value to enable boost power mode.
- #define EMBER_TX_POWER_MODE_ALTERNATE 0x0002

  The application should call emberSetTxPowerMode() with the txPowerMode parameter set to this value to enable the alternate transmitter output.
- #define EMBER_TX_POWER_MODE_BOOST_AND_ALTERNATE

  The application should call emberSetTxPowerMode() with the txPowerMode parameter set to this value to enable both boost mode and the alternate transmitter output.

Generic Messages

These messages are system wide.

- #define EMBER_SUCCESS(x00)

  The generic "no error" message.
- #define EMBER_ERR_FATAL(x01)

  The generic "fatal error" message.
- #define EMBER_BAD_ARGUMENT(x02)

  An invalid value was passed as an argument to a function.
- #define EMBER_EEPROM_MFG_STACK_VERSION_MISMATCH(x04)

  The manufacturing and stack token format in non-volatile memory is different than what the stack expects (returned at initialization).
- #define EMBER_INCOMPATIBLE_STATIC_MEMORY_DEFINITIONS(x05)

  The static memory definitions in ember-static-memory.h are incompatible with this stack version.
- #define EMBER_EEPROM_MFG_VERSION_MISMATCH(x06)

  The manufacturing token format in non-volatile memory is different than what the stack expects (returned at initialization).
- #define EMBER_EEPROM_STACK_VERSION_MISMATCH(x07)

  The stack token format in non-volatile memory is different than what the stack expects (returned at initialization).

Packet Buffer Module Errors

- #define EMBER_NO_BUFFERS(x18)

  There are no more buffers.

Serial Manager Errors

- #define EMBER_SERIAL_INVALID_BAUD_RATE(x20)

  Specified an invalid baud rate.
- #define EMBER_SERIAL_INVALID_PORT(x21)

  Specified an invalid serial port.
- #define EMBER_SERIAL_TX_OVERFLOW(x22)

  Tried to send too much data.
- #define EMBER_SERIAL_RX_OVERFLOW(x23)

  There was not enough space to store a received character and the character was dropped.
- #define EMBER_SERIAL_RX_FRAME_ERROR(x24)

  Detected a UART framing error.
- #define EMBER_SERIAL_RX_PARITY_ERROR(x25)

  Detected a UART parity error.
- #define EMBER_SERIAL_RX_EMPTY(x26)

  There is no received data to process.
- #define EMBER_SERIAL_RX_OVERRUN_ERROR(x27)

  The receive interrupt was not handled in time, and a character was dropped.

MAC Errors

- #define EMBER_MAC_TRANSMIT_QUEUE_FULL(x39)

  The MAC transmit queue is full.
- #define EMBER_MAC_UNKNOWN_HEADER_TYPE(x3A)

  MAC header FCF error on receive.
- #define EMBER_MAC_ACK_HEADER_TYPE(x3B)

  MAC ACK header received.
- #define EMBER_MAC_SCANNING(x3D)

  The MAC can't complete this task because it is scanning.
- #define EMBER_MAC_NO_DATA(x31)

  No pending data exists for device doing a data poll.
- #define EMBER_MAC_JOINED_NETWORK(x32)

  Attempt to scan when we are joined to a network.
- #define EMBER_MAC_BAD_SCAN_DURATION(x33)

  Scan duration must be 0 to 14 inclusive. Attempt was made to scan with an incorrect duration value.
- #define EMBER_MAC_INCORRECT_SCAN_TYPE(x34)

  emberStartScan was called with an incorrect scan type.
- #define EMBER_MAC_INVALID_CHANNEL_MASK(x35)

  emberStartScan was called with an invalid channel mask.
- #define EMBER_MAC_COMMAND_TRANSMIT_FAILURE(x36)

  Failed to scan current channel because we were unable to transmit the relevent MAC command.
- #define EMBER_MAC_NO_ACK_RECEIVED(x40)

  We expected to receive an ACK following the transmission, but the MAC level ACK was never received.
- #define EMBER_MAC_INDIRECT_TIMEOUT(x42)

  Indirect data message timed out before polled.

Simulated EEPROM Errors

- #define EMBER_SIM_EEPROM_ERASE_PAGE_GREEN(x43)

  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.
- #define EMBER_SIM_EEPROM_ERASE_PAGE_RED(x44)

  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.
- #define EMBER_SIM_EEPROM_FULL(x45)

  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.
- #define EMBER_SIM_EEPROM_INIT_1_FAILED(x48)

  Attempt 1 to initialize the Simulated EEPROM has failed.
- #define EMBER_SIM_EEPROM_INIT_2_FAILED(x49)

  Attempt 2 to initialize the Simulated EEPROM has failed.
- #define EMBER_SIM_EEPROM_INIT_3_FAILED(x4A)

  Attempt 3 to initialize the Simulated EEPROM has failed.
- #define EMBER_SIM_EEPROM_REPAIRING(x4D)

  The Simulated EEPROM is repairing itself.

Flash Errors

- #define EMBER_ERR_FLASH_WRITE_INHIBITED(x46)

  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.
- #define EMBER_ERR_FLASH_VERIFY_FAILED(x47)

  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 cycles of the flash.
- #define EMBER_ERR_FLASH_PROG_FAIL(x4B)
- #define EMBER_ERR_FLASH_ERASE_FAIL(x4C)

Bootloader Errors

- #define EMBER_ERR_BOOTLOADER_TRAP_TABLE_BAD(x58)

  The bootloader received an invalid message (failed attempt to go into bootloader).
- #define EMBER_ERR_BOOTLOADER_TRAP_UNKNOWN(x59)

  Bootloader received an invalid message (failed attempt to go into bootloader).
- #define EMBER_ERR_BOOTLOADER_NO_IMAGE(x05A)

  The bootloader cannot complete the bootload operation because either an image was not found or the image exceeded memory bounds.

Transport Errors

- #define EMBER_DELIVERY_FAILED(x66)

  The APS layer attempted to send or deliver a message, but it failed.
- #define EMBER_BINDING_INDEX_OUT_OF_RANGE(x69)

  This binding index is out of range for the current binding table.
- #define EMBER_ADDRESS_TABLE_INDEX_OUT_OF_RANGE(x6A)

  This address table index is out of range for the current address table.

- #define EMBER_INVALID_BINDING_INDEX(x6C)

  An invalid binding table index was given to a function.
- #define EMBER_INVALID_CALL(x70)

  The API call is not allowed given the current state of the stack.
- #define EMBER_COST_NOT_KNOWN(x71)

  The link cost to a node is not known.
- #define EMBER_MAX_MESSAGE_LIMIT_REACHED(x72)

  The maximum number of in-flight messages (i.e. ::EMBER_APS_UNICAST_MESSAGE_COUNT) has been reached.
- #define EMBER_MESSAGE_TOO_LONG(x74)

  The message to be transmitted is too big to fit into a single over-the-air packet.
- #define EMBER_BINDING_IS_ACTIVE(x75)

  The application is trying to delete or overwrite a binding that is in use.
- #define EMBER_ADDRESS_TABLE_ENTRY_IS_ACTIVE(x76)

  The application is trying to overwrite an address table entry that is in use.

HAL Module Errors

- #define EMBER_ADC_CONVERSION_DONE(x80)

  Conversion is complete.
- #define EMBER_ADC_CONVERSION_BUSY(x81)

  Conversion cannot be done because a request is being processed.
- #define EMBER_ADC_CONVERSION_DEFERRED(x82)

  Conversion is deferred until the current request has been processed.
- #define EMBER_ADC_NO_CONVERSION_PENDING(x84)

  No results are pending.
- #define EMBER_SLEEP_INTERRUPTED(x85)

  Sleeping (for a duration) has been abnormally interrupted and exited prematurely.

PHY Errors

- #define EMBER_PHY_TX_UNDERFLOW(x88)

  The transmit hardware buffer underflowed.
- #define EMBER_PHY_TX_INCOMPLETE(x89)

  The transmit hardware did not finish transmitting a packet.
- #define EMBER_PHY_INVALID_CHANNEL(x8A)

  An unsupported channel setting was specified.
- #define EMBER_PHY_INVALID_POWER(x8B)

  An unsupported power setting was specified.
- #define EMBER_PHY_TX_BUSY(x8C)

  The requested operation cannot be completed because the radio is currently busy, either transmitting a packet or performing calibration.
- #define EMBER_PHY_TX_CCA_FAIL(x8D)

  The transmit attempt failed because all CCA attempts indicated that the channel was busy.
- #define EMBER_PHY_OSCILLATOR_CHECK_FAILED(x8E)

  The software installed on the hardware doesn't recognize the hardware radio type.
- #define EMBER_PHY_ACK_RECEIVED(x8F)

  The expected ACK was received after the last transmission.

Return Codes Passed to emberStackStatusHandler()

See also ::emberStackStatusHandler().

- #define EMBER_NETWORK_UP(x90)

  The stack software has completed initialization and is ready to send and receive packets over the air.
- #define EMBER_NETWORK_DOWN(x91)

  The network is not operating.
- #define EMBER_JOIN_FAILED(x94)

  An attempt to join a network failed.
- #define EMBER_MOVE_FAILED(x96)

  After moving, a mobile node's attempt to re-establish contact with the network failed.
- #define EMBER_CANNOT_JOIN_AS_ROUTER(x98)

  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.
- #define EMBER_NODE_ID_CHANGED(x99)

  The local node ID has changed. The application can obtain the new node ID by calling ::emberGetNodeId().
- #define EMBER_PAN_ID_CHANGED(x9A)

  The local PAN ID has changed. The application can obtain the new PAN ID by calling emberGetPanId().
- #define EMBER_CHANNEL_CHANGED(x9B)

  The channel has changed.
- #define EMBER_NO_BEACONS(xAB)

  An attempt to join or rejoin the network failed because no router beacons could be heard by the joining node.
- #define EMBER_RECEIVED_KEY_IN_THE_CLEAR(xAC)

  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_REQUIRE_ENCRYPTED_KEY).
- #define EMBER_NO_NETWORK_KEY_RECEIVED(xAD)

  An attempt was made to join a Secured Network, but the device did not receive a Network Key.
- #define EMBER_NO_LINK_KEY_RECEIVED(xAE)

  After a device joined a Secured Network, a Link Key was requested (::EMBER_GET_LINK_KEY_WHEN_JOINING) but no response was ever received.
- #define EMBER_PRECONFIGURED_KEY_REQUIRED(xAF)

  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.

Security Errors

- #define EMBER_KEY_INVALID(xB2)

  The passed key data is not valid. A key of all zeros or all F's are reserved values and cannot be used.
- #define EMBER_INVALID_SECURITY_LEVEL(x95)

  The chosen security level (the value of EMBER_SECURITY_LEVEL) is not supported by the stack.
- #define EMBER_APS_ENCRYPTION_ERROR(xA6)

  There was an error in trying to encrypt at the APS Level.
- #define EMBER_TRUST_CENTER_MASTER_KEY_NOT_SET(xA7)

  There was an attempt to form a network using High security without setting the Trust Center master key first.
- #define EMBER_SECURITY_STATE_NOT_SET(xA8)

  There was an attempt to form or join a network with security without calling ::emberSetInitialSecurityState() first.
- #define EMBER_KEY_TABLE_INVALID_ADDRESS(xB3)

  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 not valid addresses in 802.15.4.

- #define EMBER_SECURITY_CONFIGURATION_INVALID(xB7)

  There was an attempt to set a security configuration that is not valid given the other security settings.
- #define EMBER_TOO_SOON_FOR_SWITCH_KEY(xB8)

  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.
- #define EMBER_SIGNATURE_VERIFY_FAILURE(xB9)

  The received signature corresponding to the message that was passed to the CBKE Library failed verification, it is not valid.
- #define EMBER_KEY_NOT_AUTHORIZED(xBB)

  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).
- #define EMBER_MAC_COUNTER_ERROR(xDB)

  MAC encryption failed.
- #define EMBER_SECURITY_DATA_INVALID(xBD)

  The security data provided was not valid, or an integrity check failed.

Miscellaneous Network Errors

- #define EMBER_NOT_JOINED(x93)

  The node has not joined a network.
- #define EMBER_NETWORK_BUSY(xA1)

  A message cannot be sent because the network is currently overloaded.
- #define EMBER_INVALID_ENDPOINT(xA3)

  The application tried to send a message using an endpoint that it has not defined.
- #define EMBER_BINDING_HAS_CHANGED(xA4)

  The application tried to use a binding that has been remotely modified and the change has not yet been reported to the application.
- #define EMBER_INSUFFICIENT_RANDOM_DATA(xA5)

  An attempt to generate random bytes failed because of insufficient random data from the radio.
- #define EMBER_ROUTE_FAILURE(xA9)

  A route could not be found.
- #define EMBER_MANY_TO_ONE_ROUTE_FAILURE(xAA)

Miscellaneous Utility Errors

- #define EMBER_STACK_AND_HARDWARE_MISMATCH(xB0)

  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.
- #define EMBER_INDEX_OUT_OF_RANGE(xB1)

  An index was passed into the function that was larger than the valid range.
- #define EMBER_TABLE_FULL(xB4)

  There are no empty entries left in the table.
- #define EMBER_TABLE_ENTRY_ERASED(xB6)

  The requested table entry has been erased and contains no valid data.
- #define EMBER_LIBRARY_NOT_PRESENT(xB5)

  The requested function cannot be executed because the library that contains the necessary functionality is not present.
- #define EMBER_OPERATION_IN_PROGRESS(xBA)

  The stack accepted the command and is currently processing the request. The results will be returned via an appropriate handler.
- #define EMBER_TRUST_CENTER_EUI_HAS_CHANGED(xBC)

  The EUI of the Trust center has changed due to a successful rejoin. The device may need to perform other authentication to verify the new TC is authorized to take over.

Application Errors

These error codes are available for application use.

- #define EMBER_APPLICATION_ERROR_0(xF0)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_1(xF1)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_2(xF2)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_3(xF3)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_4(xF4)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_5(xF5)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_6(xF6)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_7(xF7)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_8(xF8)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_9(xF9)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_10(xFA)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_11(xFB)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_12(xFC)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_13(xFD)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_14(xFE)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_15(xFF)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.

Radio-specific Functions

- void emberRadioNeedsCalibratingHandler (void)

  This function enables boost power mode and/or the alternate transmit path.
- void emberCalibrateCurrentChannel (void)

  This function calibrates the current channel. The stack will notify the application of the need for channel calibration via the emberRadioNeedsCalibratingHandler() callback function during emberTick(). This function should only be called from within the context of the emberRadioNeedsCalibratingHandler() callback function. Calibration can take up to 150 ms. Note, if this function is called when the radio is off, it will turn the radio on and leave it on.

6.7.1 Detailed Description

All configurations have defaults, therefore many applications may not need to do anything special. However, you can override these defaults by creating a CONFIGURATION_HEADER and within this header, defining the appropriate macro to a different size. For example, to increase the child table size from 16 (the default) to 32:

```
1 #define EMBER_CHILD_TABLE_SIZE 32
```

The convenience stubs provided in `hal/ember-configuration.c` can be overridden by defining the appropriate macro and providing the corresponding callback function. For example, an application with custom debug channel input must implement `emberDebugHandler()` to process it. Along with the function definition, the application should provide the following line in its CONFIGURATION_HEADER:

```
1 #define EMBER_APPLICATION_HAS_DEBUG_HANDLER
```

See ember-configuration-defaults.h for source code.

See ember-types.h for source code.

Many Thread API functions return an EmberStatus value to indicate the success or failure of the call. Return codes are one byte long. This page documents the possible status codes and their meanings.

See error-def.h for source code.

See also error.h for information on how the values for the return codes are built up from these definitions. The file error-def.h is separated from error.h because utilities will use this file to parse the return codes.

Note

Do not include error-def.h directly. It is included by error.h inside an enum typedef, which is in turn included by ember.h.

See stack-info.h for source code.

6.7.2 Macro Definition Documentation

6.7.2.1 #define __EMBERSTATUS_TYPE__

6.7.2.2 #define DEFAULT_SCAN_DURATION 5

The value is the exponent of the number of scan periods, where a scan period is 960 symbols and a symbol is 16 microseconds. The scan lasts for ((2^duration) + 1) scan periods. The value of this duration must be less than 15. The time corresponding to the first few values is as follows: 0 = 31 msec, 1 = 46 msec, 2 = 77 msec, 3 = 138 msec, 4 = 261 msec, 5 = 507 msec, 6 = 998 msec.

6.7.2.3 #define EMBER_ADC_CONVERSION_BUSY(*x81*)

6.7.2.4 #define EMBER_ADC_CONVERSION_DEFERRED(*x82*)

6.7.2.5 #define EMBER_ADC_CONVERSION_DONE(*x80*)

6.7.2.6 #define EMBER_ADC_NO_CONVERSION_PENDING(*x84*)

6.7.2.7 #define EMBER_ADDRESS_TABLE_ENTRY_IS_ACTIVE(*x76*)

6.7.2.8 #define EMBER_ADDRESS_TABLE_INDEX_OUT_OF_RANGE(*x6A*)

6.7.2.9 #define EMBER_ALL_802_15_4_CHANNELS_MASK 0x07FFF800UL

6.7.2.10 #define EMBER_APPLICATION_ERROR_0(*xF0*)

6.7.2.11 #define EMBER_APPLICATION_ERROR_1(*xF1*)

6.7.2.12 #define EMBER_APPLICATION_ERROR_10(*xFA*)

6.7.2.13 #define EMBER_APPLICATION_ERROR_11(*xFB*)

6.7.2.14 #define EMBER_APPLICATION_ERROR_12(*xFC*)

6.7.2.15 #define EMBER_APPLICATION_ERROR_13(*xFD*)

6.7.2.16 #define EMBER_APPLICATION_ERROR_14(*xFE*)

6.7.2.17 #define EMBER_APPLICATION_ERROR_15(*xFF*)

6.7.2.18 #define EMBER_APPLICATION_ERROR_2(*xF2*)

6.7.2.19 #define EMBER_APPLICATION_ERROR_3(*xF3*)

6.7.2.20 #define EMBER_APPLICATION_ERROR_4(*xF4*)

6.7.2.21 #define EMBER_APPLICATION_ERROR_5(*xF5*)

6.7.2.22 #define EMBER_APPLICATION_ERROR_6(*xF6*)

6.7.2.23 #define EMBER_APPLICATION_ERROR_7(*xF7*)

6.7.2.24 #define EMBER_APPLICATION_ERROR_8(*xF8*)

6.7.2.25 #define EMBER_APPLICATION_ERROR_9(*xF9*)

6.7.2.26 #define EMBER_APS_ENCRYPTION_ERROR(*xA6*)

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.

6.7.2.27 #define EMBER_ASSERT_SERIAL_PORT 1

The output can be suppressed by defining EMBER_ASSERT_OUTPUT_DISABLED. The serial port to which the output is sent can be changed by defining EMBER_ASSERT_SERIAL_PORT as the desired port.

The default is to have assert output on and sent to serial port 1.

6.7.2.28 #define EMBER_BAD_ARGUMENT(*x02*)

6.7.2.29 #define EMBER_BINDING_HAS_CHANGED(*xA4*)

6.7.2.30 #define EMBER_BINDING_INDEX_OUT_OF_RANGE(*x69*)

6.7.2.31 #define EMBER_BINDING_IS_ACTIVE(*x75*)

6.7.2.32 #define EMBER_BROADCAST_ADDRESS 0xFFFC

Broadcast to all routers.

6.7.2.33 #define EMBER_CANNOT_JOIN_AS_ROUTER(*x98*)

6.7.2.34 #define EMBER_CHANNEL_CHANGED(*x9B*)

6.7.2.35 #define EMBER_CHILD_TABLE_SIZE 16

Note: We do not support greater than 32 children, so the maximum value for this configuration setting is 32.

6.7.2.36 #define EMBER_COST_NOT_KNOWN(*x71*)

6.7.2.37 #define EMBER_COUNTER_STRINGS

6.7.2.38 #define EMBER_DELIVERY_FAILED(*x66*)

6.7.2.39 #define EMBER_EEPROM_MFG_STACK_VERSION_MISMATCH(*x04*)

6.7.2.40 #define EMBER_EEPROM_MFG_VERSION_MISMATCH(*x06*)

6.7.2.41 #define EMBER_EEPROM_STACK_VERSION_MISMATCH(*x07*)

6.7.2.42 #define EMBER_ENCRYPTION_KEY_SIZE 16

6.7.2.43 #define EMBER_END_DEVICE_POLL_TIMEOUT 240

The default is 240 seconds.

If no poll is heard within this time, then the parent removes the EMBER_END_DEVICE from its tables.

6.7.2.44 #define EMBER_ERR_BOOTLOADER_NO_IMAGE(*x05A*)

6.7.2.45 #define EMBER_ERR_BOOTLOADER_TRAP_TABLE_BAD(*x58*)

6.7.2.46 #define EMBER_ERR_BOOTLOADER_TRAP_UNKNOWN(*x59*)

6.7.2.47 #define EMBER_ERR_FATAL(*x01*)

6.7.2.48 #define EMBER_ERR_FLASH_ERASE_FAIL(*x4C*)

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.

6.7.2.49 #define EMBER_ERR_FLASH_PROG_FAIL(*x4B*)

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.

6.7.2.50 #define EMBER_ERR_FLASH_VERIFY_FAILED(*x47*)

Referenced by halSimEepromCallback().

6.7.2.51 #define EMBER_ERR_FLASH_WRITE_INHIBITED(*x46*)

Referenced by halSimEepromCallback().

6.7.2.52 #define EMBER_HEAP_SIZE 6000

6.7.2.53 #define EMBER_INCOMPATIBLE_STATIC_MEMORY_DEFINITIONS(*x05*)

6.7.2.54 #define EMBER_INDEX_OUT_OF_RANGE(*xB1*)

6.7.2.55 #define EMBER_INDIRECT_TRANSMISSION_TIMEOUT 30

The default is 30 quarter seconds (7.5 seconds). The maximum value is 30000 quarter seconds (125 minutes). Larger values will cause rollover confusion.

6.7.2.56 #define EMBER_INSUFFICIENT_RANDOM_DATA(*xA5*)

6.7.2.57 #define EMBER_INVALID_BINDING_INDEX(*x6C*)

6.7.2.58 #define EMBER_INVALID_CALL(*x70*)

6.7.2.59 #define EMBER_INVALID_ENDPOINT(*xA3*)

6.7.2.60 #define EMBER_INVALID_SECURITY_LEVEL(*x95*)

6.7.2.61 #define EMBER_JOIN_FAILED(*x94*)

6.7.2.62 #define EMBER_JOIN_KEY_MAX_SIZE 32

6.7.2.63 #define EMBER_KEY_INVALID(*xB2*)

6.7.2.64 #define EMBER_KEY_NOT_AUTHORIZED(*xBB*)

6.7.2.65 #define EMBER_KEY_TABLE_INVALID_ADDRESS(*xB3*)

6.7.2.66 #define EMBER_LIBRARY_NOT_PRESENT(*xB5*)

6.7.2.67 #define EMBER_MAC_ACK_HEADER_TYPE(*x3B*)

6.7.2.68 #define EMBER_MAC_BAD_SCAN_DURATION(*x33*)

6.7.2.69 #define EMBER_MAC_COMMAND_TRANSMIT_FAILURE(*x36*)

6.7.2.70 #define EMBER_MAC_COUNTER_ERROR(*xDB*)

6.7.2.71 #define EMBER_MAC_INCORRECT_SCAN_TYPE(*x34*)

6.7.2.72 #define EMBER_MAC_INDIRECT_TIMEOUT(*x42*)

6.7.2.73 #define EMBER_MAC_INVALID_CHANNEL_MASK(*x35*)

6.7.2.74 #define EMBER_MAC_JOINED_NETWORK(*x32*)

6.7.2.75 #define EMBER_MAC_NO_ACK_RECEIVED(*x40*)

6.7.2.76 #define EMBER_MAC_NO_DATA(*x31*)

6.7.2.77 #define EMBER_MAC_SCANNING(*x3D*)

6.7.2.78 #define EMBER_MAC_TRANSMIT_QUEUE_FULL(*x39*)

6.7.2.79 #define EMBER_MAC_UNKNOWN_HEADER_TYPE(*x3A*)

6.7.2.80 #define EMBER_MALLOC_HEAP_SIZE_BYTES 32768

6.7.2.81 #define EMBER_MANY_TO_ONE_ROUTE_FAILURE(*xAA*)

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.

6.7.2.82 #define EMBER_MAX_802_15_4_CHANNEL_NUMBER 26

6.7.2.83 #define EMBER_MAX_MESSAGE_LIMIT_REACHED(*x72*)

6.7.2.84 #define EMBER_MESSAGE_TOO_LONG(*x74*)

6.7.2.85 #define EMBER_MFG_RX_NCP_TO_HOST_INTERVAL 50

The default value is 50 packets.

6.7.2.86 #define EMBER_MIN_802_15_4_CHANNEL_NUMBER 11

6.7.2.87 #define EMBER_MOVE_FAILED(*x96*)

6.7.2.88 #define EMBER_NETWORK_BUSY(*xA1*)

6.7.2.89 #define EMBER_NETWORK_DOWN(*x91*)

6.7.2.90 #define EMBER_NETWORK_ID_SIZE 16

6.7.2.91 #define EMBER_NETWORK_UP(*x90*)

6.7.2.92 #define EMBER_NO_BEACONS(*xAB*)

6.7.2.93 #define EMBER_NO_BUFFERS(*x18*)

6.7.2.94 #define EMBER_NO_LINK_KEY_RECEIVED(*xAE*)

6.7.2.95 #define EMBER_NO_NETWORK_KEY_RECEIVED(*xAD*)

6.7.2.96 #define EMBER_NODE_ID_CHANGED(*x99*)

6.7.2.97 #define EMBER_NOT_JOINED(*x93*)

6.7.2.98 #define EMBER_NULL_NODE_ID 0xFFFF

6.7.2.99 #define EMBER_NUM_802_15_4_CHANNELS (EMBER_MAX_802_15_4_CHANNEL_NUMBER - EMBER_MIN_802_15_4_CHANNEL_NUMBER + 1)

6.7.2.100 #define EMBER_OPERATION_IN_PROGRESS(*xBA*)

6.7.2.101 #define EMBER_PAN_ID_CHANGED(*x9A*)

6.7.2.102 #define EMBER_PHY_ACK_RECEIVED(*x8F*)

6.7.2.103 #define EMBER_PHY_INVALID_CHANNEL(*x8A*)

6.7.2.104 #define EMBER_PHY_INVALID_POWER(*x8B*)

6.7.2.105 #define EMBER_PHY_OSCILLATOR_CHECK_FAILED(*x8E*)

6.7.2.106 #define EMBER_PHY_TX_BUSY(*x8C*)

6.7.2.107 #define EMBER_PHY_TX_CCA_FAIL(*x8D*)

6.7.2.108 #define EMBER_PHY_TX_INCOMPLETE(*x89*)

6.7.2.109 #define EMBER_PHY_TX_UNDERFLOW(*x88*)

6.7.2.110 #define EMBER_PRECONFIGURED_KEY_REQUIRED(*xAF*)

6.7.2.111 #define EMBER_RECEIVED_KEY_IN_THE_CLEAR(*xAC*)

6.7.2.112 #define EMBER_RETRY_QUEUE_SIZE 8

6.7.2.113 #define EMBER_ROUTE_FAILURE(*xA9*)

6.7.2.114 #define EMBER_RX_ON_WHEN_IDLE_BROADCAST_ADDRESS 0xFFFD

Broadcast to all non-sleepy devices.

6.7.2.115 #define EMBER_SECURITY_CONFIGURATION_INVALID(*xB7*)

6.7.2.116 #define EMBER_SECURITY_DATA_INVALID(*xBD*)

6.7.2.117 #define EMBER_SECURITY_LEVEL 5

6.7.2.118 #define EMBER_SECURITY_STATE_NOT_SET(*xA8*)

6.7.2.119 #define EMBER_SECURITY_TO_HOST false

6.7.2.120 #define EMBER_SERIAL_INVALID_BAUD_RATE(*x20*)

6.7.2.121 #define EMBER_SERIAL_INVALID_PORT(*x21*)

6.7.2.122 #define EMBER_SERIAL_RX_EMPTY(*x26*)

6.7.2.123 #define EMBER_SERIAL_RX_FRAME_ERROR(*x24*)

6.7.2.124 #define EMBER_SERIAL_RX_OVERFLOW(*x23*)

6.7.2.125 #define EMBER_SERIAL_RX_OVERRUN_ERROR(*x27*)

6.7.2.126 #define EMBER_SERIAL_RX_PARITY_ERROR(*x25*)

6.7.2.127 #define EMBER_SERIAL_TX_OVERFLOW(*x22*)

6.7.2.128 #define EMBER_SIGNATURE_VERIFY_FAILURE(*xB9*)

6.7.2.129 #define EMBER_SIM_EEPROM_ERASE_PAGE_GREEN(*x43*)

The application should call the function halSimEepromErasePage() when it can to erase a page.

Referenced by halSimEepromCallback().

6.7.2.130 #define EMBER_SIM_EEPROM_ERASE_PAGE_RED(*x44*)

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.

Referenced by halSimEepromCallback().

6.7.2.131 #define EMBER_SIM_EEPROM_FULL(*x45*)

The application must call the function halSimEepromErasePage() to make room for any further calls to set a token.

Referenced by halSimEepromCallback().

6.7.2.132 #define EMBER_SIM_EEPROM_INIT_1_FAILED(*x48*)

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.

6.7.2.133 #define EMBER_SIM_EEPROM_INIT_2_FAILED(*x49*)

This failure means Attempt 1 failed, and the token system failed to properly reload default tokens and reset the Simulated EEPROM.

6.7.2.134 #define EMBER_SIM_EEPROM_INIT_3_FAILED(*x4A*)

This failure means one or both of the tokens ::TOKEN_MFG_NVDATA_VERSION or ::TOKEN_STACK_NVDAT↩A_VERSION were incorrect and the token system failed to properly reload default tokens and reset the Simulated EEPROM.

6.7.2.135 #define EMBER_SIM_EEPROM_REPAIRING(*x4D*)

While there's nothing for an app to do when the SimEE is going to repair itself (SimEE has to be fully functional for the rest of the system to work), alert the application to the fact that repairing is occurring. There are debugging scenarios where an app might want to know that repairing is happening; such as monitoring frequency.

Note

Common situations will trigger an expected repair, such as using an erased chip or changing token definitions.

Referenced by halSimEepromCallback().

6.7.2.136 #define EMBER_SLEEP_INTERRUPTED(*x85*)

6.7.2.137 #define EMBER_SLEEPY_BROADCAST_ADDRESS 0xFFFF

Broadcast to all devices, including sleepy end devices.

6.7.2.138 #define EMBER_SLEEPY_CHILD_POLL_TIMEOUT 240

The default is 240 seconds. The maximum value is 2^32 - 1 (136 years).

This value is determined by the child and communicated to the parent via the MLE protocol.

6.7.2.139 #define EMBER_STACK_AND_HARDWARE_MISMATCH(*xB0*)

6.7.2.140 #define EMBER_SUCCESS(*x00*)

6.7.2.141 #define EMBER_TABLE_ENTRY_ERASED(*xB6*)

6.7.2.142 #define EMBER_TABLE_FULL(*xB4*)

6.7.2.143 #define EMBER_TASK_COUNT (3)

6.7.2.144 #define EMBER_TOO_SOON_FOR_SWITCH_KEY(*xB8*)

6.7.2.145 #define EMBER_TRUST_CENTER_EUI_HAS_CHANGED(*xBC*)

6.7.2.146 #define EMBER_TRUST_CENTER_MASTER_KEY_NOT_SET(*xA7*)

6.7.2.147 #define EMBER_TX_POWER_MODE_ALTERNATE 0x0002

6.7.2.148 #define EMBER_TX_POWER_MODE_BOOST 0x0001

6.7.2.149 #define EMBER_TX_POWER_MODE_BOOST_AND_ALTERNATE

Value:

```
(EMBER_TX_POWER_MODE_BOOST \
                                                | EMBER_TX_POWER_MODE_ALTERNATE
      )
```

6.7.2.150 #define EMBER_TX_POWER_MODE_DEFAULT 0x0000

6.7.2.151 #define EMBER_USE_DIRECT_IP_CALLBACK false

6.7.2.152 #define EMBER_VERSION_NAME "Thread"

The default version name for an application.

6.7.2.153 #define EMBER_VERSION_TYPE_MAX EMBER_VERSION_TYPE_LEGACY

6.7.2.154 #define EMBER_VERSION_TYPE_NAMES

Value:

```
"Internal",                   \
  "Alpha",                     \
  "Beta",                      \
  "GA",                        \
  "Special",                   \
  "Legacy",                    \
```

6.7.2.155 #define EMBER_ZIGBEE_COORDINATOR_ADDRESS 0x0000

6.7.2.156 #define EUI64_SIZE 8

6.7.2.157 #define EXTENDED_PAN_ID_SIZE 8

6.7.2.158 #define INT16U_MAX ((uint16_t)(∼(uint16_t)0))

6.7.2.159 #define LEADER_SIZE EUI64_SIZE

6.7.2.160 #define NULL_BUFFER 0x0000

6.7.2.161 #define RIP_MAX_LURKERS 0

6.7.2.162 #define TLS_MASTER_SECRET_SIZE 48

6.7.2.163 #define TLS_SESSION_ID_SIZE 32

6.7.3 Typedef Documentation

6.7.3.1 typedef uint16_t Buffer

6.7.3.2 typedef uint16_t ChildStatusFlags

6.7.3.3 typedef uint8_t EmberEUI64[EUI64_SIZE]

6.7.3.4 typedef { ... } EmberEventData

An application typically creates an array of events along with their handlers. The main loop passes the array to ::emberRunEvents() to call the handlers of any events whose time has arrived.

6.7.3.5 typedef uint16_t EmberMessageBuffer

6.7.3.6 typedef uint16_t EmberNodeId

6.7.3.7 typedef uint16_t EmberPanId

6.7.3.8 typedef uint8_t EmberStatus

6.7.3.9 typedef uint8_t EmberTaskId

brief An identifier for a task

6.7.3.10 typedef struct Event_s Event

6.7.3.11 typedef const struct EventActions_s EventActions

6.7.3.12 typedef struct EventQueue_s EventQueue

6.7.3.13 typedef Buffer PacketHeader

6.7.4 Enumeration Type Documentation

6.7.4.1 enum EmberCounterType

Enumerator

EMBER_COUNTER_PHY_IN_PACKETS Every packet that comes in over the radio (except MAC acks).

EMBER_COUNTER_PHY_OUT_PACKETS Every packet that goes out over the radio (except MAC acks).

EMBER_COUNTER_PHY_IN_OCTETS Every incoming byte, including the 802.15.4 length byte. Note MAC acks are not counted.

EMBER_COUNTER_PHY_OUT_OCTETS Every outgoing byte, including the 802.15.4 length byte. MAC retries contribute to the count, but not MAC acks.

EMBER_COUNTER_MAC_IN_UNICAST Incoming MAC unicasts post duplicate detection.

EMBER_COUNTER_MAC_IN_BROADCAST

EMBER_COUNTER_MAC_OUT_UNICAST_SUCCESS Outgoing MAC unicasts for which an ack was received possibly after retrying.

EMBER_COUNTER_MAC_OUT_UNICAST_ACK_FAIL Outgoing unicasts for which ack was never received even after retrying.

EMBER_COUNTER_MAC_OUT_UNICAST_CCA_FAIL Outgoing MAC packets which were never transmitted because clear channel assessment always returned busy.

EMBER_COUNTER_MAC_OUT_UNICAST_EXT_FAIL Outgoing unicasts that failed even after extended MAC retries.

EMBER_COUNTER_MAC_OUT_UNICAST_RETRY Outgoing unicast retries. This does not count the initial transmission. Note a single MAC transmission can result in multiple retries.

EMBER_COUNTER_MAC_OUT_BROADCAST

EMBER_COUNTER_MAC_OUT_BROADCAST_CCA_FAIL

EMBER_COUNTER_MAC_DROP_IN_MEMORY Dropped incoming MAC packets (out of memory).

EMBER_COUNTER_MAC_DROP_IN_NO_EUI Dropped incoming MAC packets (no EUI).

EMBER_COUNTER_MAC_DROP_IN_FRAME_COUNTER Dropped incoming MAC packets (invalid frame counter).

EMBER_COUNTER_MAC_DROP_IN_DECRYPT Dropped incoming MAC packets (can't decrypt).

EMBER_COUNTER_MAC_DROP_IN_DUPLICATE Dropped incoming MAC packets (duplicate message).

EMBER_COUNTER_IP_IN_UNICAST IP packets

EMBER_COUNTER_IP_OUT_UNICAST

EMBER_COUNTER_IP_IN_MULTICAST

EMBER_COUNTER_IP_OUT_MULTICAST

EMBER_COUNTER_UDP_IN Application UDP messages. Excludes DNS, PANA, and MLE.

EMBER_COUNTER_UDP_OUT

EMBER_COUNTER_UART_IN_DATA UART in and out data

EMBER_COUNTER_UART_IN_MANAGEMENT

EMBER_COUNTER_UART_IN_FAIL

EMBER_COUNTER_UART_OUT_DATA

EMBER_COUNTER_UART_OUT_MANAGEMENT

EMBER_COUNTER_UART_OUT_FAIL

EMBER_COUNTER_ROUTE_2_HOP_LOOP

EMBER_COUNTER_BUFFER_ALLOCATION_FAIL

EMBER_ASH_V3_ACK_SENT ASHv3

EMBER_ASH_V3_ACK_RECEIVED

EMBER_ASH_V3_NACK_SENT

EMBER_ASH_V3_NACK_RECEIVED

EMBER_ASH_V3_RESEND

EMBER_ASH_V3_BYTES_SENT

EMBER_ASH_V3_TOTAL_BYTES_RECEIVED

EMBER_ASH_V3_VALID_BYTES_RECEIVED

EMBER_ASH_V3_PAYLOAD_BYTES_SENT

EMBER_COUNTER_PTA_LO_PRI_REQUESTED The number of times a low-priority packet traffic arbitration request has been made.

EMBER_COUNTER_PTA_HI_PRI_REQUESTED The number of times a high-priority packet traffic arbitration request has been made.

EMBER_COUNTER_PTA_LO_PRI_DENIED The number of times a low-priority packet traffic arbitration request has been denied.

EMBER_COUNTER_PTA_HI_PRI_DENIED The number of times a high-priority packet traffic arbitration request has been denied.

EMBER_COUNTER_PTA_LO_PRI_TX_ABORTED The number of times a low-priority packet traffic arbitration transmission has been aborted.

EMBER_COUNTER_PTA_HI_PRI_TX_ABORTED The number of times a high-priority packet traffic arbitration transmission has been aborted.

EMBER_COUNTER_TYPE_COUNT A placeholder giving the number of Ember counter types.

EMBER_COUNTER_ALL

6.7.4.2 enum EmberEventUnits

Enumerator

EMBER_EVENT_INACTIVE The event is not scheduled to run.

EMBER_EVENT_MS_TIME The execution time is in approximate milliseconds.

EMBER_EVENT_QS_TIME The execution time is in 'binary' quarter seconds (256 approximate milliseconds each).

EMBER_EVENT_MINUTE_TIME The execution time is in 'binary' minutes (65536 approximate milliseconds each).

EMBER_EVENT_ZERO_DELAY The event is scheduled to run at the earliest opportunity.

6.7.4.3 enum EmberIcmpCode

Enumerator

ICMP_CODE_NO_ROUTE_TO_DESTINATION

ICMP_CODE_ERROR_IN_SOURCE_ROUTING_HEADER

6.7.4.4 enum EmberIcmpType

Enumerator

ICMP_DESTINATION_UNREACHABLE

ICMP_PACKET_TOO_BIG

ICMP_TIME_EXCEEDED

ICMP_PARAMETER_PROBLEM

ICMP_PRIVATE_EXPERIMENTATION_0

ICMP_ECHO_REQUEST

ICMP_ECHO_REPLY

ICMP_ROUTER_SOLICITATION

ICMP_ROUTER_ADVERTISEMENT

ICMP_NEIGHBOR_SOLICITATION

ICMP_NEIGHBOR_ADVERTISEMENT

ICMP_RPL

ICMP_DUPLICATE_ADDRESS_REQUEST

ICMP_DUPLICATE_ADDRESS_CONFIRM

6.7.4.5 enum EmberIpv6NextHeader

Enumerator

IPV6_NEXT_HEADER_ICMP

IPV6_NEXT_HEADER_TCP

IPV6_NEXT_HEADER_UDP

IPV6_NEXT_HEADER_IPV6

IPV6_NEXT_HEADER_ICMPV6

IPV6_NEXT_HEADER_NO_NEXT

IPV6_NEXT_HEADER_MOBILITY

IPV6_NEXT_HEADER_HOP_BY_HOP

IPV6_NEXT_HEADER_DESTINATION

IPV6_NEXT_HEADER_ROUTING

IPV6_NEXT_HEADER_FRAGMENT

IPV6_NEXT_HEADER_UNKNOWN

6.7.4.6 enum EmberJoinFailureReason

This information is passed up to the application via the emberNetworkStatusHandler callback.

Enumerator

EMBER_JOIN_FAILURE_REASON_NONE No failure. This indicates that the network status change occurred as part of regular network operation.

EMBER_JOIN_FAILURE_REASON_FORM_SCAN The operation emberFormNetwork failed while performing an energy scan on a channel.

EMBER_JOIN_FAILURE_REASON_ACTIVE_SCAN emberJoinNetwork or emberJoinCommissioned failed while performing an active scan on a channel. This indicates that discovery failed due to no network matching the network parameters being filtered on.

EMBER_JOIN_FAILURE_REASON_COMMISSIONING emberJoinNetwork failed during commissioning. This usually indicates that either the commissioning step timed out or a mismatch occurred with one of the parameters passed into emberJoinNetwork.

EMBER_JOIN_FAILURE_REASON_SECURITY emberJoinNetwork failed during the DTLS handshake to establish a shared key. This usually indicates that either the join key (EMBER_JOIN_KEY_OPTION) passed in this call is incorrect or some other fatal error occurred, such as a timeout.

6.7.4.7 enum EmberNetworkScanType

Enumerator

EMBER_ENERGY_SCAN An energy scan for each channel looks for its RSSI value.

EMBER_ACTIVE_SCAN An active scan for each channel looks for available networks.

6.7.4.8 enum EmberNetworkStatus

Enumerator

EMBER_NO_NETWORK The node is not associated with a network in any way.
EMBER_SAVED_NETWORK The node was part of a network prior to reset.
EMBER_JOINING_NETWORK The node is currently attempting to join a network.
EMBER_JOINED_NETWORK_ATTACHED The node is joined and attached to a network.
EMBER_JOINED_NETWORK_NO_PARENT The node is joined but without a parent.
EMBER_JOINED_NETWORK_ATTACHING The node is joined but is currently attaching.

6.7.4.9 enum EmberVersionType

Enumerator

EMBER_VERSION_TYPE_INTERNAL
EMBER_VERSION_TYPE_ALPHA
EMBER_VERSION_TYPE_BETA
EMBER_VERSION_TYPE_GA
EMBER_VERSION_TYPE_SPECIAL
EMBER_VERSION_TYPE_LEGACY

6.7.5 Function Documentation

6.7.5.1 void emberCalibrateCurrentChannel (void)

Referenced by emberRadioNeedsCalibratingHandler().

6.7.5.2 uint8_t emberGetRadioChannel (void)

Returns

A current radio channel.

6.7.5.3 void emberRadioNeedsCalibratingHandler (void)

Boost power mode is a high-performance radio mode which offers increased transmit power and receive sensitivity at the cost of an increase in power consumption. The alternate transmit output path allows for simplified connection to an external power amplifier via the RF_TX_ALT_P and RF_TX_ALT_N pins on the em250. emberInit() calls this function using the power mode and transmitter output settings as specified in the MFG_PHY_CONFIG token (with each bit inverted so that the default token value of 0xffff corresponds to normal power mode and bi-directional RF transmitter output). The application only needs to call emberSetTxPowerMode() to use a power mode or transmitter output setting different from that specified in the MFG_PHY_CONFIG token. After this initial call to emberSetTxPowerMode(), the stack will automatically maintain the specified power mode configuration across sleep/wake cycles.

Note

This function does not alter the MFG_PHY_CONFIG token. The MFG_PHY_CONFIG token must be properly configured to ensure optimal radio performance when the standalone bootloader runs in recovery mode. The MFG_PHY_CONFIG can only be set using external tools. IF YOUR PRODUCT USES BOOST MODE OR THE ALTERNATE TRANSMITTER OUTPUT AND THE STANDALONE BOOTLOADER YOU MUST SET THE PHY_CONFIG TOKEN INSTEAD OF USING THIS FUNCTION. Contact support@ember.com for instructions to set the MFG_PHY_CONFIG token appropriately.

Parameters

txPowerMode	Specifies which of the transmit power mode options are to be activated. This parameter should be set to one of the literal values described in stack/include/ember-types.h. Any power option not specified in the txPowerMode parameter will be deactivated.

Returns

EMBER_SUCCESS if successful; an error code otherwise.This function returns the current configuration of boost power mode and alternate transmitter output.
The current tx power mode.The radio calibration callback function.

The Voltage Controlled Oscillator (VCO) can drift with temperature changes. During every call to emberTick(), the stack will check to see if the VCO has drifted. If the VCO has drifted, the stack will call emberRadioNeeds↩ CalibratingHandler() to inform the application that it should perform calibration of the current channel as soon as possible. Calibration can take up to 150 ms. The default callback function implementation provided here performs the calibration immediately. If the application wishes, it can define its own callback by defining ::EMBER_AP↩ PLICATION_HAS_CUSTOM_RADIO_CALIBRATION_CALLBACK in its CONFIGURATION_HEADER. It can then failsafe any critical processes or peripherals before calling emberCalibrateCurrentChannel(). The application must call emberCalibrateCurrentChannel() in response to this callback to maintain an expected radio performance.

This function enables boost power mode and/or the alternate transmit path.

The Voltage Controlled Oscillator (VCO) can drift with temperature changes. During every call to emberTick(), the stack will check to see if the VCO has drifted. If the VCO has drifted, the stack will call emberRadioNeeds↩ CalibratingHandler() to inform the application that it should perform calibration of the current channel as soon as possible. Calibration can take up to 150ms. The default callback function implementation provided here performs calibration immediately. If the application wishes, it can define its own callback by defining ::EMBER_APPLICAT↩ ION_HAS_CUSTOM_RADIO_CALIBRATION_CALLBACK in its CONFIGURATION_HEADER. It can then failsafe any critical processes or peripherals before calling emberCalibrateCurrentChannel(). The application must call emberCalibrateCurrentChannel() in response to this callback to maintain expected radio performance.

References emberCalibrateCurrentChannel().

6.7.5.4 EmberStatus emberSetRadioChannel (uint8_t *channel*)

Note: Care should be taken when using this API, as all devices on a network must use the same channel.

Parameters

channel	A desired radio channel.

Returns

An EmberStatus value indicating the success or failure of the command.

6.7.6 Variable Documentation

6.7.6.1 EventActions* Event_s::actions

6.7.6.2 uint16_t EmberVersion::build

6.7.6.3 bool EmberTaskControl::busy

6.7.6.4 uint8_t EmberEui64::bytes[EUI64_SIZE]

6.7.6.5 uint8_t EmberIpv6Prefix::bytes[8]

6.7.6.6 uint8_t EmberIpv6Address::bytes[16]

6.7.6.7 const uint8_t* DeviceCertificate::certificate

6.7.6.8 const uint16_t DeviceCertificate::certificateSize

6.7.6.9 uint32_t EmberVersion::change

6.7.6.10 uint8_t EmberKeyData::contents[EMBER_ENCRYPTION_KEY_SIZE]

This is the key byte data.

6.7.6.11 uint8_t Bytes8::contents[8]

6.7.6.12 uint8_t Bytes16::contents[16]

6.7.6.13 EmberEventControl* control

The control structure for the event.

6.7.6.14 EmberEventControl* { ... } control

The control structure for the event.

6.7.6.15 uint8_t Ipv6Header::destination[16]

6.7.6.16 uint16_t Ipv6Header::destinationPort

6.7.6.17 Event* EventQueue_s::events

6.7.6.18 EmberEventData* EmberTaskControl::events

6.7.6.19 uint32_t Ipv6Header::flowLabel

6.7.6.20 void(* EventActions_s::handler) (struct Event_s *)

6.7.6.21 void(* { ... } handler) (void)

The procedure to call when the event fires.

6.7.6.22 void(∗ handler) (void)

The procedure to call when the event fires.

6.7.6.23 uint8_t Ipv6Header::hopLimit

6.7.6.24 uint8_t Ipv6Header::icmpCode

6.7.6.25 uint8_t Ipv6Header::icmpType

6.7.6.26 uint16_t TlsSessionState::id[(TLS_SESSION_ID_SIZE+1)/2]

6.7.6.27 uint8_t TlsSessionState::idLength

6.7.6.28 uint8_t∗ Ipv6Header::ipPayload

6.7.6.29 uint16_t Ipv6Header::ipPayloadLength

6.7.6.30 Event∗ EventQueue_s::isrEvents

6.7.6.31 uint8_t EmberVersion::major

6.7.6.32 void(∗ EventActions_s::marker) (struct Event_s ∗)

6.7.6.33 uint8_t TlsSessionState::master[TLS_MASTER_SECRET_SIZE]

6.7.6.34 uint8_t CertificateAuthority::maxPathLength

6.7.6.35 uint8_t EmberVersion::minor

6.7.6.36 const uint8_t∗ CertificateAuthority::name

6.7.6.37 const char∗ EventActions_s::name

6.7.6.38 uint16_t CertificateAuthority::nameLength

6.7.6.39 struct Event_s∗ Event_s::next

6.7.6.40 uint32_t EmberTaskControl::nextEventTime

6.7.6.41 uint8_t Ipv6Header::nextHeader

6.7.6.42 uint8_t EmberVersion::patch

6.7.6.43 const uint8_t∗ DeviceCertificate::privateKey

6.7.6.44 uint8_t* CertificateAuthority::publicKey

6.7.6.45 struct EventQueue_s* EventActions_s::queue

6.7.6.46 bool EventQueue_s::running

6.7.6.47 uint32_t EventQueue_s::runTime

6.7.6.48 uint8_t Ipv6Header::source[16]

6.7.6.49 uint16_t Ipv6Header::sourcePort

6.7.6.50 EmberEventUnits EmberEventControl::status

The event's status, either inactive or the units for timeToExecute.

6.7.6.51 EmberTaskId EmberEventControl::taskid

The ID of the task this event belongs to.

6.7.6.52 uint32_t Event_s::timeToExecute

6.7.6.53 uint32_t EmberEventControl::timeToExecute

How long before the event fires. Units are always in milliseconds.

6.7.6.54 uint8_t Ipv6Header::trafficClass

6.7.6.55 uint8_t* Ipv6Header::transportHeader

6.7.6.56 uint16_t Ipv6Header::transportHeaderLength

6.7.6.57 uint8_t* Ipv6Header::transportPayload

6.7.6.58 uint16_t Ipv6Header::transportPayloadLength

6.7.6.59 uint8_t Ipv6Header::transportProtocol

6.7.6.60 EmberVersionType EmberVersion::type

6.8 AES crypto routines

Macros

- #define EMBER_AES_BLOCK_SIZE_BYTES 16

  The number of bytes in a 128-bit AES block.

Functions

- void emberAesEcbEncryptBlock (uint8_t *block, const uint8_t *key, bool sameKey)

  This function performs a standalone-mode "electronic code book" (ECB) AES-128 encryption of the 16-byte plaintext `block` *using the 128-bit (16-byte)* `key`*. The resulting 16 byte ciphertext overwrites the plaintext* `block`*.*
- void emberAesCtrCryptData (uint8_t *nonce, const uint8_t *key, uint8_t *data, uint32_t dataLen, uint32_t dataDid)

  This function performs a counter-mode (CTR) AES-128 encrypt/decrypt of the `data` *for* `dataLen` *bytes, using the 128-bit (16-byte)* `key` *and 128-bit (16-byte)* `nonce`*. The resulting encrypted/decrypted data overwrites the* `data` *passed in.*

6.8.1 Detailed Description

See aes.h for source code.

6.8.2 Macro Definition Documentation

6.8.2.1 #define EMBER_AES_BLOCK_SIZE_BYTES 16

6.8.3 Function Documentation

6.8.3.1 void emberAesCtrCryptData (uint8_t * *nonce*, const uint8_t * *key*, uint8_t * *data*, uint32_t *dataLen*, uint32_t *dataDid*)

Parameters

nonce	The big-endian nonce (MSB is nonce[0] and LSB is nonce[15]) serves as a 128-bit block counter for every 16-byte block of `data`. It is incremented by the number of blocks processed ((dataLen+15)/16).
key	A pointer to the 128-bit key to be used for the nonce encryption.
data	A pointer to the plain- or cypher-text to be encrypted/decrypted in place.
dataLen	Indicates the number of bytes of data. It need not be a multiple of 16 bytes.
dataDid	This parameter allows splitting a CTR operation across multiple calls. The first call passes in `dataDid` of 0 to start a fresh CTR. Then subsequent calls pass in `dataDid` of the sum of the previous calls' `dataLen` values (with `data` and `dataLen` representing the new portion to encrypt/decrypt). A non-zero `dataDid` indicates a continuation of the prior CTR operation which will pick up where the earlier one left off.

Note

If your `nonce` is divided into a fixed and counter portion, ensure that the counter value passed in is such that when incremented by the number of blocks ((dataLen+15)/16) it won't overflow the counter portion into the fixed portion of the nonce. It may be necessary to split the operation across multiple calls to emberAesCtr↩CryptData() to satisfy this criteria.

6.8.3.2 void emberAesEcbEncryptBlock (uint8_t * *block*, const uint8_t * *key*, bool *sameKey*)

Parameters

block	A pointer to the 128-bit data in RAM to be encrypted in place.
key	A pointer to the 128-bit key to be used for the encryption.
sameKey	If true, indicates that the 128-bit `key` value is the same as it was in a prior call to this routine and serves as a hint that the key needn't be reloaded into the AES hardware engine. Otherwise, the `key` value is considered new and will always be loaded.

6.9 Messaging

Sending unicasts and multicasts.

Modules

- ICMP messages

  Sending and receiving ICMP messages.
- UDP messages

  Sending and receiving UDP messages.
- Constrained Application Protocol API
- DTLS API

6.9.1 Detailed Description

6.10 ICMP messages

Sending and receiving ICMP messages.

Functions

- EmberStatus emberIcmpListen (const uint8_t *address)

  This function sets up a listener for ICMP messages for the given address.
- bool emberIpPing (uint8_t *destination, uint16_t id, uint16_t sequence, uint16_t length, uint8_t hopLimit)

  This function sends an ICMP ECHO REQUEST message.
- void emberIncomingIcmpHandler (Ipv6Header *ipHeader)

  An application callback for an incoming ICMP message.

6.10.1 Detailed Description

See icmp.h for source code.

6.10.2 Function Documentation

6.10.2.1 EmberStatus emberIcmpListen (const uint8_t * *address*)

Parameters

address	The IPv6 address, which is listened to.

Returns

EMBER_SUCCESS if successful EMBER_TABLE_FULL if failed to set up a listener EMBER_ERR_FATAL other fatal failure

6.10.2.2 void emberIncomingIcmpHandler (Ipv6Header * *ipHeader*)

Parameters

ipHeader	Pointer to an IPv6 buffer

An application callback for an incoming ICMP message.

Parameters

ipHeader	Pointer to an IPV6 buffer

6.10.2.3 bool emberIpPing (uint8_t * *destination*, uint16_t *id*, uint16_t *sequence*, uint16_t *length*, uint8_t *hopLimit*)

Parameters

destination	IPv6 destination address
id	IPv6 unique ID
sequence	IPv6 unique sequence
length	Payload length
hopLimit	IPv6 hop limit

Returns

Returns true if the ICMP echo request was succesfully submitted to the IP stack and false otherwise.

6.11 UDP messages

Sending and receiving UDP messages.

Functions

- EmberStatus emberUdpListen (uint16_t port, const uint8_t *address)

  This function sets up a listener for UDP messages for a given address.
- EmberStatus emberSendUdp (const uint8_t *destination, uint16_t sourcePort, uint16_t destinationPort, uint8_t *payload, uint16_t payloadLength)

  This function sends a UDP message.
- void emberUdpHandler (const uint8_t *destination, const uint8_t *source, uint16_t localPort, uint16_t remotePort, const uint8_t *payload, uint16_t payloadLength)

  An application callback for an incoming UDP message.
- void emberUdpMulticastHandler (const uint8_t *destination, const uint8_t *source, uint16_t localPort, uint16_t remotePort, const uint8_t *payload, uint16_t payloadLength)

  An application callback for an incoming UDP multicast.

6.11.1 Detailed Description

See udp.h for source code.

6.11.2 Function Documentation

6.11.2.1 EmberStatus emberSendUdp (const uint8_t * *destination*, uint16_t *sourcePort*, uint16_t *destinationPort*, uint8_t * *payload*, uint16_t *payloadLength*)

Parameters

destination	IPv6 destination address
sourcePort	UDP source port
destinationPort	UDP destination port
payload	UDP transport payload
payloadLength	Payload length

Returns

EMBER_SUCCESS if successful EMBER_NO_BUFFERS if failed to allocate a buffer EMBER_ERR_FATAL other fatal failure

6.11.2.2 void emberUdpHandler (const uint8_t * *destination*, const uint8_t * *source*, uint16_t *localPort*, uint16_t *remotePort*, const uint8_t * *payload*, uint16_t *payloadLength*)

Parameters

destination	IPv6 destination address

Parameters

source	IPv6 source address
localPort	UDP source port
remotePort	UDP destination port
payload	UDP transport payload
payloadLength	payload length

An application callback for an incoming UDP message.

Parameters

destination	IPV6 destination address
source	IPV6 source address
localPort	UDP source port
remotePort	UDP destination port
payload	UDP transport payload
payloadLength	payload length

Referenced by emberUdpMulticastHandler().

6.11.2.3 EmberStatus emberUdpListen (uint16_t *port,* const uint8_t * *address*)

Parameters

port	A port to bind the UDP address to
address	The IPv6 address listened to.

Returns

EMBER_SUCCESS if successful EMBER_TABLE_FULL if we failed to set up a listener EMBER_ERR_FATAL other fatal failure

6.11.2.4 void emberUdpMulticastHandler (const uint8_t * *destination,* const uint8_t * *source,* uint16_t *localPort,* uint16_t *remotePort,* const uint8_t * *payload,* uint16_t *payloadLength*)

Parameters

destination	IPv6 destination address
source	IPv6 source address
localPort	UDP source port
remotePort	UDP destination port
payload	UDP transport payload
payloadLength	payload length

An application callback for an incoming UDP multicast.

Parameters

destination	IPV6 destination address
source	IPV6 source address
localPort	UDP source port
remotePort	UDP destination port
payload	UDP transport payload
payloadLength	payload length

References emberUdpHandler().

6.12 Constrained Application Protocol API

Modules

- Callbacks
- Diagnostic Callbacks

Data Structures

- struct EmberCoapOption

  Structure that includes options in outgoing requests and responses.
- struct EmberCoapResponseInfo

  Additional information about an incoming response.
- struct EmberCoapSendInfo

  Optional information when sending a message.
- struct EmberCoapRequestInfo

  Additional information about an incoming request.
- struct EmberCoapBlockOption

Macros

- #define MAKE_COAP_CODE(class, detail) ((class << 5) | detail)
- #define GET_COAP_CLASS(code) (((code) & 0xE0) >> 5)
- #define GET_COAP_DETAIL(code) ((code) & 0x1F)
- #define EMBER_COAP_PORT 5683
- #define EMBER_COAP_SECURE_PORT 5684
- #define EMBER_COAP_MAX_TOKEN_LENGTH 8
- #define EMBER_COAP_DEFAULT_TIMEOUT_MS 90000
- #define emberBlockOptionSize(option) (1 << (option)->logSize)

Typedefs

- typedef struct EmberCoapReadOptions_s EmberCoapReadOptions

  This function encapsulates incoming CoAP options.
- typedef bool(* EmberCoapTransmitHandler) (const uint8_t *payload, uint16_t payloadLength, const EmberIpv6Address *localAddress, uint16_t localPort, const EmberIpv6Address *remoteAddress, uint16_t remotePort, void *transmitHandlerData)

  Function type for alternative transports.
- typedef void(* EmberCoapResponseHandler) (EmberCoapStatus status, EmberCoapCode code, EmberCoapReadOptions *options, uint8_t *payload, uint16_t payloadLength, EmberCoapResponseInfo *info)

  Type definition for callback handlers for a response.

Enumerations

- enum EmberCoapClass {
  EMBER_COAP_CLASS_REQUEST = 0,
  EMBER_COAP_CLASS_SUCCESS_RESPONSE = 2,
  EMBER_COAP_CLASS_CLIENT_ERROR_RESPONSE = 4,
  EMBER_COAP_CLASS_SERVER_ERROR_RESPONSE = 5 }
- enum EmberCoapCode {
  EMBER_COAP_CODE_EMPTY = MAKE_COAP_CODE(0, 0),
  EMBER_COAP_CODE_GET = MAKE_COAP_CODE(0, 1),
  EMBER_COAP_CODE_POST = MAKE_COAP_CODE(0, 2),
  EMBER_COAP_CODE_PUT = MAKE_COAP_CODE(0, 3),
  EMBER_COAP_CODE_DELETE = MAKE_COAP_CODE(0, 4),
  EMBER_COAP_CODE_201_CREATED = MAKE_COAP_CODE(2, 1),
  EMBER_COAP_CODE_202_DELETED = MAKE_COAP_CODE(2, 2),
  EMBER_COAP_CODE_203_VALID = MAKE_COAP_CODE(2, 3),
  EMBER_COAP_CODE_204_CHANGED = MAKE_COAP_CODE(2, 4),
  EMBER_COAP_CODE_205_CONTENT = MAKE_COAP_CODE(2, 5),
  EMBER_COAP_CODE_400_BAD_REQUEST = MAKE_COAP_CODE(4, 0),
  EMBER_COAP_CODE_401_UNAUTHORIZED = MAKE_COAP_CODE(4, 1),
  EMBER_COAP_CODE_402_BAD_OPTION = MAKE_COAP_CODE(4, 2),
  EMBER_COAP_CODE_403_FORBIDDEN = MAKE_COAP_CODE(4, 3),
  EMBER_COAP_CODE_404_NOT_FOUND = MAKE_COAP_CODE(4, 4),
  EMBER_COAP_CODE_405_METHOD_NOT_ALLOWED = MAKE_COAP_CODE(4, 5),
  EMBER_COAP_CODE_406_NOT_ACCEPTABLE = MAKE_COAP_CODE(4, 6),
  EMBER_COAP_CODE_412_PRECONDITION_FAILED = MAKE_COAP_CODE(4, 12),
  EMBER_COAP_CODE_413_REQUEST_ENTITY_TOO_LARGE = MAKE_COAP_CODE(4, 13),
  EMBER_COAP_CODE_415_UNSUPPORTED_CONTENT_FORMAT = MAKE_COAP_CODE(4, 15),
  EMBER_COAP_CODE_500_INTERNAL_SERVER_ERROR = MAKE_COAP_CODE(5, 0),
  EMBER_COAP_CODE_501_NOT_IMPLEMENTED = MAKE_COAP_CODE(5, 1),
  EMBER_COAP_CODE_502_BAD_GATEWAY = MAKE_COAP_CODE(5, 2),
  EMBER_COAP_CODE_503_SERVICE_UNAVAILABLE = MAKE_COAP_CODE(5, 3),
  EMBER_COAP_CODE_504_GATEWAY_TIMEOUT = MAKE_COAP_CODE(5, 4),
  EMBER_COAP_CODE_505_PROXYING_NOT_SUPPORTED = MAKE_COAP_CODE(5, 5) }
- enum EmberCoapOptionType {
  EMBER_COAP_NO_OPTION = 0,
  EMBER_COAP_OPTION_IF_MATCH = 1,
  EMBER_COAP_OPTION_URI_HOST = 3,
  EMBER_COAP_OPTION_ETAG = 4,
  EMBER_COAP_OPTION_IF_NONE_MATCH = 5,
  EMBER_COAP_OPTION_OBSERVE = 6,
  EMBER_COAP_OPTION_URI_PORT = 7,
  EMBER_COAP_OPTION_LOCATION_PATH = 8,
  EMBER_COAP_OPTION_URI_PATH = 11,
  EMBER_COAP_OPTION_CONTENT_FORMAT = 12,
  EMBER_COAP_OPTION_MAX_AGE = 14,
  EMBER_COAP_OPTION_URI_QUERY = 15,
  EMBER_COAP_OPTION_ACCEPT = 17,
  EMBER_COAP_OPTION_LOCATION_QUERY = 20,
  EMBER_COAP_OPTION_BLOCK2 = 23,
  EMBER_COAP_OPTION_BLOCK1 = 27,
  EMBER_COAP_OPTION_SIZE2 = 28,
  EMBER_COAP_OPTION_PROXY_URI = 35,
  EMBER_COAP_OPTION_PROXY_SCHEME = 39,
  EMBER_COAP_OPTION_SIZE1 = 60 }
- enum EmberCoapContentFormatType {

EMBER_COAP_CONTENT_FORMAT_TEXT_PLAIN = 0,
EMBER_COAP_CONTENT_FORMAT_LINK_FORMAT = 40,
EMBER_COAP_CONTENT_FORMAT_XML = 41,
EMBER_COAP_CONTENT_FORMAT_OCTET_STREAM = 42,
EMBER_COAP_CONTENT_FORMAT_EXI = 47,
EMBER_COAP_CONTENT_FORMAT_JSON = 50,
EMBER_COAP_CONTENT_FORMAT_CBOR = 60,
EMBER_COAP_CONTENT_FORMAT_LINK_FORMAT_PLUS_CBOR = 65064,
EMBER_COAP_CONTENT_FORMAT_NONE = -1 }

- enum EmberCoapStatus {
  EMBER_COAP_MESSAGE_TIMED_OUT,
  EMBER_COAP_MESSAGE_ACKED,
  EMBER_COAP_MESSAGE_RESET,
  EMBER_COAP_MESSAGE_RESPONSE }

  Status values passed to response handlers.

Functions

- bool emberCoapIsSuccessResponse (EmberCoapCode code)

  This function indicates whether the code represents a successful response.
- bool emberCoapIsClientErrorResponse (EmberCoapCode code)

  This function indicates whether the code represents a client error response.
- bool emberCoapIsServerErrorResponse (EmberCoapCode code)

  This function indicates whether the code represents a server error response.
- bool emberCoapIsRequest (EmberCoapCode code)

  This function indicates whether the code represents a request.
- bool emberCoapIsResponse (EmberCoapCode code)

  This function indicates whether the code represents a response.
- EmberCoapOptionType emberReadNextOption (EmberCoapReadOptions *options, const uint8_t **value↩PointerLoc, uint16_t *valueLengthLoc)

  This function reads the next option from an incoming message.
- void emberResetReadOptionPointer (EmberCoapReadOptions *options)

  This function resets the internal pointer back to the first option.
- uint32_t emberReadOptionValue (const uint8_t *value, uint16_t valuelength)

  This function decodes the value of an integer option.
- bool emberReadIntegerOption (EmberCoapReadOptions *options, EmberCoapOptionType type, uint32_↩t *valueLoc)

  This function reads the value of an integer option.
- bool emberReadBytesOption (EmberCoapReadOptions *options, EmberCoapOptionType type, const uint8↩_t **valueLoc, uint16_t *valueLengthLoc)

  This function reads the value of an option.
- int16_t emberReadLocationPath (EmberCoapReadOptions *options, uint8_t *pathBuffer, uint16_t path↩BufferLength)

  This function converts path options to a string.
- EmberStatus emberCoapSend (const EmberIpv6Address *destination, EmberCoapCode code, const uint8↩_t *path, const uint8_t *payload, uint16_t payloadLength, EmberCoapResponseHandler responseHandler, const EmberCoapSendInfo *info)

  This function sends a request.
- EmberStatus emberCoapGet (const EmberIpv6Address *destination, const uint8_t *path, EmberCoap↩ResponseHandler responseHandler, const EmberCoapSendInfo *info)

  This function sends a GET request.

- EmberStatus emberCoapPut (const EmberIpv6Address *destination, const uint8_t *path, const uint8_t *payload, uint16_t payloadLength, EmberCoapResponseHandler responseHandler, const EmberCoapSendInfo *info)

  This function sends a PUT request.
- EmberStatus emberCoapPost (const EmberIpv6Address *destination, const uint8_t *path, const uint8_t *payload, uint16_t payloadLength, EmberCoapResponseHandler responseHandler, const EmberCoapSendInfo *info)

  This function sends a POST request.
- EmberStatus emberCoapDelete (const EmberIpv6Address *destination, const uint8_t *path, EmberCoapResponseHandler responseHandler, const EmberCoapSendInfo *info)

  This function sends a DELETE request.
- void emberCoapRequestHandler (EmberCoapCode code, uint8_t *uri, EmberCoapReadOptions *options, const uint8_t *payload, uint16_t payloadLength, const EmberCoapRequestInfo *info)

  Callback for incoming requests.
- EmberStatus emberCoapRespond (const EmberCoapRequestInfo *requestInfo, EmberCoapCode code, const EmberCoapOption *options, uint8_t numberOfOptions, const uint8_t *payload, uint16_t payloadLength)

  Sending a response.
- EmberStatus emberCoapRespondWithPath (const EmberCoapRequestInfo *requestInfo, EmberCoapCode code, const uint8_t *path, const EmberCoapOption *options, uint8_t numberOfOptions, const uint8_t *payload, uint16_t payloadLength)

  Sending a response that includes a location path.
- EmberStatus emberCoapRespondWithCode (const EmberCoapRequestInfo *requestInfo, EmberCoapCode code)

  Sending a response that consists of just a code.
- EmberStatus emberCoapRespondWithPayload (const EmberCoapRequestInfo *requestInfo, EmberCoapCode code, const uint8_t *payload, uint16_t payloadLength)

  Sending a response that consists of a code and a payload.
- void emberSaveRequestInfo (const EmberCoapRequestInfo *from, EmberCoapRequestInfo *to)

  This function saves a `EmberCoapRequestInfo` for later use.
- void emberProcessCoap (const uint8_t *message, uint16_t messageLength, EmberCoapRequestInfo *info)

  This function processes a CoAP message received over an alternate transport.
- uint32_t emberBlockOptionOffset (EmberCoapBlockOption *option)
- bool emberReadBlockOption (EmberCoapReadOptions *options, EmberCoapOptionType type, EmberCoapBlockOption *option)
- void emberParseBlockOptionValue (uint32_t value, EmberCoapBlockOption *option)
- uint32_t emberBlockOptionValue (bool more, uint8_t logSize, uint32_t number)
- void emberInitCoapOption (EmberCoapOption *option, EmberCoapOptionType type, uint32_t value)
- bool emberVerifyBlockOption (const EmberCoapBlockOption *blockOption, uint16_t payloadLength, uint8_t expectedLogSize)
- EmberStatus emberCoapRequestNextBlock (EmberCoapCode code, const uint8_t *path, EmberCoapBlockOption *block2Option, EmberCoapResponseHandler responseHandler, EmberCoapResponseInfo *responseInfo)

6.12.1 Detailed Description

The CoAP API was updated for the Silicon Labs Thread 2.2 release. The new API is simpler and more flexible. These notes provide a brief overview of the key parts of the API to help customers migrate from the old API to the new API.

Sending messages

The old API had two functions for sending messages:

```
1 EmberStatus emberCoapSend(EmberCoapMessage *message,
2                           const uint8_t *uri,
3                           uint32_t responseTimeoutMs,
4                           EmberCoapResponseHandler responseHandler,
5                           void *applicationData,
6                           uint16_t applicationDataLength);
7
8 EmberStatus emberCoapSendUri(EmberCoapCode code,
9                               const EmberIpv6Address *destination,
10                               const uint8_t *uri,
11                               const uint8_t *body,
12                               uint16_t bodyLength,
13                               EmberCoapResponseHandler responseHandler);
```

If the responseHandler was NULL, the message was sent as a non-confirmable message (NON). Otherwise, the message was sent as a confirmable message (CON) and responseHandler was called when a response was received or a timeout occurred.

Now a single function is used for sending:

```
1 EmberStatus emberCoapSend(const EmberIpv6Address *destination,
2                           EmberCoapCode code,
3                           const uint8_t *path,
4                           const uint8_t *payload,
5                           uint16_t payloadLength,
6                           EmberCoapResponseHandler responseHandler,
7                           const EmberCoapSendInfo *info);
```

Common settings are passed directly as arguments to the function with other settings passed through a structure. CON or NON is determined by a field in the structure. Default settings, which include sending a CON to unicast addresses or NON to multicast addresses, can be obtained by zeroing the structure:

```
1 // Clear the structure to use default settings.
2 EmberCoapSendInfo info;
3 MEMSET(&info, 0, sizeof(EmberCoapSendInfo));
4 EmberStatus status = emberCoapSend(destination,
5                                    EMBER_COAP_CODE_GET,
6                                    "an/example/path",
7                                    payload,
8                                    payloadLength,
9                                    myResponseHandler,
10                                    &info);
```

The old API had a number of utility functions for sending specific types of requests:

```
1 EmberStatus emberCoapGet(const EmberIpv6Address *destination,
2                           const uint8_t *uri,
3                           EmberCoapResponseHandler responseHandler);
4
5 EmberStatus emberCoapPost(const EmberIpv6Address *destination,
6                            const uint8_t *uri,
7                            const uint8_t *body,
8                            uint16_t bodyLength,
9                            EmberCoapResponseHandler responseHandler);
10
11 EmberStatus emberCoapPostNonconfirmable(const EmberIpv6Address *destination,
12                                         const uint8_t *uri,
13                                         const uint8_t *body,
14                                         uint16_t bodyLength);
```

Similar utility functions exist in the new API.

```
1 EmberStatus emberCoapGet(const EmberIpv6Address *destination,
2                          const uint8_t *path,
3                          EmberCoapResponseHandler responseHandler,
4                          const EmberCoapSendInfo *info);
5
6 EmberStatus emberCoapPut(const EmberIpv6Address *destination,
7                          const uint8_t *path,
8                          const uint8_t *payload,
9                          uint16_t payloadLength,
10                          EmberCoapResponseHandler responseHandler,
11                          const EmberCoapSendInfo *info);
12
13 EmberStatus emberCoapPost(const EmberIpv6Address *destination,
14                           const uint8_t *path,
15                           const uint8_t *payload,
16                           uint16_t payloadLength,
17                           EmberCoapResponseHandler responseHandler,
18                           const EmberCoapSendInfo *info);
19
20 EmberStatus emberCoapDelete(const EmberIpv6Address *destination,
21                             const uint8_t *path,
22                             EmberCoapResponseHandler responseHandler,
23                             const EmberCoapSendInfo *info);
```

Receiving messages

The old API had a single handler for receiving messages:

```
1 void emberCoapMessageHandler(EmberCoapMessage *message);
```

All details of the message were contained in a single structure. Responses had to be sent from within the context of the handler.

The new API also has a single handler for receiving messages:

```
1 void emberCoapRequestHandler(EmberCoapCode code,
2                              uint8_t *uri,
3                              EmberCoapReadOptions *options,
4                              const uint8_t *payload,
5                              uint16_t payloadLength,
6                              const EmberCoapRequestInfo *info);
```

Common settings are passed directly as arguments to the handler, with other settings passed via structures. The EmberCoapRequestInfo structure is used to send a response to the request. By saving the contents of the structure, the application can send a delayed response to the message. If a response is sent within the context of the handler, the stack will send a piggybacked response, which has the acknowledgement and the response in a single message. If a response is not sent within the context of the handler, the stack will send an acknowledgement only. The application can send a separate response later.

Responding to messages

The old API had a number of functions for sending responses:

```
1 EmberStatus emberCoapRespond(EmberCoapCode responseCode,
2                              const uint8_t *payload,
3                              uint16_t payloadLength);
4
5 EmberStatus emberCoapRespondFormat(EmberCoapCode responseCode,
6                                    EmberCoapContentFormatType format,
7                                    const uint8_t *payload,
8                                    uint16_t payloadLength);
9
10 EmberStatus emberCoapRespondCreated(const uint8_t *locationPath);
```

Each of these were required to be called from within the context of emberCoapMessageHandler.

The new API has a single function for sending responses:

```
1 EmberStatus emberCoapRespondWithPath(const EmberCoapRequestInfo *requestInfo,
2                                       EmberCoapCode code,
3                                       const uint8_t *path,
4                                       const EmberCoapOption *options,
5                                       uint8_t numberOfOptions,
6                                       const uint8_t *payload,
7                                       uint16_t payloadLength);
```

The new API also has utility functions for sending specific types of responses.

```
1 EmberStatus emberCoapRespond(const EmberCoapRequestInfo *requestInfo,
2                               EmberCoapCode code,
3                               const EmberCoapOption *options,
4                               uint8_t numberOfOptions,
5                               const uint8_t *payload,
6                               uint16_t payloadLength);
7
8 EmberStatus emberCoapRespondWithCode(const EmberCoapRequestInfo *requestInfo,
9                                       EmberCoapCode code);
10
11 EmberStatus emberCoapRespondWithPayload(const EmberCoapRequestInfo *requestInfo,
12                                          EmberCoapCode code,
13                                          const uint8_t *payload,
14                                          uint16_t payloadLength);
```

The EmberCoapRequestInfo structure from the request is passed in when sending a response. To send a piggybacked response, call one of the response APIs from within the handler:

```
1 void emberCoapRequestHandler(EmberCoapCode code,
2                               uint8_t *uri,
3                               EmberCoapReadOptions *options,
4                               const uint8_t *payload,
5                               uint16_t payloadLength,
6                               const EmberCoapRequestInfo *info)
7 {
8   emberCoapRespondWithCode(info, EMBER_COAP_CODE_204_CHANGED);
9 }
```

To send a separate response, save the structure and use it to reply later. A utility function can be used to save the structure:

```
1 void emberSaveRequestInfo(const EmberCoapRequestInfo *from,
2                            EmberCoapRequestInfo *to);
3
4 static EmberCoapRequestInfo myInfo;
5 void emberCoapRequestHandler(EmberCoapCode code,
6                               uint8_t *uri,
7                               EmberCoapReadOptions *options,
8                               const uint8_t *payload,
9                               uint16_t payloadLength,
10                               const EmberCoapRequestInfo *info)
11 {
12   emberSaveRequestInfo(info, &myInfo);
13 }
14
15 static void sendResponse(void)
16 {
17   emberCoapRespondWithCode(&myInfo, EMBER_COAP_CODE_204_CHANGED);
18 }
```

6.12.2 Macro Definition Documentation

6.12.2.1 #define EMBER_COAP_DEFAULT_TIMEOUT_MS 90000

6.12.2.2 #define EMBER_COAP_MAX_TOKEN_LENGTH 8

6.12.2.3 #define EMBER_COAP_PORT 5683

6.12.2.4 #define EMBER_COAP_SECURE_PORT 5684

6.12.2.5 #define emberBlockOptionSize(*option*) (1 << (option)->logSize)

6.12.2.6 #define GET_COAP_CLASS(*code*) (((code) & 0xE0) >> 5)

6.12.2.7 #define GET_COAP_DETAIL(*code*) ((code) & 0x1F)

6.12.2.8 #define MAKE_COAP_CODE(*class, detail*) ((class << 5) | detail)

6.12.3 Typedef Documentation

6.12.3.1 typedef struct EmberCoapReadOptions_s EmberCoapReadOptions

The EmberCoapReadOptions type encapsulates the incoming options from a message. It is opaque to the application and contains an interal pointer that can be used to walk down the list of options received. Options can be read sequentially, using emberReadNextOption(), or individually, using emberReadIntegerOption() or emberReadBytesOption().

6.12.3.2 typedef void(* EmberCoapResponseHandler) (EmberCoapStatus status, EmberCoapCode code, EmberCoapReadOptions *options, uint8_t *payload, uint16_t payloadLength, EmberCoapResponseInfo *info)

The arguments`options`, `payload`, and `payloadLength` are meaningful only when `status` is `EMBER_COAP_MESSAGE_RESPONSE`.

6.12.3.3 typedef bool(* EmberCoapTransmitHandler) (const uint8_t *payload, uint16_t payloadLength, const EmberIpv6Address *localAddress, uint16_t localPort, const EmberIpv6Address *remoteAddress, uint16_t remotePort, void *transmitHandlerData)

CoAP messages can be sent via transports other than UDP, such as DTLS, by providing a function of this type. The CoAP code calls the provided function whenever a message needs to be sent.

The addresses, ports, and `transmitHandlerData` are the values that were passed to the original call to `emberCoapSend()` (if the message is a request) or to `emberProcessCoap()` (if the message is a response). Their values are specific to the underlying transport and need not be actual IPv6 addresses or ports.

Returns

true if transmission was initiated and false otherwise

6.12.4 Enumeration Type Documentation

6.12.4.1 enum EmberCoapClass

Enumerator

EMBER_COAP_CLASS_REQUEST

EMBER_COAP_CLASS_SUCCESS_RESPONSE

EMBER_COAP_CLASS_CLIENT_ERROR_RESPONSE

EMBER_COAP_CLASS_SERVER_ERROR_RESPONSE

6.12.4.2 enum EmberCoapCode

Enumerator

EMBER_COAP_CODE_EMPTY

EMBER_COAP_CODE_GET

EMBER_COAP_CODE_POST

EMBER_COAP_CODE_PUT

EMBER_COAP_CODE_DELETE

EMBER_COAP_CODE_201_CREATED

EMBER_COAP_CODE_202_DELETED

EMBER_COAP_CODE_203_VALID

EMBER_COAP_CODE_204_CHANGED

EMBER_COAP_CODE_205_CONTENT

EMBER_COAP_CODE_400_BAD_REQUEST

EMBER_COAP_CODE_401_UNAUTHORIZED

EMBER_COAP_CODE_402_BAD_OPTION

EMBER_COAP_CODE_403_FORBIDDEN

EMBER_COAP_CODE_404_NOT_FOUND

EMBER_COAP_CODE_405_METHOD_NOT_ALLOWED

EMBER_COAP_CODE_406_NOT_ACCEPTABLE

EMBER_COAP_CODE_412_PRECONDITION_FAILED

EMBER_COAP_CODE_413_REQUEST_ENTITY_TOO_LARGE

EMBER_COAP_CODE_415_UNSUPPORTED_CONTENT_FORMAT

EMBER_COAP_CODE_500_INTERNAL_SERVER_ERROR

EMBER_COAP_CODE_501_NOT_IMPLEMENTED

EMBER_COAP_CODE_502_BAD_GATEWAY

EMBER_COAP_CODE_503_SERVICE_UNAVAILABLE

EMBER_COAP_CODE_504_GATEWAY_TIMEOUT

EMBER_COAP_CODE_505_PROXYING_NOT_SUPPORTED

6.12.4.3 enum EmberCoapContentFormatType

Enumerator

EMBER_COAP_CONTENT_FORMAT_TEXT_PLAIN

EMBER_COAP_CONTENT_FORMAT_LINK_FORMAT

EMBER_COAP_CONTENT_FORMAT_XML

EMBER_COAP_CONTENT_FORMAT_OCTET_STREAM

EMBER_COAP_CONTENT_FORMAT_EXI

EMBER_COAP_CONTENT_FORMAT_JSON

EMBER_COAP_CONTENT_FORMAT_CBOR

EMBER_COAP_CONTENT_FORMAT_LINK_FORMAT_PLUS_CBOR

EMBER_COAP_CONTENT_FORMAT_NONE

6.12.4.4 enum EmberCoapOptionType

Enumerator

EMBER_COAP_NO_OPTION

EMBER_COAP_OPTION_IF_MATCH

EMBER_COAP_OPTION_URI_HOST

EMBER_COAP_OPTION_ETAG

EMBER_COAP_OPTION_IF_NONE_MATCH

EMBER_COAP_OPTION_OBSERVE

EMBER_COAP_OPTION_URI_PORT

EMBER_COAP_OPTION_LOCATION_PATH

EMBER_COAP_OPTION_URI_PATH

EMBER_COAP_OPTION_CONTENT_FORMAT

EMBER_COAP_OPTION_MAX_AGE

EMBER_COAP_OPTION_URI_QUERY

EMBER_COAP_OPTION_ACCEPT

EMBER_COAP_OPTION_LOCATION_QUERY

EMBER_COAP_OPTION_BLOCK2

EMBER_COAP_OPTION_BLOCK1

EMBER_COAP_OPTION_SIZE2

EMBER_COAP_OPTION_PROXY_URI

EMBER_COAP_OPTION_PROXY_SCHEME

EMBER_COAP_OPTION_SIZE1

6.12.4.5 enum EmberCoapStatus

For unicast requests EmberCoapResponseHandler() will usually be called exactly once, with one of the following three status values:

EMBER_COAP_MESSAGE_RESPONSE EMBER_COAP_MESSAGE_TIMED_OUT EMBER_COAP_MESSAGE↩ _RESET

If the server sends an ACK before any other response, EmberCoapResponseHandler() will be called twice, the first time with status EMBER_COAP_MESSAGE_ACKED and the second time with either status EMBER_CO↩ AP_MESSAGE_RESPONSE (if a response arrives after the ACK and before the timeout) or status EMBER_COA↩ P_MESSAGE_TIMED_OUT (if no response arrives after the ACK and before the timeout).

For multicast requests, EmberCoapResponseHandler() will be called with status EMBER_COAP_MESSA↩ GE_RESPONSE for each response that arrives and a final time with status EMBER_COAP_MESSAGE_TIMED↩ _OUT.

Enumerator

EMBER_COAP_MESSAGE_TIMED_OUT

EMBER_COAP_MESSAGE_ACKED

EMBER_COAP_MESSAGE_RESET

EMBER_COAP_MESSAGE_RESPONSE

6.12.5 Function Documentation

6.12.5.1 uint32_t emberBlockOptionOffset (EmberCoapBlockOption * *option*)

6.12.5.2 uint32_t emberBlockOptionValue (bool *more*, uint8_t *logSize*, uint32_t *number*)

6.12.5.3 EmberStatus emberCoapDelete (const EmberIpv6Address * *destination*, const uint8_t * *path*, EmberCoapResponseHandler *responseHandler*, const EmberCoapSendInfo * *info*)

See `emberCoapSend()` for more information.

6.12.5.4 EmberStatus emberCoapGet (const EmberIpv6Address * *destination*, const uint8_t * *path*, EmberCoapResponseHandler *responseHandler*, const EmberCoapSendInfo * *info*)

See `emberCoapSend()` for more information.

6.12.5.5 bool emberCoapIsClientErrorResponse (EmberCoapCode *code*)

6.12.5.6 bool emberCoapIsRequest (EmberCoapCode *code*)

6.12.5.7 bool emberCoapIsResponse (EmberCoapCode *code*)

6.12.5.8 bool emberCoapIsServerErrorResponse (EmberCoapCode *code*)

6.12.5.9 bool emberCoapIsSuccessResponse (EmberCoapCode *code*)

6.12.5.10 EmberStatus emberCoapPost (const EmberIpv6Address * *destination*, const uint8_t * *path*, const uint8_t * *payload*, uint16_t *payloadLength*, EmberCoapResponseHandler *responseHandler*, const EmberCoapSendInfo * *info*)

See `emberCoapSend()` for more information.

6.12.5.11 EmberStatus emberCoapPut (const EmberIpv6Address * *destination*, const uint8_t * *path*, const uint8_t * *payload*, uint16_t *payloadLength*, EmberCoapResponseHandler *responseHandler*, const EmberCoapSendInfo * *info*)

See `emberCoapSend()` for more information.

6.12.5.12 void emberCoapRequestHandler (EmberCoapCode *code*, uint8_t * *uri*, EmberCoapReadOptions * *options*, const uint8_t * *payload*, uint16_t *payloadLength*, const EmberCoapRequestInfo * *info*)

`info` can be passed as-is when sending an immediate response from within the call to `emberCoapRequestHandler()`. To send a delayed response, `info` data must be copied to a more permanent location using `emberSaveRequestInfo()`.

For more information, see stack/include/coap.h

`info` can be passed as-is when sending an immediate response from within the call to `emberCoapRequestHandler()`. To send a delayed response, the `info` data must be copied to a more permanent location using `emberSaveRequestInfo()`.

6.12.5.13 **EmberStatus** emberCoapRequestNextBlock (**EmberCoapCode** *code,* const uint8_t * *path,* **EmberCoapBlockOption** * *block2Option,* **EmberCoapResponseHandler** *responseHandler,* **EmberCoapResponseInfo** * *responseInfo*)

6.12.5.14 **EmberStatus** emberCoapRespond (const **EmberCoapRequestInfo** * *requestInfo,* **EmberCoapCode** *code,* const **EmberCoapOption** * *options,* uint8_t *numberOfOptions,* const uint8_t * *payload,* uint16_t *payloadLength*)

6.12.5.15 **EmberStatus** emberCoapRespondWithCode (const **EmberCoapRequestInfo** * *requestInfo,* **EmberCoapCode** *code*)

6.12.5.16 **EmberStatus** emberCoapRespondWithPath (const **EmberCoapRequestInfo** * *requestInfo,* **EmberCoapCode** *code,* const uint8_t * *path,* const **EmberCoapOption** * *options,* uint8_t *numberOfOptions,* const uint8_t * *payload,* uint16_t *payloadLength*)

6.12.5.17 **EmberStatus** emberCoapRespondWithPayload (const **EmberCoapRequestInfo** * *requestInfo,* **EmberCoapCode** *code,* const uint8_t * *payload,* uint16_t *payloadLength*)

6.12.5.18 **EmberStatus** emberCoapSend (const **EmberIpv6Address** * *destination,* **EmberCoapCode** *code,* const uint8_t * *path,* const uint8_t * *payload,* uint16_t *payloadLength,* **EmberCoapResponseHandler** *responseHandler,* const **EmberCoapSendInfo** * *info*)

Any response is passed to `responseHandler`. For unicast requests, at most one response is processed. For multicast requests, the response handler is called once for each response that arrives before the response timeout.

Returns

`EMBER_SUCCESS` if no errors occured.

6.12.5.19 **void** emberInitCoapOption (**EmberCoapOption** * *option,* **EmberCoapOptionType** *type,* uint32_t *value*)

6.12.5.20 **void** emberParseBlockOptionValue (uint32_t *value,* **EmberCoapBlockOption** * *option*)

6.12.5.21 **void** emberProcessCoap (const uint8_t * *message,* uint16_t *messageLength,* **EmberCoapRequestInfo** * *info*)

Called to process a CoAP message that arrives via DTLS or other alternative transport. Only the address, port, and transmit handler fields of `info` are used. The token and ackData fields are ignored.

This function processes a CoAP message received over an alternate transport.

Called to process a CoAP message that arrived via DTLS or other alternative transport. Only the address, port and transmit handler fields of `info` are used. The token and ackData fields are ignored.

6.12.5.22 **bool** emberReadBlockOption (**EmberCoapReadOptions** * *options,* **EmberCoapOptionType** *type,* **EmberCoapBlockOption** * *option*)

6.12.5.23 **bool** emberReadBytesOption (**EmberCoapReadOptions** * *options,* **EmberCoapOptionType** *type,* const uint8_t ** *valueLoc,* uint16_t * *valueLengthLoc*)

This function searches from the beginning of the options and leaves `option`s internal pointer pointing to the option following the one returned.

Returns

true if the option was found

6.12.5.24 bool emberReadIntegerOption (EmberCoapReadOptions * *options*, EmberCoapOptionType *type*, uint32_t * *valueLoc*)

This function searches from the beginning of the options and leaves `options` internal pointer pointing to the option following the one returned.

Returns

true if the option was found

6.12.5.25 int16_t emberReadLocationPath (EmberCoapReadOptions * *options*, uint8_t * *pathBuffer*, uint16_t *pathBufferLength*)

Any path options are copied to `pathBuffer` with a '/' between each pair of path elements.

If the path is longer than `pathBufferLength` only the first `pathBufferLength` bytes are stored in `pathBuffer`.

Returns

the length of the complete path

6.12.5.26 EmberCoapOptionType emberReadNextOption (EmberCoapReadOptions * *options*, const uint8_t ** *valuePointerLoc*, uint16_t * *valueLengthLoc*)

Reads the next option from an incoming message, advancing the internal pointer to the following option. Returns EMBER_COAP_NO_OPTION if there are no more options to read.

6.12.5.27 uint32_t emberReadOptionValue (const uint8_t * *value*, uint16_t *valuelength*)

This function should be passed the value and length returned by emberReadNextOption().

6.12.5.28 void emberResetReadOptionPointer (EmberCoapReadOptions * *options*)

6.12.5.29 void emberSaveRequestInfo (const EmberCoapRequestInfo * *from*, EmberCoapRequestInfo * *to*)

This saves necesary fields from `from` to `to` so that `to` may later be used to send a response.

6.12.5.30 bool emberVerifyBlockOption (const EmberCoapBlockOption * *blockOption*, uint16_t *payloadLength*, uint8_t *expectedLogSize*)

6.13 Callbacks

Functions

- void emberCoapRequestHandler (EmberCoapCode code, uint8_t *uri, EmberCoapReadOptions *options, const uint8_t *payload, uint16_t payloadLength, const EmberCoapRequestInfo *info)

  Callback for incoming requests.

6.13.1 Detailed Description

These callbacks were contributed by the coap API.

6.13.2 Function Documentation

6.13.2.1 void emberCoapRequestHandler (EmberCoapCode *code*, uint8_t * *uri*, EmberCoapReadOptions * *options*, const uint8_t * *payload*, uint16_t *payloadLength*, const EmberCoapRequestInfo * *info*)

For more information, see stack/include/coap.h

`info` can be passed as-is when sending an immediate response from within the call to `emberCoapRequestHandler()`. To send a delayed response, the `info` data must be copied to a more permanent location using `emberSaveRequestInfo()`.

6.14 Diagnostic Callbacks

6.15 DTLS API

Macros

- #define EMBER_DTLS_MODE_CERT 0x01

  Define the various modes of a DTLS connection.
- #define EMBER_DTLS_MODE_PSK 0x02
- #define EMBER_DTLS_MODE_PKEY 0x04

Typedefs

- typedef uint8_t EmberDtlsMode

Functions

- void emberSetDtlsDeviceCertificate (const CertificateAuthority **certAuthority, const DeviceCertificate *deviceCert)

  Set a device certificate to be used to create a certificate based secure session on the application. The expected arguments are DER encoded X.509 certificates. If this succeeds, emberSetDtlsDeviceCertificateReturn should return 0.
- void emberSetDtlsDeviceCertificateReturn (uint32_t result)

  Provides the result of a call to emberSetDtlsDeviceCertificate().
- void emberSetDtlsPresharedKey (const uint8_t *key, uint8_t keyLength, const EmberIpv6Address *remote↩ Address)

  Set a key to be used to create a PSK based secure session on the application. The maximum length of the key is 32 bytes.
- void emberSetDtlsPresharedKeyReturn (EmberStatus status)

  Provides the result of a call to emberSetDtlsPresharedKey().
- void emberOpenDtlsConnection (EmberDtlsMode dtlsMode, const EmberIpv6Address *remoteAddress, uint16_t localPort, uint16_t remotePort)

  Establish a DTLS connection with a peer on the Thread network. When established, this session can be used to send secure CoAP data. The device requesting the connection acts as a DTLS client.
- void emberOpenDtlsConnectionReturn (uint32_t result, const EmberIpv6Address *remoteAddress, uint16_t localPort, uint16_t remotePort)

  Provides the result of a call to emberOpenDtlsConnection().
- void emberDtlsSecureSessionEstablished (uint8_t flags, uint8_t sessionId, const EmberIpv6Address *local↩ Address, const EmberIpv6Address *remoteAddress, uint16_t localPort, uint16_t remotePort)

  Indicates to the application that a secure connection was successfully established.
- void emberGetSecureDtlsSessionId (const EmberIpv6Address *remoteAddress, uint16_t localPort, uint16↩ _t remotePort)

  Request the session ID given connection parameters.
- void emberGetSecureDtlsSessionIdReturn (uint8_t sessionId, const EmberIpv6Address *remoteAddress, uint16_t localPort, uint16_t remotePort)

  Provides the result of a call to emberGetSecureDtlsSessionId().
- void emberCloseDtlsConnection (uint8_t sessionId)

  Close a currently active secure session on the application. When successful, emberCloseDtlsConnectionReturn should be called on both ends of the connection with EMBER_SUCCESS.
- void emberCloseDtlsConnectionReturn (uint8_t sessionId, EmberStatus status)

  Provides the result of a call to emberCloseDtlsConnection(), or indicates that the connection was closed on the other end.
- bool emberDtlsTransmitHandler (const uint8_t *payload, uint16_t payloadLength, const EmberIpv6Address *localAddress, uint16_t localPort, const EmberIpv6Address *remoteAddress, uint16_t remotePort, void *transmitHandlerData)

  Public DTLS transmit handler to be set in emberCoapSend. The secure payload is delivered via emberProcessCoap on the other end, with a matching session ID in the transmitHandlerData of its CoapRequestInfo. See ember↩ ProcessCoap (stack/include/coap.h)

6.15.1 Detailed Description

See dtls.h for source code.

6.15.2 Macro Definition Documentation

6.15.2.1 #define EMBER_DTLS_MODE_CERT 0x01

Note: Please configure either the CERT or PSK modes, as the public key option is currently unavailable.

6.15.2.2 #define EMBER_DTLS_MODE_PKEY 0x04

6.15.2.3 #define EMBER_DTLS_MODE_PSK 0x02

6.15.3 Typedef Documentation

6.15.3.1 typedef uint8_t EmberDtlsMode

6.15.4 Function Documentation

6.15.4.1 void emberCloseDtlsConnection (uint8_t *sessionId*)

Parameters

sessionId	sessionId used for secure CoAP transport.

6.15.4.2 void emberCloseDtlsConnectionReturn (uint8_t *sessionId,* EmberStatus *status*)

Parameters

sessionId	sessionId used for secure CoAP transport.
status	• EMBER_SUCCESS - Successfully closed the connection • EMBER_INVALID_CALL - Invalid session ID • EMBER_ERR_FATAL - Fatal error closing the connection

6.15.4.3 void emberDtlsSecureSessionEstablished (uint8_t *flags,* uint8_t *sessionId,* const EmberIpv6Address * *localAddress,* const EmberIpv6Address * *remoteAddress,* uint16_t *localPort,* uint16_t *remotePort*)

Parameters

flags	1 = server, 0 = client (possibly other info later)

Parameters

sessionId	sessionId used for secure CoAP transport
localAddress	local IPv6 address
remoteAddress	remote IPv6 address
localPort	local port
remotePort	remote port

6.15.4.4 bool emberDtlsTransmitHandler (const uint8_t * *payload*, uint16_t *payloadLength*, const EmberIpv6Address * *localAddress*, uint16_t *localPort*, const EmberIpv6Address * *remoteAddress*, uint16_t *remotePort*, void * *transmitHandlerData*)

Parameters

payload	CoAP payload to be sent securely
payloadLength	payload length
localAddress	local IPv6 address
localPort	local port
remoteAddress	remote IPv6 address
remotePort	remote port
transmitHandlerData	session ID of the secure connection (see emberDtlsSecureSessionEstablished or emberGetSecureDtlsSessionId above)

6.15.4.5 void emberGetSecureDtlsSessionId (const EmberIpv6Address * *remoteAddress*, uint16_t *localPort*, uint16_t *remotePort*)

Parameters

remoteAddress	remote IPv6 address
localPort	local port
remotePort	remote port

6.15.4.6 void emberGetSecureDtlsSessionIdReturn (uint8_t *sessionId*, const EmberIpv6Address * *remoteAddress*, uint16_t *localPort*, uint16_t *remotePort*)

Parameters

sessionId	sessionId used for secure CoAP transport
remoteAddress	remote IPv6 address
localPort	local port
remotePort	remote port

6.15.4.7 void emberOpenDtlsConnection (EmberDtlsMode *dtlsMode*, const EmberIpv6Address * *remoteAddress*, uint16_t *localPort*, uint16_t *remotePort*)

(For DotDot applications, the local port and remote port are both EMBER_COAP_SECURE_PORT)

Parameters

dtlsMode	DTLS connection mode (see EMBER_DTLS_MODE_∗ above)
remoteAddress	IPv6 address of the server
localPort	local port
remotePort	remote port

6.15.4.8 void emberOpenDtlsConnectionReturn (uint32_t *result*, const EmberIpv6Address ∗ *remoteAddress*, uint16_t *localPort*, uint16_t *remotePort*)

Parameters

result	error code • an EmberStatus value if using Silicon Labs TLS • an mbed TLS error code if using mbed TLS library (see mbedtls:include/mbedtls/ssl.h)
remoteAddress	IPv6 address of the server
localPort	local port
remotePort	remote port

6.15.4.9 void emberSetDtlsDeviceCertificate (const CertificateAuthority ∗∗ *certAuthority*, const DeviceCertificate ∗ *deviceCert*)

Parameters

certAuthority	the certificate authority
deviceCert	the certificate

6.15.4.10 void emberSetDtlsDeviceCertificateReturn (uint32_t *result*)

Parameters

result	• ::0 The certificate was set successfully. • ::result error code – an EmberStatus value if using Silicon Labs TLS – an mbed TLS error code if using mbed TLS library (see mbedtls:include/mbedtls/ssl.h)

6.15.4.11 void emberSetDtlsPresharedKey (const uint8_t ∗ *key*, uint8_t *keyLength*, const EmberIpv6Address ∗ *remoteAddress*)

Note: Up to 32 pre-shared keys can be stored.

If this succeeds, emberSetDtlsPresharedKeyReturn will return EMBER_SUCCESS. Otherwise, a failure status is indicated.

Parameters

key	the pre-shared key
keyLength	length
remoteAddress	IPv6 address of peer

6.15.4.12 void emberSetDtlsPresharedKeyReturn (EmberStatus *status*)

Parameters

result	• ::status An EmberStatus value

6.16 Command Interpreter

The command interpreter.

Data Structures

- struct EmberCommandEntry

  Command entry for a command table.
- struct EmberCommandState

  For use when declaring a separate command streams. The fields are not accessed directly by the application.

Macros

- #define MAX_TOKEN_COUNT (EMBER_MAX_COMMAND_ARGUMENTS + 1)
- #define MAX_COMMAND_TABLE_NESTING 16
- #define emberBinaryCommand emberBinaryCommandEntryAction
- #define emberBinaryNestedCommand emberBinaryCommandEntrySubMenu
- #define emberBinaryCommandEntryAction(identifier, command, arguments, description) { { (PGM_↩P)identifier }, command, arguments, description }
- #define emberBinaryCommandEntrySubMenu(identifier, nestedCommands, description) { { (PGM_↩P)identifier }, NULL, description, nestedCommands }
- #define emberCommandEntryAction(name, command, arguments, description) { { name }, command, arguments, description }
- #define emberCommandEntrySubMenu(name, nestedCommands, description) { { name }, NULL, (PGM_↩P)nestedCommands, description }
- #define emberCommandEntryTerminator() { { NULL }, NULL, NULL, NULL }
- #define emberCommand emberCommandEntryAction
- #define emberNestedCommand emberCommandEntrySubMenu
- #define emberProcessCommandInput(port) emberProcessCommandString(NULL, port)

  This function processes input coming in on the given serial port.

Typedefs

- typedef void(* CommandAction) (void)
- typedef void EmberCommandErrorHandler(EmberCommandStatus status, EmberCommandEntry *command)

  Type of error handlers; the command argument is currently always NULL.

Enumerations

- enum EmberCommandStatus {
  EMBER_CMD_SUCCESS,
  EMBER_CMD_ERR_PORT_PROBLEM,
  EMBER_CMD_ERR_NO_SUCH_COMMAND,
  EMBER_CMD_ERR_WRONG_NUMBER_OF_ARGUMENTS,
  EMBER_CMD_ERR_ARGUMENT_OUT_OF_RANGE,
  EMBER_CMD_ERR_ARGUMENT_SYNTAX_ERROR,
  EMBER_CMD_ERR_STRING_TOO_LONG,
  EMBER_CMD_ERR_INVALID_ARGUMENT_TYPE }

  Command error states.

Functions

- void emberCommandErrorHandler (EmberCommandStatus status, EmberCommandEntry *command)
- void emberPrintCommandUsage (EmberCommandEntry *entry)
- void emberPrintCommandUsageNotes (void)
- void emberPrintCommandTable (void)
- void emberCommandClearBuffer (void)
- void emberCommandReaderInit (void)

  This function nitializes the command interpreter.
- bool emberProcessCommandString (const uint8_t *input, uint16_t sizeOrPort)

  This function processes the given string as a command.
- void emberInitializeCommandState (EmberCommandState *state)

  This function must be called to initialize a command state before passing it to emberRunBinaryCommandInterpreter() or emberRunAsciiCommandInterpreter().
- bool emberRunBinaryCommandInterpreter (EmberCommandState *state, EmberCommandEntry *commands, EmberCommandErrorHandler *errorHandler, const uint8_t *input, uint16_t sizeOrPort)

  For use to process binary commands when additional different command streams are being used.
- bool emberRunAsciiCommandInterpreter (EmberCommandState *state, EmberCommandEntry *commands, EmberCommandErrorHandler *errorHandler, const uint8_t *input, uint16_t sizeOrPort)

  For use to process ASCII commands when additional different command streams are being used.
- void emberCommandReaderSetDefaultBase (uint8_t base)

Variables

- EmberCommandEntry emberCommandTable [ ]

Command Table Settings

- #define EMBER_MAX_COMMAND_ARGUMENTS 14
- #define EMBER_COMMAND_BUFFER_LENGTH 100
- #define EMBER_CUSTOM_COMMAND_BUFFER_LENGTH (EMBER_COMMAND_BUFFER_LENGTH - 3)
- #define EMBER_COMMAND_INTERPRETER_HAS_DESCRIPTION_FIELD
- #define EMBER_COMMAND_INTERPRETER_NO_ERROR_MESSAGE

Functions to Retrieve Arguments

Use the following functions in your functions that process commands to retrieve arguments from the command interpreter. These functions pull out unsigned integers, signed integers, and strings, and hex strings. Index 0 is the first command argument.

- uint8_t emberCommandArgumentCount (void)
- uint32_t emberUnsignedCommandArgument (uint8_t argNum)
- int32_t emberSignedCommandArgument (uint8_t argNum)
- uint8_t * emberStringCommandArgument (int8_t argNum, uint8_t *length)
- uint8_t * emberLongStringCommandArgument (int8_t argNum, uint16_t *length)
- const char * emberCommandName (void)
- uint8_t emberGetStringArgument (int8_t argNum, uint8_t *destination, uint8_t maxLength, bool leftPad)
- bool emberGetIpArgument (uint8_t index, uint8_t *target)
- bool emberGetIpv6AddressArgument (uint8_t index, EmberIpv6Address *dst)

  This function parses an IPv6 address in a CLI command.
- bool emberGetIpv6PrefixArgument (uint8_t index, EmberIpv6Address *dst, uint8_t *dstPrefixBits)

  This function parses an IPv6 prefix in a CLI command.
- void emberGetExtendedPanIdArgument (int8_t index, uint8_t *extendedPanId)
- #define emberGetKeyArgument(index, keyDataPointer)
- #define emberGetEui64Argument(index, eui64) emberGetExtendedPanIdArgument(index, (eui64)->bytes)

6.16.1 Detailed Description

Processes commands coming from the serial port.

Interpret serial port commands. See command-interpreter2.c for source code.

See the following application usage example followed by a brief explanation.

```
1 // Usage: network form 22 0xAB12 -3 { 00 01 02 A3 A4 A5 A6 A7 }
2 void formCommand(void)
3 {
4   uint8_t channel = emberUnsignedCommandArgument(0);
5   uint16_t panId  = emberUnsignedCommandArgument(1);
6   int8_t power    = emberSignedCommandArgument(2);
7   uint8_t length;
8   uint8_t *eui64  = emberStringCommandArgument(3, &length);
9   ...
10   ... call emberFormNetwork() etc
11   ...
12 }
13
14 // The table of network commands.
15 EmberCommandEntry networkCommands[] = {
16   emberCommandEntryAction("form", formCommand, "uvsh", "Form a network"),
17   emberCommandEntryAction("join", joinCommand, "uvsh", "join a network"),
18   ...
19   emberCommandEntryTerminator()
20 };
21
22 EmberCommandEntry systemCommands[] = {
23   emberCommandEntrySubMenu("network",  networkCommands, "Network form/join commands"),
24   ...
25   emberCommandEntryTerminator()
26 };
27
28 // The main command table.  A SubMenu with an empty command name processes
29 // the commands in the subtable as if they were in the main table.
30 EmberCommandEntry emberCommandTable[] = {
31   emberCommandEntrySubMenu("",  systemCommands, ""),
32   emberCommandEntryAction("status",    statusCommand,   "Prints application status),
33   ...
34   emberCommandEntryTerminator()
35 };
36
37 void main(void)
38 {
39    emberCommandReaderInit();
40    while(0) {
41      ...
42      // Process input and print prompt if it returns true.
43      if (emberProcessCommandInput(serialPort)) {
44         emberSerialPrintf(1, "%p>", PROMPT);
45      }
46      ...
47    }
48 }
```

1. Applications specify the commands that can be interpreted by defining the emberCommandTable array of type EmberCommandEntry. The table includes the following information for each command:
   (a) The full command name.
   (b) Your application's function name that implements the command.
   (c) An EmberCommandEntry::argumentTypes string specifies the number and types of arguments the command accepts. See ::argumentTypes for details.
   (d) A description string explains the command.
2. A default error handler emberCommandErrorHandler() is provided to deal with incorrect command input. Applications may override it.
3. The application calls emberCommandReaderInit() to initalize, and emberProcessCommandInput() in its main loop.

4. Within the application's command functions, use emberXXXCommandArgument() functions to retrieve command arguments.

The command interpreter does extensive processing and validation of the command input before calling the function that implements the command. It checks that the number, type, syntax, and range of all arguments are correct. It performs any conversions necessary (for example, converting integers and strings input in hexadecimal notation into the corresponding bytes), so that no additional parsing is necessary within command functions. If there is an error in the command input, emberCommandErrorHandler() is called rather than a command function.

The command interpreter allows inexact matches of command names. The input command may be either shorter or longer than the actual command. However, if more than one inexact match is found and there is no exact match, an error of type EMBER_CMD_ERR_NO_SUCH_COMMAND will be generated. To disable this feature, define EMBER_REQUIRE_EXACT_COMMAND_NAME in the application configuration header.

6.16.2 Macro Definition Documentation

6.16.2.1 #define EMBER_COMMAND_BUFFER_LENGTH 100

The maximum number of arguments a command can have. A nested command counts as an argument.

6.16.2.2 #define EMBER_COMMAND_INTERPRETER_HAS_DESCRIPTION_FIELD

Whether or not the command entry structure will include descriptions for the commands and error information. This consumes additional CONST space. By default descriptions are not included.

6.16.2.3 #define EMBER_COMMAND_INTERPRETER_NO_ERROR_MESSAGE

Whether or not error messages are included. This consumes additional CONST space. By default error messages are included.

6.16.2.4 #define EMBER_CUSTOM_COMMAND_BUFFER_LENGTH (EMBER_COMMAND_BUFFER_LENGTH - 3)

The maximum message size for custom commands reserves three bytes for the length (1 bytes) and the custom command identifier (2 bytes).

6.16.2.5 #define EMBER_MAX_COMMAND_ARGUMENTS 14

The maximum number of arguments a command can have. A nested command counts as an argument.

6.16.2.6 #define emberBinaryCommand emberBinaryCommandEntryAction

6.16.2.7 #define emberBinaryCommandEntryAction(*identifier, command, arguments, description*) { { (PGM_P)identifier }, command, arguments, description }

6.16.2.8 #define emberBinaryCommandEntrySubMenu(*identifier, nestedCommands, description*) { { (PGM_P)identifier }, NULL, description, nestedCommands }

6.16.2.9 #define emberBinaryNestedCommand emberBinaryCommandEntrySubMenu

6.16.2.10 #define emberCommand emberCommandEntryAction

6.16.2.11 #define emberCommandEntryAction(*name, command, arguments, description*) { { name }, command, arguments, description }

6.16.2.12 #define emberCommandEntrySubMenu(*name, nestedCommands, description*) { { name }, NULL, (PGM_P)nestedCommands, description }

6.16.2.13 #define emberCommandEntryTerminator( ) { { NULL }, NULL, NULL, NULL }

6.16.2.14 #define emberGetEui64Argument(*index, eui64*) emberGetExtendedPanIdArgument(index, (eui64)->bytes)

This function copies the EUI64 argument to the given EmberEui64 destination, reversing the bytes. EUI64's are stored little endian so reversing the bytes means they are big endian in the input command string.

6.16.2.15 #define emberGetKeyArgument(*index, keyDataPointer*)

Value:

```
(emberGetStringArgument((index),                            \
                        emberKeyContents((keyDataPointer)), \
                        EMBER_ENCRYPTION_KEY_SIZE,          \
                        true))
```

A convenience macro for copying security key arguments to an EmberKeyData pointer.

6.16.2.16 #define emberNestedCommand emberCommandEntrySubMenu

6.16.2.17 #define emberProcessCommandInput(*port*) emberProcessCommandString(NULL, port)

Returns

true if an end of line character was read. If the application uses a command line prompt, this indicates it is time to print the prompt.

```
1 void emberProcessCommandInput(uint8_t port);
```

6.16.2.18 #define MAX_COMMAND_TABLE_NESTING 16

6.16.2.19 #define MAX_TOKEN_COUNT (EMBER_MAX_COMMAND_ARGUMENTS + 1)

6.16.3 Typedef Documentation

6.16.3.1 typedef void(* CommandAction) (void)

6.16.3.2 typedef void EmberCommandErrorHandler(EmberCommandStatus status, EmberCommandEntry *command)

6.16.4 Enumeration Type Documentation

6.16.4.1 enum EmberCommandStatus

If you change this list, ensure you also change the strings that describe these errors in the array emberCommand↩ ErrorNames[] in command-interpreter2-error.c.

Enumerator

EMBER_CMD_SUCCESS

EMBER_CMD_ERR_PORT_PROBLEM

EMBER_CMD_ERR_NO_SUCH_COMMAND

EMBER_CMD_ERR_WRONG_NUMBER_OF_ARGUMENTS

EMBER_CMD_ERR_ARGUMENT_OUT_OF_RANGE

EMBER_CMD_ERR_ARGUMENT_SYNTAX_ERROR

EMBER_CMD_ERR_STRING_TOO_LONG

EMBER_CMD_ERR_INVALID_ARGUMENT_TYPE

6.16.5 Function Documentation

6.16.5.1 uint8_t emberCommandArgumentCount (void)

This function returns the number of arguments for the current command.

6.16.5.2 void emberCommandClearBuffer (void)

6.16.5.3 void emberCommandErrorHandler (EmberCommandStatus *status,* EmberCommandEntry * *command*)

6.16.5.4 const char* emberCommandName (void)

A convenience macro for copying security key arguments to an EmberKeyData pointer.

6.16.5.5 void emberCommandReaderInit (void)

6.16.5.6 void emberCommandReaderSetDefaultBase (uint8_t *base*)

6.16.5.7 void emberGetExtendedPanIdArgument (int8_t *index*, uint8_t * *extendedPanId*)

This function copies the extended PAN ID argument to the given destination, reversing the bytes. Extended PAN ids are stored little endian so reversing the bytes means they are big endian in the input command string.

6.16.5.8 bool emberGetIpArgument (uint8_t *index*, uint8_t * *target*)

This function parses and returns, via target, an IP address at the provided index. Returns true if an IP address was successfully parsed Return false otherwise

6.16.5.9 bool emberGetIpv6AddressArgument (uint8_t *index*, EmberIpv6Address * *dst*)

Parameters

index	An index of the IPv6 address to parse
dst	the EmberIpv6Address a location where the address will be written

Returns

true if the IPv6 address was successfully parsed.

6.16.5.10 bool emberGetIpv6PrefixArgument (uint8_t *index*, EmberIpv6Address * *dst*, uint8_t * *dstPrefixBits*)

Parameters

index	An index of the IPv6 prefix to parse
dst	the EmberIpv6Address a location where the address will be written
prefixBits	the number of prefix bits in the string

Returns

true if the IPv6 prefix was successfully parsed.

6.16.5.11 uint8_t emberGetStringArgument (int8_t *argNum*, uint8_t * *destination*, uint8_t *maxLength*, bool *leftPad*)

This function copies the string argument to the given destination up to maxLength. If the argument length is nonzero but less than maxLength and leftPad is true, leading zeroes are prepended to bring the total length of the target up to maxLength. If the argument is longer than the maxLength, it is truncated to maxLength. This function returns the minimum of the argument length and maxLength. ASCII strings are null terminated, but the null terminator is not included in the returned length.

This function is commonly used for reading in hexadecimal strings such as EUI64 or key data and left padding them with zeroes. See emberGetKeyArgument and emberGetEui64Argument for convenience macros for this purpose.

6.16.5.12 void emberInitializeCommandState (EmberCommandState * *state*)

6.16.5.13 uint8_t* emberLongStringCommandArgument (int8_t *argNum,* uint16_t * *length*)

A convenience macro for copying security key arguments to an EmberKeyData pointer.

6.16.5.14 void emberPrintCommandTable (void)

6.16.5.15 void emberPrintCommandUsage (EmberCommandEntry * *entry*)

6.16.5.16 void emberPrintCommandUsageNotes (void)

6.16.5.17 bool emberProcessCommandString (const uint8_t * *input,* uint16_t *sizeOrPort*)

6.16.5.18 bool emberRunAsciiCommandInterpreter (EmberCommandState * *state,* EmberCommandEntry * *commands,* EmberCommandErrorHandler * *errorHandler,* const uint8_t * *input,* uint16_t *sizeOrPort*)

6.16.5.19 bool emberRunBinaryCommandInterpreter (EmberCommandState * *state,* EmberCommandEntry * *commands,* EmberCommandErrorHandler * *errorHandler,* const uint8_t * *input,* uint16_t *sizeOrPort*)

6.16.5.20 int32_t emberSignedCommandArgument (uint8_t *argNum*)

This function retrieves signed integer arguments.

6.16.5.21 uint8_t* emberStringCommandArgument (int8_t *argNum,* uint8_t * *length*)

This function retrieves quoted string or hexadecimal string arguments. Hexadecimal strings have already been converted into binary. ASCII strings have been null terminated. The null terminator is not included in the returned length argument. To retrieve the name of the command, use an argNum of -1. For example, to retrieve the first character of the command, do: uint8_t firstChar = emberStringCommandArgument(-1, NULL)[0]. If the command is nested, an index of -2, -3, will work to retrieve the higher level command names. Note that [-1] only returns the text entered. If an abbreviated command name is entered, only the text entered will be returned with [-1].

6.16.5.22 uint32_t emberUnsignedCommandArgument (uint8_t *argNum*)

This function retrieves unsigned integer arguments.

6.16.6 Variable Documentation

6.16.6.1 EmberCommandEntry emberCommandTable[ ]

6.17 Debugging Utilities

Debugging Utilities.

Macros

- #define NO_DEBUG 0
- #define BASIC_DEBUG 1
- #define FULL_DEBUG 2
- #define emberDebugInit(port) do {} while (false)

  This function is obsolete and no longer required to initialize the debug system.

Functions

- void emberDebugAssert (PGM_P filename, int linenumber)

  Prints the filename and line number to the debug serial port.
- void emberDebugMemoryDump (uint8_t *start, uint8_t *end)

  Prints the contents of RAM to the debug serial port.
- void emberDebugBinaryPrintf (PGM_P formatString,...)

  Prints binary data to the debug channel.
- void emDebugSendVuartMessage (const uint8_t *buff, uint8_t len)

  An internal debug command used by the HAL to send vuart data out the the debug channel.
- void emberDebugError (EmberStatus code)

  Prints an EmberStatus return code to the serial port.
- bool emberDebugReportOff (void)

  Turns off all debug output.
- void emberDebugReportRestore (bool state)

  Restores the state of the debug output.
- void emberDebugPrintf (PGM_P formatString,...)

  Prints text debug messages.

6.17.1 Detailed Description

A set of utilities for printing to the debug backchannel.

See ember-debug.h for source code.

6.17.2 Macro Definition Documentation

6.17.2.1 #define BASIC_DEBUG 1

6.17.2.2 #define emberDebugInit(*port*) do {} while (false)

Parameters

port	Ignored because the port used for debug communication is automatically determined for each platform.

6.17.2.3 #define FULL_DEBUG 2

6.17.2.4 #define NO_DEBUG 0

6.17.3 Function Documentation

6.17.3.1 void emberDebugAssert (PGM_P *filename,* int *linenumber*)

Parameters

filename	The name of the file where the assert occurred.
linenumber	The line number in the file where the assert occurred.

6.17.3.2 void emberDebugBinaryPrintf (PGM_P *formatString,* ...)

This function does not use the normal printf format conventions. To print text debug messages, use emberDebug↩ Printf(). The format string must contain only these conversion specification characters:

- B - uint8_t value.
- W - uint16_t value, printed least significant byte first.
- D - uint32_t value, printed least significant byte first.
- F - pointer to null terminated string in Flash (PGM_P).
- xxxp - pointer to RAM, length is xxx (max 255).
- lp - pointer to RAM, length is uint8_t argument.
- xxxf - pointer to Flash (PGM_P), length is xxx (max 255).
- lf - pointer to Flash (PGM_P), length is uint8_t argument.
- b - EmberMessageBuffer.

Examples:

```
1 emberDebugBinaryPrintf("BWD", status, panId, channelMask);
2 emberDebugBinaryPrintf("F8p", "string example", eui64);
3 emberDebugBinaryPrintf("lp64fb", length, bytes, dataTable, buffer);
```

Parameters

formatString	A string of conversion specification characters describing the arguments to be printed.
...	The arguments to be printed.

6.17.3.3 void emberDebugError (EmberStatus *code*)

Parameters

code	The EmberStatus code to print.

6.17.3.4 void emberDebugMemoryDump (uint8_t * *start*, uint8_t * *end*)

Parameters

start	The start address of the block of RAM to dump.
end	The end address of the block of RAM to dump (address of the last byte).

6.17.3.5 void emberDebugPrintf (PGM_P *formatString*, ...)

Parameters

formatString	Takes the following:

%%	Percent sign
%c	Single-byte char
%s	RAM string
%p	Flash string (does not follow the printf standard)
%u	Two-byte unsigned decimal
%d	Two-byte signed decimal
%x, %%2x, %%4x	1-, 2-, 4-byte hex value (always 0 padded; does not follow the printf standard)

6.17.3.6 bool emberDebugReportOff (void)

Returns

The current state (true for on, false for off).

6.17.3.7 void emberDebugReportRestore (bool *state*)

Parameters

state	The state returned from emberDebugReportOff(). This is done so that debug output is not blindly turned on.

6.17.3.8 void emDebugSendVuartMessage (const uint8_t * *buff*, uint8_t *len*)

Parameters

buff	A pointer to the data to send
len	Lenght of the data to send

6.18 MFGLIB

Manufacturing Library.

Functions

- MfgStatus mfglibStart (void(∗mfglibRxCallback)(uint8_t ∗packet, uint8_t linkQuality, int8_t rssi))

  Activates use of MFGLIB test routines and enables the radio receiver to report packets it receives to the caller-specified mfglibRxCallback() routine.
- MfgStatus mfglibEnd (void)

  Deactivates use of MFGLIB test routines.
- MfgStatus mfglibStartTone (void)

  Starts transmitting the tone feature of the radio.
- MfgStatus mfglibStopTone (void)

  Stops transmitting a tone started by mfglibStartTone().
- MfgStatus mfglibStartStream (void)

  Starts transmitting a random stream of characters. This is so that the radio modulation can be measured.
- MfgStatus mfglibStopStream (void)

  Stops transmitting a random stream of characters started by mfglibStartStream().
- MfgStatus mfglibSendPacket (uint8_t ∗packet, uint16_t repeat)

  Sends a single packet, (repeat + 1) times.
- MfgStatus mfglibSetChannel (uint8_t channel)

  Selects the radio channel. The channel range is from 11 to 26.
- MfgStatus_U mfglibGetChannel (void)

  Get the current radio channel, as previously set via mfglibSetChannel().
- MfgStatus mfglibSetPower (uint16_t txPowerMode, int8_t power)

  Set the transmit power mode and the radio transmit power.
- MfgStatus_S mfglibGetPower (void)

  Get the current radio power setting as previously set via mfglibSetPower().
- MfgStatus_UU mfglibGetPowerMode (void)

  Get the radio transmit power mode setting as previously set via mfglibSetPower().
- void mfglibSetSynOffset (int8_t synOffset)

  Set the synth offset in 11.7kHz steps.
- MfgStatus_S mfglibGetSynOffset (void)

  Get the current synth offset in 11.7kHz steps.
- void mfglibTestContModCal (uint8_t channel, uint32_t duration)

  Run mod DAC calibration on the given channel for the given amount of time.
- MfgStatus mfglibSetOptions (uint8_t options)

  Set manufacturing library options.
- MfgStatus_U mfglibGetOptions (void)

  Get the current manufacturing library options, as previously set via mfglibSetOptions().
- void mfglibStartReturn (EmberStatus status)

  This function provides the result of a call to mfglibStart().
- void mfglibEndReturn (EmberStatus status, uint32_t receiveCount)

  This function provides the result of a call to mfglibEnd().
- void mfglibStartToneReturn (EmberStatus status)

  This function provides the result of a call to mfglibStartTone().
- void mfglibStopToneReturn (EmberStatus status)

  This function provides the result of a call to mfglibStopTone().
- void mfglibStartStreamReturn (EmberStatus status)

This function provides the result of a call to mfglibStartStream().

- void mfglibStopStreamReturn (EmberStatus status)

  This function provides the result of a call to mfglibStopStream().
- void mfglibSendPacketReturn (EmberStatus status)

  This function provides the result of a call to mfglibSendPacket().
- void mfglibSetChannelReturn (EmberStatus status)

  This function provides the result of a call to mfglibSetChannel().
- void mfglibGetChannelReturn (uint8_t channel)

  This function provides the result of a call to mfglibGetChannel().
- void mfglibSetPowerReturn (EmberStatus status)

  This function provides the result of a call to mfglibSetPower().
- void mfglibGetPowerReturn (int8_t power)

  This function provides the result of a call to mfglibGetPower().
- void mfglibGetPowerModeReturn (uint16_t txPowerMode)

  This function provides the result of a call to mfglibGetPowerMode().
- void mfglibGetSynOffsetReturn (int8_t synthOffset)

  This function provides the result of a call to mfglibGetSynOffset().
- void mfglibSetOptionsReturn (EmberStatus status)

  This function provides the result of a call to mfglibSetOptions().
- void mfglibGetOptionsReturn (uint8_t options)

  This function provides the result of a call to mfglibGetOptions().
- void mfglibRxHandler (uint8_t *packet, uint8_t linkQuality, int8_t rssi)

  RX Handler for the mfglib test library.

6.18.1 Detailed Description

This is a manufacturing and functional test library for testing and verifying the RF component of products at manufacture time.

See mfglib.h for source code.

Developers can optionally include this library in their application code. The goal is that in most cases, this will eliminate the need for developers to load multiple images into their hardware at manufacturing time.

This library can optionally be compiled into the developer's production code and run at manufacturing time. Any interface to the library is handled by the application.

This library cannot assist in hardware start up.

Many functions in this file return an EmberStatus value.

This is a universal library for both SoCs and hosts. Since host-NCP communication involves communication of return values from NCP to host, the return types for some of these methods on the host are voids. In this case, we use the corresponding ...Return() methods so the NCP can return status to the host.

To account for the differences between the host and SoC interfaces the following macros are defined.

Host apps will have these defines:

```
1 #define MfgStatus    void
2 #define MfgStatus_U  void
3 #define MfgStatus_UU void
4 #define MfgStatus_S  void
```

SoC apps will have these defines:

```
1 #define MfgStatus    EmberStatus
2 #define MfgStatus_U  uint8_t
3 #define MfgStatus_UU uint16_t
4 #define MfgStatus_S  int8_t
```

See error-def.h for definitions of all EmberStatus values.

6.18.2 Function Documentation

6.18.2.1 MfgStatus mfglibEnd (void)

This restores the hardware to the state it was in prior to mfglibStart() and stops receiving packets started by mfglibStart() at the same time.

Use this function to exit the mfg test mode.

Note: It may be desirable to also reboot after use of manufacturing mode to ensure all application state is properly re-initialized.

Returns

For host apps see mfglibEndReturn(). For SoC apps one of the following:

- EMBER_SUCCESS if the mfg test mode has been exited.
- EMBER_ERR_FATAL if the mfg test mode cannot be exited.

6.18.2.2 void mfglibEndReturn (EmberStatus *status*, uint32_t *receiveCount*)

Parameters

status	• EMBER_SUCCESS if the mfg test mode has been exited. • EMBER_ERR_FATAL if the mfg test mode cannot be exited.
receiveCount	The total number of packets received during the test.

6.18.2.3 MfgStatus_U mfglibGetChannel (void)

Use this function to get current channel.

Returns

For host apps see mfglibGetChannelReturn(). For SoC apps the current channel.

6.18.2.4 void mfglibGetChannelReturn (uint8_t *channel*)

Parameters

channel	The current channel.

6.18.2.5 MfgStatus_U mfglibGetOptions (void)

Use this function to get library options.

Returns

For host apps see mfglibGetOptionsReturn(). For SoC apps the current test mode.

6.18.2.6 void mfglibGetOptionsReturn (uint8_t *options*)

Parameters

options	The current options based on the current test mode.

6.18.2.7 MfgStatus_S mfglibGetPower (void)

Use this function to get current power setting.

Returns

For host apps see mfglibGetPowerReturn(). For SoC apps the current power setting.

6.18.2.8 MfgStatus_UU mfglibGetPowerMode (void)

Use this function to get current power mode setting.

Returns

For host apps see mfglibGetPowerModeReturn(). For SoC apps the current power mode setting.

6.18.2.9 void mfglibGetPowerModeReturn (uint16_t *txPowerMode*)

Parameters

txPowerMode	The current power mode setting.

6.18.2.10 void mfglibGetPowerReturn (int8_t *power*)

Parameters

power	The current power setting.

6.18.2.11 MfgStatus_S mfglibGetSynOffset (void)

See mfglibSetSynOffset() for details.

Use this function to get the current setting for tolerances in the crystal oscillator or capacitors.

Returns

For host apps see mfglibGetSynOffsetReturn(). For SoC apps the synth offset in 11.7kHz steps

6.18.2.12 void mfglibGetSynOffsetReturn (int8_t *synthOffset*)

Parameters

synthOffset	The synth offset in 11.7kHz steps.

6.18.2.13 void mfglibRxHandler (uint8_t * *packet,* uint8_t *linkQuality,* int8_t *rssi*)

Parameters

packet	incoming packet
linkQuality	link quality as a numeric value
rssi	RSSI in dBm

6.18.2.14 MfgStatus mfglibSendPacket (uint8_t * *packet,* uint16_t *repeat*)

Use this function to send raw data. Note that *packet* array must be word-aligned (begin at even address), such that *((((uint16_t)packet) & 1) == 0)* holds true. (This is generally done by either declaring *packet* as a local variable or putting it in a global declaration immediately following the declaration of an uint16_t.)

Parameters

packet	Packet to be sent. First byte of the packet is always the length byte, whose value does not include itself but does include the 16-bit CRC in the length calculation. The CRC gets appended automatically by the radio as it transmits the packet, so the host does not need to provide this as part of packetContents. The total length of packet contents (Length Byte+1) going out the radio should not be >128 or <6 bytes. Note that the packet array should not include the CRC, as this appended by the radio automatically.
repeat	Number of times to repeat sending the packet after having been sent once. A value of 0 means send once and don't repeat.

Returns

For host apps see mfglibSendPacketReturn(). For SoC apps one of the following:

- EMBER_SUCCESS if the packet was sent.
- EMBER_ERR_FATAL if the mfg test mode is not available or TONE or STREAM test is running.

6.18.2.15 void mfglibSendPacketReturn (EmberStatus *status*)

Parameters

status	• EMBER_SUCCESS if the packet was sent. • EMBER_ERR_FATAL if the mfg test mode is not available or TONE or STREAM test is running.

6.18.2.16 MfgStatus mfglibSetChannel (uint8_t *channel*)

Customers can set any valid channel they want. Calibration occurs if this is the first time after power up.

Use this function to change channels.

Parameters

channel	Valid values depend upon the radio used.

Returns

For host apps see mfglibSetChannelReturn(). For SoC apps one of the following:

- EMBER_SUCCESS if the channel has been set.
- EMBER_PHY_INVALID_CHANNEL if the channel requested is invalid.
- EMBER_ERR_FATAL if the mfg test mode is not available or TONE or STREAM test is running.

6.18.2.17 void mfglibSetChannelReturn (EmberStatus *status*)

Parameters

status	• EMBER_SUCCESS if the channel has been set. • EMBER_PHY_INVALID_CHANNEL if the channel requested is invalid. • EMBER_ERR_FATAL if the mfg test mode is not available or TONE or STREAM test is running.
status	• EMBER_SUCCESS if the channel has been set. • ::EMBER_ERROR_INVALID_CHANNEL if the channel requested is invalid. • EMBER_ERR_FATAL if the mfg test mode is not available or TONE or STREAM test is running.

6.18.2.18 MfgStatus mfglibSetOptions (uint8_t *options*)

Use this function to set manufacturing library options.

Parameters

options	bitmask. 0 == non-CSMA transmits, 1 == CSMA transmits

Returns

For host apps see mfglibSetOptionsReturn(). For SoC apps one of the following:

- EMBER_SUCCESS if the options have been set.
- EMBER_BAD_ARGUMENT if any options are unavailable.
- EMBER_ERR_FATAL if the mfg test mode is not available or TONE or STREAM test is running.

6.18.2.19 void mfglibSetOptionsReturn (EmberStatus *status*)

Parameters

status	• EMBER_SUCCESS if the options have been set. • EMBER_BAD_ARGUMENT if any options are unavailable. • EMBER_ERR_FATAL if the mfg test mode is not available or TONE or STREAM test is running.

6.18.2.20 MfgStatus mfglibSetPower (uint16_t *txPowerMode,* int8_t *power*)

Valid power settings depend upon the specific radio in use. Silabs 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 mfglib↩ GetPower().

Use this function to adjust the transmit power.

Parameters

txPowerMode	boost mode or external PA.
power	Power in units of dBm, which can be negative.

Returns

For host apps see mfglibSetPowerReturn(). For SoC apps one of the following:

- EMBER_SUCCESS if the power has been set.
- ::EMBER_ERROR_INVALID_POWER if the power requested is invalid.
- EMBER_ERR_FATAL if the mfg test mode is not available or TONE or STREAM test is running.

6.18.2.21 void mfglibSetPowerReturn (EmberStatus *status*)

Parameters

status	• EMBER_SUCCESS if the power has been set. • ::EMBER_ERROR_INVALID_POWER if the power requested is invalid. • EMBER_ERR_FATAL if the mfg test mode is not available or TONE or STREAM test is running.

6.18.2.22 void mfglibSetSynOffset (int8_t *synOffset*)

This function does NOT write the new synth offset to the token, it only changes it in memory. It can be changed as many times as you like, and the setting will be lost when a reset occurs. The value will survive deep sleep, but will not survive a reset, thus it will not take effect in the bootloader. If you would like it to be permanent (and accessible to the bootloader), you must write the TOKEN_MFG_SYNTH_FREQ_OFFSET token using the token API or em3xx_load –patch.

Use this function to compensate for tolerances in the crystal oscillator or capacitors. This function does not effect a permanent change; once you have found the offset you want, you must write it to a token using the token API for it to be permanent.

Parameters

synOffset	the number of 11.7kHz steps to offset the carrier frequency (may be negative)

Returns

None.

6.18.2.23 MfgStatus mfglibStart (void(∗)(uint8_t ∗packet, uint8_t linkQuality, int8_t rssi) *mfglibRxCallback*)

It is legal to pass in a NULL. These packets will not be passed up with a CRC failure. The first byte of the packet in the callback is the length. All other functions will return an error until mfglibStart() has been called.

Use this function to enter test mode.

Note: This function should only be called shortly after initialization and prior to forming or joining a network.

Parameters

mfglibRxCallback	Function pointer to callback routine. On SoCs this function is invoked whenever a valid packet is received. On Hosts, in order not to flood the serial connection between the NCP and the host, this function is called once for every EMBER_MFG_RX_NCP_TO_HOST_INTERVAL packets. EMBER_MFG_RX_NCP_TO_HOST_INTERVAL is defined in ember-configuration-defaults.h and can be modified when building NCP images. The default value is 50. The total number of packets received is returned in the mfglibEndReturn() callback. emberTick() must be called routinely for this callback to function correctly.

Returns

For host apps see mfglibStartReturn(). For SoC apps one of the following:

- EMBER_SUCCESS if the mfg test mode has been enabled.
- EMBER_ERR_FATAL if the mfg test mode is not available.

6.18.2.24 void mfglibStartReturn (EmberStatus *status*)

Parameters

status	• EMBER_SUCCESS if the mfg test mode has been enabled. • EMBER_ERR_FATAL if the mfg test mode is not available.

6.18.2.25 MfgStatus mfglibStartStream (void)

Use this function to enable the measurement of radio modulation.

Returns

For host apps see mfglibStartStreamReturn(). For SoC apps one of the following:

- EMBER_SUCCESS if the transmit stream has started.
- EMBER_ERR_FATAL if the stream cannot be started.

6.18.2.26 void mfglibStartStreamReturn (EmberStatus *status*)

Parameters

status	• EMBER_SUCCESS if the transmit stream has started. • EMBER_ERR_FATAL if the stream cannot be started.

6.18.2.27 MfgStatus mfglibStartTone (void)

In this mode, the radio will transmit an unmodulated tone on the currently set channel and power level. Upon successful return, the tone will be transmitting. To stop transmitting a tone, the application must call mfglibStop↩ Tone(), allowing it the flexibility to determine its own criteria for tone duration, such as time, event, and so on.

Use this function to transmit a tone.

Returns

For host apps see mfglibStartToneReturn(). For SoC apps one of the following:

- EMBER_SUCCESS if the transmit tone has started.
- EMBER_ERR_FATAL if the tone cannot be started.

6.18.2.28 void mfglibStartToneReturn (EmberStatus *status*)

Parameters

status	• EMBER_SUCCESS if the transmit tone has started. • EMBER_ERR_FATAL if the tone cannot be started.

6.18.2.29 MfgStatus mfglibStopStream (void)

Use this function to end the measurement of radio modulation.

Returns

For host apps see mfglibStopStreamReturn(). For SoC apps one of the following:
- EMBER_SUCCESS if the transmit stream has stopped.
- EMBER_ERR_FATAL if the stream cannot be stopped.

6.18.2.30 void mfglibStopStreamReturn (EmberStatus *status*)

Parameters

status	• EMBER_SUCCESS if the transmit stream has stopped. • EMBER_ERR_FATAL if the stream cannot be stopped.

6.18.2.31 MfgStatus mfglibStopTone (void)

Use this function to stop transmitting a tone.

Returns

For host apps see mfglibStopToneReturn(). For SoC apps one of the following:
- EMBER_SUCCESS if the transmit tone has stopped.
- EMBER_ERR_FATAL if the tone cannot be stopped.

6.18.2.32 void mfglibStopToneReturn (EmberStatus *status*)

Parameters

status	• EMBER_SUCCESS if the transmit tone has stopped. • EMBER_ERR_FATAL if the tone cannot be stopped.

6.18.2.33 void mfglibTestContModCal (uint8_t *channel,* uint32_t *duration*)

If the duration argument == 0, this test will run forever (until the chip is reset).

Use this function to run the active transmit part of mod DAC calibration.

Parameters

channel	Selects the channel to transmit on.
duration	Duration in ms, 0 == infinite.

6.19 Hardware Abstraction Layer (HAL) API Reference

Modules

- Common Microcontroller Functions
- Token Access
- Sample APIs for Peripheral Access
- USB Device Stack Library
- System Timer Control

  Functions that provide access to the system clock.
- Symbol Timer Control
- HAL Configuration
- HAL Utilities
- Bootloader Interfaces
- Custom Bootloader HAL

Macros

- #define TOKEN_NEXT_ADDRESS(region, address)

  By default, tokens are automatically located after the previous token.
- #define CURRENT_STACK_TOKEN_VERSION 0x03FC

  The current version number of the stack tokens. MSB is the version, LSB is a complement.

Convenience Macros

The following convenience macros are used to simplify the definition process for commonly specified parameters to the basic TOKEN_DEF macro. Please see hal/micro/token.h for a more complete explanation.

- #define DEFINE_BASIC_TOKEN(name, type, ...) TOKEN_DEF(name, CREATOR_##name, 0, 0, type, 1, __VA_ARGS__)
- #define DEFINE_COUNTER_TOKEN(name, type, ...) TOKEN_DEF(name, CREATOR_##name, 1, 0, type, 1, __VA_ARGS__)
- #define DEFINE_INDEXED_TOKEN(name, type, arraysize, ...) TOKEN_DEF(name, CREATOR_##name, 0, 1, type, (arraysize), __VA_ARGS__)
- #define DEFINE_FIXED_BASIC_TOKEN(name, type, address, ...)
- #define DEFINE_FIXED_COUNTER_TOKEN(name, type, address, ...)
- #define DEFINE_FIXED_INDEXED_TOKEN(name, type, arraysize, address, ...)
- #define DEFINE_MFG_TOKEN(name, type, address, ...)

Creator Codes

The CREATOR is used as a distinct identifier tag for the token.

The CREATOR is necessary because the token name is defined differently depending on the hardware platform, therefore the CREATOR makes sure that token definitions and data stay tagged and known. The only requirement is that each creator definition must be unique. Please see hal/micro/token.h for a more complete explanation.

- #define CREATOR_STACK_NVDATA_VERSION 0xFF01
- #define CREATOR_STACK_BOOT_COUNTER 0xE263

- #define CREATOR_STACK_NONCE_COUNTER 0xE563
- #define CREATOR_STACK_ANALYSIS_REBOOT 0xE162
- #define CREATOR_STACK_KEYS 0xEB79
- #define CREATOR_STACK_NODE_DATA 0xEE64
- #define CREATOR_STACK_CLASSIC_DATA 0xE364
- #define CREATOR_STACK_ALTERNATE_KEY 0xE475
- #define CREATOR_STACK_APS_FRAME_COUNTER 0xE123
- #define CREATOR_STACK_TRUST_CENTER 0xE124
- #define CREATOR_STACK_NETWORK_MANAGEMENT 0xE125
- #define CREATOR_STACK_BINDING_TABLE 0xE274
- #define CREATOR_STACK_CHILD_TABLE 0xFF0D
- #define CREATOR_STACK_KEY_TABLE 0xE456
- #define CREATOR_STACK_CERTIFICATE_TABLE 0xE500
- #define CREATOR_STACK_PSL_DATA 0xE501
- #define CREATOR_STACK_HOST_REGISTRY 0xE502

6.19.1 Detailed Description

The tokens listed here are divided into three sections (the three main types of tokens mentioned in token.h):

- Manufacturing
- Stack
- Application

For a full explanation of the tokens, see hal/micro/token.h. See token-stack.h for source code.

A set of tokens is predefined in the APPLICATION DATA section at the end of token-stack.h because these tokens are required by the stack, but they are classified as application tokens since they are sized by the application via its CONFIGURATION_HEADER.

The user application can include its own tokens in a header file similar to this one. The macro ::APPLICATION_↩ TOKEN_HEADER should be defined to equal the name of the header file in which application tokens are defined. See the APPLICATION DATA section at the end of token-stack.h for examples of token definitions.

Since token-stack.h contains both the typedefs and the token defs, there are two #defines used to select which one is needed when this file is included. #define DEFINETYPES is used to select the type definitions and #define DEFINETOKENS is used to select the token definitions. Refer to token.h and token.c to see how these are used.

EM35x Microprocessors

HAL function names have the following prefix conventions:

halCommon: API that is used by the EmberZNet stack and can also be called from an application. This API must be implemented. Custom applications can change the implementation of the API but its functionality must remain the same.

hal: API that is used by sample applications. Custom applications can remove this API or change its implementation as they see fit.

halStack: API used only by the EmberZNet stack. This API must be implemented and should not be directly called from any application. Custom applications can change the implementation of the API, but its functionality must remain the same.

halInternal: API that is internal to the HAL. The EmberZNet stack and applications must never call this API directly. Custom applications can change this API as they see fit. However, be careful not to impact the functionalty of any halStack or halCommon APIs.

See also hal.h.

6.19.2 Macro Definition Documentation

6.19.2.1 #define CREATOR_STACK_ALTERNATE_KEY 0xE475

6.19.2.2 #define CREATOR_STACK_ANALYSIS_REBOOT 0xE162

6.19.2.3 #define CREATOR_STACK_APS_FRAME_COUNTER 0xE123

6.19.2.4 #define CREATOR_STACK_BINDING_TABLE 0xE274

6.19.2.5 #define CREATOR_STACK_BOOT_COUNTER 0xE263

6.19.2.6 #define CREATOR_STACK_CERTIFICATE_TABLE 0xE500

6.19.2.7 #define CREATOR_STACK_CHILD_TABLE 0xFF0D

6.19.2.8 #define CREATOR_STACK_CLASSIC_DATA 0xE364

6.19.2.9 #define CREATOR_STACK_HOST_REGISTRY 0xE502

6.19.2.10 #define CREATOR_STACK_KEY_TABLE 0xE456

6.19.2.11 #define CREATOR_STACK_KEYS 0xEB79

6.19.2.12 #define CREATOR_STACK_NETWORK_MANAGEMENT 0xE125

6.19.2.13 #define CREATOR_STACK_NODE_DATA 0xEE64

6.19.2.14 #define CREATOR_STACK_NONCE_COUNTER 0xE563

6.19.2.15 #define CREATOR_STACK_NVDATA_VERSION 0xFF01

6.19.2.16 #define CREATOR_STACK_PSL_DATA 0xE501

6.19.2.17 #define CREATOR_STACK_TRUST_CENTER 0xE124

6.19.2.18 #define CURRENT_STACK_TOKEN_VERSION 0x03FC

Please see hal/micro/token.h for a more complete explanation.

6.19.2.19 #define DEFINE_BASIC_TOKEN(*name, type, ...*) TOKEN_DEF(name, CREATOR_##name, 0, 0, type, 1, __VA_ARGS__)

6.19.2.20 #define DEFINE_COUNTER_TOKEN(*name, type, ...*) TOKEN_DEF(name, CREATOR_##name, 1, 0, type, 1, __VA_ARGS__)

6.19.2.21 #define DEFINE_FIXED_BASIC_TOKEN(*name, type, address, ...*)

Value:

```
TOKEN_NEXT_ADDRESS(name, (address))                       \
  TOKEN_DEF(name, CREATOR_##name, 0, 0, type, 1, __VA_ARGS__)
```

6.19.2.22 #define DEFINE_FIXED_COUNTER_TOKEN(*name, type, address, ...*)

Value:

```
TOKEN_NEXT_ADDRESS(name, (address))                         \
  TOKEN_DEF(name, CREATOR_##name, 1, 0, type, 1, __VA_ARGS__)
```

6.19.2.23 #define DEFINE_FIXED_INDEXED_TOKEN(*name, type, arraysize, address, ...*)

Value:

```
TOKEN_NEXT_ADDRESS(name, (address))
      \
  TOKEN_DEF(name, CREATOR_##name, 0, 1, type, (arraysize), __VA_ARGS__)
```

6.19.2.24 #define DEFINE_INDEXED_TOKEN(*name, type, arraysize, ...*) TOKEN_DEF(name, CREATOR_##name, 0, 1, type, (arraysize), __VA_ARGS__)

6.19.2.25 #define DEFINE_MFG_TOKEN(*name, type, address, ...*)

Value:

```
TOKEN_NEXT_ADDRESS(name, (address))                 \
  TOKEN_MFG(name, CREATOR_##name, 0, 0, type, 1, __VA_ARGS__)
```

6.19.2.26 #define TOKEN_NEXT_ADDRESS(*region, address*)

If a token needs to be placed at a specific location, one of the DEFINE_FIXED_∗ definitions should be used. This macro is inherently used in the DEFINE_FIXED_∗ definition to locate a token, and under special circumstances (such as manufacturing tokens) it may be explicitely used.

Parameters

region	A name for the next region being located.
address	The address of the beginning of the next region.

6.20 Common Microcontroller Functions

Data Structures

- struct RTCCRamData

Macros

- #define PTA_SUPPORT
- #define PTA_GPIOCFG_INPUT GPIOCFG_IN_PUD
- #define PTA_GPIOCFG_OUTPUT GPIOCFG_OUT
- #define PTA_GPIOCFG_WIRED_OR GPIOCFG_OUT_OD
- #define PTA_GPIOCFG_WIRED_AND GPIOCFG_OUT_OD
- #define MICRO_DISABLE_WATCH_DOG_KEY 0xA5U

  The value that must be passed as the single parameter to halInternalDisableWatchDog() in order to successfully disable the watchdog timer.
- #define GPIO_MASK_SIZE 24
- #define GPIO_MASK 0xFFFFFF
- #define WAKE_GPIO_MASK GPIO_MASK
- #define WAKE_GPIO_SIZE GPIO_MASK_SIZE
- #define WAKE_MASK_INVALID (-1)
- #define WAKE_EVENT_SIZE WakeMask
- #define DEBUG_TOGGLE(n)
- #define halGetEm2xxResetInfo() halGetResetInfo()

  Calls halGetExtendedResetInfo() and translates the EM35x or COBRA reset code to the corresponding value used by the EM2XX HAL. Any reset codes not present in the EM2XX are returned after being OR'ed with 0x80.

Typedefs

- typedef uint32_t WakeEvents
- typedef uint32_t WakeMask

Enumerations

- enum SleepModes {
  SLEEPMODE_RUNNING = 0U,
  SLEEPMODE_IDLE = 1U,
  SLEEPMODE_WAKETIMER = 2U,
  SLEEPMODE_MAINTAINTIMER = 3U,
  SLEEPMODE_NOTIMER = 4U,
  SLEEPMODE_HIBERNATE = 5U,
  SLEEPMODE_RESERVED = 6U,
  SLEEPMODE_POWERDOWN = 7U,
  SLEEPMODE_POWERSAVE = 8U }

  Enumerations for the possible microcontroller sleep modes.

Functions

- void halInternalSysReset (uint16_t extendedCause)

  Records the specified reset cause then forces a reboot.
- uint16_t halGetExtendedResetInfo (void)

  Returns the Extended Reset Cause information.
- PGM_P halGetExtendedResetString (void)

  Calls halGetExtendedResetInfo() and supplies a string describing the extended cause of the reset. halGetReset↩String() should also be called to get the string for the base reset cause.
- EmberStatus halSetRadioHoldOff (bool enable)

  Enables or disables Radio HoldOff support.
- bool halGetRadioHoldOff (void)

  Returns whether Radio HoldOff has been enabled or not.
- void halStackRadioPowerDownBoard (void)

  To assist with saving power when the radio automatically powers down, this function allows the stack to tell the HAL to put pins specific to radio functionality in their powerdown state. The pin state used is the state used by halInternal↩PowerDownBoard, but applied only to the pins identified in the global variable gpioRadioPowerBoardMask. The stack will automatically call this function as needed, but it will only change GPIO state based on gpioRadioPowerBoard↩Mask. Most commonly, the bits set in gpioRadioPowerBoardMask petain to using a Front End Module. This function is often called from interrupt context.
- void halStackRadio2PowerDownBoard (void)

  To assist with saving power when radio2 automatically powers down, this function allows the stack to tell the HAL to put pins specific to radio functionality in their powerdown state. The pin state used is the state used by halInternal↩PowerDownBoard, but applied only to the pins identified in the global variable gpioRadioPowerBoardMask. The stack will automatically call this function as needed, but it will only change GPIO state based on gpioRadioPowerBoard↩Mask. Most commonly, the bits set in gpioRadioPowerBoardMask petain to using a Front End Module. This function is often called from interrupt context.
- void halStackRadioPowerUpBoard (void)

  To assist with saving power when the radio automatically powers up, this function allows the stack to tell the HAL to put pins specific to radio functionality in their powerup state. The pin state used is the state used by halInternal↩PowerUpBoard, but applied only to the pins identified in the global variable gpioRadioPowerBoardMask. The stack will automatically call this function as needed, but it will only change GPIO state based on gpioRadioPowerBoardMask. Most commonly, the bits set in gpioRadioPowerBoardMask petain to using a Front End Module. This function can be called from interrupt context.
- void halStackRadio2PowerUpBoard (void)

  To assist with saving power when radio2 automatically powers up, this function allows the stack to tell the HAL to put pins specific to radio functionality in their powerup state. The pin state used is the state used by halInternalPower↩UpBoard, but applied only to the pins identified in the global variable gpioRadioPowerBoardMask. The stack will automatically call this function as needed, but it will only change GPIO state based on gpioRadioPowerBoardMask. Most commonly, the bits set in gpioRadioPowerBoardMask petain to using a Front End Module. This function can be called from interrupt context.
- void halStackRadioPowerMainControl (bool powerUp)

  This function is called automatically by the stack prior to Radio power-up and after Radio power-down. It can be used to prepare for the radio being powered on and to clean up after it's been powered off. Unlike halStackRadioPower↩UpBoard() and halStackRadioPowerDownBoard(), which can be called from interrupt context, this function is only called from main-line context.
- void halRadioPowerUpHandler (void)

  Handler called in main context prior to radio being powered on.
- void halRadioPowerDownHandler (void)

  Handler called in main context after radio has been powered off.
- void halInit (void)

  Initializes microcontroller-specific peripherals.
- void halReboot (void)

  Restarts the microcontroller and therefore everything else.
- void halPowerUp (void)

  Powers up microcontroller peripherals and board peripherals.

- void halPowerDown (void)

  Powers down microcontroller peripherals and board peripherals.
- void halResume (void)

  Resumes microcontroller peripherals and board peripherals.
- void halSuspend (void)

  Suspends microcontroller peripherals and board peripherals.
- void halInternalEnableWatchDog (void)

  Enables the watchdog timer.
- void halInternalDisableWatchDog (uint8_t magicKey)

  Disables the watchdog timer.
- bool halInternalWatchDogEnabled (void)

  Determines whether the watchdog has been enabled or disabled.
- void halSleep (SleepModes sleepMode)

  Puts the microcontroller to sleep in a specified mode.
- void halCommonDelayMicroseconds (uint16_t us)

  Blocks the current thread of execution for the specified amount of time, in microseconds.
- void halCommonDisableVreg1v8 (void)

  Disable the 1.8V regulator. This function is to be used when the 1.8V supply is provided externally. Disabling the regulator saves current consumption. Disabling the regulator will cause ADC readings of external signals to be wrong. These exteranl signals include analog sources ADC0 thru ADC5 and VDD_PADS/4.
- void halCommonEnableVreg1v8 (void)

  Enable the 1.8V regulator. Normally the 1.8V regulator is enabled out of reset. This function is only needed if the 1.8V regulator has been disabled and ADC conversions on external signals are needed. These exteranl signals include analog sources ADC0 thru ADC5 and VDD_PADS/4. The state of 1v8 survives deep sleep.
- void halBeforeEM4 (uint32_t duration, RTCCRamData input)
- RTCCRamData halAfterEM4 (void)
- void halStackProcessBootCount (void)

  Called from emberInit and provides a means for the HAL to increment a boot counter, most commonly in non-volatile memory.
- uint8_t halGetResetInfo (void)

  Gets information about what caused the microcontroller to reset.
- PGM_P halGetResetString (void)

  Calls halGetResetInfo() and supplies a string describing it.

Variables

- volatile int8_t halCommonVreg1v8EnableCount

  Helper variable to track the state of 1.8V regulator.

Vector Table Index Definitions

These are numerical definitions for vector table. Indices 0 through 15 are Cortex-M3 standard exception vectors and indices 16 through 35 are EM3XX specific interrupt vectors.

- #define STACK_VECTOR_INDEX 0U

  A numerical definition for a vector.
- #define RESET_VECTOR_INDEX 1U

  A numerical definition for a vector.
- #define NMI_VECTOR_INDEX 2U

  A numerical definition for a vector.

- #define HARD_FAULT_VECTOR_INDEX 3U

  A numerical definition for a vector.
- #define MEMORY_FAULT_VECTOR_INDEX 4U

  A numerical definition for a vector.
- #define BUS_FAULT_VECTOR_INDEX 5U

  A numerical definition for a vector.
- #define USAGE_FAULT_VECTOR_INDEX 6U

  A numerical definition for a vector.
- #define RESERVED07_VECTOR_INDEX 7U

  A numerical definition for a vector.
- #define RESERVED08_VECTOR_INDEX 8U

  A numerical definition for a vector.
- #define RESERVED09_VECTOR_INDEX 9U

  A numerical definition for a vector.
- #define RESERVED10_VECTOR_INDEX 10U

  A numerical definition for a vector.
- #define SVCALL_VECTOR_INDEX 11U

  A numerical definition for a vector.
- #define DEBUG_MONITOR_VECTOR_INDEX 12U

  A numerical definition for a vector.
- #define RESERVED13_VECTOR_INDEX 13U

  A numerical definition for a vector.
- #define PENDSV_VECTOR_INDEX 14U

  A numerical definition for a vector.
- #define SYSTICK_VECTOR_INDEX 15U

  A numerical definition for a vector.
- #define TIMER1_VECTOR_INDEX 16U

  A numerical definition for a vector.
- #define TIMER2_VECTOR_INDEX 17U

  A numerical definition for a vector.
- #define MANAGEMENT_VECTOR_INDEX 18U

  A numerical definition for a vector.
- #define BASEBAND_VECTOR_INDEX 19U

  A numerical definition for a vector.
- #define SLEEP_TIMER_VECTOR_INDEX 20U

  A numerical definition for a vector.
- #define SC1_VECTOR_INDEX 21U

  A numerical definition for a vector.
- #define SC2_VECTOR_INDEX 22U

  A numerical definition for a vector.
- #define SECURITY_VECTOR_INDEX 23U

  A numerical definition for a vector.
- #define MAC_TIMER_VECTOR_INDEX 24U

  A numerical definition for a vector.
- #define MAC_TX_VECTOR_INDEX 25U

  A numerical definition for a vector.
- #define MAC_RX_VECTOR_INDEX 26U

  A numerical definition for a vector.
- #define ADC_VECTOR_INDEX 27U

  A numerical definition for a vector.
- #define IRQA_VECTOR_INDEX 28U

A numerical definition for a vector.

- #define IRQB_VECTOR_INDEX 29U

  A numerical definition for a vector.
- #define IRQC_VECTOR_INDEX 30U

  A numerical definition for a vector.
- #define IRQD_VECTOR_INDEX 31U

  A numerical definition for a vector.
- #define DEBUG_VECTOR_INDEX 32U

  A numerical definition for a vector.
- #define SC3_VECTOR_INDEX 33U

  A numerical definition for a vector.
- #define SC4_VECTOR_INDEX 34U

  A numerical definition for a vector.
- #define USB_VECTOR_INDEX 35U

  A numerical definition for a vector.
- #define VECTOR_TABLE_LENGTH 36U

  Number of vectors.

6.20.1 Detailed Description

See also hal/micro/cortexm3/micro.h for source code.

Many of the supplied example applications use these microcontroller functions. See hal/micro/micro-common.h for source code.

Many of the supplied example applications use these microcontroller functions. See hal/micro/micro.h for source code.

Note

The term SFD refers to the Start Frame Delimiter.

6.20.2 Macro Definition Documentation

6.20.2.1 #define ADC_VECTOR_INDEX 27U

6.20.2.2 #define BASEBAND_VECTOR_INDEX 19U

6.20.2.3 #define BUS_FAULT_VECTOR_INDEX 5U

6.20.2.4 #define DEBUG_MONITOR_VECTOR_INDEX 12U

6.20.2.5 #define DEBUG_TOGGLE(*n*)

6.20.2.6 #define DEBUG_VECTOR_INDEX 32U

6.20.2.7 #define GPIO_MASK 0xFFFFFF

6.20.2.8 #define GPIO_MASK_SIZE 24

6.20.2.9 #define halGetEm2xxResetInfo( ) halGetResetInfo()

Used by the EZSP host as a platform-independent NCP reset code.

Returns

The EM2XX-compatible reset code. If not supported by the EM2XX, return the platform-specific code with B7 set.

6.20.2.10 #define HARD_FAULT_VECTOR_INDEX 3U

6.20.2.11 #define IRQA_VECTOR_INDEX 28U

6.20.2.12 #define IRQB_VECTOR_INDEX 29U

6.20.2.13 #define IRQC_VECTOR_INDEX 30U

6.20.2.14 #define IRQD_VECTOR_INDEX 31U

6.20.2.15 #define MAC_RX_VECTOR_INDEX 26U

6.20.2.16 #define MAC_TIMER_VECTOR_INDEX 24U

6.20.2.17 #define MAC_TX_VECTOR_INDEX 25U

6.20.2.18 #define MANAGEMENT_VECTOR_INDEX 18U

6.20.2.19 #define MEMORY_FAULT_VECTOR_INDEX 4U

6.20.2.20 #define MICRO_DISABLE_WATCH_DOG_KEY 0xA5U

6.20.2.21 #define NMI_VECTOR_INDEX 2U

6.20.2.22 #define PENDSV_VECTOR_INDEX 14U

6.20.2.23 #define PTA_GPIOCFG_INPUT GPIOCFG_IN_PUD

6.20.2.24 #define PTA_GPIOCFG_OUTPUT GPIOCFG_OUT

6.20.2.25 #define PTA_GPIOCFG_WIRED_AND GPIOCFG_OUT_OD

6.20.2.26 #define PTA_GPIOCFG_WIRED_OR GPIOCFG_OUT_OD

6.20.2.27 #define PTA_SUPPORT

6.20.2.28 #define RESERVED07_VECTOR_INDEX 7U

6.20.2.29 #define RESERVED08_VECTOR_INDEX 8U

6.20.2.30 #define RESERVED09_VECTOR_INDEX 9U

6.20.2.31 #define RESERVED10_VECTOR_INDEX 10U

6.20.2.32 #define RESERVED13_VECTOR_INDEX 13U

6.20.2.33 #define RESET_VECTOR_INDEX 1U

6.20.2.34 #define SC1_VECTOR_INDEX 21U

6.20.2.35 #define SC2_VECTOR_INDEX 22U

6.20.2.36 #define SC3_VECTOR_INDEX 33U

6.20.2.37 #define SC4_VECTOR_INDEX 34U

6.20.2.38 #define SECURITY_VECTOR_INDEX 23U

6.20.2.39 #define SLEEP_TIMER_VECTOR_INDEX 20U

6.20.2.40 #define STACK_VECTOR_INDEX 0U

6.20.2.41 #define SVCALL_VECTOR_INDEX 11U

6.20.2.42 #define SYSTICK_VECTOR_INDEX 15U

6.20.2.43 #define TIMER1_VECTOR_INDEX 16U

6.20.2.44 #define TIMER2_VECTOR_INDEX 17U

6.20.2.45 #define USAGE_FAULT_VECTOR_INDEX 6U

6.20.2.46 #define USB_VECTOR_INDEX 35U

6.20.2.47 #define VECTOR_TABLE_LENGTH 36U

6.20.2.48 #define WAKE_EVENT_SIZE WakeMask

Note

The preprocessor symbol WAKE_EVENT_SIZE has been deprecated. Please use WakeMask instead.

6.20.2.49 #define WAKE_GPIO_MASK GPIO_MASK

6.20.2.50 #define WAKE_GPIO_SIZE GPIO_MASK_SIZE

6.20.2.51 #define WAKE_MASK_INVALID (-1)

6.20.3 Typedef Documentation

6.20.3.1 typedef uint32_t WakeEvents

6.20.3.2 typedef uint32_t WakeMask

6.20.4 Enumeration Type Documentation

6.20.4.1 enum SleepModes

- SLEEPMODE_RUNNING Everything is active and running. In practice this mode is not used, but it is defined for completeness of information.
- SLEEPMODE_IDLE Only the CPU is idled. The rest of the chip continues running normally. The chip will wake from any interrupt.
- SLEEPMODE_WAKETIMER The sleep timer clock sources remain running. The RC is always running and the 32kHz XTAL depends on the board header. Wakeup is possible from both GPIO and the sleep timer. System time is maintained. The sleep timer is assumed to be configured properly for wake events.
- SLEEPMODE_MAINTAINTIMER The sleep timer clock sources remain running. The RC is always running and the 32kHz XTAL depends on the board header. Wakeup is possible from only GPIO. System time is maintained. NOTE: This mode is not available on EM2XX chips.
- SLEEPMODE_NOTIMER The sleep timer clock sources (both RC and XTAL) are turned off. Wakeup is possible from only GPIO. System time is lost.
- SLEEPMODE_HIBERNATE This maps to EM4 Hibernate on the EFM32/EFR32 devices. RAM is not retained in SLEEPMODE_HIBERNATE so waking up from this sleepmode will behave like a reset. NOTE: This mode is only available on EFM32/EFR32

Enumerator

SLEEPMODE_RUNNING

SLEEPMODE_IDLE

SLEEPMODE_WAKETIMER

SLEEPMODE_MAINTAINTIMER

SLEEPMODE_NOTIMER

SLEEPMODE_HIBERNATE

SLEEPMODE_RESERVED

SLEEPMODE_POWERDOWN

SLEEPMODE_POWERSAVE

6.20.5 Function Documentation

6.20.5.1 RTCCRamData halAfterEM4 (void)

6.20.5.2 void halBeforeEM4 (uint32_t *duration,* RTCCRamData *input*)

6.20.5.3 void halCommonDelayMicroseconds (uint16_t *us*)

The function is implemented with cycle-counted busy loops and is intended to create the short delays required when interfacing with hardware peripherals.

The accuracy of the timing provided by this function is not specified, but a general rule is that when running off of a crystal oscillator it will be within 10us. If the micro is running off of another type of oscillator (e.g. RC) the timing accuracy will potentially be much worse.

Parameters

us	The specified time, in microseconds. Values should be between 1 and 65535 microseconds.

6.20.5.4 void halCommonDisableVreg1v8 (void)

Note

: Only used when DISABLE_INTERNAL_1V8_REGULATOR is defined.

6.20.5.5 void halCommonEnableVreg1v8 (void)

Note

: Only used when DISABLE_INTERNAL_1V8_REGULATOR is defined.

6.20.5.6 uint16_t halGetExtendedResetInfo (void)

Returns

A 16-bit code identifying the base and extended cause of the reset

6.20.5.7 PGM_P halGetExtendedResetString (void)

Useful for diagnostic printing of text just after program initialization.

Returns

A pointer to a program space string.

6.20.5.8 bool halGetRadioHoldOff (void)

Returns

true if Radio HoldOff has been enabled, false otherwise.

6.20.5.9 uint8_t halGetResetInfo (void)

Returns

A code identifying the cause of the reset.

6.20.5.10 PGM_P halGetResetString (void)

Useful for diagnostic printing of text just after program initialization.

Returns

A pointer to a program space string.

6.20.5.11 void halInit (void)

6.20.5.12 void halInternalDisableWatchDog (uint8_t *magicKey*)

Note

To prevent the watchdog from being disabled accidentally, a magic key must be provided.

Parameters

magicKey	A value (MICRO_DISABLE_WATCH_DOG_KEY) that enables the function.

6.20.5.13 void halInternalEnableWatchDog (void)

6.20.5.14 void halInternalSysReset (uint16_t *extendedCause*)

Referenced by halSimEepromCallback().

6.20.5.15 bool halInternalWatchDogEnabled (void)

Returns

A bool value indicating if the watchdog is current enabled.

6.20.5.16 void halPowerDown (void)

6.20.5.17 void halPowerUp (void)

6.20.5.18 void halRadioPowerDownHandler (void)

Handler called in main context after radio has been powered off.

6.20.5.19 void halRadioPowerUpHandler (void)

Handler called in main context prior to radio being powered on.

6.20.5.20 void halReboot (void)

6.20.5.21 void halResume (void)

6.20.5.22 EmberStatus halSetRadioHoldOff (bool *enable*)

Parameters

enable	When true, configures ::RHO_GPIO in BOARD_HEADER as an input which, when asserted, will prevent the radio from transmitting. When false, configures ::RHO_GPIO for its original default purpose.

Returns

EMBER_SUCCESS if Radio HoldOff was configured as desired or EMBER_BAD_ARGUMENT if requesting it be enabled but RHO has not been configured by the BOARD_HEADER.

6.20.5.23 void halSleep (SleepModes *sleepMode*)

Note

This routine always enables interrupts.

Parameters

sleepMode	A microcontroller sleep mode

See also

SleepModes

Referenced by usbSuspendDsr().

6.20.5.24 void halStackProcessBootCount (void)

This is useful while debugging to determine the number of resets that might be seen over a period of time. Exposing this functionality allows the application to disable or alter processing of the boot counter if, for example, the application is expecting a lot of resets that could wear out non-volatile storage or some

Called from emberInit only as helpful debugging information. This should be left enabled by default, but this function can also be reduced to a simple return statement if boot counting is not desired.

6.20.5.25 void halStackRadio2PowerDownBoard (void)

6.20.5.26 void halStackRadio2PowerUpBoard (void)

6.20.5.27 void halStackRadioPowerDownBoard (void)

6.20.5.28 void halStackRadioPowerMainControl (bool *powerUp*)

6.20.5.29 void halStackRadioPowerUpBoard (void)

6.20.5.30 void halSuspend (void)

6.20.6 Variable Documentation

6.20.6.1 volatile int8_t halCommonVreg1v8EnableCount

Note

: Only used when DISABLE_INTERNAL_1V8_REGULATOR is defined.

6.21 Token Access

Modules

- Tokens
- Simulated EEPROM
- Simulated EEPROM 2

6.21.1 Detailed Description

The token system stores such non-volatile information as the manufacturing ID, channel number, transmit power, and various pieces of information that the application needs to be persistent between device power cycles. The token system is design to abstract implementation details and simplify interacting with differing non-volatile systems. The majority of tokens are stored in Simulated EEPROM (in Flash) where they can be rewritten. Manufacturing tokens are stored in dedicated regions of flash and are not designed to be rewritten.

Refer to the Tokens module for a detailed description of the token system.
Refer to the Simulated EEPROM module for a detailed description of the necessary support functions for Simulated EEPROM.
Refer to the Simulated EEPROM 2 module for a detailed description of the necessary support functions for Simulated EEPROM, version 2.
Refer to token-stack.h for stack token definitions.
Refer to token-manufacturing.h for manufaturing token definitions.

Note

Simulated EEPROM, version 2 is only supported on EM335x chips.

6.22 Tokens

Macros

- #define halCommonGetToken(data, token)

  Macro that copies the token value from non-volatile storage into a RAM location. This macro can only be used with tokens that are defined using DEFINE_BASIC_TOKEN.
- #define halCommonGetMfgToken(data, token)

  Macro that copies the token value from non-volatile storage into a RAM location. This macro can only be used with tokens that are defined using DEFINE_MFG_TOKEN.
- #define halCommonGetIndexedToken(data, token, index)

  Macro that copies the token value from non-volatile storage into a RAM location. This macro can only be used with tokens that are defined using DEFINE_INDEXED_TOKEN.
- #define halCommonSetToken(token, data)

  Macro that sets the value of a token in non-volatile storage. This macro can only be used with tokens that are defined using DEFINE_BASIC_TOKEN.
- #define halCommonSetIndexedToken(token, index, data)

  Macro that sets the value of a token in non-volatile storage. This macro can only be used with tokens that are defined using DEFINE_INDEXED_TOKEN.
- #define halCommonIncrementCounterToken(token)

  Macro that increments the value of a token that is a counter. This macro can only be used with tokens that are defined using either DEFINE_COUNTER_TOKEN.

Functions

- EmberStatus halStackInitTokens (void)

  Initializes and enables the token system. Checks if the manufacturing and stack non-volatile data versions are correct.

6.22.1 Detailed Description

There are three main types of tokens:

- **Manufacturing tokens:** Tokens that are set at the factory and must not be changed through software operations.
- **Stack-level tokens:** Tokens that can be changed via the appropriate stack API calls.
- **Application level tokens:** Tokens that can be set via the token system API calls in this file.

The token system API controls writing tokens to non-volatile data and reading tokens from non-volatile data. If an application wishes to use application specific normal tokens, it must do so by creating its own token header file similar to token-stack.h. The macro `APPLICATION_TOKEN_HEADER` should be defined to equal the name of the header file in which application tokens are defined. If an application wishes to use application specific manufacturing tokens, it must do so by creating its own manufacturing token header file similar to token-manufacturing.h. The macro `APPLICATION_MFG_TOKEN_HEADER` should be defined to equal the name of the header file in which manufacturing tokens are defined.

Because the token system is based on memory locations within non-volatile storage, the token information could become out of sync without some kind of version tracking. The two defines, `CURRENT_MFG_TOKEN_VERSION` and `CURRENT_STACK_TOKEN_VERSION`, are used to make sure the stack stays in sync with the proper token set. If the application defines its own tokens, it is recommended that the application also define an application token to be a application version to ensure the application stays in sync with the proper token set.

The most general format of a token definition is:

```
1 #define CREATOR_name 16bit_value
2 #ifdef DEFINETYPES
3    typedef data_type type
4 #endif
5 #ifdef DEFINETOKENS
6    DEFINE_*_TOKEN(name, type, ... ,defaults)
7 #endif
```

The defined CREATOR is used as a distinct identifier tag for the token. The CREATOR is necessary because the token name is defined differently depending on underlying implementation, so the CREATOR makes sure token definitions and data stay tagged and known. The only requirement on these creator definitions is that they all must be unique. A favorite method for picking creator codes is to use two ASCII characters inorder to make the codes more memorable. The 'name' part of the `#define CREATOR_name` must match the 'name' provided in the `DEFINE_*_TOKEN` because the token system uses this name to automatically link the two.

The typedef provides a convenient and efficient abstraction of the token data. Since some tokens are structs with multiple pieces of data inside of them, type defining the token type allows more efficient and readable local copies of the tokens throughout the code.

The typedef is wrapped with an `#ifdef DEFINETYPES` because the typdefs and token defs live in the same file, and DEFINETYPES is used to select only the typedefs when the file is included. Similarly, the `DEFINE_*_TOKEN` is wrapped with an `#ifdef DEFINETOKENS` as a method for selecting only the token definitions when the file is included.

The abstract definition, `DEFINE_*_TOKEN(name, type, ... ,defaults)`, has seven possible complete definitions:

- `DEFINE_BASIC_TOKEN(name, type, ...)`
- `DEFINE_INDEXED_TOKEN(name, type, arraysize, ...)`
- `DEFINE_COUNTER_TOKEN(name, type, ...)`
- `DEFINE_MFG_TOKEN(name, type, address, ...)`

The three fields common to all `DEFINE_*_TOKEN` are:
name - The name of the token, which all information is tied to.
type - Type of the token which is the same as the typedef mentioned before.
... - The default value to which the token is set upon initialization.

Note

The old DEFINE_FIXED* token definitions are no longer used. They remain defined for backwards compatibility. In current systems, the Simulated EEPROM is used for storing non-manufacturing tokens and the Simulated EEPROM intelligently manages where tokens are stored to provide wear leveling across the flash memory and increase the number of write cycles. Manufacturing tokens live at a fixed address, but they must use DEFINE_MFG_TOKEN so the token system knows they are manufacturing tokens.

DEFINE_BASIC_TOKEN is the simplest definition and will be used for the majority of tokens (tokens that are not indexed, not counters, and not manufacturing). Basic tokens are designed for data storage that is always accessed as a single element.

DEFINE_INDEXED_TOKEN should be used on tokens that look like arrays. For example, data storage that looks like:

```
uint32_t myData[5]
```

This example data storage can be a token with typedef of uint32_t and defined as INDEXED with arraysize of 5. The extra field in this token definition is: arraysize - The number of elements in the indexed token. Indexed tokens are designed for data storage that is logically grouped together, but elements are accessed individually.

DEFINE_COUNTER_TOKEN should be used on tokens that are simple numbers where the majority of operations on the token is to increment the count. The reason for using DEFINE_COUNTER_TOKEN instead of DEFINE_↩ BASIC_TOKEN is the special support that the token system provides for incrementing counters. The function call `halCommonIncrementCounterToken()` only operates on counter tokens and is more efficient in terms of speed, data compression, and write cyles for incrementing simple numbers in the token system.

DEFINE_MFG_TOKEN is a DEFINE_BASIC_TOKEN token at a specific address and the token is manufacturing data that is written only once. The major difference is this token is designated manufacturing, which means the token system treats it differently from stack or app tokens. Primarily, a manufacturing token is written only once and lives at a fixed address outside of the Simulated EEPROM system. Being a write once token, the token system will also aid in debugging by asserting if there is an attempt to write a manufacturing token.

Here is an example of two application tokens:

```
1 #define CREATOR_SENSOR_NAME        0x5354
2 #define CREATOR_SENSOR_PARAMETERS  0x5350
3 #ifdef DEFINETYPES
4   typedef uint8_t tokTypeSensorName[10];
5   typedef struct {
6     uint8_t initValues[5];
7     uint8_t reportInterval;
8     uint16_t calibrationValue;
9   } tokTypeSensorParameters;
10 #endif
11 #ifdef DEFINETOKENS
12   DEFINE_BASIC_TOKEN(SENSOR_NAME,
13                      tokTypeSensorName,
14                      {'U','N','A','M','E','D',' ',' ',' ',' '})
15   DEFINE_BASIC_TOKEN(SENSOR_PARAMETERS,
16                      tokTypeSensorParameters,
17                      {{0x01,0x02,0x03,0x04,0x05},5,0x0000})
18 #endif
```

Here is an example of how to use the two application tokens:

```
1 {
2   tokTypeSensorName sensor;
3   tokTypeSensorParameters params;
4
5   halCommonGetToken(&sensor, TOKEN_SENSOR_NAME);
6   halCommonGetToken(&params, TOKEN_SENSOR_PARAMETERS);
7   if(params.calibrationValue == 0xBEEF) {
8     params.reportInterval = 5;
9   }
10   halCommonSetToken(TOKEN_SENSOR_PARAMETERS, &params);
11 }
```

See token-stack.h to see the default set of tokens and their values.

The nodetest utility app can be used for generic manipulation such as loading default token values, viewing tokens, and writing tokens. **The nodetest utility cannot work with customer defined application tokens or manufacturing tokens. Using the nodetest utility will erase customer defined application tokens in the Simulated EEPROM.**

The Simulated EEPROM will initialize tokens to their default values if the token does not yet exist, the token's creator code is changed, or the token's size changes.

Changing the number indexes in an INDEXED token will not alter existing entries. If the number of indexes is reduced, the entires that still fit in the token will retain their data and the entries that no longer fit will be erased. If the number of indexes is increased, the existing entries retain their data and the new entries are initialized to the token's defaults.

Further details on exact implementation can be found in code comments in token-stack.h file, the platform specific token-manufacturing.h file, the platform specific token.h file, and the platform specific token.c file.

Some functions in this file return an EmberStatus value. See error-def.h for definitions of all EmberStatus return values.

See hal/micro/token.h for source code.

6.22.2 Macro Definition Documentation

6.22.2.1 #define halCommonGetIndexedToken(*data, token, index*)

Note

To better understand the parameters of this macro, refer to the example of token usage above.

Parameters

data	A pointer to where the token data should be placed.
token	The token name used in `DEFINE_*_TOKEN`, prepended with `TOKEN_`.
index	The index to access in the indexed token.

6.22.2.2 #define halCommonGetMfgToken(*data, token*)

Note

To better understand the parameters of this macro, refer to the example of token usage above.

Parameters

data	A pointer to where the token data should be placed.
token	The token name used in `DEFINE_*_TOKEN`, prepended with `TOKEN_`.

6.22.2.3 #define halCommonGetToken(*data, token*)

Note

To better understand the parameters of this macro, refer to the example of token usage above.

Parameters

data	A pointer to where the token data should be placed.
token	The token name used in `DEFINE_*_TOKEN`, prepended with `TOKEN_`.

6.22.2.4 #define halCommonIncrementCounterToken(*token*)

Note

To better understand the parameters of this macro, refer to the example of token usage above.

Parameters

token	The token name used in `DEFINE_*_TOKEN`, prepended with `TOKEN_`.

6.22.2.5 #define halCommonSetIndexedToken(*token, index, data*)

Note

To better understand the parameters of this macro, refer to the example of token usage above.

Parameters

token	The token name used in `DEFINE_*_TOKEN`, prepended with `TOKEN_`.
index	The index to access in the indexed token.
data	A pointer to where the token data should be placed.

6.22.2.6 #define halCommonSetToken(*token, data*)

Note

To better understand the parameters of this macro, refer to the example of token usage above.

Parameters

token	The token name used in `DEFINE_*_TOKEN`, prepended with `TOKEN_`.
data	A pointer to the data being written.

6.22.3 Function Documentation

6.22.3.1 EmberStatus halStackInitTokens (void)

Returns

An EmberStatus value indicating the success or failure of the command.

6.23 Simulated EEPROM

Functions

- void halSimEepromCallback (EmberStatus status)

  The Simulated EEPROM callback function, implemented by the application.
- uint8_t halSimEepromErasePage (void)

  Erases a hardware flash page, if needed.
- uint8_t halSimEepromPagesRemainingToBeErased (void)

  Get count of pages to be erased.
- void halSimEepromStatus (uint16_t *freeWordsUntilFull, uint16_t *totalPageUseCount)

  Provides two basic statistics.

6.23.1 Detailed Description

The Simulated EEPROM system (typically referred to as SimEE) is designed to operate under the Token Access API and provide a non-volatile storage system. Since the flash write cycles are finite, the Simulated EEPROM's primary purpose is to perform wear leveling across several hardware flash pages, ultimately increasing the number of times tokens may be written before a hardware failure.

The Simulated EEPROM needs to periodically perform a page erase operation to recover storage area for future token writes. The page erase operation requires an ATOMIC block of 21ms. Since this is such a long time to not be able to service any interrupts, the page erase operation is under application control providing the application the opportunity to decide when to perform the operation and complete any special handling needed that might be needed.

Note

The best, safest, and recommended practice is for the application to regularly and always call the function halSimEepromErasePage() when the application can expect and deal with the page erase delay. halSimEepromErasePage() will immediately return if there is nothing to erase. If there is something that needs to be erased, doing so as regularly and as soon as possible will keep the SimEE in the healthiest state possible.

::ERASE_CRITICAL_THRESHOLD is the metric the freePtr is compared against. This metric is set to about 3/4 full. The freePtr is a marker used internally by the Simulated EEPROM to track where data ends and where available write space begins. If the freePtr crosses this threhold, halSimEepromCallback() will be called with an EmberStatus of EMBER_SIM_EEPROM_ERASE_PAGE_RED, indicating a critical need for the application to call halSimEepromErasePage() which will erase a hardware page and provide fresh storage for the Simulated EEPROM to write token data. If freePtr is less than the threshold, the callback will have an EmberStatus of EMBER_SIM_EEPROM_ERASE_PAGE_GREEN indicating the application should call halSimEepromErasePage() at its earliest convenience, but doing so is not critically important at this time.

Some functions in this file return an EmberStatus value. See error-def.h for definitions of all EmberStatus return values.

See hal/plugin/sim-eeprom/sim-eeprom.h for source code.

6.23.2 Function Documentation

6.23.2.1 void halSimEepromCallback (EmberStatus *status*)

Parameters

status	An EmberStatus error code indicating one of the conditions described below.

This callback will report an EmberStatus of EMBER_SIM_EEPROM_ERASE_PAGE_GREEN whenever a token is set and a page needs to be erased. If the main application loop does not periodically call halSimEepromErase↩ Page(), it is best to then erase a page in response to EMBER_SIM_EEPROM_ERASE_PAGE_GREEN.

This callback will report an EmberStatus of EMBER_SIM_EEPROM_ERASE_PAGE_RED when the pages *must* be erased to prevent data loss. halSimEepromErasePage() needs to be called until it returns 0 to indicate there are no more pages that need to be erased. Ignoring this indication and not erasing the pages will cause dropping the new data trying to be written.

This callback will report an EmberStatus of EMBER_SIM_EEPROM_FULL when the new data cannot be written due to unerased pages. **Not erasing pages regularly, not erasing in response to EMBER_SIM_EEPROM_E↩ RASE_PAGE_GREEN, or not erasing in response to EMBER_SIM_EEPROM_ERASE_PAGE_RED will cause EMBER_SIM_EEPROM_FULL and the new data will be lost!.** Any future write attempts will be lost as well.

This callback will report an EmberStatus of EMBER_SIM_EEPROM_REPAIRING when the Simulated EEPROM needs to repair itself. While there's nothing for an app to do when the SimEE is going to repair itself (SimEE has to be fully functional for the rest of the system to work), alert the application to the fact that repairing is occuring. There are debugging scenarios where an app might want to know that repairing is happening; such as monitoring frequency.

Note

Common situations will trigger an expected repair, such as using a new chip or changing token definitions.

If the callback ever reports the status EMBER_ERR_FLASH_WRITE_INHIBITED or EMBER_ERR_FLASH_V↩ ERIFY_FAILED, this indicates a catastrophic failure in flash writing, meaning either the address being written is not empty or the write itself has failed. If EMBER_ERR_FLASH_WRITE_INHIBITED is encountered, the function ::halInternalSimEeRepair(false) should be called and the chip should then be reset to allow proper initialization to recover. If EMBER_ERR_FLASH_VERIFY_FAILED is encountered the Simulated EEPROM (and tokens) on the specific chip with this error should not be trusted anymore.

References assert, EMBER_ERR_FLASH_VERIFY_FAILED, EMBER_ERR_FLASH_WRITE_INHIBITED, EM↩ BER_SIM_EEPROM_ERASE_PAGE_GREEN, EMBER_SIM_EEPROM_ERASE_PAGE_RED, EMBER_SIM_E↩ EPROM_FULL, EMBER_SIM_EEPROM_REPAIRING, halInternalSysReset(), halSimEepromErasePage(), and halSimEepromPagesRemainingToBeErased().

6.23.2.2 uint8_t halSimEepromErasePage (void)

This function can be called at anytime from anywhere in the application (except ISRs) and will only take effect if needed (otherwise it will return immediately). Since this function takes 21ms to erase a hardware page during which interrupts cannot be serviced, it is preferable to call this function while in a state that can withstand being unresponsive for so long. The Simulated EEPROM will periodically request through the halSimEepromCallback() that a page be erased. The Simulated EEPROM will never erase a page (which could result in data loss) and relies entirely on the application to call this function to approve a page erase (only one erase per call to this function).

The Simulated EEPROM depends on the ability to move between two Virtual Pages, which are comprised of multiple hardware pages. Before moving to the unused Virtual Page, all hardware pages comprising the unused Virtual Page must be erased first. The erase time of a hardware flash page is 21ms. During this time the chip will be unresponsive and unable to service an interrupt or execute any code (due to the flash being unavailable during the erase procedure). This function is used to trigger a page erase.

Returns

A count of how many hardware pages are left to be erased. This return value allows for calling code to easily loop over this function until the function returns 0.

Referenced by halSimEepromCallback().

6.23.2.3 uint8_t halSimEepromPagesRemainingToBeErased (void)

This function returns the same value halSimEepromErasePage() would return, but without modifying/erasing any flash.

Returns

A count of how many hardware pages are left to be erased. This code assist with loops wanting to know how much is left to erase.

Referenced by halSimEepromCallback().

6.23.2.4 void halSimEepromStatus (uint16_t * *freeWordsUntilFull,* uint16_t * *totalPageUseCount*)

- The number of unused words until SimEE is full
- The total page use count

There is a lot of management and state processing involved with the Simulated EEPROM, and most of it has no practical purpose in the application. These two parameters provide a simple metric for knowing how soon the Simulated EEPROM will be full (::freeWordsUntilFull) and how many times (approximatly) SimEE has rotated pysical flash pages (::totalPageUseCount).

Parameters

freeWordsUntilFull	Number of unused words available to SimEE until the SimEE is full and would trigger an EMBER_SIM_EEPROM_ERASE_PAGE_RED then EMBER_SIM_EEPROM_FULL callback.
totalPageUseCount	The value of the highest page counter indicating how many times the Simulated EEPROM has rotated physical flash pages (and approximate write cycles).

6.24 Simulated EEPROM 2

6.25 Sample APIs for Peripheral Access

Modules

- Serial UART Communication

  This API contains the HAL interfaces that applications must implement for the high-level serial code.
- Button Control

  Sample API functions for using push-buttons.
- Buzzer Control

  Sample API functions for playing tunes on a piezo buzzer.
- LED Control

  Sample API funtions for controlling LEDs.
- Flash Memory Control

  Definition and description of public flash manipulation routines.

6.25.1 Detailed Description

These are sample API for accessing peripherals and can be modified as needed for your applications.

6.26 Serial UART Communication

This API contains the HAL interfaces that applications must implement for the high-level serial code.

Modules

- ASHv3 Functionality for realiable UART communication

Enumerations

- enum SerialBaudRate {
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0 }

  Assign numerical values for variables that hold Baud Rate parameters.
- enum SerialParity {
  DEFINE_PARITY =(NONE) = 0U,
  DEFINE_PARITY =(NONE) = 0U,
  DEFINE_PARITY =(NONE) = 0U }

  CORTEXM3_EFM32_MICRO.

Functions

- void halHostFlushBuffers (void)
- uint16_t halHostEnqueueTx (const uint8_t ∗data, uint16_t length)
- void halHostFlushTx (void)
- uint16_t serialCopyFromRx (const uint8_t ∗data, uint16_t length)
- void emLoadSerialTx (void)

Serial Mode Definitions

These are numerical definitions for the possible serial modes so that code can test for the one being used. There may be additional modes defined in the micro-specific micro.h.

- #define EMBER_SERIAL_UNUSED 0

  A numerical definition for a possible serial mode the code can test for.
- #define EMBER_SERIAL_FIFO 1

  A numerical definition for a possible serial mode the code can test for.
- #define EMBER_SERIAL_BUFFER 2

  A numerical definition for a possible serial mode the code can test for.
- #define EMBER_SERIAL_LOWLEVEL 3

  A numerical definition for a possible serial mode the code can test for.

FIFO Utility Macros

These macros manipulate the FIFO queue data structures to add and remove data.

- #define FIFO_ENQUEUE(queue, data, size)

  Macro that enqueues a byte of data in a FIFO queue.
- #define FIFO_DEQUEUE(queue, size)

  Macro that de-queues a byte of data from a FIFO queue.

Serial HAL APIs

These functions must be implemented by the HAL in order for the serial code to operate. Only the higher-level serial code uses these functions, so they should not be called directly. The HAL should also implement the appropriate interrupt handlers to drain the TX queues and fill the RX FIFO queue.

- EmberStatus halInternalUartInit (uint8_t port, SerialBaudRate rate, SerialParity parity, uint8_t stopBits)

  Initializes the UART to the given settings (same parameters as ::emberSerialInit()).
- void halInternalPowerDownUart (void)

  This function is typically called by halPowerDown() and it is responsible for performing all the work internal to the UART needed to stop the UART before a sleep cycle.
- void halInternalPowerUpUart (void)

  This function is typically called by halPowerUp() and it is responsible for performing all the work internal to the UART needed to restart the UART after a sleep cycle.
- void halInternalStartUartTx (uint8_t port)

  Called by serial code whenever anything is queued for transmission to start any interrupt-driven transmission. May be called when transmission is already in progess.
- void halInternalStopUartTx (uint8_t port)

  Called by serial code to stop any interrupt-driven serial transmission currently in progress.
- EmberStatus halInternalForceWriteUartData (uint8_t port, uint8_t *data, uint8_t length)

  Directly writes a byte to the UART for transmission, regardless of anything currently queued for transmission. Should wait for anything currently in the UART hardware registers to finish transmission first, and block until `data` is finished being sent.
- EmberStatus halInternalForceReadUartByte (uint8_t port, uint8_t *dataByte)

  Directly reads a byte from the UART for reception, regardless of anything currently queued for reception. Does not block if a data byte has not been received.

- void halInternalWaitUartTxComplete (uint8_t port)

  Blocks until the UART has finished transmitting any data in its hardware registers.
- void halInternalRestartUart (void)

  This function is typically called by halInternalPowerUpBoard() and it is responsible for performing all the work internal to the UART needed to restart the UART after a sleep cycle. (For example, resyncing the DMA hardware and the serial FIFO.)
- bool halInternalUartFlowControlRxIsEnabled (uint8_t port)

  Checks to see if the host is allowed to send serial data to the ncp - i.e., it is not being held off by nCTS or an XOFF. Returns true is the host is able to send.
- bool halInternalUartXonRefreshDone (uint8_t port)

  When Xon/Xoff flow control is used, returns true if the host is not being held off and XON refreshing is complete.
- bool halInternalUartTxIsIdle (uint8_t port)

  Returns true if the uart transmitter is idle, including the transmit shift register.
- bool serialDropPacket (void)

  Testing function implemented by the upper layer. Determines whether the next packet should be dropped. Returns true if the next packet should be dropped, false otherwise.
- #define halInternalUartFlowControl(port) do {} while (false)

  This function is used in FIFO mode when flow control is enabled. It is called from emberSerialReadByte(), and based on the number of bytes used in the uart receive queue, decides when to tell the host it may resume transmission.
- #define halInternalUartRxPump(port) do {} while (false)

  This function exists only in software UART (SOFTUART) mode on the EM3xx. This function is called by ::ember↩ SerialReadByte(). It is responsible for maintaining synchronization between the emSerialRxQueue and the UART DMA.
- #define halInternalUart1FlowControlRxIsEnabled() halInternalUartFlowControlRxIsEnabled(1)

  This function is used in FIFO mode when flow control is enabled. It is called from emberSerialReadByte(), and based on the number of bytes used in the uart receive queue, decides when to tell the host it may resume transmission.
- #define halInternalUart1XonRefreshDone() halInternalUartXonRefreshDone(1)

  This function is used in FIFO mode when flow control is enabled. It is called from emberSerialReadByte(), and based on the number of bytes used in the uart receive queue, decides when to tell the host it may resume transmission.
- #define halInternalUart1TxIsIdle() halInternalUartTxIsIdle(1)

  This function is used in FIFO mode when flow control is enabled. It is called from emberSerialReadByte(), and based on the number of bytes used in the uart receive queue, decides when to tell the host it may resume transmission.

Buffered Serial Utility APIs

The higher-level serial code implements these APIs, which the HAL uses to deal with buffered serial output.

- void emSerialBufferNextMessageIsr (EmSerialBufferQueue *q)

  When new serial transmission is started and `bufferQueue->nextByte` is equal to NULL, this can be called to set up `nextByte` and `lastByte` for the next message.
- void emSerialBufferNextBlockIsr (EmSerialBufferQueue *q, uint8_t port)

  When a serial transmission is in progress and `bufferQueue->nextByte` has been sent and incremented leaving it equal to lastByte, this should be called to set up `nextByte` and `lastByte` for the next block.

Virtual UART API

API used by the stack in debug builds to receive data arriving over the virtual UART.

- void halStackReceiveVuartMessage (uint8_t *data, uint8_t length)

  When using a debug build with virtual UART support, this API is called by the stack when virtual UART data has been received over the debug channel.

6.26.1 Detailed Description

This header describes the interface between the high-level serial APIs in serial/serial.h and the low level UART implementation.

Some functions in this file return an EmberStatus value. See error-def.h for definitions of all EmberStatus return values.

See hal/micro/serial.h for source code.

6.26.2 Macro Definition Documentation

6.26.2.1 #define EMBER_SERIAL_BUFFER 2

6.26.2.2 #define EMBER_SERIAL_FIFO 1

6.26.2.3 #define EMBER_SERIAL_LOWLEVEL 3

6.26.2.4 #define EMBER_SERIAL_UNUSED 0

6.26.2.5 #define FIFO_DEQUEUE(*queue, size*)

Value:

```
(queue)->fifo[(queue)->tail];                    \
  (queue)->tail = (((queue)->tail + 1) % (size)); \
  (queue)->used--
```

Parameters

queue	Pointer to the FIFO queue.
size	Size used to control the wrap-around of the FIFO pointers.

Referenced by USBD_RemoteWakeup().

6.26.2.6 #define FIFO_ENQUEUE(*queue, data, size*)

Value:

```
do {                                             \
    (queue)->fifo[(queue)->head] = (data);        \
    (queue)->head = (((queue)->head + 1) % (size)); \
    (queue)->used++;                              \
  } while (0)
```

Parameters

queue	Pointer to the FIFO queue.
data	Data byte to be enqueued.
size	Size used to control the wrap-around of the FIFO pointers.

6.26.2.7 #define halInternalUart1FlowControlRxIsEnabled( ) halInternalUartFlowControlRxIsEnabled(1)

Parameters

port	Serial port number (0 or 1). (Does nothing for port 0)

6.26.2.8 #define halInternalUart1TxIsIdle( ) halInternalUartTxIsIdle(1)

Parameters

port	Serial port number (0 or 1). (Does nothing for port 0)

6.26.2.9 #define halInternalUart1XonRefreshDone( ) halInternalUartXonRefreshDone(1)

Parameters

port	Serial port number (0 or 1). (Does nothing for port 0)

6.26.2.10 #define halInternalUartFlowControl(*port*) do {} while (false)

Parameters

port	Serial port number (0 or 1). (Does nothing for port 0)

6.26.2.11 #define halInternalUartRxPump(*port*) do {} while (false)

Parameters

port	Serial port number (0 or 1).

6.26.3 Enumeration Type Documentation

6.26.3.1 enum SerialBaudRate

Enumerator

DEFINE_BAUD
DEFINE_BAUD
DEFINE_BAUD
DEFINE_BAUD
DEFINE_BAUD
DEFINE_BAUD
DEFINE_BAUD
DEFINE_BAUD
DEFINE_BAUD

DEFINE_BAUD
DEFINE_BAUD
DEFINE_BAUD
DEFINE_BAUD
DEFINE_BAUD
DEFINE_BAUD
DEFINE_BAUD
DEFINE_BAUD
DEFINE_BAUD
DEFINE_BAUD

6.26.3.2 enum SerialParity

Assign numerical values for the types of parity. Use for variables that hold Parity parameters.

Enumerator

DEFINE_PARITY
DEFINE_PARITY
DEFINE_PARITY

6.26.4 Function Documentation

6.26.4.1 void emLoadSerialTx (void)

6.26.4.2 void emSerialBufferNextBlockIsr (EmSerialBufferQueue * *q,* uint8_t *port*)

Parameters

q	Pointer to the buffer queue structure for the port.
port	Serial port number (0 or 1).

6.26.4.3 void emSerialBufferNextMessageIsr (EmSerialBufferQueue * *q*)

Parameters

q	Pointer to the buffer queue structure for the port.

6.26.4.4 uint16_t halHostEnqueueTx (const uint8_t * *data,* uint16_t *length*)

6.26.4.5 void halHostFlushBuffers (void)

6.26.4.6 void halHostFlushTx (void)

6.26.4.7 EmberStatus halInternalForceReadUartByte (uint8_t *port*, uint8_t * *dataByte*)

Parameters

port	Serial port number (0 or 1).
dataByte	The byte to receive data into.

6.26.4.8 EmberStatus halInternalForceWriteUartData (uint8_t *port,* uint8_t * *data,* uint8_t *length*)

Parameters

port	Serial port number (0 or 1).
data	Pointer to the data to be transmitted.
length	The length of data to be transmitted

6.26.4.9 void halInternalPowerDownUart (void)

6.26.4.10 void halInternalPowerUpUart (void)

6.26.4.11 void halInternalRestartUart (void)

6.26.4.12 void halInternalStartUartTx (uint8_t *port*)

Parameters

port	Serial port number (0 or 1).

6.26.4.13 void halInternalStopUartTx (uint8_t *port*)

Parameters

port	Serial port number (0 or 1).

6.26.4.14 bool halInternalUartFlowControlRxIsEnabled (uint8_t *port*)

6.26.4.15 EmberStatus halInternalUartInit (uint8_t *port,* SerialBaudRate *rate,* SerialParity *parity,* uint8_t *stopBits*)

Parameters

port	Serial port number (0 or 1).
rate	Baud rate (see SerialBaudRate).
parity	Parity value (see SerialParity).
stopBits	Number of stop bits.

Returns

An error code if initialization failed (such as invalid baud rate), otherise EMBER_SUCCESS.

6.26.4.16 bool halInternalUartTxIsIdle (uint8_t *port*)

6.26.4.17 bool halInternalUartXonRefreshDone (uint8_t *port*)

6.26.4.18 void halInternalWaitUartTxComplete (uint8_t *port*)

Parameters

port	Serial port number (0 or 1).

6.26.4.19 void halStackReceiveVuartMessage (uint8_t * *data,* uint8_t *length*)

Parameters

data	Pointer to the the data received
length	Length of the data received

6.26.4.20 uint16_t serialCopyFromRx (const uint8_t * *data,* uint16_t *length*)

6.26.4.21 bool serialDropPacket (void)

6.27 ASHv3 Functionality for realiable UART communication

Variables

- uint8_t AshTxDmaBuffer::data [MAX_ASH_PACKET_SIZE]
- uint8_t * AshTxDmaBuffer::finger
- AshTxDmaBufferState AshTxDmaBuffer::state
- bool AshTxDmaBuffer::resend
- uint8_t AshTxDmaBuffer::resendCount
- bool AshTxDmaBuffer::isCorrupt
- AshTxDmaBuffer AshTxState::dmaBufferA
- AshTxDmaBuffer AshTxState::dmaBufferB
- AshTxDmaBuffer * AshTxState::dmaBuffer
- uint8_t AshTxState::outgoingFrameCounter
- uint8_t AshTxState::ackNackFrameCounter
- bool AshTxState::serialLayerReplied
- uint8_t AshRxState::payload [MAX_ASH_PAYLOAD_SIZE]
- uint8_t AshRxState::payloadIndex
- uint8_t AshRxState::escapedPayloadIndex
- uint8_t AshRxState::payloadLength
- uint8_t AshRxState::controlByte
- uint8_t AshRxState::headerEscapeByte
- uint16_t AshRxState::computedCrc
- uint8_t AshRxState::highCrcByte
- uint8_t AshRxState::inBetweenCrcByte
- AshRxFrameState AshRxState::frameState
- bool AshRxState::escapeNextByte

Application Functions

Implement these functions in your application

The following functions are only for builds that support software flow control and that are compiled with EMBER_↩ APPLICATION_SUPPORTS_SOFTWARE_FLOW_CONTROL

- void emberXOnHandler (void)

  Tell the application that we received an XON.
- void emberXOffHandler (void)

  Tell the application that we received an XOFF.

6.27.1 Detailed Description

```
*    ASHv3 Header (4 bytes)
*           0                   1                   2                   3
+-+-+-+-+-+-+-+-+|+-+-+-+-+-+-+-+-+|+-+-+-+-+-+-+-+-+|+-+-+-+-+-+-+-+-+
+   ASH_FLAG     |  HEADER ESCAPE  |  CONTROL BYTE   | PAYLOAD LENGTH +
+-+-+-+-+-+-+-+-+|+-+-+-+-+-+-+-+-+|+-+-+-+-+-+-+-+-+|+-+-+-+-+-+-+-+-+
+
+    ASH_FLAG is defined below
+
+    Header Escape bytes contains escape data for the control byte and the payload
+    length byte, and is encoded as:
+
+-+-+-+-+-+-+-+-+
+A B             +
+-+-+-+-+-+-+-+-+
+
+     where A and B are booleans representing:
+     A = is the control byte escaped?          (1 << 7)
+     B = is the payload length byte escaped?   (1 << 6)
+
+    The control byte has the syntax:
+
+-+-+-+-+-+-+-+-+
+ T + OFC + AFC +
+-+-+-+-+-+-+-+-+
+
+    where:
+
+     T = type (see AshMessageType below)
+     OFC = outgoing frame counter
+     AFC = ack/nack frame counter
+
+    Payload length is in the range: [0, MAX_ASH_PAYLOAD_SIZE] (defined below)
+
+    The CRC is 3 bytes and contains 2 bytes of data. It is stored in such a way
+    that each of its bytes never needs escaping.
+
+    The escape bytes are:
+
+     ASH_FLAG             0b01111110
+     ASH_ESC              0b01111101
+     ASH_XON              0b00010001
+     ASH_XOFF             0b00010011
+     ASH_COBRA_FORCE_BOOT 0b11111000
+                               ^ They all have this bit set
+
+    The CRC is expanded in the following way:
+
+-+-+-+-+-+-+-+-+|+-+-+-+-+-+-+-+-+
+  High CRC      +     Low CRC     +
+-+-+-+-+-+-+-+-+|+-+-+-+-+-+-+-+-+
+    F E D C B A 9 8   7 6 5 4 3 2 1 0
+          ^                 ^-------------+
+-----------------------------+ |
+-+-+-+-+-+-+-+-+|+-+-+-+-+-+-+-+-+|+-+-+-+-+-+-+-+-+
+  New High CRC  |   New Low CRC   |   Bits         +
+-+-+-+-+-+-+-+-+|+-+-+-+-+-+-+-+-+|+-+-+-+-+-+-+-+-+
+                                     7 6 5 4 3 2 1 0
+
+    ------------------------------------------------------------------------------
+    PROTOCOL
+
* Design
*
*    The upward interface sends and receives a stream of bytes. The upper layer
*    writes and reads data in blocks, but the block boundaries are not preserved
*    over the wire. The UART framing is independent of the upward interface; the
*    data is treated as a stream.
*
*    The protocol is organized around using a single DMA operation to send a
*    frame. The maximum size is of a DMA buffer is 100 bytes. Getting good
*    throughput requires allowing two frames in flight; having more does not help.
```

```
*    The UART is responsible for ACKing and retrying frames over the wire. Each
*    frame contains a sequence number, the sequence number of the most recently
*    received frame, and a flag indicating whether or not corrupt data was
*    received after that frame.
*
*    An OFC value of 0 is never used because 0 is the AFC's default/uninitialized
*    value.
*
* Resets
*
*    To reset the UART link a device sends a RESET frame and the peer responds
*    with a RESET_ACK frame. RESET and RESET_ACK frames have a single data byte that
*    gives the type of reset. For a RESET packet, the OFC is set to 1 and the AFC
*    is set to 0. For a RESET_ACK packet with no payload, the OFC is 1 and the AFC
*    is 1. For an RESET_ACK with non-empty payload, the OFC is 2 and the AFC is 1.
*
* ACKs, NACKs and retransmission
*
*    The driver retains transmitted frames until a matching ACK is received.
*    Frames are retransmitted if a NACK is received or, in case an ACK is lost,
*    if a resend timer expires.
*
*    ACK, NACK, and RESET_ACK frames can carry data. The ACK/NACK counter has the
*    frame counter from the last correctly received frame. A NACK frame is sent if
*    corrupt data or a corrupt frame is received. An ACK frame is sent whenever a
*    data-containing frame is received correctly. If an ACK/NACK has no payload,
*    its OFC is ignored and no ACK is sent in response. Note that we record an
*    RESET_ACK's OFC whether or not its payload is populated. An ACK/NACK/RESET_ACK with
*    empty payload must not have an incremented OFC.
*
*    After sending a RESET frame all incoming ACK and NACK frames are ignored
*    until an RESET_ACK is received. Additional incoming RESET frames are answered
*    with a matching RESET_ACK.
*
* Waking
*
*    When using the UART to wake up the other device, the byte 0xFF (ASH_WAKEUP)
*    can be sent in betweem frames. This only applies between frames, so 0xFF does
*    not need to be escaped within a frame. Any number of wake bytes can be sent
*    between the ASH_FLAG at the end of one frame and an ASH_FLAG at the beginning
*    of the next frame.
*
```

6.27.2 Function Documentation

6.27.2.1 void emberXOffHandler (void)

6.27.2.2 void emberXOnHandler (void)

6.27.3 Variable Documentation

6.27.3.1 uint8_t AshTxState::ackNackFrameCounter

6.27.3.2 uint16_t AshRxState::computedCrc

6.27.3.3 uint8_t AshRxState::controlByte

6.27.3.4 uint8_t AshTxDmaBuffer::data[MAX_ASH_PACKET_SIZE]

6.27.3.5 AshTxDmaBuffer* AshTxState::dmaBuffer

6.27.3.6 **AshTxDmaBuffer** AshTxState::dmaBufferA

6.27.3.7 **AshTxDmaBuffer** AshTxState::dmaBufferB

6.27.3.8 uint8_t AshRxState::escapedPayloadIndex

6.27.3.9 bool AshRxState::escapeNextByte

6.27.3.10 uint8_t* AshTxDmaBuffer::finger

6.27.3.11 **AshRxFrameState** AshRxState::frameState

6.27.3.12 uint8_t AshRxState::headerEscapeByte

6.27.3.13 uint8_t AshRxState::highCrcByte

6.27.3.14 uint8_t AshRxState::inBetweenCrcByte

6.27.3.15 bool AshTxDmaBuffer::isCorrupt

6.27.3.16 uint8_t AshTxState::outgoingFrameCounter

6.27.3.17 uint8_t AshRxState::payload[**MAX_ASH_PAYLOAD_SIZE**]

6.27.3.18 uint8_t AshRxState::payloadIndex

6.27.3.19 uint8_t AshRxState::payloadLength

6.27.3.20 bool AshTxDmaBuffer::resend

6.27.3.21 uint8_t AshTxDmaBuffer::resendCount

6.27.3.22 bool AshTxState::serialLayerReplied

6.27.3.23 **AshTxDmaBufferState** AshTxDmaBuffer::state

6.28 Button Control

Sample API functions for using push-buttons.

Functions

- void halInternalInitButton (void)

  Initializes the buttons. This function is automatically called by halInit().
- uint8_t halButtonState (uint8_t button)

  Returns the current state (pressed or released) of a button.
- uint8_t halButtonPinState (uint8_t button)

  Returns the current state (pressed or released) of the pin associated with a button.
- void halButtonIsr (uint8_t button, uint8_t state)

  A callback called in interrupt context whenever a button changes its state.

Button State Definitions

A set of numerical definitions for use with the button APIs indicating the state of a button.

- #define BUTTON_PRESSED 1

  Button state is pressed.
- #define BUTTON_RELEASED 0

  Button state is released.

6.28.1 Detailed Description

See button.h for source code.

6.28.2 Macro Definition Documentation

6.28.2.1 #define BUTTON_PRESSED 1

6.28.2.2 #define BUTTON_RELEASED 0

6.28.3 Function Documentation

6.28.3.1 void halButtonIsr (uint8_t *button,* uint8_t *state*)

Must be implemented by the application. This function should contain the functionality to be executed in response to changes of state in each of the buttons, or callbacks to the appropriate functionality.

Parameters

button	The button which has changed state, either BUTTON0 or BUTTON1 as defined in the appropriate BOARD_HEADER.
state	The new state of the button referenced by the button parameter, either BUTTON_PRESSED if the button has been pressed or BUTTON_RELEASED if the button has been released.

6.28.3.2 uint8_t halButtonPinState (uint8_t *button*)

This reads the actual state of the pin and can be used on startup to determine the initial position of the buttons.

Parameters

button	The button being queried, either BUTTON0 or BUTTON1 as defined in the appropriate BOARD_HEADER.

Returns

BUTTON_PRESSED if the button is pressed or BUTTON_RELEASED if the button is not pressed.

6.28.3.3 uint8_t halButtonState (uint8_t *button*)

Note

This function is correlated with halButtonIsr() and so returns the shadow state rather than reading the actual state of the pin.

Parameters

button	The button being queried, either BUTTON0 or BUTTON1 as defined in the appropriate BOARD_HEADER.

Returns

BUTTON_PRESSED if the button is pressed or BUTTON_RELEASED if the button is not pressed.

6.28.3.4 void halInternalInitButton (void)

6.29 Buzzer Control

Sample API functions for playing tunes on a piezo buzzer.

Functions

- void halPlayTune_P (uint8_t PGM *tune, bool bkg)

  Plays a tune on the piezo buzzer.
- void halStackIndicatePresence (void)

  Causes something to happen on a node (such as playing a tune on the buzzer) that can be used to indicate where it physically is.

Note Definitions

Flats are used instead of sharps because # is a special character.

- #define NOTE_C3 119

  A note which can be used in tune structure definitions.
- #define NOTE_Db3 112

  A note which can be used in tune structure definitions.
- #define NOTE_D3 106

  A note which can be used in tune structure definitions.
- #define NOTE_Eb3 100

  A note which can be used in tune structure definitions.
- #define NOTE_E3 94

  A note which can be used in tune structure definitions.
- #define NOTE_F3 89

  A note which can be used in tune structure definitions.
- #define NOTE_Gb3 84

  A note which can be used in tune structure definitions.
- #define NOTE_G3 79

  A note which can be used in tune structure definitions.
- #define NOTE_Ab3 74

  A note which can be used in tune structure definitions.
- #define NOTE_A3 70

  A note which can be used in tune structure definitions.
- #define NOTE_Bb3 66

  A note which can be used in tune structure definitions.
- #define NOTE_B3 63

  A note which can be used in tune structure definitions.
- #define NOTE_C4 59

  A note which can be used in tune structure definitions.
- #define NOTE_Db4 55

  A note which can be used in tune structure definitions.
- #define NOTE_D4 52

  A note which can be used in tune structure definitions.
- #define NOTE_Eb4 49

  A note which can be used in tune structure definitions.

- #define NOTE_E4 46

  A note which can be used in tune structure definitions.
- #define NOTE_F4 44

  A note which can be used in tune structure definitions.
- #define NOTE_Gb4 41

  A note which can be used in tune structure definitions.
- #define NOTE_G4 39

  A note which can be used in tune structure definitions.
- #define NOTE_Ab4 37

  A note which can be used in tune structure definitions.
- #define NOTE_A4 35

  A note which can be used in tune structure definitions.
- #define NOTE_Bb4 33

  A note which can be used in tune structure definitions.
- #define NOTE_B4 31

  A note which can be used in tune structure definitions.
- #define NOTE_C5 29

  A note which can be used in tune structure definitions.
- #define NOTE_Db5 27

  A note which can be used in tune structure definitions.
- #define NOTE_D5 26

  A note which can be used in tune structure definitions.
- #define NOTE_Eb5 24

  A note which can be used in tune structure definitions.
- #define NOTE_E5 23

  A note which can be used in tune structure definitions.
- #define NOTE_F5 21

  A note which can be used in tune structure definitions.
- #define NOTE_Gb5 20

  A note which can be used in tune structure definitions.
- #define NOTE_G5 19

  A note which can be used in tune structure definitions.
- #define NOTE_Ab5 18

  A note which can be used in tune structure definitions.
- #define NOTE_A5 17

  A note which can be used in tune structure definitions.
- #define NOTE_Bb5 16

  A note which can be used in tune structure definitions.
- #define NOTE_B5 15

  A note which can be used in tune structure definitions.

6.29.1 Detailed Description

See buzzer.h for source code.

6.29.2 Macro Definition Documentation

6.29.2.1 #define NOTE_A3 70

6.29.2.2 #define NOTE_A4 35

6.29.2.3 #define NOTE_A5 17

6.29.2.4 #define NOTE_Ab3 74

6.29.2.5 #define NOTE_Ab4 37

6.29.2.6 #define NOTE_Ab5 18

6.29.2.7 #define NOTE_B3 63

6.29.2.8 #define NOTE_B4 31

6.29.2.9 #define NOTE_B5 15

6.29.2.10 #define NOTE_Bb3 66

6.29.2.11 #define NOTE_Bb4 33

6.29.2.12 #define NOTE_Bb5 16

6.29.2.13 #define NOTE_C3 119

6.29.2.14 #define NOTE_C4 59

6.29.2.15 #define NOTE_C5 29

6.29.2.16 #define NOTE_D3 106

6.29.2.17 #define NOTE_D4 52

6.29.2.18 #define NOTE_D5 26

6.29.2.19 #define NOTE_Db3 112

6.29.2.20 #define NOTE_Db4 55

6.29.2.21 #define NOTE_Db5 27

6.29.2.22 #define NOTE_E3 94

6.29.2.23 #define NOTE_E4 46

6.29.2.24 #define NOTE_E5 23

6.29.2.25 #define NOTE_Eb3 100

6.29.2.26 #define NOTE_Eb4 49

6.29.2.27 #define NOTE_Eb5 24

6.29.2.28 #define NOTE_F3 89

6.29.2.29 #define NOTE_F4 44

6.29.2.30 #define NOTE_F5 21

6.29.2.31 #define NOTE_G3 79

6.29.2.32 #define NOTE_G4 39

6.29.2.33 #define NOTE_G5 19

6.29.2.34 #define NOTE_Gb3 84

6.29.2.35 #define NOTE_Gb4 41

6.29.2.36 #define NOTE_Gb5 20

6.29.3 Function Documentation

6.29.3.1 void halPlayTune_P (uint8_t PGM * *tune*, bool *bkg*)

The tune is played in the background if ::bkg is true. Otherwise, the API blocks until the playback of the tune is complete. halPlayTune_P() is not meant to be called back-to-back.

Parameters

tune	A pointer to tune to play.
bkg	Determines whether the tune plays in the background. If true, tune plays in background; if false, tune plays in foreground.

A tune is implemented as follows:

```
1 uint8_t PGM hereIamTune[] = {  //All tunes are stored in flash.
2    NOTE_B4,  1,                //Plays the note B4 for 100 milliseconds.
3    0,        1,                //Pause for 100 milliseconds.
4    NOTE_B5,  5,                //Plays the note B5 for 500 milliseconds.
5    0,        0                 //NULL terminates the tune.
6  };
```

6.29.3.2 void halStackIndicatePresence (void)

6.30 LED Control

Sample API funtions for controlling LEDs.

Typedefs

- typedef enum HalBoardLedPins HalBoardLed

  Ensures that the definitions from the BOARD_HEADER are always used as parameters to the LED functions.

Functions

- void halInternalInitLed (void)

  Configures GPIOs pertaining to the control of LEDs.
- void halToggleLed (HalBoardLed led)

  Atomically wraps an XOR or similar operation for a single GPIO pin attached to an LED.
- void halSetLed (HalBoardLed led)

  Turns on (sets) a GPIO pin connected to an LED so that the LED turns on.
- void halClearLed (HalBoardLed led)

  Turns off (clears) a GPIO pin connected to an LED, which turns off the LED.
- void halStackIndicateActivity (bool turnOn)

  Called by the stack to indicate activity over the radio (for both transmission and reception). It is called once with `turnOn` *true and shortly thereafter with* `turnOn` *false.*

6.30.1 Detailed Description

When specifying an LED to use, always use the BOARDLEDx definitions that are defined within the BOARD_HE↩ ADER.

See led.h for source code.

6.30.2 Typedef Documentation

6.30.2.1 typedef enum HalBoardLedPins HalBoardLed

6.30.3 Function Documentation

6.30.3.1 void halClearLed (HalBoardLed *led*)

Parameters

led	Identifier (from BOARD_HEADER) for the LED to turn off.

6.30.3.2 void halInternalInitLed (void)

6.30.3.3 void halSetLed (HalBoardLed *led*)

Parameters

led	Identifier (from BOARD_HEADER) for the LED to turn on.

6.30.3.4 void halStackIndicateActivity (bool *turnOn*)

Typically does something interesting, such as change the state of an LED.

Parameters

turnOn	See Usage.

6.30.3.5 void halToggleLed (**HalBoardLed** *led*)

Parameters

led	Identifier (from BOARD_HEADER) for the LED to be toggled.

6.31 Flash Memory Control

Definition and description of public flash manipulation routines.

Functions

- bool halFlashEraseIsActive (void)

  Tells the calling code if a Flash Erase operation is active.

6.31.1 Detailed Description

Note

During an erase or a write the flash is not available, which means code will not be executable from flash. These routines still execute from flash, though, since the bus architecture can support doing so. **Additonally, this also means all interrupts will be disabled.**

Hardware documentation indicates 40us for a write and 21ms for an erase.

See flash.h for source code.

6.31.2 Function Documentation

6.31.2.1 bool halFlashEraseIsActive (void)

This state is import to know because Flash Erasing is ATOMIC for 21ms and could disrupt interrupt latency. But if an ISR can know that it wasn't serviced immediately due to Flash Erasing, then the ISR has the opportunity to correct in whatever manner it needs to.

Returns

A bool flag: true if Flash Erase is active, false otherwise.

6.32 USB Device Stack Library

Modules

- USB Common API

  Sample API functions for using USB.
- USB Device API

  USB DEVICE protocol stack, see USB Device Stack Library page for detailed documentation.

  .

6.32.1 Detailed Description

The source files for the USB device stack resides in the usb directory and follows the naming convention: em_↩ usbd*nnn*.c/h.

- Introduction
- The device stack API
- Configuring the device stack

6.32.2 Introduction

The USB device protocol stack provides an API which makes it possible to create USB devices with a minimum of effort. The device stack supports control, bulk and interrupt transfers.

The stack is highly configurable to suit various needs, it does also contain useful debugging features together with several demonstration projects to get you started fast.

We recommend that you read through this documentation, then proceed to build and test a few example projects before you start designing your own device.

6.32.3 The device stack API

This section contains brief descriptions of the functions in the API. You will find detailed information on input and output parameters and return values by clicking on the hyperlinked function names. It is also a good idea to study the code in the USB demonstration projects.

Your application code must include one header file: *em_usb.h*.

All functions defined in the API can be called from within interrupt handlers.

Pitfalls:
The USB peripheral will fill your receive buffers in quantities of WORD's (4 bytes). Transmit and receive buffers must be WORD aligned, in addition when allocating storage for receive buffers, round size up to next WORD boundary. If it is possible that the host will send more data than your device expects, round buffer size up to the next multiple of maxpacket size for the relevant endpoint to avoid data corruption.

Transmit buffers passed to must be statically allocated because only initiates the transfer. When the host decide to actually perform the transfer, your data must be available.

USBD_Init()
This function is called to register your device and all its properties with the device stack. The application must fill in a USBD_Init_TypeDef structure prior to calling. Refer to DeviceInitCallbacks for the optional callback functions defined within this structure. When this function has been called your device is ready to be enumerated by the USB host.

USBD_Read(), USBD_Write()
These functions initiate data transfers.
initiate a transfer of data *from* host *to* device (an *OUT* transfer in USB terminology).
initiate a transfer of data *from* device *to* host (an *IN* transfer).

When the USB host actually performs the transfer, your application will be notified by means of a callback function which you provide (optionally). Refer to TransferCallback for details of the callback functionality.

USBD_AbortTransfer(), USBD_AbortAllTransfers()
These functions terminate transfers that are initiated, but has not yet taken place. If a transfer is initiated with but the USB host never actually peform the transfers, these functions will deactivate the transfer setup to make the USB device endpoint hardware ready for new (and potentially) different transfers.

USBD_Connect(), USBD_Disconnect()
These functions turns the data-line (D+ or D-) pullup on or off. They can be used to force reenumeration. It's good practice to delay at least one second between to allow the USB host to unload the currently active device driver.

USBD_EpIsBusy()
Check if an endpoint is busy.

USBD_StallEp(), USBD_UnStallEp()
These functions stalls or un-stalls an endpoint. This functionality may not be needed by your application, but the USB device stack use them in response to standard setup commands SET_FEATURE and CLEAR_FEATURE. They may be useful when implementing some USB classes, e.g. a mass storage device use them extensively.

USBD_RemoteWakeup()
Used in SUSPENDED state (see USB_Status_TypeDef) to signal resume to host. It's the applications responsibility to adhere to the USB standard which states that a device can not signal resume before it has been SUSPENDED for at least 5 ms. The function will also check the configuration descriptor defined by the application to see if it is legal for the device to signal resume.

USBD_GetUsbState()
Returns the device USB state (see USBD_State_TypeDef). Refer to Figure 9-1. "Device State Diagram" in the USB revision 2.0 specification.

USBD_GetUsbStateName()
Returns a text string naming a given USB device state.

The transfer complete callback function:

USB_XferCompleteCb_TypeDef() is called when a transfer completes. It is called with three parameters, the status of the transfer, the number of bytes transferred and the number of bytes remaining. It may not always be needed to have a callback on transfer completion, but you should keep in mind that a transfer may be aborted when you least expect it. A transfer will be aborted if host stalls the endpoint, if host resets your device, if host unconfigures your device or if you unplug your device cable and the device is selfpowered. is also called if your application use calls.

Note

This callback is called from within an interrupt handler with interrupts disabled.

Optional callbacks passed to the stack via the USBD_Init() function:

These callbacks are all optional, and it is up to the application programmer to decide if the application needs the functionality they provide.

Note

These callbacks are all called from within an interrupt handler with interrupts disabled.

USBD_UsbResetCb_TypeDef() is called each time reset signalling is sensed on the USB wire.

USBD_SofIntCb_TypeDef() is called with framenumber as a parameter on each SOF interrupt.

USBD_DeviceStateChangeCb_TypeDef() is called whenever the device state change. Useful for detecting e.g. SUSPENDED state change in order to reduce current consumption of buspowered devices. The USB HID keyboard example project has a good example on how to use this callback.

USBD_IsSelfPoweredCb_TypeDef() is called by the device stack when host queries the device with a standard setup GET_STATUS command to check if the device is currently selfpowered or buspowered. This feature is only applicable on selfpowered devices which also works when only buspower is available.

USBD_SetupCmdCb_TypeDef() is called each time a setup command is received from host. Use this callback to override or extend the default handling of standard setup commands, and to implement class or vendor specific setup commands. The USB HID keyboard example project has a good example on how to use this callback.

6.32.4 Configuring the device stack

Your application must provide a header file named *usbconfig.h*. This file must contain the following #define:

```
#define NUM_EP_USED n     // Your application use 'n' endpoints in
*                           // addition to endpoint 0.
```

6.33 USB Common API

Sample API functions for using USB.

Data Structures

- struct USB_Setup_TypeDef
  USB Setup request package.
- struct USB_DeviceDescriptor_TypeDef
  USB Device Descriptor.
- struct USB_ConfigurationDescriptor_TypeDef
  USB Configuration Descriptor.
- struct USB_InterfaceDescriptor_TypeDef
  USB Interface Descriptor.
- struct USB_EndpointDescriptor_TypeDef
  USB Endpoint Descriptor.
- struct USB_StringDescriptor_TypeDef
  USB String Descriptor.

Macros

- #define USB_SETUP_DIR_OUT 0
- #define USB_SETUP_DIR_IN 1
- #define USB_SETUP_DIR_MASK 0x80
- #define USB_SETUP_DIR_D2H 0x80
- #define USB_SETUP_DIR_H2D 0x00
- #define USB_SETUP_TYPE_STANDARD 0
- #define USB_SETUP_TYPE_CLASS 1
- #define USB_SETUP_TYPE_VENDOR 2
- #define USB_SETUP_TYPE_STANDARD_MASK 0x00
- #define USB_SETUP_TYPE_CLASS_MASK 0x20
- #define USB_SETUP_TYPE_VENDOR_MASK 0x40
- #define USB_SETUP_RECIPIENT_DEVICE 0
- #define USB_SETUP_RECIPIENT_INTERFACE 1
- #define USB_SETUP_RECIPIENT_ENDPOINT 2
- #define USB_SETUP_RECIPIENT_OTHER 3
- #define GET_STATUS 0
- #define CLEAR_FEATURE 1
- #define SET_FEATURE 3
- #define SET_ADDRESS 5
- #define GET_DESCRIPTOR 6
- #define SET_DESCRIPTOR 7
- #define GET_CONFIGURATION 8
- #define SET_CONFIGURATION 9
- #define GET_INTERFACE 10
- #define SET_INTERFACE 11
- #define SYNCH_FRAME 12
- #define USB_HID_GET_REPORT 0x01
- #define USB_HID_GET_IDLE 0x02
- #define USB_HID_SET_REPORT 0x09
- #define USB_HID_SET_IDLE 0x0A

- #define USB_HID_SET_PROTOCOL 0x0B
- #define USB_CDC_SETLINECODING 0x20
- #define USB_CDC_GETLINECODING 0x21
- #define USB_CDC_SETCTRLLINESTATE 0x22
- #define USB_MSD_BOTRESET 0xFF
- #define USB_MSD_GETMAXLUN 0xFE
- #define USB_DEVICE_DESCRIPTOR 1
- #define USB_CONFIG_DESCRIPTOR 2
- #define USB_STRING_DESCRIPTOR 3
- #define USB_INTERFACE_DESCRIPTOR 4
- #define USB_ENDPOINT_DESCRIPTOR 5
- #define USB_DEVICE_QUALIFIER_DESCRIPTOR 6
- #define USB_OTHER_SPEED_CONFIG_DESCRIPTOR 7
- #define USB_INTERFACE_POWER_DESCRIPTOR 8
- #define USB_HUB_DESCRIPTOR 0x29
- #define USB_HID_DESCRIPTOR 0x21
- #define USB_HID_REPORT_DESCRIPTOR 0x22
- #define USB_CS_INTERFACE_DESCRIPTOR 0x24
- #define USB_DEVICE_DESCSIZE 18
- #define USB_CONFIG_DESCSIZE 9
- #define USB_INTERFACE_DESCSIZE 9
- #define USB_ENDPOINT_DESCSIZE 7
- #define USB_DEVICE_QUALIFIER_DESCSIZE 10
- #define USB_OTHER_SPEED_CONFIG_DESCSIZE 9
- #define USB_HID_DESCSIZE 9
- #define USB_CDC_HEADER_FND_DESCSIZE 5
- #define USB_CDC_CALLMNG_FND_DESCSIZE 5
- #define USB_CDC_ACM_FND_DESCSIZE 4
- #define USB_EP0_SIZE 8
- #define USB_EP1_SIZE 8
- #define USB_EP2_SIZE 8
- #define USB_EP3_SIZE 64
- #define USB_EP4_SIZE 32
- #define USB_EP5_SIZE 64
- #define USB_EP6_SIZE 512
- #define USB_MAX_EP_SIZE 64
- #define USB_EPTYPE_CTRL 0
- #define USB_EPTYPE_ISOC 1
- #define USB_EPTYPE_BULK 2
- #define USB_EPTYPE_INTR 3
- #define USB_EP_DIR_IN 0x80
- #define USB_SETUP_PKT_SIZE 8
- #define USB_EPNUM_MASK 0x0F
- #define USB_LANGID_ENUS 0x0409
- #define USB_MAX_DEVICE_ADDRESS 127
- #define CONFIG_DESC_BM_REMOTEWAKEUP 0x20
- #define CONFIG_DESC_BM_SELFPOWERED 0x40
- #define CONFIG_DESC_BM_RESERVED_D7 0x80
- #define CONFIG_DESC_BM_TRANSFERTYPE 0x03
- #define CONFIG_DESC_MAXPOWER_mA(x) (((x) + 1) / 2)
- #define DEVICE_IS_SELFPOWERED 0x0001
- #define REMOTE_WAKEUP_ENABLED 0x0002
- #define USB_FEATURE_ENDPOINT_HALT 0
- #define USB_FEATURE_DEVICE_REMOTE_WAKEUP 1
- #define HUB_FEATURE_PORT_RESET 4

- #define HUB_FEATURE_PORT_POWER 8
- #define HUB_FEATURE_C_PORT_CONNECTION 16
- #define HUB_FEATURE_C_PORT_RESET 20
- #define HUB_FEATURE_PORT_INDICATOR 22
- #define USB_CLASS_CDC 2
- #define USB_CLASS_CDC_DATA 0x0A
- #define USB_CLASS_CDC_ACM 2
- #define USB_CLASS_CDC_HFN 0
- #define USB_CLASS_CDC_CMNGFN 1
- #define USB_CLASS_CDC_ACMFN 2
- #define USB_CLASS_CDC_UNIONFN 6
- #define USB_CLASS_HID 3
- #define USB_CLASS_HID_KEYBOARD 1
- #define USB_CLASS_HID_MOUSE 2
- #define USB_CLASS_HUB 9
- #define USB_CLASS_MSD 8
- #define USB_CLASS_MSD_BOT_TRANSPORT 0x50
- #define USB_CLASS_MSD_SCSI_CMDSET 6
- #define USB_CLASS_MSD_CSW_CMDPASSED 0
- #define USB_CLASS_MSD_CSW_CMDFAILED 1
- #define USB_CLASS_MSD_CSW_PHASEERROR 2
- #define PORT_FULL_SPEED 1
- #define PORT_LOW_SPEED 2
- #define nibble2Ascii(n) ((n) + (((n) < 10) ? '0' : 'A' - 10));
- #define STATIC_CONST_STRING_DESC(_name, ...)
- #define STATIC_CONST_STRING_DESC_LANGID(_name, x, y)
- #define UBUF(x, y) EFM32_ALIGN(4) uint8_t x[((y) + 3) & ~3]
- #define STATIC_UBUF(x, y) EFM32_ALIGN(4) static uint8_t x[((y) + 3) & ~3]

Typedefs

- typedef int(* USB_XferCompleteCb_TypeDef) (USB_Status_TypeDef status, uint32_t xferred, uint32_t remaining)

  USB transfer callback function.
- typedef void(* USBTIMER_Callback_TypeDef) (void)

  USBTIMER callback function.

Enumerations

- enum USB_Status_TypeDef {
  USB_STATUS_OK = 0,
  USB_STATUS_REQ_ERR = -1,
  USB_STATUS_EP_BUSY = -2,
  USB_STATUS_REQ_UNHANDLED = -3,
  USB_STATUS_ILLEGAL = -4,
  USB_STATUS_EP_STALLED = -5,
  USB_STATUS_EP_ABORTED = -6,
  USB_STATUS_EP_ERROR = -7,
  USB_STATUS_EP_NAK = -8,
  USB_STATUS_DEVICE_UNCONFIGURED = -9,
  USB_STATUS_DEVICE_SUSPENDED = -10,
  USB_STATUS_DEVICE_RESET = -11,
  USB_STATUS_TIMEOUT = -12,
  USB_STATUS_DEVICE_REMOVED = -13,
  USB_STATUS_HC_BUSY = -14,
  USB_STATUS_DEVICE_MALFUNCTION = -15,
  USB_STATUS_PORT_OVERCURRENT = -16 }

  USB transfer status enumerator.

Functions

- void USBTIMER_DelayMs (uint32_t msec)
- void USBTIMER_DelayUs (uint32_t usec)
- void USBTIMER_Init (void)
- void USBTIMER_Start (uint32_t id, uint32_t timeout, USBTIMER_Callback_TypeDef callback)
- void USBTIMER_Stop (uint32_t id)

6.33.1 Detailed Description

See em_usb.h for source code.

6.33.2 Macro Definition Documentation

6.33.2.1 #define CLEAR_FEATURE 1

Standard setup request CLEAR_FEATURE.

6.33.2.2 #define CONFIG_DESC_BM_REMOTEWAKEUP 0x20

Configuration descriptor attribute macro.

6.33.2.3 #define CONFIG_DESC_BM_RESERVED_D7 0x80

Configuration descriptor attribute macro.

6.33.2.4 #define CONFIG_DESC_BM_SELFPOWERED 0x40

Configuration descriptor attribute macro.

6.33.2.5 #define CONFIG_DESC_BM_TRANSFERTYPE 0x03

Configuration descriptor transfer type bitmask.

Referenced by USBD_Init().

6.33.2.6 #define CONFIG_DESC_MAXPOWER_mA(*x*) (((x) + 1) / 2)

Configuration descriptor power macro.

6.33.2.7 #define DEVICE_IS_SELFPOWERED 0x0001

Standard request GET_STATUS bitmask.

6.33.2.8 #define GET_CONFIGURATION 8

Standard setup request GET_CONFIGURATION.

6.33.2.9 #define GET_DESCRIPTOR 6

Standard setup request GET_DESCRIPTOR.

6.33.2.10 #define GET_INTERFACE 10

Standard setup request GET_INTERFACE.

6.33.2.11 #define GET_STATUS 0

Standard setup request GET_STATUS.

6.33.2.12 #define HUB_FEATURE_C_PORT_CONNECTION 16

HUB class request CLEAR/SET_PORT_FEATURE feature selector.

6.33.2.13 #define HUB_FEATURE_C_PORT_RESET 20

HUB class request CLEAR/SET_PORT_FEATURE feature selector.

6.33.2.14 #define HUB_FEATURE_PORT_INDICATOR 22

HUB class request CLEAR/SET_PORT_FEATURE feature selector.

6.33.2.15 #define HUB_FEATURE_PORT_POWER 8

HUB class request CLEAR/SET_PORT_FEATURE feature selector.

6.33.2.16 #define HUB_FEATURE_PORT_RESET 4

HUB class request CLEAR/SET_PORT_FEATURE feature selector.

6.33.2.17 #define nibble2Ascii(*n*) ((n) + (((n) < 10) ? '0' : 'A' - 10));

6.33.2.18 #define PORT_FULL_SPEED 1

Full speed return value for USBH_GetPortSpeed().

6.33.2.19 #define PORT_LOW_SPEED 2

Low speed return value for USBH_GetPortSpeed().

6.33.2.20 #define REMOTE_WAKEUP_ENABLED 0x0002

Standard request GET_STATUS bitmask.

6.33.2.21 #define SET_ADDRESS 5

Standard setup request SET_ADDRESS.

6.33.2.22 #define SET_CONFIGURATION 9

Standard setup request SET_CONFIGURATION.

6.33.2.23 #define SET_DESCRIPTOR 7

Standard setup request SET_DESCRIPTOR.

6.33.2.24 #define SET_FEATURE 3

Standard setup request SET_FEATURE.

6.33.2.25 #define SET_INTERFACE 11

Standard setup request SET_INTERFACE.

6.33.2.26 #define STATIC_CONST_STRING_DESC(*_name, ...*)

Value:

```
EFM32_PACK_START(1)                                             \
  typedef struct                                                \
  {                                                             \
    uint8_t len;                                                \
    uint8_t type;                                               \
    char16_t name[1 + sizeof((char16_t[]){ __VA_ARGS__ }) / 2]; \
  } __attribute__ ((packed)) _##_name;                          \
  EFM32_PACK_END()                                              \
  EFM32_ALIGN(4)                                                \
  EFM32_PACK_START(1)                                           \
  static const _##_name _name =                                 \
  {                                                             \
    .len  = sizeof(_##_name) - 2,                               \
    .type = USB_STRING_DESCRIPTOR,                              \
    .name = { __VA_ARGS__ },                                    \
    .name[((sizeof(_##_name) - 2) / 2) - 1] = '\0'              \
  }                                                             \
  EFM32_PACK_END()
```

6.33.2.27 #define STATIC_CONST_STRING_DESC_LANGID(*_name, x, y*)

Value:

```
EFM32_PACK_START(1)                                             \
  typedef struct                                                \
  {                                                             \
    uint8_t len;                                                \
    uint8_t type;                                               \
    uint8_t name[2];                                            \
  } _##_name;                                                   \
  EFM32_PACK_END()                                              \
  EFM32_ALIGN(4)                                                \
  EFM32_PACK_START(1)                                           \
  static const _##_name _name __attribute__ ((aligned(4))) =    \
  {                                                             \
    .len = 4,                                                   \
    .type = USB_STRING_DESCRIPTOR,                              \
    .name = { y, x }                                            \
  }                                                             \
  EFM32_PACK_END()
```

Macro for creating USB compliant language string descriptors.
Example: STATIC_CONST_STRING_DESC_LANGID(langID, 0x04, 0x09);

6.33.2.28 #define STATIC_UBUF(*x, y*) EFM32_ALIGN(4) static uint8_t x[((y) + 3) & ~3]

6.33.2.29 #define SYNCH_FRAME 12

Standard setup request SYNCH_FRAME.

6.33.2.30 #define UBUF(*x, y*) EFM32_ALIGN(4) uint8_t x[((y) + 3) & ~3]

Macro for creating WORD (4 byte) aligned uint8_t array with size which is a multiple of WORD size.
Example:
UBUF(rxBuffer, 37); => uint8_t rxBuffer[40];

6.33.2.31 #define USB_CDC_ACM_FND_DESCSIZE 4

CDC Abstract Control Management functional descriptor size.

6.33.2.32 #define USB_CDC_CALLMNG_FND_DESCSIZE 5

CDC Call Management functional descriptor size.

6.33.2.33 #define USB_CDC_GETLINECODING 0x21

CDC class setup request GET_LINE_CODING.

6.33.2.34 #define USB_CDC_HEADER_FND_DESCSIZE 5

CDC Header functional descriptor size.

6.33.2.35 #define USB_CDC_SETCTRLLINESTATE 0x22

CDC class setup request SET_CONTROL_LINE_STATE.

6.33.2.36 #define USB_CDC_SETLINECODING 0x20

CDC class setup request SET_LINE_CODING.

6.33.2.37 #define USB_CLASS_CDC 2

CDC device/interface class code.

6.33.2.38 #define USB_CLASS_CDC_ACM 2

CDC Abstract Control Model interface subclass code.

6.33.2.39 #define USB_CLASS_CDC_ACMFN 2

CDC class Abstract Control Management Functional Descriptor subtype.

6.33.2.40 #define USB_CLASS_CDC_CMNGFN 1

CDC class Call Management Functional Descriptor subtype.

6.33.2.41 #define USB_CLASS_CDC_DATA 0x0A

CDC Data interface class code.

6.33.2.42 #define USB_CLASS_CDC_HFN 0

CDC class Header Functional Descriptor subtype.

6.33.2.43 #define USB_CLASS_CDC_UNIONFN 6

CDC class Union Functional Descriptor subtype.

6.33.2.44 #define USB_CLASS_HID 3

HID device/interface class code.

6.33.2.45 #define USB_CLASS_HID_KEYBOARD 1

HID keyboard interface protocol code.

6.33.2.46 #define USB_CLASS_HID_MOUSE 2

HID mouse interface protocol code.

6.33.2.47 #define USB_CLASS_HUB 9

HUB device/interface class code.

6.33.2.48 #define USB_CLASS_MSD 8

MSD device/interface class code.

6.33.2.49 #define USB_CLASS_MSD_BOT_TRANSPORT 0x50

MSD Bulk Only Transport protocol.

6.33.2.50 #define USB_CLASS_MSD_CSW_CMDFAILED 1

MSD BOT Command status wrapper command failed code.

6.33.2.51 #define USB_CLASS_MSD_CSW_CMDPASSED 0

MSD BOT Command status wrapper command passed code.

6.33.2.52 #define USB_CLASS_MSD_CSW_PHASEERROR 2

MSD BOT Command status wrapper cmd phase error code.

6.33.2.53 #define USB_CLASS_MSD_SCSI_CMDSET 6

MSD Subclass SCSI transparent command set.

6.33.2.54 #define USB_CONFIG_DESCRIPTOR 2

CONFIGURATION descriptor value.

6.33.2.55 #define USB_CONFIG_DESCSIZE 9

Configuration descriptor size.

6.33.2.56 #define USB_CS_INTERFACE_DESCRIPTOR 0x24

Audio Class-specific Descriptor Type.

6.33.2.57 #define USB_DEVICE_DESCRIPTOR 1

DEVICE descriptor value.

6.33.2.58 #define USB_DEVICE_DESCSIZE 18

Device descriptor size.

6.33.2.59 #define USB_DEVICE_QUALIFIER_DESCRIPTOR 6

DEVICE_QUALIFIER descriptor value.

6.33.2.60 #define USB_DEVICE_QUALIFIER_DESCSIZE 10

Device qualifier descriptor size.

6.33.2.61 #define USB_ENDPOINT_DESCRIPTOR 5

ENDPOINT descriptor value.

Referenced by USBD_Init().

6.33.2.62 #define USB_ENDPOINT_DESCSIZE 7

Endpoint descriptor size.

6.33.2.63 #define USB_EP0_SIZE 8

The size of endpoint 0.

Referenced by USBD_Init().

6.33.2.64 #define USB_EP1_SIZE 8

The size of endpoint 1.

6.33.2.65 #define USB_EP2_SIZE 8

The size of endpoint 2.

6.33.2.66 #define USB_EP3_SIZE 64

The size of endpoint 3.

6.33.2.67 #define USB_EP4_SIZE 32

The size of endpoint 4.

6.33.2.68 #define USB_EP5_SIZE 64

The size of endpoint 5.

6.33.2.69 #define USB_EP6_SIZE 512

The size of endpoint 6.

6.33.2.70 #define USB_EP_DIR_IN 0x80

Endpoint direction mask.

6.33.2.71 #define USB_EPNUM_MASK 0x0F

Endpoint number mask.

Referenced by USBD_Init().

6.33.2.72 #define USB_EPTYPE_BULK 2

Endpoint type bulk.

6.33.2.73 #define USB_EPTYPE_CTRL 0

Endpoint type control.

Referenced by USBD_Init().

6.33.2.74 #define USB_EPTYPE_INTR 3

Endpoint type interrupt.

6.33.2.75 #define USB_EPTYPE_ISOC 1

Endpoint type isochron.

6.33.2.76 #define USB_FEATURE_DEVICE_REMOTE_WAKEUP 1

Standard request CLEAR/SET_FEATURE bitmask.

6.33.2.77 #define USB_FEATURE_ENDPOINT_HALT 0

Standard request CLEAR/SET_FEATURE bitmask.

6.33.2.78 #define USB_HID_DESCRIPTOR 0x21

HID descriptor value.

6.33.2.79 #define USB_HID_DESCSIZE 9

HID descriptor size.

6.33.2.80 #define USB_HID_GET_IDLE 0x02

HID class setup request GET_IDLE.

6.33.2.81 #define USB_HID_GET_REPORT 0x01

HID class setup request GET_REPORT.

6.33.2.82 #define USB_HID_REPORT_DESCRIPTOR 0x22

HID REPORT descriptor value.

6.33.2.83 #define USB_HID_SET_IDLE 0x0A

HID class setup request SET_IDLE.

6.33.2.84 #define USB_HID_SET_PROTOCOL 0x0B

HID class setup request SET_PROTOCOL.

6.33.2.85 #define USB_HID_SET_REPORT 0x09

HID class setup request SET_REPORT.

6.33.2.86 #define USB_HUB_DESCRIPTOR 0x29

HUB descriptor value.

6.33.2.87 #define USB_INTERFACE_DESCRIPTOR 4

INTERFACE descriptor value.

6.33.2.88 #define USB_INTERFACE_DESCSIZE 9

Interface descriptor size.

6.33.2.89 #define USB_INTERFACE_POWER_DESCRIPTOR 8

INTERFACE_POWER descriptor value.

6.33.2.90 #define USB_LANGID_ENUS 0x0409

English-United States language id.

6.33.2.91 #define USB_MAX_DEVICE_ADDRESS 127

Maximum allowable device address.

6.33.2.92 #define USB_MAX_EP_SIZE 64

The max size of any full speed endpoint.

6.33.2.93 #define USB_MSD_BOTRESET 0xFF

MSD class setup request Bulk only transfer reset.

6.33.2.94 #define USB_MSD_GETMAXLUN 0xFE

MSD class setup request Get Max LUN.

6.33.2.95 #define USB_OTHER_SPEED_CONFIG_DESCRIPTOR 7

OTHER_SPEED_CONFIGURATION descriptor value.

6.33.2.96 #define USB_OTHER_SPEED_CONFIG_DESCSIZE 9

Device other speed configuration descriptor size.

6.33.2.97 #define USB_SETUP_DIR_D2H 0x80

Setup request data stage IN direction mask.

6.33.2.98 #define USB_SETUP_DIR_H2D 0x00

Setup request data stage OUT direction mask.

6.33.2.99 #define USB_SETUP_DIR_IN 1

Setup request data stage IN direction value.

6.33.2.100 #define USB_SETUP_DIR_MASK 0x80

Setup request data stage direction mask.

Referenced by USBD_Init().

6.33.2.101 #define USB_SETUP_DIR_OUT 0

Setup request data stage OUT direction value.

6.33.2.102 #define USB_SETUP_PKT_SIZE 8

Setup request packet size.

6.33.2.103 #define USB_SETUP_RECIPIENT_DEVICE 0

Setup request device recipient value.

6.33.2.104 #define USB_SETUP_RECIPIENT_ENDPOINT 2

Setup request endpoint recipient value.

6.33.2.105 #define USB_SETUP_RECIPIENT_INTERFACE 1

Setup request interface recipient value.

6.33.2.106 #define USB_SETUP_RECIPIENT_OTHER 3

Setup request other recipient value.

6.33.2.107 #define USB_SETUP_TYPE_CLASS 1

Class setup request value.

6.33.2.108 #define USB_SETUP_TYPE_CLASS_MASK 0x20

Class setup request mask.

6.33.2.109 #define USB_SETUP_TYPE_STANDARD 0

Standard setup request value.

6.33.2.110 #define USB_SETUP_TYPE_STANDARD_MASK 0x00

Standard setup request mask.

6.33.2.111 #define USB_SETUP_TYPE_VENDOR 2

Vendor setup request value.

6.33.2.112 #define USB_SETUP_TYPE_VENDOR_MASK 0x40

Vendor setup request mask.

6.33.2.113 #define USB_STRING_DESCRIPTOR 3

STRING descriptor value.

6.33.3 Typedef Documentation

6.33.3.1 typedef int(* USB_XferCompleteCb_TypeDef) (USB_Status_TypeDef status, uint32_t xferred, uint32_t remaining)

The callback function is called when a transfer has completed. An application should check the status, xferred and optionally the remaining parameters before deciding if the transfer is usable. In the case where the transfer is part of a control request data stage, the callback function should return an appropriate USB_Status_TypeDef status.

Parameters

in	*status*	The transfer status. See USB_Status_TypeDef.
in	*xferred*	Number of bytes actually transferred.
in	*remaining*	Number of bytes not transferred.

Returns

USB_STATUS_OK on success, else an appropriate error code.

6.33.3.2 typedef void(* USBTIMER_Callback_TypeDef) (void)

The callback function is called when an USBTIMER has expired. The callback is done with interrupts disabled.

6.33.4 Enumeration Type Documentation

6.33.4.1 enum USB_Status_TypeDef

Enumerator

USB_STATUS_OK No errors detected.
USB_STATUS_REQ_ERR Setup request error.
USB_STATUS_EP_BUSY Endpoint is busy.
USB_STATUS_REQ_UNHANDLED Setup request not handled.
USB_STATUS_ILLEGAL Illegal operation attempted.
USB_STATUS_EP_STALLED Endpoint is stalled.
USB_STATUS_EP_ABORTED Endpoint transfer was aborted.
USB_STATUS_EP_ERROR Endpoint transfer error.
USB_STATUS_EP_NAK Endpoint NAK'ed transfer request.
USB_STATUS_DEVICE_UNCONFIGURED Device is unconfigured.
USB_STATUS_DEVICE_SUSPENDED Device is suspended.
USB_STATUS_DEVICE_RESET Device is/was reset.
USB_STATUS_TIMEOUT Transfer timeout.
USB_STATUS_DEVICE_REMOVED Device was removed.
USB_STATUS_HC_BUSY Host channel is busy.
USB_STATUS_DEVICE_MALFUNCTION Malfunctioning device attached.
USB_STATUS_PORT_OVERCURRENT VBUS shortcircuit/overcurrent failure.

6.33.5 Function Documentation

6.33.5.1 void USBTIMER_DelayMs (uint32_t *msec*)

6.33.5.2 void USBTIMER_DelayUs (uint32_t *usec*)

6.33.5.3 void USBTIMER_Init (void)

Referenced by USBD_Init().

6.33.5.4 void USBTIMER_Start (uint32_t *id,* uint32_t *timeout,* USBTIMER_Callback_TypeDef *callback*)

6.33.5.5 void USBTIMER_Stop (uint32_t *id*)

6.34 USB Device API

USB DEVICE protocol stack, see USB Device Stack Library page for detailed documentation.

.

Data Structures

- struct USBD_Init_TypeDef

  USB Device stack initialization structure.
- struct USBD_Callbacks_TypeDef

  USB Device stack callback structure.

Typedefs

- typedef void(∗ USBD_UsbResetCb_TypeDef) (void)

  USB Reset callback function.
- typedef void(∗ USBD_SofIntCb_TypeDef) (uint16_t sofNr)

  USB Start Of Frame (SOF) interrupt callback function.
- typedef void(∗ USBD_DeviceStateChangeCb_TypeDef) (USBD_State_TypeDef oldState, USBD_State_TypeDef newState)

  USB State change callback function.
- typedef bool(∗ USBD_IsSelfPoweredCb_TypeDef) (void)

  USB power mode callback function.
- typedef int(∗ USBD_SetupCmdCb_TypeDef) (const USB_Setup_TypeDef ∗setup)

  USB setup request callback function.
- typedef struct USBD_Callbacks_TypeDef USBD_Callbacks_TypeDef

  USB Device stack callback structure.

Enumerations

- enum USBD_State_TypeDef {
  USBD_STATE_NONE = 0,
  USBD_STATE_ATTACHED = 1,
  USBD_STATE_POWERED = 2,
  USBD_STATE_DEFAULT = 3,
  USBD_STATE_ADDRESSED = 4,
  USBD_STATE_CONFIGURED = 4,
  USBD_STATE_SUSPENDED = 6,
  USBD_STATE_LASTMARKER = 7 }

  USB device state enumerator.

Functions

- void USBD_AbortAllTransfers (void)

  Abort all pending transfers.
- int USBD_AbortTransfer (int epAddr)

  Abort a pending transfer on a specific endpoint.
- void USBD_Connect (void)

  Start USB device operation.
- void USBD_Disconnect (void)

  Stop USB device operation.
- bool USBD_EpIsBusy (int epAddr)

  Check if an endpoint is busy doing a transfer.
- USBD_State_TypeDef USBD_GetUsbState (void)

  Get current USB device state.
- const char * USBD_GetUsbStateName (USBD_State_TypeDef state)

  Get a string naming a device USB state.
- int USBD_Init (const USBD_Init_TypeDef *p)

  Initializes USB device hardware and internal protocol stack data structures, then connects the data-line (D+ or D-) pullup resistor to signal host that enumeration can begin.
- int USBD_Read (int epAddr, void *data, int byteCount, USB_XferCompleteCb_TypeDef callback)

  Start a read (OUT) transfer on an endpoint.
- int USBD_RemoteWakeup (void)

  Perform a remote wakeup signalling sequence.
- bool USBD_SafeToEnterEM2 (void)
- int USBD_StallEp (int epAddr)

  Set an endpoint in the stalled (halted) state.
- void USBD_Stop (void)

  Stop USB device stack operation.
- int USBD_UnStallEp (int epAddr)

  Reset stall state on a stalled (halted) endpoint.
- int USBD_Write (int epAddr, void *data, int byteCount, USB_XferCompleteCb_TypeDef callback)

  Start a write (IN) transfer on an endpoint.
- void usbSuspendDsr (void)

  USB suspend delayed service routine.

6.34.1 Detailed Description

See em_usbd.c for source code.

6.34.2 Typedef Documentation

6.34.2.1 typedef struct USBD_Callbacks_TypeDef USBD_Callbacks_TypeDef

Callback functions used by the device stack to signal events or query status to/from the application. See USBD_Init_TypeDef. Assign members to NULL if your application don't need a specific callback.

6.34.2.2 typedef void(* USBD_DeviceStateChangeCb_TypeDef) (USBD_State_TypeDef oldState, USBD_State_TypeDef newState)

Called whenever the device change state.

Parameters

in	*oldState*	The device USB state just leaved. See USBD_State_TypeDef.
in	*newState*	New (the current) USB device state. See USBD_State_TypeDef.

6.34.2.3 typedef bool(∗ USBD_IsSelfPoweredCb_TypeDef) (void)

Called whenever the device stack needs to query if the device is currently self- or bus-powered. Typically when host has issued an GET_STATUS setup command.

Returns

True if self-powered, false otherwise.

6.34.2.4 typedef int(∗ USBD_SetupCmdCb_TypeDef) (const USB_Setup_TypeDef ∗setup)

Called on each setup request received from host. This gives the application a possibility to extend or override standard requests, and to implement class or vendor specific requests. Return USB_STATUS_OK if the request is handled, return USB_STATUS_REQ_ERR if it is an illegal request or return USB_STATUS_REQ_UNHANDLED to pass the request on to the default request handler.

Parameters

in	*setup*	Pointer to an USB setup packet. See USB_Setup_TypeDef.

Returns

An appropriate status/error code. See USB_Status_TypeDef.

6.34.2.5 typedef void(∗ USBD_SofIntCb_TypeDef) (uint16_t sofNr)

Called at each SOF interrupt (if enabled),

Parameters

in	*sofNr*	Current frame number. The value rolls over to 0 after 16383 (0x3FFF).

6.34.2.6 typedef void(∗ USBD_UsbResetCb_TypeDef) (void)

Called whenever USB reset signalling is detected on the USB port.

6.34.3 Enumeration Type Documentation

6.34.3.1 enum USBD_State_TypeDef

Enumerator

USBD_STATE_NONE Device state is undefined/unknown.

USBD_STATE_ATTACHED Device state is ATTACHED.

USBD_STATE_POWERED Device state is POWERED.

USBD_STATE_DEFAULT Device state is DEFAULT.

USBD_STATE_ADDRESSED Device state is ADDRESSED.

USBD_STATE_CONFIGURED Device state is CONFIGURED.

USBD_STATE_SUSPENDED Device state is SUSPENDED.

USBD_STATE_LASTMARKER Device state enum end marker.

6.34.4 Function Documentation

6.34.4.1 void USBD_AbortAllTransfers (void)

Aborts transfers for all endpoints currently in use. Pending transfers on the default endpoint (EP0) are not aborted.

References USB_STATUS_EP_ABORTED.

6.34.4.2 int USBD_AbortTransfer (int *epAddr*)

Parameters

in	*epAddr*	The address of the endpoint to abort.

References assert, NULL, USB_STATUS_EP_ABORTED, USB_STATUS_OK, USBD_STATE_ADDRESSED, and USBD_STATE_CONFIGURED.

6.34.4.3 void USBD_Connect (void)

Device operation is started by connecting a pullup resistor on the appropriate USB data line.

6.34.4.4 void USBD_Disconnect (void)

Device operation is stopped by disconnecting the pullup resistor from the appropriate USB data line. Often referred to as a "soft" disconnect.

References USBD_STATE_SUSPENDED.

Referenced by USBD_Stop().

6.34.4.5 bool USBD_EpIsBusy (int *epAddr*)

Parameters

in	*epAddr*	The address of the endpoint to check.

Returns

True if endpoint is busy, false otherwise.

References assert, and NULL.

Referenced by USBD_RemoteWakeup().

6.34.4.6 USBD_State_TypeDef USBD_GetUsbState (void)

Returns

Device USB state. See USBD_State_TypeDef.

Referenced by USBD_Read(), and USBD_Write().

6.34.4.7 const char * USBD_GetUsbStateName (USBD_State_TypeDef *state*)

Parameters

in	*state*	Device USB state. See USBD_State_TypeDef.

Returns

State name string pointer.

References USBD_STATE_LASTMARKER.

6.34.4.8 int USBD_Init (const USBD_Init_TypeDef * *p*)

Note

You may later use USBD_Disconnect() and USBD_Connect() to force reenumeration.

Parameters

in	*p*	Pointer to device initialization struct. See USBD_Init_TypeDef.

Returns

USB_STATUS_OK on success, else an appropriate error code.

References assert, USB_EndpointDescriptor_TypeDef::bEndpointAddress, USB_EndpointDescriptor_TypeDef::bmAttributes, USBD_Init_TypeDef::bufferingMultiplier, USBD_Init_TypeDef::callbacks, CONFIG_DESC_BM_TRANSFERTYPE, USBD_Init_TypeDef::configDescriptor, USBD_Init_TypeDef::deviceDescriptor, MEMSET, NULL,

USBD_Init_TypeDef::numberOfStrings, USBD_Init_TypeDef::stringDescriptors, USB_ENDPOINT_DESCRIPTOR, USB_EP0_SIZE, USB_EPNUM_MASK, USB_EPTYPE_CTRL, USB_REMOTEWKUPEN_STATE, USB_SETUP_DIR_MASK, USB_STATUS_ILLEGAL, USB_STATUS_OK, USBD_STATE_LASTMARKER, USBD_STATE_NONE, USBTIMER_Init(), and USB_EndpointDescriptor_TypeDef::wMaxPacketSize.

6.34.4.9 int USBD_Read (int *epAddr*, void * *data*, int *byteCount*, USB_XferCompleteCb_TypeDef *callback*)

Note

The transfer buffer length must be a multiple of 4 bytes in length and WORD (4 byte) aligned. When allocating the buffer, round buffer length up. If it is possible that the host will send more data than your device expects, round buffer size up to the next multiple of maxpacket size.

Parameters

in	*epAddr*	Endpoint address.
in	*data*	Pointer to transfer data buffer.
in	*byteCount*	Transfer length.
in	*callback*	Function to be called on transfer completion. Supply NULL if no callback is needed. See USB_XferCompleteCb_TypeDef.

Returns

USB_STATUS_OK on success, else an appropriate error code.

References assert, NULL, USB_STATUS_DEVICE_UNCONFIGURED, USB_STATUS_EP_BUSY, USB_STATUS_EP_STALLED, USB_STATUS_OK, USBD_GetUsbState(), and USBD_STATE_CONFIGURED.

6.34.4.10 int USBD_RemoteWakeup (void)

Note

It is the responsibility of the application to ensure that remote wakeup is not attempted before the device has been suspended for at least 5 miliseconds. This function should not be called from within an interrupt handler.

Returns

USB_STATUS_OK on success, else an appropriate error code.

References elapsedTimeInt16u, FIFO_DEQUEUE, halCommonGetInt16uMillisecondTick(), MEMSET, NULL, USB_STATUS_ILLEGAL, USB_STATUS_OK, USB_STATUS_TIMEOUT, USBD_EpIsBusy(), USBD_STATE_SUSPENDED, USBD_Write(), usbForceTxData(), and usbTxData().

6.34.4.11 bool USBD_SafeToEnterEM2 (void)

6.34.4.12 int USBD_StallEp (int *epAddr*)

Parameters

in	*epAddr*	The address of the endpoint to stall.

Returns

USB_STATUS_OK on success, else an appropriate error code.

References assert, NULL, USB_STATUS_ILLEGAL, and USB_STATUS_OK.

6.34.4.13 void USBD_Stop (void)

The data-line pullup resistor is turned off, USB interrupts are disabled, and finally the USB pins are disabled.

References USBD_Disconnect(), and USBD_STATE_NONE.

6.34.4.14 int USBD_UnStallEp (int *epAddr*)

Parameters

in	*epAddr*	The address of the endpoint to un-stall.

Returns

USB_STATUS_OK on success, else an appropriate error code.

References assert, NULL, USB_STATUS_ILLEGAL, and USB_STATUS_OK.

6.34.4.15 int USBD_Write (int *epAddr*, void * *data*, int *byteCount*, USB_XferCompleteCb_TypeDef *callback*)

Parameters

in	*epAddr*	Endpoint address.
in	*data*	Pointer to transfer data buffer. This buffer must be WORD (4 byte) aligned.
in	*byteCount*	Transfer length.
in	*callback*	Function to be called on transfer completion. Supply NULL if no callback is needed. See USB_XferCompleteCb_TypeDef.

Returns

USB_STATUS_OK on success, else an appropriate error code.

References assert, NULL, USB_STATUS_DEVICE_UNCONFIGURED, USB_STATUS_EP_BUSY, USB_STAT↩US_EP_STALLED, USB_STATUS_ILLEGAL, USB_STATUS_OK, USBD_GetUsbState(), and USBD_STATE_C↩ONFIGURED.

Referenced by USBD_RemoteWakeup().

6.34.4.16 void usbSuspendDsr (void)

This function keeps the device in a low power state in order to meet USB specification during USB suspend state.

References emberStackPowerDown(), halSleep(), SLEEPMODE_IDLE, and USBD_STATE_SUSPENDED.

6.35 System Timer Control

Functions that provide access to the system clock.

Macros

- #define halIdleForMilliseconds(duration) halCommonIdleForMilliseconds((duration))

Functions

- uint16_t halInternalStartSystemTimer (void)

  Initializes the system tick.
- uint16_t halCommonGetInt16uMillisecondTick (void)

  Returns the current system time in system ticks, as a 16-bit value.
- uint32_t halCommonGetInt32uMillisecondTick (void)

  Returns the current system time in system ticks, as a 32-bit value.
- uint16_t halCommonGetInt16uQuarterSecondTick (void)

  Returns the current system time in quarter second ticks, as a 16-bit value.
- EmberStatus halSleepForQuarterSeconds (uint32_t *duration)

  Uses the system timer to enter SLEEPMODE_WAKETIMER for approximately the specified amount of time (provided in quarter seconds).
- EmberStatus halSleepForMilliseconds (uint32_t *duration)

  Uses the system timer to enter SLEEPMODE_WAKETIMER for approximately the specified amount of time (provided in milliseconds). Note that since the system timer ticks at a rate of 1024Hz, a second is comprised of 1024 milliseconds in this function.
- EmberStatus halCommonIdleForMilliseconds (uint32_t *duration)

  Uses the system timer to enter SLEEPMODE_IDLE for approximately the specified amount of time (provided in milliseconds).

6.35.1 Detailed Description

A single system tick (as returned by halCommonGetInt16uMillisecondTick() and halCommonGetInt32u↩MillisecondTick()) is approximately 1 millisecond.

- When used with a 32.768kHz crystal, the system tick is 0.976 milliseconds.
- When used with a 3.6864MHz crystal, the system tick is 1.111 milliseconds.

A single quarter-second tick (as returned by halCommonGetInt16uQuarterSecondTick()) is approximately 0.25 seconds.

The values used by the time support functions will wrap after an interval. The length of the interval depends on the length of the tick and the number of bits in the value. However, there is no issue when comparing time deltas of less than half this interval with a subtraction, if all data types are the same.

See system-timer.h for source code.

6.35.2 Macro Definition Documentation

6.35.2.1 #define halIdleForMilliseconds(*duration*) halCommonIdleForMilliseconds((duration))

6.35.3 Function Documentation

6.35.3.1 uint16_t halCommonGetInt16uMillisecondTick (void)

Returns

The least significant 16 bits of the current system time, in system ticks.

Referenced by USBD_RemoteWakeup().

6.35.3.2 uint16_t halCommonGetInt16uQuarterSecondTick (void)

Returns

The least significant 16 bits of the current system time, in system ticks multiplied by 256.

6.35.3.3 uint32_t halCommonGetInt32uMillisecondTick (void)

Returns

The least significant 32 bits of the current system time, in system ticks.

6.35.3.4 EmberStatus halCommonIdleForMilliseconds (uint32_t * *duration*)

This function returns EMBER_SUCCESS and the duration parameter is decremented to 0 after idling for the specified amount of time. If an interrupt occurs that brings the chip out of idle, the function returns EMBER_SLEEP_INTERRUPTED and the duration parameter reports the amount of time remaining out of the original request.

Note

This routine always enables interrupts.

Parameters

duration	The amount of time, expressed in milliseconds, that the micro should be placed into SLEEPMODE_IDLE. When the function returns, this parameter provides the amount of time remaining out of the original idle time request (normally the return value will be 0).

Returns

An EmberStatus value indicating the success or failure of the command.

6.35.3.5 uint16_t halInternalStartSystemTimer (void)

Returns

Time to update the async registers after RTC is started (units of 100 microseconds).

6.35.3.6 EmberStatus halSleepForMilliseconds (uint32_t * *duration*)

This function returns EMBER_SUCCESS and the duration parameter is decremented to 0 after sleeping for the specified amount of time. If an interrupt occurs that brings the chip out of sleep, the function returns EMBER_SLEEP_INTERRUPTED and the duration parameter reports the amount of time remaining out of the original request.

Note

This routine always enables interrupts.
This function is not implemented on AVR-based platforms.
Sleep durations less than 3 milliseconds are not allowed on on EM2XX-based platforms. Any attempt to sleep for less than 3 milliseconds on EM2XX-based platforms will cause the function to immediately exit without sleeping and return EMBER_SLEEP_INTERRUPTED.
The maximum sleep time of the hardware is limited on EM2XX-based platforms to 32 seconds. Any sleep duration greater than this limit will wake up briefly (e.g. 16 microseconds) to reenable another sleep cycle. Due to this limitation, this function should not be used with durations within 3 milliseconds of a multiple 32 seconds. The short sleep cycle that results from such durations is not handled reliably by the system timer on EM2XX-based platforms. If a sleep duration within 3 milliseconds of a multiple of 32 seconds is desired, halSleepForQuarterSeconds should be used.

Parameters

duration	The amount of time, expressed in milliseconds (1024 milliseconds = 1 second), that the micro should be placed into SLEEPMODE_WAKETIMER. When the function returns, this parameter provides the amount of time remaining out of the original sleep time request (normally the return value will be 0).

Returns

An EmberStatus value indicating the success or failure of the command.

6.35.3.7 EmberStatus halSleepForQuarterSeconds (uint32_t * *duration*)

This function returns EMBER_SUCCESS and the duration parameter is decremented to 0 after sleeping for the specified amount of time. If an interrupt occurs that brings the chip out of sleep, the function returns EMBER_SLEEP_INTERRUPTED and the duration parameter reports the amount of time remaining out of the original request.

Note

This routine always enables interrupts.
The maximum sleep time of the hardware is limited on AVR-based platforms to 8 seconds, on EM2XX-based platforms to 64 seconds, and on EM35x platforms to 48.5 days. Any sleep duration greater than this limit will wake up briefly (e.g. 16 microseconds) to reenable another sleep cycle.

The EM2xx has a 16 bit sleep timer, which normally runs at 1024Hz. In order to support long sleep durations, the chip will periodically wake up to manage a larger timer in software. This periodic wakeup is normally triggered once every 32 seconds. However, this period can be extended to once every 2.275 hours by building with **ENABLE_LONG_SLEEP_CYCLES** defined. This definition enables the use of a prescaler when sleeping for more than 63 seconds at a time. However, this define also imposes the following limitations:

1. The chip may only wake up from the sleep timer. (External GPIO wake events may not be used)
2. Each time a sleep cycle is performed, a loss of accuracy up to +/-750ms will be observed in the system timer.

Parameters

duration	The amount of time, expressed in quarter seconds, that the micro should be placed into SLEEPMODE_WAKETIMER. When the function returns, this parameter provides the amount of time remaining out of the original sleep time request (normally the return value will be 0).

Returns

An EmberStatus value indicating the success or failure of the command.

6.36 Symbol Timer Control

Symbol Timer Functions

- void halInternalStartSymbolTimer (void)

  Initializes the symbol timer. When a dedicated symbol timer peripheral exists (e.g. EM2xx, EM3xx) this initialization is generally performed directly by the PHY, so this routine may be a no-op.
- uint32_t halStackGetInt32uSymbolTick (void)

  Returns the current symbol time in symbol ticks (units are platform-dependent, but typically on the order of microseconds).
- bool halStackInt32uSymbolTickGTorEqual (uint32_t st1, uint32_t st2)

  Returns true if symbol tick time st1 is greater than symbol tick time st2, as determined by half the range of the symbol timer. Can only account for 1 wrap around between st1 and st2 before it is wrong.
- uint32_t halStackGetSymbolTicksPerSecond (void)

  Obtains the number of symtol timer ticks in one second of real time. Can be used for conversion between real time and symbol ticks.

MAC Timer Support Functions

These functions are used for MAC layer timing and symbol-based delays.

Applications should not directly call these functions. They are used internally by the operation of the stack.

- enum EmHalSymbolDelayChannel_t {
  EM_HAL_SYMBOL_DELAY_CHANNEL_A,
  EM_HAL_SYMBOL_DELAY_CHANNEL_B,
  EM_HAL_SYMBOL_DELAY_CHANNELS }

  Specifies two independent channels for symbol delay operations.
- typedef void(∗ EmHalSymbolDelayCallback_t) (EmHalSymbolDelayChannel_t delayChan)

  Specifies the callback API triggered when the symbol timer channel expires.
- uint32_t halStackOrderSymbolDelay (EmHalSymbolDelayChannel_t delayChan, EmHalSymbolDelay↩Callback_t callback, uint32_t microseconds)

  Sets up a delay timer to call the indicated interrupt-context callback when it expires.
- void halStackCancelSymbolDelay (EmHalSymbolDelayChannel_t delayChan, EmHalSymbolDelay↩Callback_t callback)

  Cancels the delay set up by an earlier halStackOrderSymbolDelay() call.
- void halStackOrderInt16uSymbolDelayA (uint16_t symbols)

  Sets up a timer and calls an interrupt-context callback when it expires.
- void halStackCancelSymbolDelayA (void)

  Cancels the timer set up by halStackOrderInt16uSymbolDelayA().
- void halStackSymbolDelayAIsr (void)

  This is the interrupt level callback into the stack that is called when the timers set by halStackOrderInt16uSymbol↩DelayA expire.

6.36.1 Detailed Description

See symbol-timer.h for source code.

6.36.2 Typedef Documentation

6.36.2.1 typedef void(* EmHalSymbolDelayCallback_t) (EmHalSymbolDelayChannel_t delayChan)

6.36.3 Enumeration Type Documentation

6.36.3.1 enum EmHalSymbolDelayChannel_t

Enumerator

EM_HAL_SYMBOL_DELAY_CHANNEL_A

EM_HAL_SYMBOL_DELAY_CHANNEL_B

EM_HAL_SYMBOL_DELAY_CHANNELS

6.36.4 Function Documentation

6.36.4.1 void halInternalStartSymbolTimer (void)

6.36.4.2 void halStackCancelSymbolDelay (EmHalSymbolDelayChannel_t *delayChan,* EmHalSymbolDelayCallback_t *callback*)

Parameters

delayChan	The symbol timer delay channel specified in the earlier halStackOrderSymbolDelay() call.
callback	The callback specified in the earlier halStackOrderSymbolDelay() call.

Note

If cancelled prior to the delay expiring, the original callback will not be called.

6.36.4.3 void halStackCancelSymbolDelayA (void)

6.36.4.4 uint32_t halStackGetInt32uSymbolTick (void)

Returns

The least significant 32 bits of the current symbol time in symbol timer ticks.

6.36.4.5 uint32_t halStackGetSymbolTicksPerSecond (void)

Returns

How many symbol ticks occur in a second.

6.36.4.6 bool halStackInt32uSymbolTickGTorEqual (uint32_t *st1,* uint32_t *st2*)

6.36.4.7 void halStackOrderInt16uSymbolDelayA (uint16_t *symbols*)

Used by the MAC to request an interrupt callback at a specified amount of time in the future.

Parameters

symbols	The delay, in symbols.

6.36.4.8 uint32_t halStackOrderSymbolDelay (EmHalSymbolDelayChannel_t *delayChan,* EmHalSymbolDelayCallback_t *callback,* uint32_t *microseconds*)

Used by the MAC to request an interrupt callback at a specified number of symbol timer ticks in the future.

Parameters

delayChan	The symbol timer delay channel to use.
callback	The callback to call in interrupt context when the delay expires. If NULL, the timer is not actually started, but the return value is accurate; useful for polling.
microseconds	The delay, in units of microseconds.

Returns

The absolute symbol tick value of the delay (usually in the future).

Note

Internal conversion of microseconds to symbol ticks will use a rounding function.
No minimum delay is enforced; short delays may trigger the callback immediately, before this routine returns to its caller.

6.36.4.9 void halStackSymbolDelayAIsr (void)

6.37 HAL Configuration

Modules

- Sample Breakout Board Configuration

  Functions and definitions specific to the breakout board.
- IAR PLATFORM_HEADER Configuration

  Compiler and Platform specific definitions and typedefs for the IAR ARM C compiler.
- Common PLATFORM_HEADER Configuration

  Compiler and Platform specific definitions and typedefs common to all platforms.
- NVIC Configuration
- Reset Cause Type Definitions

6.37.1 Detailed Description

6.38 Sample Breakout Board Configuration

Functions and definitions specific to the breakout board.

Macros

- #define PWRUP_CFG_SC1_TXD _GPIO_P_CFGL_Px0_OUT_ALT

  Give GPIO SC1 TXD and nRTS configurations friendly names.
- #define PWRDN_OUT_SC1_nRTS 1

Custom Baud Rate Definitions

Application Framework NCP Configuration Board Header

This board header (dev0680) is not supported in framework NCP applications. NCP applications must use either the dev0680spi or dev0680uart board headers when creating custom NCP applications through the framework.

The following define is used with defining a custom baud rate for the UART. This define provides a simple hook into the definition of the baud rates used with the UART. The baudSettings[] array in uart.c links the BAUD_∗ defines with the actual register values needed for operating the UART. The array baudSettings[] can be edited directly for a custom baud rate or another entry (the register settings) can be provided here with this define.

- #define EMBER_SERIAL_BAUD_CUSTOM 13

  This define is the register setting for generating a baud of.

LED Definitions

The following are used to aid in the abstraction with the LED connections. The microcontroller-specific sources use these definitions so they are able to work across a variety of boards which could have different connections. The names and ports/pins used below are intended to match with a schematic of the system to provide the abstraction.

The HalBoardLedPins enum values should always be used when manipulating the state of LEDs, as they directly refer to the GPIOs to which the LEDs are connected.

Note: LEDs 0 and 1 are on the RCM.

Note: LED 2 is on the breakout board (dev0680).

Note: LED 3 simply redirects to LED 2.

- enum HalBoardLedPins {
  BOARDLED0 = PORTA_PIN(6),
  BOARDLED1 = PORTA_PIN(7),
  BOARDLED2 = PORTC_PIN(5),
  BOARDLED3 = BOARDLED2,
  BOARD_ACTIVITY_LED = BOARDLED0,
  BOARD_HEARTBEAT_LED = BOARDLED1 }

  Assign each GPIO with an LED connected to a convenient name. BOARD_ACTIVITY_LED and BOARD_HEART↩ BEAT_LED provide a further layer of abstraction on top of the 3 LEDs for verbose coding.

Button Definitions

The following are used to aid in the abstraction with the Button connections. The microcontroller-specific sources use these definitions so they are able to work across a variety of boards which could have different connections. The names and ports/pins used below are intended to match with a schematic of the system to provide the abstraction.

The BUTTONn macros should always be used with manipulating the buttons as they directly refer to the GPIOs to which the buttons are connected.

Note

The GPIO number must match the IRQ letter

- #define BUTTON0 PORTB_PIN(6)

  The actual GPIO BUTTON0 is connected to. This define should be used whenever referencing BUTTON0.
- #define BUTTON0_IN (GPIO->P[1].IN)

  The GPIO input register for BUTTON0.
- #define BUTTON0_SEL() do {} while (0)

  Point the proper IRQ at the desired pin for BUTTON0.
- #define BUTTON0_ISR halIrqBIsr

  The interrupt service routine for BUTTON0.
- #define BUTTON0_INTCFG (EVENT_GPIO->CFGB)

  The interrupt configuration register for BUTTON0.
- #define BUTTON0_INT_EN_IRQN IRQB_IRQn

  The interrupt enable bit for BUTTON0.
- #define BUTTON0_INT_EN_BIT BIT32(BUTTON0_INT_EN_IRQN)

  The actual GPIO BUTTON0 is connected to. This define should be used whenever referencing BUTTON0.
- #define BUTTON0_FLAG_BIT EVENT_GPIO_FLAG_IRQB

  The interrupt flag bit for BUTTON0.
- #define BUTTON0_MISS_BIT EVENT_MISS_MISS_IRQB

  The missed interrupt bit for BUTTON0.
- #define BUTTON1 PORTC_PIN(6)

  The actual GPIO BUTTON1 is connected to. This define should be used whenever referencing BUTTON1, such as controlling if pieces are compiled in. Remember there may be other things that might want to use IRQC.
- #define BUTTON1_IN (GPIO->P[2].IN)

  The GPIO input register for BUTTON1.
- #define BUTTON1_SEL() do { GPIO->IRQCSEL = PORTC_PIN(6); } while (0)

  Point the proper IRQ at the desired pin for BUTTON1. Remember there may be other things that might want to use IRQC.
- #define BUTTON1_ISR halIrqCIsr

  The interrupt service routine for BUTTON1. Remember there may be other things that might want to use IRQC.
- #define BUTTON1_INTCFG (EVENT_GPIO->CFGC)

  The interrupt configuration register for BUTTON1.
- #define BUTTON1_INT_EN_IRQN IRQC_IRQn

  The interrupt enable bit for BUTTON1.
- #define BUTTON1_INT_EN_BIT BIT32(BUTTON1_INT_EN_IRQN)

  The actual GPIO BUTTON0 is connected to. This define should be used whenever referencing BUTTON0.
- #define BUTTON1_FLAG_BIT EVENT_GPIO_FLAG_IRQC

  The interrupt flag bit for BUTTON1.
- #define BUTTON1_MISS_BIT EVENT_MISS_MISS_IRQC

  The missed interrupt bit for BUTTON1.

USB Power State

Define if the USB is self powered or bus powered since the configuration descriptor needs to report to the host the powered state.

Note

VBUS Monitoring is required for USB to function when the EM358 device is configured as self-powered.

- #define USB_SELFPWRD_STATE (1)

  The USB power state.

USB Remote Wakeup Enable

If the USB device needs to awake the host from suspend, then it needs to have remote wakeup enable.

Note

The host can deny remote wakeup, keeping the device in suspend.

If the device has remote wakeup enabled the configuration descriptor needs to report this fact to the host. Additionally, the USB core in the chip needs to be directly told. Set the define USB_REMOTEWKUPEN_STATE to 0 if remote wake is disabled or 1 if enabled.

- #define USB_REMOTEWKUPEN_STATE (1)

  USB Remote Wakeup Enable.

USB Maximum Power Consumption

The USB device must report the maximum power it will draw from the bus. This is done via the bMaxPower parameter in the Configuration Descriptor reported to the host. The value used is in units of 2mA.

Self-powered devices are low power devices and must draw less than 100mA.

Systems that have components such as a FEM are likely to consume more than 100mA and are considered high power and therefore must be bus-powered.

- #define USB_MAX_POWER (50)

  USB Max Power parameter (bMaxPower) the driver will report to the host in the Configuration Descriptor.

USB Enumeration Control

The following are used to aid in the abstraction of which GPIO is used for controlloing the pull-up resistor for enumeation.

The hardware setup connects the D+ signal to a GPIO via a 1.5kOhm pull-up resistor. Any GPIO can be used since it just needs to be a simple push-pull output configuration.

- #define ENUMCTRL PORTA_PIN(2)

  The actual GPIO ENUMCTRL is connected to. The GPIO only needs to be a simple push-pull output or input.
- #define ENUMCTRL_SETCFG(cfg)

  Set the GPIO's configuration to the provided state. The two states used are GPIOCFG_OUT when the device is enumerated and GPIOCFG_IN when the device is not enumerated.
- #define ENUMCTRL_SET() do { GPIO->P[0].SET = GPIO_P_SET_Px2; } while (0)

  When the GPIO used for enumeration is configured as push-pull, this macro makes it easy to set the output state high.
- #define ENUMCTRL_CLR() do { GPIO->P[0].CLR = GPIO_P_SET_Px2; } while (0)

  When the GPIO used for enumeration is configured as push-pull, this macro makes it easy to clear the output state low.

USB VBUS Monitoring Support

Note

VBUS Monitoring is required for USB to function when the EM358 device is configured as self-powered.

The following are used to aid in the abstraction of which GPIO and IRQ is used for VBUS Monitoring.

Remember that IRQA and IRQB are fixed to GPIO PB0 and PB6 respectively while IRQC and IRQD can be assigned to any GPIO. Since USB's D- and D+ data pins are fixed to PA0 and PA1 respectively, SC2 can't be used so it makes sense to allocate PA2 for enumeration control and PA3 for VBUS monitoring. Therefore, using PA3 for VBUS monitoring requires IRQC or IRQD.

The driver will only try to use VBUSMON functionality if USB_SELFPWRD_STATE is set to 1.

- #define VBUSMON GPIO_P_IN_Px3

  The actual GPIO VBUSMON is connected to. Remember that other pieces might want to use PA3.
- #define VBUSMON_IN (GPIO->P[0].IN)

  The GPIO input register for VBUSMON.
- #define VBUSMON_SETCFG()

  The GPIO configuration needed for VBUSMON. The configuration needs to be a simple input that will monitor for edge tansitions.
- #define VBUSMON_SEL() do { GPIO->IRQDSEL = PORTA_PIN(3); } while (0)

  Point the proper IRQ at the desired pin for VBUSMON. Remember that other pieces that might want to use IRQC.
- #define VBUSMON_ISR halIrqDIsr

  The interrupt service routine for VBUSMON. Remember that other pieces that might want to use IRQC.
- #define VBUSMON_INTCFG (EVENT_GPIO->CFGD)

  The interrupt configuration register for VBUSMON.
- #define VBUSMON_INT_EN_IRQN IRQD_IRQn

  The interrupt enable bit for VBUSMON.
- #define VBUSMON_INT_EN_BIT BIT32(VBUSMON_INT_EN_IRQN)

  The actual GPIO VBUSMON is connected to. Remember that other pieces might want to use PA3.
- #define VBUSMON_FLAG_BIT EVENT_GPIO_FLAG_IRQD

  The interrupt flag bit for VBUSMON.
- #define VBUSMON_MISS_BIT EVENT_MISS_MISS_IRQD

  The missed interrupt bit for VBUSMON.

Radio HoldOff Configuration Definitions

This define does not equate to anything. It is used as a trigger to enable Radio HoldOff support.

The following are used to aid in the abstraction with Radio HoldOff (RHO). The microcontroller-specific sources use these definitions so they are able to work across a variety of boards which could have different connections. The names and ports/pins used below are intended to match with a schematic of the system to provide the abstraction.

The Radio HoldOff input GPIO is abstracted like BUTTON0/1.

- #define RHO_ASSERTED 1

  The actual GPIO used to control Radio HoldOff.
- #define RHO_CFG (GPIO->P[0].CFGH)

  The GPIO configuration register for Radio HoldOff.
- #define RHO_IN (GPIO->P[0].IN)

  The GPIO input register for Radio HoldOff.
- #define RHO_OUT (GPIO->P[0].OUT)

  The GPIO output register for Radio HoldOff.
- #define RHO_SEL() do { GPIO->IRQDSEL = RHO_GPIO; } while (0)

  Point the proper IRQ at the desired pin for Radio HoldOff. Remember there may be other things that might want to use this IRQ.
- #define RHO_ISR halIrqDIsr

  The interrupt service routine for Radio HoldOff. Remember there may be other things that might want to use this IRQ.
- #define RHO_INTCFG (EVENT_GPIO->CFGD)

  The interrupt configuration register for Radio HoldOff.
- #define RHO_INT_EN_IRQN IRQD_IRQn

  The interrupt enable bit for Radio HoldOff.
- #define RHO_INT_EN_BIT BIT32(RHO_INT_EN_IRQN)

  The actual GPIO used to control Radio HoldOff.
- #define RHO_FLAG_BIT EVENT_GPIO_FLAG_IRQD

  The interrupt flag bit for Radio HoldOff.
- #define RHO_MISS_BIT EVENT_MISS_MISS_IRQD

  The missed interrupt bit for Radio HoldOff.
- #define PWRUP_CFG_DFL_RHO_FOR_RHO _GPIO_P_CFGL_Px0_IN_PUD

  Configuration of GPIO for Radio HoldOff operation.
- #define PWRUP_OUT_DFL_RHO_FOR_RHO 0 /* Deassert */

  The actual GPIO used to control Radio HoldOff.
- #define PWRDN_CFG_DFL_RHO_FOR_RHO _GPIO_P_CFGL_Px0_IN_PUD

  The actual GPIO used to control Radio HoldOff.
- #define PWRDN_OUT_DFL_RHO_FOR_RHO 0 /* Deassert */

  The actual GPIO used to control Radio HoldOff.
- #define PWRUP_CFG_DFL_RHO_FOR_DFL _GPIO_P_CFGL_Px0_OUT

  Configuration of GPIO for default behavior.
- #define PWRUP_OUT_DFL_RHO_FOR_DFL 1 /* LED default off */

  The actual GPIO used to control Radio HoldOff.
- #define PWRDN_CFG_DFL_RHO_FOR_DFL _GPIO_P_CFGL_Px0_OUT

  The actual GPIO used to control Radio HoldOff.
- #define PWRDN_OUT_DFL_RHO_FOR_DFL 1 /* LED off */

  The actual GPIO used to control Radio HoldOff.
- #define PWRUP_CFG_DFL_RHO PWRUP_CFG_DFL_RHO_FOR_DFL

  The following definitions are helpers for managing Radio HoldOff and should not be modified.

- #define PWRUP_OUT_DFL_RHO PWRUP_OUT_DFL_RHO_FOR_DFL

  The actual GPIO used to control Radio HoldOff.
- #define PWRDN_CFG_DFL_RHO PWRDN_CFG_DFL_RHO_FOR_DFL

  The actual GPIO used to control Radio HoldOff.
- #define PWRDN_OUT_DFL_RHO PWRDN_OUT_DFL_RHO_FOR_DFL

  The actual GPIO used to control Radio HoldOff.
- #define halInternalInitRadioHoldOff() /∗ no-op ∗/

  The actual GPIO used to control Radio HoldOff.

Temperature sensor ADC channel

Define the analog input channel connected to the LM-20 temperature sensor. The scale factor compensates for different platform input ranges. PB5/ADC0 must be an analog input. PC7 must be an output and set to a high level to power the sensor.

- #define TEMP_SENSOR_ADC_CHANNEL ADC_SOURCE_ADC0_VREF2

  The analog input channel to use for the temperature sensor.
- #define TEMP_SENSOR_SCALE_FACTOR 1

  The scale factor to compensate for different input ranges.

Packet Trace

When PACKET_TRACE is defined, ::GPIO_PACFGH will automatically be setup by halInit() to enable Packet Trace support on PA4 and PA5, in addition to the configuration specified below.

Note

This define will override any settings for PA4 and PA5.

- #define PACKET_TRACE

  This define does not equate to anything. It is used as a trigger to enable Packet Trace support on the breakout board (dev0680).

ENABLE_OSC32K

When ENABLE_OSC32K is defined, halInit() will configure system timekeeping to utilize the external 32.768 kHz crystal oscillator rather than the internal 1 kHz RC oscillator.

Note

ENABLE_OSC32K is mutually exclusive with ENABLE_ALT_FUNCTION_NTX_ACTIVE since they define conflicting usage of GPIO PC6.

On initial powerup the 32.768 kHz crystal oscillator will take a little while to start stable oscillation. This only happens on initial powerup, not on wake-from-sleep, since the crystal usually stays running in deep sleep mode.

When ENABLE_OSC32K is defined the crystal oscillator is started as part of halInit(). After the crystal is started we delay for OSC32K_STARTUP_DELAY_MS (time in milliseconds). This delay allows the crystal oscillator to stabilize before we start using it for system timing.

If you set OSC32K_STARTUP_DELAY_MS to less than the crystal's startup time:

- The system timer won't produce a reliable one millisecond tick before the crystal is stable.
- You may see some number of ticks of unknown period occur before the crystal is stable.
- halInit() will complete and application code will begin running, but any events based on the system timer will not be accurate until the crystal is stable.
- An unstable system timer will only affect the APIs in system-timer.h.

Typical 32.768 kHz crystals measured by Ember take about 400 milliseconds to stabilize. Be sure to characterize your particular crystal's stabilization time since crystal behavior can vary.

- #define OSC32K_STARTUP_DELAY_MS (0)

Packet Trace Configuration Defines

Provide the proper set of pin configuration for when the Packet Trace is enabled (look above for the define which enables it). When Packet Trace is not enabled, leave the two PTI pins in their default configuration. If Packet Trace is not being used, feel free to set the pin configurations as desired. The config shown here is simply the Power On Reset defaults.

- #define PWRUP_CFG_PTI_EN _GPIO_P_CFGL_Px0_OUT_ALT

  Give the packet trace configuration a friendly name.
- #define PWRUP_OUT_PTI_EN 0

  Give the packet trace configuration a friendly name.
- #define PWRDN_CFG_PTI_EN _GPIO_P_CFGL_Px0_IN_PUD

  Give the packet trace configuration a friendly name.
- #define PWRDN_OUT_PTI_EN 0

  Give the packet trace configuration a friendly name.
- #define PWRUP_CFG_PTI_DATA _GPIO_P_CFGL_Px0_OUT_ALT

  Give the packet trace configuration a friendly name.
- #define PWRUP_OUT_PTI_DATA 1

  Give the packet trace configuration a friendly name.
- #define PWRDN_CFG_PTI_DATA _GPIO_P_CFGL_Px0_IN_PUD

  Give the packet trace configuration a friendly name.
- #define PWRDN_OUT_PTI_DATA 1

  Give the packet trace configuration a friendly name.

32kHz Oscillator and nTX_ACTIVE Configuration Defines

Since the 32kHz Oscillator and nTX_ACTIVE both share PC6, their configuration defines are linked and instantiated together. Look above for the defines that enable the 32kHz Oscillator and nTX_ACTIVE.

Note

ENABLE_OSC32K is mutually exclusive with ENABLE_ALT_FUNCTION_NTX_ACTIVE since they define conflicting usage of GPIO PC6.

When using the 32kHz, configure PC6 and PC7 for analog for the XTAL.

When using nTX_ACTIVE, configure PC6 for alternate output while awake and a low output when deepsleeping. Also, configure PC7 for TEMP_EN.

When not using the 32kHz or nTX_ACTIVE, configure PC6 and PC7 for Button1 and TEMP_EN.

- #define PWRUP_CFG_BUTTON1 _GPIO_P_CFGL_Px0_IN_PUD

  Give GPIO PC6 configuration a friendly name.
- #define PWRUP_OUT_BUTTON1 1 /∗ Button needs a pullup ∗/

  Give GPIO PC6 configuration a friendly name.
- #define PWRDN_CFG_BUTTON1 _GPIO_P_CFGL_Px0_IN_PUD

  Give GPIO PC6 configuration a friendly name.
- #define PWRDN_OUT_BUTTON1 1 /∗ Button needs a pullup ∗/

  Give GPIO PC6 configuration a friendly name.
- #define CFG_TEMPEN _GPIO_P_CFGL_Px0_OUT

  Give GPIO PC7 configuration a friendly name.

TX_ACTIVE Configuration Defines

Provide the proper set of pin (PC5) configurations for when TX_ACTIVE is enabled (look above for the define which enables it). When TX_ACTIVE is not enabled, configure the pin for LED2.

- #define PWRUP_CFG_LED2 _GPIO_P_CFGL_Px0_OUT

  Give the TX_ACTIVE configuration a friendly name.
- #define PWRUP_OUT_LED2 1 /∗ LED default off ∗/

  Give the TX_ACTIVE configuration a friendly name.
- #define PWRDN_CFG_LED2 _GPIO_P_CFGL_Px0_OUT

  Give the TX_ACTIVE configuration a friendly name.
- #define PWRDN_OUT_LED2 1 /∗ LED default off ∗/

  Give the TX_ACTIVE configuration a friendly name.

USB Configuration Defines

Provide the proper set of pin configuration for when USB is not enumerated. Not enumerated primarily refers to the driver not being configured or deep sleep. The configuration used here is only for keeping the USB off the bus. The GPIO configuration used when active is controlled by the USB driver since the driver needs to control the enumeration process (which affects GPIO state.)

Note

: Using USB requires Serial port 3 to be defined and is only possible on EM3582/EM3586/EM3588/EM359 chips.

- #define PWRUP_CFG_USBDM _GPIO_P_CFGL_Px0_OUT_ALT

  Give the USB configuration a friendly name.
- #define PWRUP_OUT_USBDM 0

  Give the USB configuration a friendly name.
- #define PWRUP_CFG_USBDP _GPIO_P_CFGL_Px0_IN

  Give the USB configuration a friendly name.
- #define PWRUP_OUT_USBDP 0

  Give the USB configuration a friendly name.
- #define PWRUP_CFG_ENUMCTRL _GPIO_P_CFGL_Px0_OUT_ALT

  Give the USB configuration a friendly name.
- #define PWRUP_OUT_ENUMCTRL 0

  Give the USB configuration a friendly name.
- #define PWRUP_CFG_VBUSMON _GPIO_P_CFGL_Px0_OUT

  Give the USB configuration a friendly name.
- #define PWRUP_OUT_VBUSMON 1

  Give the USB configuration a friendly name.
- #define PWRDN_CFG_USBDM _GPIO_P_CFGL_Px0_IN_PUD

  Give the USB configuration a friendly name.
- #define PWRDN_OUT_USBDM 1

  Give the USB configuration a friendly name.
- #define PWRDN_CFG_USBDP _GPIO_P_CFGL_Px0_IN_PUD

  Give the USB configuration a friendly name.
- #define PWRDN_OUT_USBDP 1

  Give the USB configuration a friendly name.
- #define PWRDN_CFG_ENUMCTRL _GPIO_P_CFGL_Px0_IN_PUD

  Give the USB configuration a friendly name.
- #define PWRDN_OUT_ENUMCTRL 1

  Give the USB configuration a friendly name.
- #define PWRDN_CFG_VBUSMON _GPIO_P_CFGL_Px0_OUT

  Give the USB configuration a friendly name.
- #define PWRDN_OUT_VBUSMON 1

  Give the USB configuration a friendly name.

GPIO Configuration Macros

These macros define the GPIO configuration and initial state of the output registers for all the GPIO in the powerup and powerdown modes.

- uint16_t gpioCfgPowerUp [6]

  Define the mask for GPIO relevant to the radio in the context of power state. Each bit in the mask indicates the corresponding GPIO which should be affected when invoking halStackRadioPowerUpBoard() or halStackRadio↩ PowerDownBoard().
- uint16_t gpioCfgPowerDown [6]

  Define the mask for GPIO relevant to the radio in the context of power state. Each bit in the mask indicates the corresponding GPIO which should be affected when invoking halStackRadioPowerUpBoard() or halStackRadio↩ PowerDownBoard().
- uint8_t gpioOutPowerUp [3]

  Define the mask for GPIO relevant to the radio in the context of power state. Each bit in the mask indicates the corresponding GPIO which should be affected when invoking halStackRadioPowerUpBoard() or halStackRadio↩ PowerDownBoard().
- uint8_t gpioOutPowerDown [3]

  Define the mask for GPIO relevant to the radio in the context of power state. Each bit in the mask indicates the corresponding GPIO which should be affected when invoking halStackRadioPowerUpBoard() or halStackRadio↩ PowerDownBoard().
- GpioMaskType gpioRadioPowerBoardMask

  Define the mask for GPIO relevant to the radio in the context of power state. Each bit in the mask indicates the corresponding GPIO which should be affected when invoking halStackRadioPowerUpBoard() or halStackRadio↩ PowerDownBoard().
- #define DEFINE_GPIO_RADIO_POWER_BOARD_MASK_VARIABLE() GpioMaskType gpioRadioPower↩ BoardMask = 0

  Define the mask for GPIO relevant to the radio in the context of power state. Each bit in the mask indicates the corresponding GPIO which should be affected when invoking halStackRadioPowerUpBoard() or halStackRadio↩ PowerDownBoard().
- #define DEFINE_POWERUP_GPIO_CFG_VARIABLES()

  Initialize GPIO powerup configuration variables.
- #define DEFINE_POWERUP_GPIO_OUTPUT_DATA_VARIABLES()

  Initialize GPIO powerup output variables.
- #define DEFINE_POWERDOWN_GPIO_CFG_VARIABLES()

  Initialize powerdown GPIO configuration variables.
- #define DEFINE_POWERDOWN_GPIO_OUTPUT_DATA_VARIABLES()

  Initialize powerdown GPIO output variables.
- #define SET_POWERUP_GPIO_CFG_REGISTERS()

  Set powerup GPIO configuration registers.
- #define SET_POWERUP_GPIO_OUTPUT_DATA_REGISTERS()

  Set powerup GPIO output registers.
- #define SET_POWERDOWN_GPIO_CFG_REGISTERS()

  Set powerdown GPIO configuration registers.
- #define SET_POWERDOWN_GPIO_OUTPUT_DATA_REGISTERS()

  Set powerdown GPIO output registers.
- #define SET_RESUME_GPIO_CFG_REGISTERS()

  Set resume GPIO configuration registers. Identical to SET_POWERUP.
- #define SET_RESUME_GPIO_OUTPUT_DATA_REGISTERS()

  Set resume GPIO output registers. Identical to SET_POWERUP.
- #define SET_SUSPEND_GPIO_CFG_REGISTERS()

  Set suspend GPIO configuration registers. SET_POWERDOWN minus USB regs.
- #define SET_SUSPEND_GPIO_OUTPUT_DATA_REGISTERS()

Set suspend GPIO output registers. SET_POWERDOWN minus USB regs.

- #define CONFIGURE_EXTERNAL_REGULATOR_ENABLE() GPIO->DBGCFG &= ~GPIO_DBGCFG_E↩ XTREGEN;

  External regulator enable/disable macro.

GPIO Wake Source Definitions

A convenient define that chooses if this external signal can be used as source to wake from deep sleep. Any change in the state of the signal will wake up the CPU.

- #define WAKE_ON_PA0 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PA1 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PA2 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PA3 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PA4 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PA5 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PA6 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PA7 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PB0 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PB1 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PB2 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PB3 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PB4 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PB5 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PB6 true

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PB7 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PC0 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PC1 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PC2 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PC3 false

true if this GPIO can wake the chip from deep sleep, false if not.

- #define WAKE_ON_PC4 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PC5 false

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PC6 true

  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PC7 false

  true if this GPIO can wake the chip from deep sleep, false if not.

6.38.1 Detailed Description

Note

The file dev0680.h is intended to be copied, renamed, and customized for customer-specific hardware.

The file dev0680.h is the default BOARD_HEADER file used with the breakout board of the development kit.

The EM35x on a dev0680 BoB has the following example GPIO configuration. This board file and the default HAL setup reflects this configuration.

- PA0 - SC2MOSI
- PA1 - SC2MISO
- PA2 - SC2SCLK
- PA3 - SC2nSSEL
- PA4 - PTI_EN
- PA5 - PTI_DATA
- PA6 - LED (on RCM), or Radio HoldOff
- PA7 - LED (on RCM)
- PB0 - Power Amplifier shutdown control / TRACEDATA2
- PB1 - SC1TXD
- PB2 - SC1RXD
- PB3 - SC1nCTS
- PB4 - SC1nRTS
- PB5 - TEMP_SENSE
- PB6 - Button (IRQB fixed to PB6)
- PB7 - Buzzer (also used for DataFlash Enable)
- PC0 - JTAG (JRST) / TRACEDATA1
- PC1 - Power Amplifier antenna select control / TRACEDATA3
- PC2 - JTAG (JTDO) / SWO / TRACEDATA0
- PC3 - JTAG (JTDI) / TRACECLK
- PC4 - JTAG (JTMS) / SWDIO
- PC5 - LED (on BoB)
- PC6 - Button (IRQC pointed to PC6)
- PC7 - TEMP_EN

6.38.2 Macro Definition Documentation

6.38.2.1 #define BUTTON0 PORTB_PIN(6)

6.38.2.2 #define BUTTON0_FLAG_BIT EVENT_GPIO_FLAG_IRQB

6.38.2.3 #define BUTTON0_IN (GPIO->P[1].IN)

6.38.2.4 #define BUTTON0_INT_EN_BIT BIT32(BUTTON0_INT_EN_IRQN)

6.38.2.5 #define BUTTON0_INT_EN_IRQN IRQB_IRQn

6.38.2.6 #define BUTTON0_INTCFG (EVENT_GPIO->CFGB)

6.38.2.7 #define BUTTON0_ISR halIrqBIsr

6.38.2.8 #define BUTTON0_MISS_BIT EVENT_MISS_MISS_IRQB

6.38.2.9 #define BUTTON0_SEL( ) do {} while (0)

Note

IRQB is fixed and as such does not need any selection operation.

6.38.2.10 #define BUTTON1 PORTC_PIN(6)

6.38.2.11 #define BUTTON1_FLAG_BIT EVENT_GPIO_FLAG_IRQC

6.38.2.12 #define BUTTON1_IN (GPIO->P[2].IN)

6.38.2.13 #define BUTTON1_INT_EN_BIT BIT32(BUTTON1_INT_EN_IRQN)

6.38.2.14 #define BUTTON1_INT_EN_IRQN IRQC_IRQn

6.38.2.15 #define BUTTON1_INTCFG (EVENT_GPIO->CFGC)

6.38.2.16 #define BUTTON1_ISR halIrqCIsr

6.38.2.17 #define BUTTON1_MISS_BIT EVENT_MISS_MISS_IRQC

6.38.2.18 #define BUTTON1_SEL( ) do { GPIO->IRQCSEL = PORTC_PIN(6); } while (0)

Note

For this board, IRQC is pointed at PC6

6.38.2.19 #define CFG_TEMPEN _GPIO_P_CFGL_Px0_OUT

ENABLE_OSC32K

6.38.2.20 #define CONFIGURE_EXTERNAL_REGULATOR_ENABLE( ) GPIO->DBGCFG &= ~GPIO_DBGCFG_EXTREGEN;

6.38.2.21 #define DEFINE_GPIO_RADIO_POWER_BOARD_MASK_VARIABLE( ) GpioMaskType gpioRadioPowerBoardMask = 0

6.38.2.22 #define DEFINE_POWERDOWN_GPIO_CFG_VARIABLES( )

Value:

```
uint16_t gpioCfgPowerDown[6] = {
                                \
    ((PWRDN_CFG_USBDM           << _GPIO_P_CFGL_Px0_SHIFT)
                                \
     | (PWRDN_CFG_USBDP           << _GPIO_P_CFGL_Px1_SHIFT)
                                \
     | (PWRDN_CFG_ENUMCTRL        << _GPIO_P_CFGL_Px2_SHIFT)
                                 \
     | (PWRDN_CFG_VBUSMON         << _GPIO_P_CFGL_Px3_SHIFT)),
                                \
    ((PWRDN_CFG_PTI_EN          << _GPIO_P_CFGH_Px4_SHIFT)
                                 \
     | (PWRDN_CFG_PTI_DATA        << _GPIO_P_CFGH_Px5_SHIFT)
                                 \
     | (PWRDN_CFG_DFL_RHO         << _GPIO_P_CFGH_Px6_SHIFT)
                                \
     | (_GPIO_P_CFGL_Px0_OUT      << _GPIO_P_CFGH_Px7_SHIFT)),                                    \
    ((_GPIO_P_CFGL_Px0_OUT      << _GPIO_P_CFGL_Px0_SHIFT)                                       \
     | (_GPIO_P_CFGL_Px0_OUT      << _GPIO_P_CFGL_Px1_SHIFT)                 /* SC1TXD  */ \
     | (_GPIO_P_CFGL_Px0_IN_PUD   << _GPIO_P_CFGL_Px2_SHIFT)                 /* SC1RXD  */ \
     | (_GPIO_P_CFGL_Px0_IN_PUD   << _GPIO_P_CFGL_Px3_SHIFT)),               /* SC1nCTS */ \
    ((_GPIO_P_CFGL_Px0_OUT      << _GPIO_P_CFGH_Px4_SHIFT)                 /* SC1nRTS */ \
     |                          /* disable analog for sleep */                           \
     (_GPIO_P_CFGL_Px0_IN_PUD   << _GPIO_P_CFGH_Px5_SHIFT)                               \
     | (_GPIO_P_CFGL_Px0_IN_PUD   << _GPIO_P_CFGH_Px6_SHIFT)                             \
     |                          /* need to use pulldown for sleep */                    \
     (_GPIO_P_CFGL_Px0_IN_PUD   << _GPIO_P_CFGH_Px7_SHIFT)),                             \
    ((_GPIO_P_CFGL_Px0_IN_PUD   << _GPIO_P_CFGL_Px0_SHIFT)                              \
     | (_GPIO_P_CFGL_Px0_OUT      << _GPIO_P_CFGL_Px1_SHIFT)                           \
     | (_GPIO_P_CFGL_Px0_OUT      << _GPIO_P_CFGL_Px2_SHIFT)                           \
     | (_GPIO_P_CFGL_Px0_IN_PUD   << _GPIO_P_CFGL_Px3_SHIFT)),                          \
    ((_GPIO_P_CFGL_Px0_IN_PUD   << _GPIO_P_CFGH_Px4_SHIFT)                              \
     | (PWRDN_CFG_LED2            << _GPIO_P_CFGH_Px5_SHIFT)
                                \
     | (PWRDN_CFG_BUTTON1         << _GPIO_P_CFGH_Px6_SHIFT)
                                 \
     | (CFG_TEMPEN                << _GPIO_P_CFGH_Px7_SHIFT))
                                  \
  }
```

6.38.2.23 #define DEFINE_POWERDOWN_GPIO_OUTPUT_DATA_VARIABLES( )

6.38.2.24 #define DEFINE_POWERUP_GPIO_CFG_VARIABLES( )

Value:

```
uint16_t gpioCfgPowerUp[6] = {
        \
    ((PWRUP_CFG_USBDM           << _GPIO_P_CFGL_Px0_SHIFT)
          \
     | (PWRUP_CFG_USBDP           << _GPIO_P_CFGL_Px1_SHIFT)
           \
     | (PWRUP_CFG_ENUMCTRL        << _GPIO_P_CFGL_Px2_SHIFT)
            \
     | (PWRUP_CFG_VBUSMON         << _GPIO_P_CFGL_Px3_SHIFT)),
            \
    ((PWRUP_CFG_PTI_EN           << _GPIO_P_CFGH_Px4_SHIFT)
          \
     | (PWRUP_CFG_PTI_DATA        << _GPIO_P_CFGH_Px5_SHIFT)
           \
     | (PWRUP_CFG_DFL_RHO         << _GPIO_P_CFGH_Px6_SHIFT)
          \
     | (_GPIO_P_CFGL_Px0_OUT      << _GPIO_P_CFGH_Px7_SHIFT)),                                    \
    ((_GPIO_P_CFGL_Px0_OUT       << _GPIO_P_CFGL_Px0_SHIFT)                                      \
     | (PWRUP_CFG_SC1_TXD         << _GPIO_P_CFGL_Px1_SHIFT)                  /*
       SC1TXD  */ \
     | (_GPIO_P_CFGL_Px0_IN_PUD   << _GPIO_P_CFGL_Px2_SHIFT)                  /* SC1RXD  */ \
     | (_GPIO_P_CFGL_Px0_IN_PUD   << _GPIO_P_CFGL_Px3_SHIFT)),                /* SC1nCTS */ \
    ((_GPIO_P_CFGL_Px0_OUT_ALT  << _GPIO_P_CFGH_Px4_SHIFT)                   /* SC1nRTS */ \
     | (_GPIO_P_CFGL_Px0_ANALOG   << _GPIO_P_CFGH_Px5_SHIFT)                                     \
     | (_GPIO_P_CFGL_Px0_IN_PUD   << _GPIO_P_CFGH_Px6_SHIFT)                                     \
     | (_GPIO_P_CFGL_Px0_OUT_ALT  << _GPIO_P_CFGH_Px7_SHIFT)),                                    \
    ((_GPIO_P_CFGL_Px0_IN        << _GPIO_P_CFGL_Px0_SHIFT)                                      \
     | (_GPIO_P_CFGL_Px0_OUT      << _GPIO_P_CFGL_Px1_SHIFT)                                     \
     | (_GPIO_P_CFGL_Px0_OUT_ALT  << _GPIO_P_CFGL_Px2_SHIFT)                                     \
     | (_GPIO_P_CFGL_Px0_IN       << _GPIO_P_CFGL_Px3_SHIFT)),                                   \
    ((_GPIO_P_CFGL_Px0_IN        << _GPIO_P_CFGH_Px4_SHIFT)                                      \
     | (PWRUP_CFG_LED2            << _GPIO_P_CFGH_Px5_SHIFT)
          \
     | (PWRUP_CFG_BUTTON1         << _GPIO_P_CFGH_Px6_SHIFT)
           \
     | (CFG_TEMPEN                << _GPIO_P_CFGH_Px7_SHIFT))
      \
  }
```

6.38.2.25 #define DEFINE_POWERUP_GPIO_OUTPUT_DATA_VARIABLES( )

Value:

```
uint8_t gpioOutPowerUp[3] = {                                                                   \
    ((PWRUP_OUT_USBDM    << _GPIO_P_OUT_Px0_SHIFT)                                                \
     | (PWRUP_OUT_USBDP    << _GPIO_P_OUT_Px1_SHIFT)                                             \
     | (PWRUP_OUT_ENUMCTRL << _GPIO_P_OUT_Px2_SHIFT)
      \
     | (PWRUP_OUT_VBUSMON  << _GPIO_P_OUT_Px3_SHIFT)                                            \
     | (PWRUP_OUT_PTI_EN   << _GPIO_P_OUT_Px4_SHIFT)                                            \
     | (PWRUP_OUT_PTI_DATA << _GPIO_P_OUT_Px5_SHIFT)
      \
     | (PWRUP_OUT_DFL_RHO  << _GPIO_P_OUT_Px6_SHIFT)                                            \
     |                        /* LED default off */                                             \
     (1                   << _GPIO_P_OUT_Px7_SHIFT)),                                           \
    ((1                   << _GPIO_P_OUT_Px0_SHIFT)                                             \
     | (1                   << _GPIO_P_OUT_Px1_SHIFT)                  /* SC1TXD  */ \
     | (1                   << _GPIO_P_OUT_Px2_SHIFT)                  /* SC1RXD  */ \
     | (1                   << _GPIO_P_OUT_Px3_SHIFT)                  /* SC1nCTS */ \
     | (0                   << _GPIO_P_OUT_Px4_SHIFT)                  /* SC1nRTS */ \
     | (0                   << _GPIO_P_OUT_Px5_SHIFT)                                          \
     |                        /* PB6 has button needing a pullup */                             \
     (1                   << _GPIO_P_OUT_Px6_SHIFT)                                             \
     | (0                   << _GPIO_P_OUT_Px7_SHIFT)),                                           \
    ((0                   << _GPIO_P_OUT_Px0_SHIFT)                                             \
     | (0                   << _GPIO_P_OUT_Px1_SHIFT)                                          \
     | (1                   << _GPIO_P_OUT_Px2_SHIFT)                                          \
     | (0                   << _GPIO_P_OUT_Px3_SHIFT)                                          \
     | (0                   << _GPIO_P_OUT_Px4_SHIFT)                                          \
     | (PWRUP_OUT_LED2      << _GPIO_P_OUT_Px5_SHIFT)                                          \
     | (PWRUP_OUT_BUTTON1   << _GPIO_P_OUT_Px6_SHIFT)                                          \
     |                        /* Temp Sensor default on */                                      \
     (1                   << _GPIO_P_OUT_Px7_SHIFT))                                            \
  }
```

6.38.2.26 #define EMBER_SERIAL_BAUD_CUSTOM 13

1. Refer to the EM35x datasheet's discussion on UART baud rates for the equation used to derive this value.

6.38.2.27 #define ENUMCTRL PORTA_PIN(2)

6.38.2.28 #define ENUMCTRL_CLR( ) do { GPIO->P[0].CLR = GPIO_P_SET_Px2; } while (0)

6.38.2.29 #define ENUMCTRL_SET( ) do { GPIO->P[0].SET = GPIO_P_SET_Px2; } while (0)

6.38.2.30 #define ENUMCTRL_SETCFG(*cfg*)

Value:

```
do { SET_CMSIS_REG_FIELD(GPIO->P[0].CFGL, \
                                                    GPIO_P_CFGL_Px2, \
                                                    cfg);            \
} while (0)
```

6.38.2.31 #define halInternalInitRadioHoldOff( ) /∗ no-op ∗/

Note

If ::RHO_GPIO is not defined, then Radio HoldOff support will not be built in even for runtime use. The GPIO signal level to assert Radio HoldOff (1=high, 0=low).

6.38.2.32 #define OSC32K_STARTUP_DELAY_MS (0)

6.38.2.33 #define PACKET_TRACE

6.38.2.34 #define PWRDN_CFG_BUTTON1 _GPIO_P_CFGL_Px0_IN_PUD

6.38.2.35 #define PWRDN_CFG_DFL_RHO PWRDN_CFG_DFL_RHO_FOR_DFL

Note

If ::RHO_GPIO is not defined, then Radio HoldOff support will not be built in even for runtime use. The GPIO signal level to assert Radio HoldOff (1=high, 0=low).

6.38.2.36 #define PWRDN_CFG_DFL_RHO_FOR_DFL _GPIO_P_CFGL_Px0_OUT

Note

If ::RHO_GPIO is not defined, then Radio HoldOff support will not be built in even for runtime use. The GPIO signal level to assert Radio HoldOff (1=high, 0=low).

6.38.2.37 #define PWRDN_CFG_DFL_RHO_FOR_RHO _GPIO_P_CFGL_Px0_IN_PUD

Note

If ::RHO_GPIO is not defined, then Radio HoldOff support will not be built in even for runtime use. The GPIO signal level to assert Radio HoldOff (1=high, 0=low).

6.38.2.38 #define PWRDN_CFG_ENUMCTRL _GPIO_P_CFGL_Px0_IN_PUD

6.38.2.39 #define PWRDN_CFG_LED2 _GPIO_P_CFGL_Px0_OUT

ENABLE_ALT_FUNCTION_TX_ACTIVE

6.38.2.40 #define PWRDN_CFG_PTI_DATA _GPIO_P_CFGL_Px0_IN_PUD

6.38.2.41 #define PWRDN_CFG_PTI_EN _GPIO_P_CFGL_Px0_IN_PUD

6.38.2.42 #define PWRDN_CFG_USBDM _GPIO_P_CFGL_Px0_IN_PUD

6.38.2.43 #define PWRDN_CFG_USBDP _GPIO_P_CFGL_Px0_IN_PUD

6.38.2.44 #define PWRDN_CFG_VBUSMON _GPIO_P_CFGL_Px0_OUT

6.38.2.45 #define PWRDN_OUT_BUTTON1 1 /∗ Button needs a pullup ∗/

6.38.2.46 #define PWRDN_OUT_DFL_RHO PWRDN_OUT_DFL_RHO_FOR_DFL

Note

If ::RHO_GPIO is not defined, then Radio HoldOff support will not be built in even for runtime use. The GPIO signal level to assert Radio HoldOff (1=high, 0=low).

6.38.2.47 #define PWRDN_OUT_DFL_RHO_FOR_DFL 1 /∗ LED off ∗/

Note

If ::RHO_GPIO is not defined, then Radio HoldOff support will not be built in even for runtime use. The GPIO signal level to assert Radio HoldOff (1=high, 0=low).

6.38.2.48 #define PWRDN_OUT_DFL_RHO_FOR_RHO 0 /∗ Deassert ∗/

Note

If ::RHO_GPIO is not defined, then Radio HoldOff support will not be built in even for runtime use. The GPIO signal level to assert Radio HoldOff (1=high, 0=low).

6.38.2.49 #define PWRDN_OUT_ENUMCTRL 1

6.38.2.50 #define PWRDN_OUT_LED2 1 /∗ LED default off ∗/

ENABLE_ALT_FUNCTION_TX_ACTIVE

6.38.2.51 #define PWRDN_OUT_PTI_DATA 1

6.38.2.52 #define PWRDN_OUT_PTI_EN 0

6.38.2.53 #define PWRDN_OUT_SC1_nRTS 1

6.38.2.54 #define PWRDN_OUT_USBDM 1

6.38.2.55 #define PWRDN_OUT_USBDP 1

6.38.2.56 #define PWRDN_OUT_VBUSMON 1

6.38.2.57 #define PWRUP_CFG_BUTTON1 _GPIO_P_CFGL_Px0_IN_PUD

6.38.2.58 #define PWRUP_CFG_DFL_RHO PWRUP_CFG_DFL_RHO_FOR_DFL

(defined(RADIO_HOLDOFF) && defined(RHO_GPIO))

6.38.2.59 #define PWRUP_CFG_DFL_RHO_FOR_DFL _GPIO_P_CFGL_Px0_OUT

6.38.2.60 #define PWRUP_CFG_DFL_RHO_FOR_RHO _GPIO_P_CFGL_Px0_IN_PUD

6.38.2.61 #define PWRUP_CFG_ENUMCTRL _GPIO_P_CFGL_Px0_OUT_ALT

6.38.2.62 #define PWRUP_CFG_LED2 _GPIO_P_CFGL_Px0_OUT

ENABLE_ALT_FUNCTION_TX_ACTIVE

6.38.2.63 #define PWRUP_CFG_PTI_DATA _GPIO_P_CFGL_Px0_OUT_ALT

6.38.2.64 #define PWRUP_CFG_PTI_EN _GPIO_P_CFGL_Px0_OUT_ALT

6.38.2.65 #define PWRUP_CFG_SC1_TXD _GPIO_P_CFGL_Px0_OUT_ALT

SLEEPY_IP_MODEM_UART

6.38.2.66 #define PWRUP_CFG_USBDM _GPIO_P_CFGL_Px0_OUT_ALT

6.38.2.67 #define PWRUP_CFG_USBDP _GPIO_P_CFGL_Px0_IN

6.38.2.68 #define PWRUP_CFG_VBUSMON _GPIO_P_CFGL_Px0_OUT

6.38.2.69 #define PWRUP_OUT_BUTTON1 1 /* Button needs a pullup */

6.38.2.70 #define PWRUP_OUT_DFL_RHO PWRUP_OUT_DFL_RHO_FOR_DFL

Note

If ::RHO_GPIO is not defined, then Radio HoldOff support will not be built in even for runtime use. The GPIO signal level to assert Radio HoldOff (1=high, 0=low).

6.38.2.71 #define PWRUP_OUT_DFL_RHO_FOR_DFL 1 /* LED default off */

Note

If ::RHO_GPIO is not defined, then Radio HoldOff support will not be built in even for runtime use. The GPIO signal level to assert Radio HoldOff (1=high, 0=low).

6.38.2.72 #define PWRUP_OUT_DFL_RHO_FOR_RHO 0 /* Deassert */

Note

If ::RHO_GPIO is not defined, then Radio HoldOff support will not be built in even for runtime use. The GPIO signal level to assert Radio HoldOff (1=high, 0=low).

6.38.2.73 #define PWRUP_OUT_ENUMCTRL 0

6.38.2.74 #define PWRUP_OUT_LED2 1 /* LED default off */

ENABLE_ALT_FUNCTION_TX_ACTIVE

6.38.2.75 #define PWRUP_OUT_PTI_DATA 1

6.38.2.76 #define PWRUP_OUT_PTI_EN 0

6.38.2.77 #define PWRUP_OUT_USBDM 0

6.38.2.78 #define PWRUP_OUT_USBDP 0

6.38.2.79 #define PWRUP_OUT_VBUSMON 1

6.38.2.80 #define RHO_ASSERTED 1

Note

If ::RHO_GPIO is not defined, then Radio HoldOff support will not be built in even for runtime use. The GPIO signal level to assert Radio HoldOff (1=high, 0=low).

6.38.2.81 #define RHO_CFG (GPIO->P[0].CFGH)

6.38.2.82 #define RHO_FLAG_BIT EVENT_GPIO_FLAG_IRQD

6.38.2.83 #define RHO_IN (GPIO->P[0].IN)

6.38.2.84 #define RHO_INT_EN_BIT BIT32(RHO_INT_EN_IRQN)

Note

If ::RHO_GPIO is not defined, then Radio HoldOff support will not be built in even for runtime use. The GPIO signal level to assert Radio HoldOff (1=high, 0=low).

6.38.2.85 #define RHO_INT_EN_IRQN IRQD_IRQn

6.38.2.86 #define RHO_INTCFG (EVENT_GPIO->CFGD)

6.38.2.87 #define RHO_ISR hallrqDIsr

6.38.2.88 #define RHO_MISS_BIT EVENT_MISS_MISS_IRQD

6.38.2.89 #define RHO_OUT (GPIO->P[0].OUT)

6.38.2.90 #define RHO_SEL( ) do { GPIO->IRQDSEL = RHO_GPIO; } while (0)

6.38.2.91 #define SET_POWERDOWN_GPIO_CFG_REGISTERS( )

Value:

```
GPIO->P[0].CFGL = gpioCfgPowerDown[0];   \
  GPIO->P[0].CFGH = gpioCfgPowerDown[1];   \
  GPIO->P[1].CFGL = gpioCfgPowerDown[2];   \
  GPIO->P[1].CFGH = gpioCfgPowerDown[3];   \
  GPIO->P[2].CFGL = gpioCfgPowerDown[4];   \
  GPIO->P[2].CFGH = gpioCfgPowerDown[5];
```

6.38.2.92 #define SET_POWERDOWN_GPIO_OUTPUT_DATA_REGISTERS( )

Value:

```
GPIO->P[0].OUT = gpioOutPowerDown[0];            \
  GPIO->P[1].OUT = gpioOutPowerDown[1];            \
  GPIO->P[2].OUT = gpioOutPowerDown[2];
```

6.38.2.93 #define SET_POWERUP_GPIO_CFG_REGISTERS( )

Value:

```
GPIO->P[0].CFGL = gpioCfgPowerUp[0];   \
  GPIO->P[0].CFGH = gpioCfgPowerUp[1];   \
  GPIO->P[1].CFGL = gpioCfgPowerUp[2];   \
  GPIO->P[1].CFGH = gpioCfgPowerUp[3];   \
  GPIO->P[2].CFGL = gpioCfgPowerUp[4];   \
  GPIO->P[2].CFGH = gpioCfgPowerUp[5];
```

6.38.2.94 #define SET_POWERUP_GPIO_OUTPUT_DATA_REGISTERS( )

Value:

```
GPIO->P[0].OUT = gpioOutPowerUp[0];            \
  GPIO->P[1].OUT = gpioOutPowerUp[1];            \
  GPIO->P[2].OUT = gpioOutPowerUp[2];
```

6.38.2.95 #define SET_RESUME_GPIO_CFG_REGISTERS( )

Value:

```
/* GPIO->P[0].CFGL USB untouched! */  \
  GPIO->P[0].CFGH = gpioCfgPowerUp[1];  \
  GPIO->P[1].CFGL = gpioCfgPowerUp[2];  \
  GPIO->P[1].CFGH = gpioCfgPowerUp[3];  \
  GPIO->P[2].CFGL = gpioCfgPowerUp[4];  \
  GPIO->P[2].CFGH = gpioCfgPowerUp[5];
```

6.38.2.96 #define SET_RESUME_GPIO_OUTPUT_DATA_REGISTERS( )

Value:

```
GPIO->P[0].OUT = (GPIO->P[0].OUT & 0x0F) /*USB untouched*/ \
                 | (gpioOutPowerUp[0] & 0xF0);             \
  GPIO->P[1].OUT = gpioOutPowerUp[1];                      \
  GPIO->P[2].OUT = gpioOutPowerUp[2];
```

6.38.2.97 #define SET_SUSPEND_GPIO_CFG_REGISTERS( )

Value:

```
/* GPIO->P[0].CFGL USB untouched! */   \
  GPIO->P[0].CFGH = gpioCfgPowerDown[1]; \
  GPIO->P[1].CFGL = gpioCfgPowerDown[2]; \
  GPIO->P[1].CFGH = gpioCfgPowerDown[3]; \
  GPIO->P[2].CFGL = gpioCfgPowerDown[4]; \
  GPIO->P[2].CFGH = gpioCfgPowerDown[5];
```

6.38.2.98 #define SET_SUSPEND_GPIO_OUTPUT_DATA_REGISTERS( )

Value:

```
GPIO->P[0].OUT = (GPIO->P[0].OUT & 0x0F)  /*USB untouched*/ \
                 | (gpioOutPowerDown[0] & 0xF0);            \
  GPIO->P[1].OUT = gpioOutPowerDown[1];                     \
  GPIO->P[2].OUT = gpioOutPowerDown[2];
```

6.38.2.99 #define TEMP_SENSOR_ADC_CHANNEL ADC_SOURCE_ADC0_VREF2

6.38.2.100 #define TEMP_SENSOR_SCALE_FACTOR 1

6.38.2.101 #define USB_MAX_POWER (50)

6.38.2.102 #define USB_REMOTEWKUPEN_STATE (1)

Set the define USB_REMOTEWKUPEN_STATE: 0 remote wakeup is disabled. 1 remote wakeup is enabled.

Referenced by USBD_Init().

6.38.2.103 #define USB_SELFPWRD_STATE (1)

Set the define USB_SELFPWRD_STATE: 0 if the device is bus powered. 1 if the device self powered.

6.38.2.104 #define VBUSMON GPIO_P_IN_Px3

Leaving VBUSMON undefined will keep VBUS Monitoring functionality from being compiled in and not conflict with other pieces that might want to use the GPIO or IRQ that VBUS Monitoring needs.

6.38.2.105 #define VBUSMON_FLAG_BIT EVENT_GPIO_FLAG_IRQD

6.38.2.106 #define VBUSMON_IN (GPIO->P[0].IN)

6.38.2.107 #define VBUSMON_INT_EN_BIT BIT32(VBUSMON_INT_EN_IRQN)

Leaving VBUSMON undefined will keep VBUS Monitoring functionality from being compiled in and not conflict with other pieces that might want to use the GPIO or IRQ that VBUS Monitoring needs.

6.38.2.108 #define VBUSMON_INT_EN_IRQN IRQD_IRQn

6.38.2.109 #define VBUSMON_INTCFG (EVENT_GPIO->CFGD)

6.38.2.110 #define VBUSMON_ISR halIrqDIsr

6.38.2.111 #define VBUSMON_MISS_BIT EVENT_MISS_MISS_IRQD

6.38.2.112 #define VBUSMON_SEL( ) do { GPIO->IRQDSEL = PORTA_PIN(3); } while (0)

Note

For this board, IRQC is pointed at PA3.

6.38.2.113 #define VBUSMON_SETCFG( )

Value:

```
do { SET_CMSIS_REG_FIELD(GPIO->P[0].CFGL,       \
                                                  GPIO_P_CFGL_Px3,       \
                                                  _GPIO_P_CFGL_Px3_IN); \
} while (0)
```

6.38.2.114 #define WAKE_ON_PA0 false

6.38.2.115 #define WAKE_ON_PA1 false

6.38.2.116 #define WAKE_ON_PA2 false

6.38.2.117 #define WAKE_ON_PA3 false

6.38.2.118 #define WAKE_ON_PA4 false

6.38.2.119 #define WAKE_ON_PA5 false

6.38.2.120 #define WAKE_ON_PA6 false

6.38.2.121 #define WAKE_ON_PA7 false

6.38.2.122 #define WAKE_ON_PB0 false

6.38.2.123 #define WAKE_ON_PB1 false

6.38.2.124 #define WAKE_ON_PB2 false

6.38.2.125 #define WAKE_ON_PB3 false

6.38.2.126 #define WAKE_ON_PB4 false

6.38.2.127 #define WAKE_ON_PB5 false

6.38.2.128 #define WAKE_ON_PB6 true

6.38.2.129 #define WAKE_ON_PB7 false

6.38.2.130 #define WAKE_ON_PC0 false

6.38.2.131 #define WAKE_ON_PC1 false

6.38.2.132 #define WAKE_ON_PC2 false

6.38.2.133 #define WAKE_ON_PC3 false

6.38.2.134 #define WAKE_ON_PC4 false

6.38.2.135 #define WAKE_ON_PC5 false

6.38.2.136 #define WAKE_ON_PC6 true

6.38.2.137 #define WAKE_ON_PC7 false

6.38.3 Enumeration Type Documentation

6.38.3.1 enum HalBoardLedPins

Enumerator

BOARDLED0

BOARDLED1

BOARDLED2

BOARDLED3

BOARD_ACTIVITY_LED

BOARD_HEARTBEAT_LED

6.38.4 Variable Documentation

6.38.4.1 uint16_t gpioCfgPowerDown[6]

6.38.4.2 uint16_t gpioCfgPowerUp[6]

6.38.4.3 uint8_t gpioOutPowerDown[3]

6.38.4.4 uint8_t gpioOutPowerUp[3]

6.38.4.5 GpioMaskType gpioRadioPowerBoardMask

6.39 IAR PLATFORM_HEADER Configuration

Compiler and Platform specific definitions and typedefs for the IAR ARM C compiler.

Macros

- #define HAL_HAS_INT64

  Denotes that this platform supports 64-bit data-types.
- #define _HAL_USE_COMMON_PGM_

  Use the Master Program Memory Declarations from platform-common.h.
- #define _HAL_USE_COMMON_MEMUTILS_

  If the line below is uncommented we will use Ember memory APIs, otherwise, we will use the C Standard library (memset,memcpy,memmove) APIs.
- #define PLATCOMMONOKTOINCLUDE

  Include platform-common.h last to pick up defaults and common definitions.
- #define MAIN_FUNCTION_PARAMETERS void

  The kind of arguments the main function takes.
- #define MAIN_FUNCTION_ARGUMENTS

Functions

- void _executeBarrierInstructions (void)

Master Variable Types

These are a set of typedefs to make the size of all variable declarations explicitly known.

- typedef bool boolean

  A typedef to make the size of the variable explicitly known.
- typedef unsigned char int8u

  A typedef to make the size of the variable explicitly known.
- typedef signed char int8s

  A typedef to make the size of the variable explicitly known.
- typedef unsigned short int16u

  A typedef to make the size of the variable explicitly known.
- typedef signed short int16s

  A typedef to make the size of the variable explicitly known.
- typedef unsigned int int32u

  A typedef to make the size of the variable explicitly known.
- typedef signed int int32s

  A typedef to make the size of the variable explicitly known.
- typedef unsigned long long int64u

  A typedef to make the size of the variable explicitly known.
- typedef signed long long int64s

  A typedef to make the size of the variable explicitly known.
- typedef unsigned int PointerType

  A typedef to make the size of the variable explicitly known.

Miscellaneous Macros

- void halInternalAssertFailed (const char *filename, int linenumber)

  A prototype definition for use by the assert macro. (see hal/micro/micro.h)
- void halInternalResetWatchDog (void)

  Macro to reset the watchdog timer. Note: be very very careful when using this as you can easily get into an infinite loop if you are not careful.
- #define BIGENDIAN_CPU false

  A convenient method for code to know what endiannes processor it is running on. For the Cortex-M3, we are little endian.
- #define NTOHS(val) (__REV16(val))

  Define intrinsics for NTOHL and NTOHS to save code space by making endian.c compile to nothing.
- #define NTOHL(val) (__REV(val))

  A convenient method for code to know what endiannes processor it is running on. For the Cortex-M3, we are little endian.
- #define NO_STRIPPING __root

  A friendlier name for the compiler's intrinsic for not stripping.
- #define EEPROM errorerror

  A friendlier name for the compiler's intrinsic for eeprom reference.
- #define __SOURCEFILE__ __FILE__

  *The **SOURCEFILE** macro is used by asserts to list the filename if it isn't otherwise defined, set it to the compiler intrinsic which specifies the whole filename and path of the sourcefile.*
- #define assert(condition)

  A custom implementation of the C language assert macro. This macro implements the conditional evaluation and calls the function halInternalAssertFailed(). (see hal/micro/micro.h)
- #define halResetWatchdog() halInternalResetWatchDog()

  A convenient method for code to know what endiannes processor it is running on. For the Cortex-M3, we are little endian.
- #define __attribute__(...)

  *Define **attribute** to nothing since it isn't handled by IAR.*
- #define UNUSED

  Declare a variable as unused to avoid a warning. Has no effect in IAR builds.
- #define SIGNED_ENUM

  Some platforms need to cast enum values that have the high bit set.
- #define STACK_FILL_VALUE 0xCDCDCDCDU

  Define the magic value that is interpreted by IAR C-SPY's Stack View.
- #define RAMFUNC __ramfunc

  Define a generic RAM function identifier to a compiler specific one.
- #define NO_OPERATION() __no_operation()

  Define a generic no operation identifier to a compiler specific one.
- #define SET_REG_FIELD(reg, field, value)

  A convenience macro that makes it easy to change the field of a register to any value.
- #define SET_CMSIS_REG_FIELD(reg, field, value)

  A convenience macro that makes it easy to change the field of a register, as defined in CMSIS Device headers, to any value. Example using EM35xx: SET_CMSIS_REG_FIELD(GPIO->P[0].CFGL, GPIO_P_CFGL_Px0, _GPIO↩ _P_CFGL_Px0_OUT);.
- #define simulatedTimePasses()

  Stub for code not running in simulation.
- #define simulatedTimePassesMs(x)

  Stub for code not running in simulation.
- #define simulatedSerialTimePasses()

  Stub for code not running in simulation.
- #define _HAL_USE_COMMON_DIVMOD_

Use the Divide and Modulus Operations from platform-common.h.

- #define VAR_AT_SEGMENT(__variableDeclaration, __segmentName) __variableDeclaration @ __segmentName

  Provide a portable way to specify the segment where a variable lives.
- #define STRINGIZE(X) #X

  Convinience macro for turning a token into a string.
- #define ALIGNMENT(X) _Pragma(STRINGIZE(data_alignment = X))

  Provide a portable way to align data.
- #define WEAK(__symbol) __weak __symbol

  Provide a portable way to specify a symbol as weak.
- #define NO_INIT(__symbol) __no_init __symbol

  Provide a portable way to specify a non initialized symbol.
- #define STATIC_ASSERT(__condition, __errorstr) static_assert(__condition, __errorstr)

  Provide a portable way to specify a compile time assert.

Portable segment names

- #define __NO_INIT__ ".noinit"

  Portable segment names.
- #define __DEBUG_CHANNEL__ "DEBUG_CHANNEL"

  Portable segment names.
- #define __INTVEC__ ".intvec"

  Portable segment names.
- #define __CSTACK__ "CSTACK"

  Portable segment names.
- #define __RESETINFO__ "RESETINFO"

  Portable segment names.
- #define __DATA_INIT__ ".data_init"

  Portable segment names.
- #define __DATA__ ".data"

  Portable segment names.
- #define __BSS__ ".bss"

  Portable segment names.
- #define __APP_RAM__ "APP_RAM"

  Portable segment names.
- #define __CONST__ ".rodata"

  Portable segment names.
- #define __TEXT__ ".text"

  Portable segment names.
- #define __TEXTRW_INIT__ ".textrw_init"

  Portable segment names.
- #define __TEXTRW__ ".textrw"

  Portable segment names.
- #define __AAT__ "AAT"

  Portable segment names.
- #define __BAT__ "BAT"

  Portable segment names.
- #define __BAT_INIT__ "BAT"

  Portable segment names.
- #define __FAT__ "FAT"

Portable segment names.

- #define __RAT__ "RAT"

  Portable segment names.
- #define __NVM__ "NVM"

  Portable segment names.
- #define __SIMEE__ "SIMEE"

  Portable segment names.
- #define __PSSTORE__ "PSSTORE"

  Portable segment names.
- #define __EMHEAP__ "EMHEAP"

  Portable segment names.
- #define __EMHEAP_OVERLAY__ "EMHEAP_overlay"

  Portable segment names.
- #define __GUARD_REGION__ "GUARD_REGION"

  Portable segment names.
- #define __DLIB_PERTHREAD_INIT__ "__DLIB_PERTHREAD_init"

  Portable segment names.
- #define __DLIB_PERTHREAD_INITIALIZED_DATA__ "DLIB_PERTHREAD_INITIALIZED_DATA"

  Portable segment names.
- #define __DLIB_PERTHREAD_ZERO_DATA__ "DLIB_PERTHREAD_ZERO_DATA"

  Portable segment names.
- #define __INTERNAL_STORAGE__ "INTERNAL_STORAGE"

  Portable segment names.
- #define __UNRETAINED_RAM__ "UNRETAINED_RAM"

  Portable segment names.
- #define _NO_INIT_SEGMENT_BEGIN __segment_begin(__NO_INIT__)

  Portable segment names.
- #define _DEBUG_CHANNEL_SEGMENT_BEGIN __segment_begin(__DEBUG_CHANNEL__)

  Portable segment names.
- #define _INTVEC_SEGMENT_BEGIN __segment_begin(__INTVEC__)

  Portable segment names.
- #define _CSTACK_SEGMENT_BEGIN __segment_begin(__CSTACK__)

  Portable segment names.
- #define _RESETINFO_SEGMENT_BEGIN __segment_begin(__RESETINFO__)

  Portable segment names.
- #define _DATA_INIT_SEGMENT_BEGIN __segment_begin(__DATA_INIT__)

  Portable segment names.
- #define _DATA_SEGMENT_BEGIN __segment_begin(__DATA__)

  Portable segment names.
- #define _BSS_SEGMENT_BEGIN __segment_begin(__BSS__)

  Portable segment names.
- #define _APP_RAM_SEGMENT_BEGIN __segment_begin(__APP_RAM__)

  Portable segment names.
- #define _CONST_SEGMENT_BEGIN __segment_begin(__CONST__)

  Portable segment names.
- #define _TEXT_SEGMENT_BEGIN __segment_begin(__TEXT__)

  Portable segment names.
- #define _TEXTRW_INIT_SEGMENT_BEGIN __segment_begin(__TEXTRW_INIT__)

  Portable segment names.
- #define _TEXTRW_SEGMENT_BEGIN __segment_begin(__TEXTRW__)

  Portable segment names.

- #define _AAT_SEGMENT_BEGIN __segment_begin(__AAT__)

  Portable segment names.
- #define _BAT_SEGMENT_BEGIN __segment_begin(__BAT__)

  Portable segment names.
- #define _BAT_INIT_SEGMENT_BEGIN __segment_begin(__BAT_INIT__)

  Portable segment names.
- #define _FAT_SEGMENT_BEGIN __segment_begin(__FAT__)

  Portable segment names.
- #define _RAT_SEGMENT_BEGIN __segment_begin(__RAT__)

  Portable segment names.
- #define _NVM_SEGMENT_BEGIN __segment_begin(__NVM__)

  Portable segment names.
- #define _SIMEE_SEGMENT_BEGIN __segment_begin(__SIMEE__)

  Portable segment names.
- #define _PSSTORE_SEGMENT_BEGIN __segment_begin(__PSSTORE__)

  Portable segment names.
- #define _EMHEAP_SEGMENT_BEGIN __segment_begin(__EMHEAP__)

  Portable segment names.
- #define _EMHEAP_OVERLAY_SEGMENT_BEGIN __segment_begin(__EMHEAP_OVERLAY__)

  Portable segment names.
- #define _GUARD_REGION_SEGMENT_BEGIN __segment_begin(__GUARD_REGION__)

  Portable segment names.
- #define _DLIB_PERTHREAD_INIT_SEGMENT_BEGIN __segment_begin(__DLIB_PERTHREAD_INIT__)

  Portable segment names.
- #define _DLIB_PERTHREAD_INITIALIZED_DATA_SEGMENT_BEGIN __segment_begin(__DLIB_PERT↩HREAD_INITIALIZED_DATA__)

  Portable segment names.
- #define _DLIB_PERTHREAD_ZERO_DATA_SEGMENT_BEGIN __segment_begin(__DLIB_PERTHREA↩D_ZERO_DATA__)

  Portable segment names.
- #define _INTERNAL_STORAGE_SEGMENT_BEGIN __segment_begin(__INTERNAL_STORAGE__)

  Portable segment names.
- #define _UNRETAINED_RAM_SEGMENT_BEGIN __segment_begin(__UNRETAINED_RAM__)

  Portable segment names.
- #define _NO_INIT_SEGMENT_END __segment_end(__NO_INIT__)

  Portable segment names.
- #define _DEBUG_CHANNEL_SEGMENT_END __segment_end(__DEBUG_CHANNEL__)

  Portable segment names.
- #define _INTVEC_SEGMENT_END __segment_end(__INTVEC__)

  Portable segment names.
- #define _CSTACK_SEGMENT_END __segment_end(__CSTACK__)

  Portable segment names.
- #define _RESETINFO_SEGMENT_END __segment_end(__RESETINFO__)

  Portable segment names.
- #define _DATA_INIT_SEGMENT_END __segment_end(__DATA_INIT__)

  Portable segment names.
- #define _DATA_SEGMENT_END __segment_end(__DATA__)

  Portable segment names.
- #define _BSS_SEGMENT_END __segment_end(__BSS__)

  Portable segment names.
- #define _APP_RAM_SEGMENT_END __segment_end(__APP_RAM__)

Portable segment names.

- #define _CONST_SEGMENT_END __segment_end(__CONST__)

  Portable segment names.
- #define _TEXT_SEGMENT_END __segment_end(__TEXT__)

  Portable segment names.
- #define _TEXTRW_INIT_SEGMENT_END __segment_end(__TEXTRW_INIT__)

  Portable segment names.
- #define _TEXTRW_SEGMENT_END __segment_end(__TEXTRW__)

  Portable segment names.
- #define _AAT_SEGMENT_END __segment_end(__AAT__)

  Portable segment names.
- #define _BAT_SEGMENT_END __segment_end(__BAT__)

  Portable segment names.
- #define _BAT_INIT_SEGMENT_END __segment_end(__BAT_INIT__)

  Portable segment names.
- #define _FAT_SEGMENT_END __segment_end(__FAT__)

  Portable segment names.
- #define _RAT_SEGMENT_END __segment_end(__RAT__)

  Portable segment names.
- #define _NVM_SEGMENT_END __segment_end(__NVM__)

  Portable segment names.
- #define _SIMEE_SEGMENT_END __segment_end(__SIMEE__)

  Portable segment names.
- #define _PSSTORE_SEGMENT_END __segment_end(__PSSTORE__)

  Portable segment names.
- #define _EMHEAP_SEGMENT_END __segment_end(__EMHEAP__)

  Portable segment names.
- #define _EMHEAP_OVERLAY_SEGMENT_END __segment_end(__EMHEAP_OVERLAY__)

  Portable segment names.
- #define _GUARD_REGION_SEGMENT_END __segment_end(__GUARD_REGION__)

  Portable segment names.
- #define _DLIB_PERTHREAD_INIT_SEGMENT_END __segment_end(__DLIB_PERTHREAD_INIT__)

  Portable segment names.
- #define _DLIB_PERTHREAD_INITIALIZED_DATA_SEGMENT_END __segment_end(__DLIB_PERTHR↩EAD_INITIALIZED_DATA__)

  Portable segment names.
- #define _DLIB_PERTHREAD_ZERO_DATA_SEGMENT_END __segment_end(__DLIB_PERTHREAD_Z↩ERO_DATA__)

  Portable segment names.
- #define _INTERNAL_STORAGE_SEGMENT_END __segment_end(__INTERNAL_STORAGE__)

  Portable segment names.
- #define _UNRETAINED_RAM_SEGMENT_END __segment_end(__UNRETAINED_RAM__)

  Portable segment names.
- #define _NO_INIT_SEGMENT_SIZE __segment_size(__NO_INIT__)

  Portable segment names.
- #define _DEBUG_CHANNEL_SEGMENT_SIZE __segment_size(__DEBUG_CHANNEL__)

  Portable segment names.
- #define _INTVEC_SEGMENT_SIZE __segment_size(__INTVEC__)

  Portable segment names.
- #define _CSTACK_SEGMENT_SIZE __segment_size(__CSTACK__)

  Portable segment names.

- #define _RESETINFO_SEGMENT_SIZE __segment_size(__RESETINFO__)

  Portable segment names.
- #define _DATA_INIT_SEGMENT_SIZE __segment_size(__DATA_INIT__)

  Portable segment names.
- #define _DATA_SEGMENT_SIZE __segment_size(__DATA__)

  Portable segment names.
- #define _BSS_SEGMENT_SIZE __segment_size(__BSS__)

  Portable segment names.
- #define _APP_RAM_SEGMENT_SIZE __segment_size(__APP_RAM__)

  Portable segment names.
- #define _CONST_SEGMENT_SIZE __segment_size(__CONST__)

  Portable segment names.
- #define _TEXT_SEGMENT_SIZE __segment_size(__TEXT__)

  Portable segment names.
- #define _TEXTRW_INIT_SEGMENT_SIZE __segment_size(__TEXTRW_INIT__)

  Portable segment names.
- #define _TEXTRW_SEGMENT_SIZE __segment_size(__TEXTRW__)

  Portable segment names.
- #define _AAT_SEGMENT_SIZE __segment_size(__AAT__)

  Portable segment names.
- #define _BAT_SEGMENT_SIZE __segment_size(__BAT__)

  Portable segment names.
- #define _BAT_INIT_SEGMENT_SIZE __segment_size(__BAT_INIT__)

  Portable segment names.
- #define _FAT_SEGMENT_SIZE __segment_size(__FAT__)

  Portable segment names.
- #define _RAT_SEGMENT_SIZE __segment_size(__RAT__)

  Portable segment names.
- #define _NVM_SEGMENT_SIZE __segment_size(__NVM__)

  Portable segment names.
- #define _SIMEE_SEGMENT_SIZE __segment_size(__SIMEE__)

  Portable segment names.
- #define _PSSTORE_SEGMENT_SIZE __segment_size(__PSSTORE__)

  Portable segment names.
- #define _EMHEAP_SEGMENT_SIZE __segment_size(__EMHEAP__)

  Portable segment names.
- #define _EMHEAP_OVERLAY_SEGMENT_SIZE __segment_size(__EMHEAP_OVERLAY__)

  Portable segment names.
- #define _GUARD_REGION_SEGMENT_SIZE __segment_size(__GUARD_REGION__)

  Portable segment names.
- #define _DLIB_PERTHREAD_INIT_SEGMENT_SIZE __segment_size(__DLIB_PERTHREAD_INIT__)

  Portable segment names.
- #define _DLIB_PERTHREAD_INITIALIZED_DATA_SEGMENT_SIZE __segment_size(__DLIB_PERTHR↩EAD_INITIALIZED_DATA__)

  Portable segment names.
- #define _DLIB_PERTHREAD_ZERO_DATA_SEGMENT_SIZE __segment_size(__DLIB_PERTHREAD_↩ZERO_DATA__)

  Portable segment names.
- #define _INTERNAL_STORAGE_SEGMENT_SIZE __segment_size(__INTERNAL_STORAGE__)

  Portable segment names.
- #define _UNRETAINED_RAM_SEGMENT_SIZE __segment_size(__UNRETAINED_RAM__)

  Portable segment names.

External Declarations

These are routines that are defined in certain header files that we don't want to include, e.g. stdlib.h

- int abs (int I)

  Returns the absolute value of I (also called the magnitude of I). That is, if I is negative, the result is the opposite of I, but if I is nonnegative the result is I.

6.39.1 Detailed Description

Note

iar.h should be included first in all source files by setting the preprocessor macro PLATFORM_HEADER to point to it. iar.h automatically includes platform-common.h.

See iar.h and platform-common.h for source code.

6.39.2 Macro Definition Documentation

6.39.2.1 #define __AAT__ "AAT"

6.39.2.2 #define __APP_RAM__ "APP_RAM"

6.39.2.3 #define __attribute__(...)

6.39.2.4 #define __BAT__ "BAT"

6.39.2.5 #define __BAT_INIT__ "BAT"

6.39.2.6 #define __BSS__ ".bss"

6.39.2.7 #define __CONST__ ".rodata"

6.39.2.8 #define __CSTACK__ "CSTACK"

6.39.2.9 #define __DATA__ ".data"

6.39.2.10 #define __DATA_INIT__ ".data_init"

6.39.2.11 #define __DEBUG_CHANNEL__ "DEBUG_CHANNEL"

6.39.2.12 #define __DLIB_PERTHREAD_INIT__ "__DLIB_PERTHREAD_init"

6.39.2.13 #define __DLIB_PERTHREAD_INITIALIZED_DATA__ "DLIB_PERTHREAD_INITIALIZED_DATA"

6.39.2.14 #define __DLIB_PERTHREAD_ZERO_DATA__ "DLIB_PERTHREAD_ZERO_DATA"

6.39.2.15 #define __EMHEAP__ "EMHEAP"

6.39.2.16 #define __EMHEAP_OVERLAY__ "EMHEAP_overlay"

6.39.2.17 #define __FAT__ "FAT"

6.39.2.18 #define __GUARD_REGION__ "GUARD_REGION"

6.39.2.19 #define __INTERNAL_STORAGE__ "INTERNAL_STORAGE"

6.39.2.20 #define __INTVEC__ ".intvec"

6.39.2.21 #define __NO_INIT__ ".noinit"

6.39.2.22 #define __NVM__ "NVM"

6.39.2.23 #define __PSSTORE__ "PSSTORE"

6.39.2.24 #define __RAT__ "RAT"

6.39.2.25 #define __RESETINFO__ "RESETINFO"

6.39.2.26 #define __SIMEE__ "SIMEE"

6.39.2.27 #define __SOURCEFILE__ __FILE__

6.39.2.28 #define __TEXT__ ".text"

6.39.2.29 #define __TEXTRW__ ".textrw"

6.39.2.30 #define __TEXTRW_INIT__ ".textrw_init"

6.39.2.31 #define __UNRETAINED_RAM__ "UNRETAINED_RAM"

6.39.2.32 #define _AAT_SEGMENT_BEGIN __segment_begin(__AAT__)

6.39.2.33 #define _AAT_SEGMENT_END __segment_end(__AAT__)

6.39.2.34 #define _AAT_SEGMENT_SIZE __segment_size(__AAT__)

6.39.2.35 #define _APP_RAM_SEGMENT_BEGIN __segment_begin(__APP_RAM__)

6.39.2.36 #define _APP_RAM_SEGMENT_END __segment_end(__APP_RAM__)

6.39.2.37 #define _APP_RAM_SEGMENT_SIZE __segment_size(__APP_RAM__)

6.39.2.38 #define _BAT_INIT_SEGMENT_BEGIN __segment_begin(__BAT_INIT__)

6.39.2.39 #define _BAT_INIT_SEGMENT_END __segment_end(__BAT_INIT__)

6.39.2.40 #define _BAT_INIT_SEGMENT_SIZE __segment_size(__BAT_INIT__)

6.39.2.41 #define _BAT_SEGMENT_BEGIN __segment_begin(__BAT__)

6.39.2.42 #define _BAT_SEGMENT_END __segment_end(__BAT__)

6.39.2.43 #define _BAT_SEGMENT_SIZE __segment_size(__BAT__)

6.39.2.44 #define _BSS_SEGMENT_BEGIN __segment_begin(__BSS__)

6.39.2.45 #define _BSS_SEGMENT_END __segment_end(__BSS__)

6.39.2.46 #define _BSS_SEGMENT_SIZE __segment_size(__BSS__)

6.39.2.47 #define _CONST_SEGMENT_BEGIN __segment_begin(__CONST__)

6.39.2.48 #define _CONST_SEGMENT_END __segment_end(__CONST__)

6.39.2.49 #define _CONST_SEGMENT_SIZE __segment_size(__CONST__)

6.39.2.50 #define _CSTACK_SEGMENT_BEGIN __segment_begin(__CSTACK__)

6.39.2.51 #define _CSTACK_SEGMENT_END __segment_end(__CSTACK__)

6.39.2.52 #define _CSTACK_SEGMENT_SIZE __segment_size(__CSTACK__)

6.39.2.53 #define _DATA_INIT_SEGMENT_BEGIN __segment_begin(__DATA_INIT__)

6.39.2.54 #define _DATA_INIT_SEGMENT_END __segment_end(__DATA_INIT__)

6.39.2.55 #define _DATA_INIT_SEGMENT_SIZE __segment_size(__DATA_INIT__)

6.39.2.56 #define _DATA_SEGMENT_BEGIN __segment_begin(__DATA__)

6.39.2.57 #define _DATA_SEGMENT_END __segment_end(__DATA__)

6.39.2.58 #define _DATA_SEGMENT_SIZE __segment_size(__DATA__)

6.39.2.59 #define _DEBUG_CHANNEL_SEGMENT_BEGIN __segment_begin(__DEBUG_CHANNEL__)

6.39.2.60 #define _DEBUG_CHANNEL_SEGMENT_END __segment_end(__DEBUG_CHANNEL__)

6.39.2.61 #define _DEBUG_CHANNEL_SEGMENT_SIZE __segment_size(__DEBUG_CHANNEL__)

6.39.2.62 #define _DLIB_PERTHREAD_INIT_SEGMENT_BEGIN __segment_begin(__DLIB_PERTHREAD_INIT__)

6.39.2.63 #define _DLIB_PERTHREAD_INIT_SEGMENT_END __segment_end(__DLIB_PERTHREAD_INIT__)

6.39.2.64 #define _DLIB_PERTHREAD_INIT_SEGMENT_SIZE __segment_size(__DLIB_PERTHREAD_INIT__)

6.39.2.65 #define _DLIB_PERTHREAD_INITIALIZED_DATA_SEGMENT_BEGIN __segment_begin(__DLIB_PERTHREAD_I↩ NITIALIZED_DATA__)

6.39.2.66 #define _DLIB_PERTHREAD_INITIALIZED_DATA_SEGMENT_END __segment_end(__DLIB_PERTHREAD_INIT↩ IALIZED_DATA__)

6.39.2.67 #define _DLIB_PERTHREAD_INITIALIZED_DATA_SEGMENT_SIZE __segment_size(__DLIB_PERTHREAD_INI↩ TIALIZED_DATA__)

6.39.2.68 #define _DLIB_PERTHREAD_ZERO_DATA_SEGMENT_BEGIN __segment_begin(__DLIB_PERTHREAD_ZERO↩ _DATA__)

6.39.2.69 #define _DLIB_PERTHREAD_ZERO_DATA_SEGMENT_END __segment_end(__DLIB_PERTHREAD_ZERO_D↩ ATA__)

6.39.2.70 #define _DLIB_PERTHREAD_ZERO_DATA_SEGMENT_SIZE __segment_size(__DLIB_PERTHREAD_ZERO_D↩ ATA__)

6.39.2.71 #define _EMHEAP_OVERLAY_SEGMENT_BEGIN __segment_begin(__EMHEAP_OVERLAY__)

6.39.2.72 #define _EMHEAP_OVERLAY_SEGMENT_END __segment_end(__EMHEAP_OVERLAY__)

6.39.2.73 #define _EMHEAP_OVERLAY_SEGMENT_SIZE __segment_size(__EMHEAP_OVERLAY__)

6.39.2.74 #define _EMHEAP_SEGMENT_BEGIN __segment_begin(__EMHEAP__)

6.39.2.75 #define _EMHEAP_SEGMENT_END __segment_end(__EMHEAP__)

6.39.2.76 #define _EMHEAP_SEGMENT_SIZE __segment_size(__EMHEAP__)

6.39.2.77 #define _FAT_SEGMENT_BEGIN __segment_begin(__FAT__)

6.39.2.78 #define _FAT_SEGMENT_END __segment_end(__FAT__)

6.39.2.79 #define _FAT_SEGMENT_SIZE __segment_size(__FAT__)

6.39.2.80 #define _GUARD_REGION_SEGMENT_BEGIN __segment_begin(__GUARD_REGION__)

6.39.2.81 #define _GUARD_REGION_SEGMENT_END __segment_end(__GUARD_REGION__)

6.39.2.82 #define _GUARD_REGION_SEGMENT_SIZE __segment_size(__GUARD_REGION__)

6.39.2.83 #define _HAL_USE_COMMON_DIVMOD_

6.39.2.84 #define _HAL_USE_COMMON_MEMUTILS_

6.39.2.85 #define _HAL_USE_COMMON_PGM_

6.39.2.86 #define _INTERNAL_STORAGE_SEGMENT_BEGIN __segment_begin(__INTERNAL_STORAGE__)

6.39.2.87 #define _INTERNAL_STORAGE_SEGMENT_END __segment_end(__INTERNAL_STORAGE__)

6.39.2.88 #define _INTERNAL_STORAGE_SEGMENT_SIZE __segment_size(__INTERNAL_STORAGE__)

6.39.2.89 #define _INTVEC_SEGMENT_BEGIN __segment_begin(__INTVEC__)

6.39.2.90 #define _INTVEC_SEGMENT_END __segment_end(__INTVEC__)

6.39.2.91 #define _INTVEC_SEGMENT_SIZE __segment_size(__INTVEC__)

6.39.2.92 #define _NO_INIT_SEGMENT_BEGIN __segment_begin(__NO_INIT__)

6.39.2.93 #define _NO_INIT_SEGMENT_END __segment_end(__NO_INIT__)

6.39.2.94 #define _NO_INIT_SEGMENT_SIZE __segment_size(__NO_INIT__)

6.39.2.95 #define _NVM_SEGMENT_BEGIN __segment_begin(__NVM__)

6.39.2.96 #define _NVM_SEGMENT_END __segment_end(__NVM__)

6.39.2.97 #define _NVM_SEGMENT_SIZE __segment_size(__NVM__)

6.39.2.98 #define _PSSTORE_SEGMENT_BEGIN __segment_begin(__PSSTORE__)

6.39.2.99 #define _PSSTORE_SEGMENT_END __segment_end(__PSSTORE__)

6.39.2.100 #define _PSSTORE_SEGMENT_SIZE __segment_size(__PSSTORE__)

6.39.2.101 #define _RAT_SEGMENT_BEGIN __segment_begin(__RAT__)

6.39.2.102 #define _RAT_SEGMENT_END __segment_end(__RAT__)

6.39.2.103 #define _RAT_SEGMENT_SIZE __segment_size(__RAT__)

6.39.2.104 #define _RESETINFO_SEGMENT_BEGIN __segment_begin(__RESETINFO__)

6.39.2.105 #define _RESETINFO_SEGMENT_END __segment_end(__RESETINFO__)

6.39.2.106 #define _RESETINFO_SEGMENT_SIZE __segment_size(__RESETINFO__)

6.39.2.107 #define _SIMEE_SEGMENT_BEGIN __segment_begin(__SIMEE__)

6.39.2.108 #define _SIMEE_SEGMENT_END __segment_end(__SIMEE__)

6.39.2.109 #define _SIMEE_SEGMENT_SIZE __segment_size(__SIMEE__)

6.39.2.110 #define _TEXT_SEGMENT_BEGIN __segment_begin(__TEXT__)

6.39.2.111 #define _TEXT_SEGMENT_END __segment_end(__TEXT__)

6.39.2.112 #define _TEXT_SEGMENT_SIZE __segment_size(__TEXT__)

6.39.2.113 #define _TEXTRW_INIT_SEGMENT_BEGIN __segment_begin(__TEXTRW_INIT__)

6.39.2.114 #define _TEXTRW_INIT_SEGMENT_END __segment_end(__TEXTRW_INIT__)

6.39.2.115 #define _TEXTRW_INIT_SEGMENT_SIZE __segment_size(__TEXTRW_INIT__)

6.39.2.116 #define _TEXTRW_SEGMENT_BEGIN __segment_begin(__TEXTRW__)

6.39.2.117 #define _TEXTRW_SEGMENT_END __segment_end(__TEXTRW__)

6.39.2.118 #define _TEXTRW_SEGMENT_SIZE __segment_size(__TEXTRW__)

6.39.2.119 #define _UNRETAINED_RAM_SEGMENT_BEGIN __segment_begin(__UNRETAINED_RAM__)

6.39.2.120 #define _UNRETAINED_RAM_SEGMENT_END __segment_end(__UNRETAINED_RAM__)

6.39.2.121 #define _UNRETAINED_RAM_SEGMENT_SIZE __segment_size(__UNRETAINED_RAM__)

6.39.2.122 #define ALIGNMENT(*X*) _Pragma(STRINGIZE(data_alignment = X))

6.39.2.123 #define assert(*condition*)

Referenced by halSimEepromCallback(), USBD_AbortTransfer(), USBD_EpIsBusy(), USBD_Init(), USBD_Read(), USBD_StallEp(), USBD_UnStallEp(), and USBD_Write().

6.39.2.124 #define BIGENDIAN_CPU false

6.39.2.125 #define EEPROM errorerror

6.39.2.126 #define HAL_HAS_INT64

6.39.2.127 #define halResetWatchdog( ) halInternalResetWatchDog()

6.39.2.128 #define MAIN_FUNCTION_ARGUMENTS

6.39.2.129 #define MAIN_FUNCTION_PARAMETERS void

6.39.2.130 #define NO_INIT(*__symbol*) __no_init __symbol

6.39.2.131 #define NO_OPERATION( ) __no_operation()

6.39.2.132 #define NO_STRIPPING __root

6.39.2.133 #define NTOHL(*val*) (__REV(val))

6.39.2.134 #define NTOHS(*val*) (__REV16(val))

6.39.2.135 #define PLATCOMMONOKTOINCLUDE

6.39.2.136 #define RAMFUNC __ramfunc

6.39.2.137 #define SET_CMSIS_REG_FIELD(*reg, field, value*)

Value:

```
do {                                                                \
    reg = ((reg & (~_##field##_MASK))                               \
           | (uint32_t) (((uint32_t) value) << _##field##_SHIFT)); \
  } while (0)
```

6.39.2.138 #define SET_REG_FIELD(*reg, field, value*)

Value:

```
do {                                                    \
    reg = ((reg & (~field##_MASK))                      \
           | (uint32_t) (((uint32_t) value) << field##_BIT)); \
  } while (0)
```

6.39.2.139 #define SIGNED_ENUM

6.39.2.140 #define simulatedSerialTimePasses( )

6.39.2.141 #define simulatedTimePasses( )

6.39.2.142 #define simulatedTimePassesMs(*x*)

6.39.2.143 #define STACK_FILL_VALUE 0xCDCDCDCDU

6.39.2.144 #define STATIC_ASSERT(*__condition, __errorstr*) static_assert(__condition, __errorstr)

6.39.2.145 #define STRINGIZE(*X*) #X

6.39.2.146 #define UNUSED

6.39.2.147 #define VAR_AT_SEGMENT(*__variableDeclaration, __segmentName*) __variableDeclaration @ __segmentName

6.39.2.148 #define WEAK(*__symbol*) __weak __symbol

6.39.3 Typedef Documentation

6.39.3.1 typedef bool **boolean**

6.39.3.2 typedef signed short **int16s**

6.39.3.3 typedef unsigned short **int16u**

6.39.3.4 typedef signed int **int32s**

6.39.3.5 typedef unsigned int **int32u**

6.39.3.6 typedef signed long long **int64s**

6.39.3.7 typedef unsigned long long **int64u**

6.39.3.8 typedef signed char **int8s**

6.39.3.9 typedef unsigned char **int8u**

6.39.3.10 typedef unsigned int **PointerType**

6.39.4 Function Documentation

6.39.4.1 void _executeBarrierInstructions (void)

6.39.4.2 int abs (int *i*)

Parameters

l	An integer.

Returns

A nonnegative integer.

6.39.4.3 void halInternalAssertFailed (const char * *filename,* int *linenumber*)

6.39.4.4 void halInternalResetWatchDog (void)

6.40 Common PLATFORM_HEADER Configuration

Compiler and Platform specific definitions and typedefs common to all platforms.

Macros

- #define MEMSET(d, v, l) memset(d, v, l)

  Friendly convenience macro pointing to the C Stdlib functions.
- #define MEMCOPY(d, s, l) memcpy(d, s, l)
- #define MEMMOVE(d, s, l) memmove(d, s, l)
- #define MEMPGMCOPY(d, s, l) memcpy(d, s, l)
- #define MEMCOMPARE(s0, s1, l) memcmp(s0, s1, l)
- #define MEMPGMCOMPARE(s0, s1, l) memcmp(s0, s1, l)

Generic Types

- #define TRUE 1

  An alias for one, used for clarity.
- #define FALSE 0

  An alias for zero, used for clarity.
- #define NULL ((void *)0)

  The null pointer.

Bit Manipulation Macros

- #define BIT(x) (1U << (x))

  Useful to reference a single bit of a byte.
- #define BIT32(x) (((uint32_t) 1) << (x))

  Useful to reference a single bit of an uint32_t type.
- #define SETBIT(reg, bit) (reg) |= BIT(bit)

  Sets `bit` in the `reg` register or byte.
- #define SETBITS(reg, bits) (reg) |= (bits)

  Sets the bits in the `reg` register or the byte as specified in the bitmask `bits`.
- #define CLEARBIT(reg, bit) (reg) &= ~(BIT(bit))

  Clears a bit in the `reg` register or byte.
- #define CLEARBITS(reg, bits) (reg) &= ~(bits)

  Clears the bits in the `reg` register or byte as specified in the bitmask `bits`.
- #define READBIT(reg, bit) ((reg) & (BIT(bit)))

  Returns the value of `bit` within the register or byte `reg`.
- #define READBITS(reg, bits) ((reg) & (bits))

  Returns the value of the bitmask `bits` within the register or byte `reg`.

Byte Manipulation Macros

- #define LOW_BYTE(n) ((uint8_t)((n) & 0xFF))

  Returns the low byte of the 16-bit value `n` as an `uint8_t`.
- #define HIGH_BYTE(n) ((uint8_t)(LOW_BYTE((n) >> 8)))

  Returns the high byte of the 16-bit value `n` as an `uint8_t`.
- #define HIGH_LOW_TO_INT(high, low)

  Returns the value built from the two `uint8_t` values `high` and `low`.
- #define INT8U_TO_INT32U(byte3, byte2, byte1, byte0)

  Returns the value built from the four `uint8_t` as an `uint32_t`.
- #define BYTE_0(n) ((uint8_t)((n) & 0xFF))

  Returns the low byte of the 32-bit value `n` as an `uint8_t`.
- #define BYTE_1(n) BYTE_0((n) >> 8)

  Returns the second byte of the 32-bit value `n` as an `uint8_t`.
- #define BYTE_2(n) BYTE_0((n) >> 16)

  Returns the third byte of the 32-bit value `n` as an `uint8_t`.
- #define BYTE_3(n) BYTE_0((n) >> 24)

  Returns the high byte of the 32-bit value `n` as an `uint8_t`.
- #define BYTE_4(n) BYTE_0((n) >> 32)

  Returns the fifth byte of the 64-bit value `n` as an `uint8_t`.
- #define BYTE_5(n) BYTE_0((n) >> 40)

  Returns the sixth byte of the 64-bit value `n` as an `uint8_t`.
- #define BYTE_6(n) BYTE_0((n) >> 48)

  Returns the seventh byte of the 64-bit value `n` as an `uint8_t`.
- #define BYTE_7(n) BYTE_0((n) >> 56)

  Returns the high byte of the 64-bit value `n` as an `uint8_t`.
- #define COUNTOF(a) (sizeof(a) / sizeof(a[0]))

  Returns the number of entries in an array.

Time Manipulation Macros

- #define elapsedTimeInt8u(oldTime, newTime) ((uint8_t) ((uint8_t)(newTime) - (uint8_t)(oldTime)))

  Returns the elapsed time between two 8 bit values. Result may not be valid if the time samples differ by more than 127.
- #define elapsedTimeInt16u(oldTime, newTime) ((uint16_t) ((uint16_t)(newTime) - (uint16_t)(oldTime)))

  Returns the elapsed time between two 16 bit values. Result may not be valid if the time samples differ by more than 32767.
- #define elapsedTimeInt32u(oldTime, newTime) ((uint32_t) ((uint32_t)(newTime) - (uint32_t)(oldTime)))

  Returns the elapsed time between two 32 bit values. Result may not be valid if the time samples differ by more than 2147483647.
- #define MAX_INT8U_VALUE (0xFF)

  Returns true if t1 is greater than t2. Can only account for 1 wrap around of the variable before it is wrong.
- #define HALF_MAX_INT8U_VALUE (0x80)

  Returns the elapsed time between two 8 bit values. Result may not be valid if the time samples differ by more than 127.
- #define timeGTorEqualInt8u(t1, t2) (elapsedTimeInt8u(t2, t1) <= (HALF_MAX_INT8U_VALUE))

  Returns the elapsed time between two 8 bit values. Result may not be valid if the time samples differ by more than 127.
- #define MAX_INT16U_VALUE (0xFFFF)

  Returns true if t1 is greater than t2. Can only account for 1 wrap around of the variable before it is wrong.
- #define HALF_MAX_INT16U_VALUE (0x8000)

Returns the elapsed time between two 8 bit values. Result may not be valid if the time samples differ by more than 127.

- #define timeGTorEqualInt16u(t1, t2) (elapsedTimeInt16u(t2, t1) <= (HALF_MAX_INT16U_VALUE))

  Returns the elapsed time between two 8 bit values. Result may not be valid if the time samples differ by more than 127.
- #define MAX_INT32U_VALUE (0xFFFFFFFFUL)

  Returns true if t1 is greater than t2. Can only account for 1 wrap around of the variable before it is wrong.
- #define HALF_MAX_INT32U_VALUE (0x80000000UL)

  Returns the elapsed time between two 8 bit values. Result may not be valid if the time samples differ by more than 127.
- #define timeGTorEqualInt32u(t1, t2) (elapsedTimeInt32u(t2, t1) <= (HALF_MAX_INT32U_VALUE))

  Returns the elapsed time between two 8 bit values. Result may not be valid if the time samples differ by more than 127.

Miscellaneous Macros

- #define UNUSED_VAR(x) (void)(x)
- #define DEBUG_LEVEL BASIC_DEBUG

  Set debug level based on whether DEBUG or DEBUG_STRIPPED are defined.
- #define STATIC_ASSERT(__condition, __errorstr)

  Disable static assertions on compilers that don't support them.

6.40.1 Detailed Description

platform-common.h provides PLATFORM_HEADER defaults and common definitions. This head should never be included directly, it should only be included by the specific PLATFORM_HEADER used by your platform.

See platform-common.h for source code.

6.40.2 Macro Definition Documentation

6.40.2.1 #define BIT(*x*) (1U << (x))

6.40.2.2 #define BIT32(*x*) (((uint32_t) 1) << (x))

6.40.2.3 #define BYTE_0(*n*) ((uint8_t)((n) & 0xFF))

6.40.2.4 #define BYTE_1(*n*) BYTE_0((n) >> 8)

6.40.2.5 #define BYTE_2(*n*) BYTE_0((n) >> 16)

6.40.2.6 #define BYTE_3(*n*) BYTE_0((n) >> 24)

6.40.2.7 #define BYTE_4(*n*) BYTE_0((n) >> 32)

6.40.2.8 #define BYTE_5(*n*) BYTE_0((n) >> 40)

6.40.2.9 #define BYTE_6(*n*) BYTE_0((n) >> 48)

6.40.2.10 #define BYTE_7(*n*) BYTE_0((n) >> 56)

6.40.2.11 #define CLEARBIT(*reg, bit*) (reg) &= ~(BIT(bit))

Note

Assuming `reg` is an IO register, some platforms (such as the AVR) can implement this in a single atomic operation.

6.40.2.12 #define CLEARBITS(*reg, bits*) (reg) &= ~(bits)

Note

This is never a single atomic operation.

6.40.2.13 #define COUNTOF(*a*) (sizeof(a) / sizeof(a[0]))

6.40.2.14 #define DEBUG_LEVEL BASIC_DEBUG

6.40.2.15 #define elapsedTimeInt16u(*oldTime, newTime*) ((uint16_t) ((uint16_t)(newTime) - (uint16_t)(oldTime)))

Referenced by USBD_RemoteWakeup().

6.40.2.16 #define elapsedTimeInt32u(*oldTime, newTime*) ((uint32_t) ((uint32_t)(newTime) - (uint32_t)(oldTime)))

6.40.2.17 #define elapsedTimeInt8u(*oldTime, newTime*) ((uint8_t) ((uint8_t)(newTime) - (uint8_t)(oldTime)))

6.40.2.18 #define FALSE 0

6.40.2.19 #define HALF_MAX_INT16U_VALUE (0x8000)

6.40.2.20 #define HALF_MAX_INT32U_VALUE (0x80000000UL)

6.40.2.21 #define HALF_MAX_INT8U_VALUE (0x80)

6.40.2.22 #define HIGH_BYTE(*n*) ((uint8_t)(LOW_BYTE((n) >> 8)))

6.40.2.23 #define HIGH_LOW_TO_INT(*high, low*)

Value:

```
(       \
    ((uint16_t) (((uint16_t) (high)) << 8)) \
    + ((uint16_t) ((low) & 0xFF))           \
    )
```

6.40.2.24 #define INT8U_TO_INT32U(*byte3, byte2, byte1, byte0*)

Value:

```
( \
    (((uint32_t) (byte3)) << 24)                    \
    + (((uint32_t) (byte2)) << 16)                  \
    + (((uint32_t) (byte1)) << 8)                   \
    + ((uint32_t) ((byte0) & 0xFF))                 \
    )
```

6.40.2.25 #define LOW_BYTE(*n*) ((uint8_t)((n) & 0xFF))

6.40.2.26 #define MAX_INT16U_VALUE (0xFFFF)

6.40.2.27 #define MAX_INT32U_VALUE (0xFFFFFFFFUL)

6.40.2.28 #define MAX_INT8U_VALUE (0xFF)

6.40.2.29 #define MEMCOMPARE(*s0, s1, l*) memcmp(s0, s1, l)

6.40.2.30 #define MEMCOPY(*d, s, l*) memcpy(d, s, l)

6.40.2.31 #define MEMMOVE(*d, s, l*) memmove(d, s, l)

6.40.2.32 #define MEMPGMCOMPARE(*s0, s1, l*) memcmp(s0, s1, l)

6.40.2.33 #define MEMPGMCOPY(*d, s, l*) memcpy(d, s, l)

6.40.2.34 #define MEMSET(*d, v, l*) memset(d, v, l)

Referenced by USBD_Init(), and USBD_RemoteWakeup().

6.40.2.35 #define NULL ((void *)0)

Referenced by emberAllocateMemoryForPacketHandler(), USBD_AbortTransfer(), USBD_EpIsBusy(), USBD_Init(), USBD_Read(), USBD_RemoteWakeup(), USBD_StallEp(), USBD_UnStallEp(), and USBD_Write().

6.40.2.36 #define READBIT(*reg, bit*) ((reg) & (BIT(bit)))

6.40.2.37 #define READBITS(*reg, bits*) ((reg) & (bits))

6.40.2.38 #define SETBIT(*reg, bit*) (reg) |= BIT(bit)

Note

Assuming `reg` is an IO register, some platforms (such as the AVR) can implement this in a single atomic operation.

6.40.2.39 #define SETBITS(*reg, bits*) (reg) |= (bits)

Note

This is never a single atomic operation.

6.40.2.40 #define STATIC_ASSERT(*__condition, __errorstr*)

6.40.2.41 #define timeGTorEqualInt16u(*t1, t2*) (elapsedTimeInt16u(t2, t1) <= (HALF_MAX_INT16U_VALUE))

6.40.2.42 #define timeGTorEqualInt32u(*t1, t2*) (elapsedTimeInt32u(t2, t1) <= (HALF_MAX_INT32U_VALUE))

6.40.2.43 #define timeGTorEqualInt8u(*t1, t2*) (elapsedTimeInt8u(t2, t1) <= (HALF_MAX_INT8U_VALUE))

6.40.2.44 #define TRUE 1

6.40.2.45 #define UNUSED_VAR(*x*) (void)(x)

Useful macro for avoiding compiler warnings related to unused function arguments or unused variables.

6.41 NVIC Configuration

Nested Vectored Interrupt Controller configuration header.

This header defines the functions called by all of the NVIC exceptions/ interrupts. The following are the nine peripheral ISRs available for modification. To use one of these ISRs, it must be instantiated somewhere in the HAL. Each ISR may only be instantiated once. It is not necessary to instantiate all or any of these ISRs (a stub will be automatically generated if an ISR is not instantiated).

- `1 void halTimer1Isr(void);`
- `1 void halTimer2Isr(void);`
- `1 void halSc1Isr(void);`
- `1 void halSc1Isr(void);`
- `1 void halAdcIsr(void);`
- `1 void halIrqAIsr(void);`
- `1 void halIrqBIsr(void);`
- `1 void halIrqCIsr(void);`
- `1 void halIrqDIsr(void);`

Note

This file should **not** be modified.

6.42 Reset Cause Type Definitions

Definitions for all the reset cause types.

Reset cause types are built from a base definition and an extended. definition. The base definitions allow working with entire categories of resets while the extended definitions allow drilling down to very specific causes. The macros for the base and extended definitions are combined for use in the code and equated to their combined numberical equivalents. In addition, exach base and extended definition is given a corresponding 3 letter ASCII string to facilitate printing. The ASCII strings are best use with halGetExtendedResetString.

For example:

```
1 RESET_BASE_DEF(EXTERNAL,            0x03U,     "EXT")
2   RESET_EXT_DEF(EXTERNAL, UNKNOWN,    0x00U,     "UNK")
3   RESET_EXT_DEF(EXTERNAL, PIN,        0x01U,     "PIN")
```

results in enums which includes the entries:

```
1 RESET_EXTERNAL = 0x03U
2 RESET_EXTERNAL_PIN = 0x0301U
```

For a complete listing of all reset base and extended definitions, see reset-def.h for source code.

6.43 HAL Utilities

Modules

- Crash and Watchdog Diagnostics

  Crash and watchdog diagnostic functions.
- Cyclic Redundancy Code (CRC)

  Functions that provide access to cyclic redundancy code (CRC) calculation. See crc.h for source code.
- Random Number Generation

  Functions that provide access to random numbers.
- Network to Host Byte Order Conversion

6.43.1 Detailed Description

6.44 Crash and Watchdog Diagnostics

Crash and watchdog diagnostic functions.

Macros

- #define halResetWasCrash() (((1 << halGetResetInfo()) & RESET_CRASH_REASON_MASK) != 0U)
  Macro evaluating to true if the last reset was a crash, false otherwise.

Functions

- uint32_t halGetMainStackBytesUsed (void)
  Returns the number of bytes used in the main stack.
- void halPrintCrashSummary (uint8_t port)
  Print a summary of crash details.
- void halPrintCrashDetails (uint8_t port)
  Print the complete, decoded crash details.
- void halPrintCrashData (uint8_t port)
  Print the complete crash data.
- const HalAssertInfoType * halGetAssertInfo (void)
  If last reset was from an assert, return saved assert information.

6.44.1 Detailed Description

See diagnostic.h for source code.

6.44.2 Macro Definition Documentation

6.44.2.1 #define halResetWasCrash( ) (((1 << halGetResetInfo()) & RESET_CRASH_REASON_MASK) != 0U)

6.44.3 Function Documentation

6.44.3.1 const HalAssertInfoType* halGetAssertInfo (void)

Returns

Pointer to struct containing assert filename and line.

6.44.3.2 uint32_t halGetMainStackBytesUsed (void)

Returns

The number of bytes used in the main stack.

6.44.3.3 void halPrintCrashData (uint8_t *port*)

Parameters

port	Serial port number (0 or 1).

6.44.3.4 void halPrintCrashDetails (uint8_t *port*)

Parameters

port	Serial port number (0 or 1).

6.44.3.5 void halPrintCrashSummary (uint8_t *port*)

Parameters

port	Serial port number (0 or 1).

6.45 Cyclic Redundancy Code (CRC)

Functions that provide access to cyclic redundancy code (CRC) calculation. See crc.h for source code.

Macros

- #define INITIAL_CRC 0xFFFFFFFFL
- #define CRC32_START INITIAL_CRC
- #define CRC32_END 0xDEBB20E3L

Functions

- uint16_t halCommonCrc16 (uint8_t newByte, uint16_t prevResult)

  Calculates 16-bit cyclic redundancy code (CITT CRC 16).
- uint32_t halCommonCrc32 (uint8_t newByte, uint32_t prevResult)

  Calculates 32-bit cyclic redundancy code.

6.45.1 Detailed Description

6.45.2 Macro Definition Documentation

6.45.2.1 #define CRC32_END 0xDEBB20E3L

6.45.2.2 #define CRC32_START INITIAL_CRC

6.45.2.3 #define INITIAL_CRC 0xFFFFFFFFL

6.45.3 Function Documentation

6.45.3.1 uint16_t halCommonCrc16 (uint8_t *newByte,* uint16_t *prevResult*)

Applies the standard CITT CRC 16 polynomial to a single byte. It should support being called first with an initial value, then repeatedly until all data is processed.

Parameters

newByte	The new byte to be run through CRC.
prevResult	The previous CRC result.

Returns

The new CRC result.

6.45.3.2 uint32_t halCommonCrc32 (uint8_t *newByte,* uint32_t *prevResult*)

Note

On some radios or micros, the CRC for error detection on packet data is calculated in hardware.

Applies a CRC32 polynomial to a single byte. It should support being called first with an initial value, then repeatedly until all data is processed.

Parameters

newByte	The new byte to be run through CRC.
prevResult	The previous CRC result.

Returns

The new CRC result.

6.46 Random Number Generation

Functions that provide access to random numbers.

Functions

- void halStackSeedRandom (uint32_t seed)

  Seeds the halCommonGetRandom() pseudorandom number generator.
- uint16_t halCommonGetRandom (void)

  Runs a standard LFSR to generate pseudorandom numbers.

6.46.1 Detailed Description

These functions may be hardware accelerated, though often are not.

See random.h for source code.

6.46.2 Function Documentation

6.46.2.1 uint16_t halCommonGetRandom (void)

Called by the MAC in the stack to choose random backoff slots.

Complicated implementations may improve the MAC's ability to avoid collisions in large networks, but it is **critical** to implement this function to return quickly.

6.46.2.2 void halStackSeedRandom (uint32_t *seed*)

Called by the stack during initialization with a seed from the radio.

Parameters

seed	A seed for the pseudorandom number generator.

6.47 Network to Host Byte Order Conversion

Macros

- #define HTONL NTOHL
- #define HTONS NTOHS

Functions

- uint16_t NTOHS (uint16_t val)

  Converts a short (16-bit) value from network to host byte order.
- uint32_t NTOHL (uint32_t val)

  Converts a long (32-bit) value from network to host byte order.
- uint32_t SwapEndiannessInt32u (uint32_t val)

6.47.1 Detailed Description

Functions that provide conversions from network to host byte order. Network byte order is big endian, so these APIs are only necessary on platforms which have a natural little endian byte order. On big-endian platforms, the APIs are macro'd away to nothing. See endian.h for source code.

6.47.2 Macro Definition Documentation

6.47.2.1 #define HTONL NTOHL

6.47.2.2 #define HTONS NTOHS

6.47.3 Function Documentation

6.47.3.1 uint32_t NTOHL (uint32_t *val*)

6.47.3.2 uint16_t NTOHS (uint16_t *val*)

6.47.3.3 uint32_t SwapEndiannessInt32u (uint32_t *val*)

6.48 Bootloader Interfaces

Modules

- Common

  Common bootloader interface defines and functions.
- Standalone

  Definition of the standalone bootloader interface.
- Application

  Defiunition of the application bootloader interface.

6.48.1 Detailed Description

6.49 Common

Common bootloader interface defines and functions.

Macros

- #define BOOTLOADER_BASE_TYPE(extendedType) ((uint8_t)(((extendedType) >> 8U) & 0xFFU))

  Macro returning the base type of a bootloader when given an extended type.
- #define BOOTLOADER_MAKE_EXTENDED_TYPE(baseType, extendedSpecifier) ((uint16_t)(((uint16_t)baseType) << 8U) | (((uint16_t)extendedSpecifier) & 0xFFU))

  Macro returning the extended type of a bootloader when given a base type and extendedSpecifier.
- #define BL_EXT_TYPE_NULL ((BL_TYPE_NULL << 8U) | 0x00U)

  Macro defining the extended NULL bootloader type.
- #define BL_EXT_TYPE_STANDALONE_UNKNOWN ((BL_TYPE_STANDALONE << 8U) | 0x00U)

  Macro defining the extended standalone unknown bootloader type.
- #define BL_EXT_TYPE_SERIAL_UART ((BL_TYPE_STANDALONE << 8U) | 0x01U)

  Macro defining the extended standalone UART bootloader type.
- #define BL_EXT_TYPE_SERIAL_UART_OTA ((BL_TYPE_STANDALONE << 8U) | 0x03U)

  Macro defining the extended standalone OTA and UART bootloader type.
- #define BL_EXT_TYPE_EZSP_SPI ((BL_TYPE_STANDALONE << 8U) | 0x04U)
- #define BL_EXT_TYPE_EZSP_SPI_OTA ((BL_TYPE_STANDALONE << 8U) | 0x06U)
- #define BL_EXT_TYPE_SERIAL_USB ((BL_TYPE_STANDALONE << 8U) | 0x07U)

  Macro defining the extended standalone USB bootloader type.
- #define BL_EXT_TYPE_SERIAL_USB_OTA ((BL_TYPE_STANDALONE << 8U) | 0x08U)

  Macro defining the extended standalone OTA and USB bootloader type.
- #define BL_EXT_TYPE_APP_UNKNOWN ((BL_TYPE_APPLICATION << 8U) | 0x00U)

  Macro defining the extended application unknown bootloader type.
- #define BL_EXT_TYPE_APP_SPI ((BL_TYPE_APPLICATION << 8U) | 0x01U)

  Macro defining the extended application SPI bootloader type.
- #define BL_EXT_TYPE_APP_I2C ((BL_TYPE_APPLICATION << 8U) | 0x02U)

  Macro defining the extended application I2C bootloader type.
- #define BL_EXT_TYPE_APP_LOCAL_STORAGE ((BL_TYPE_APPLICATION << 8U) | 0x03U)

  Macro defining a type for the local storage app bootloader.
- #define BOOTLOADER_INVALID_VERSION 0xFFFF

  Define an invalid bootloader version.
- #define CUSTOMER_APPLICATION_VERSION 0

  Macro defining the customer application version stored in the ApplicationProperties_t struct.
- #define CUSTOMER_APPLICATION_CAPABILITIES 0

  Macro defining the customer application capabilities stored in the ApplicationProperties_t struct.
- #define CUSTOMER_APPLICATION_PRODUCT_ID { 0 }

  Macro defining the customer application product ID stored in the ApplicationProperties_t struct.
- #define MPSI_PLUGIN_SUPPORT 0

  Macro defining the support for the MPSI protocol stored in the capabilities field of the ApplicationProperties_t struct.
- #define APPLICATION_PROPERTIES_CAPABILITIES_MPSI_SUPPORT_BIT 31

  Macro defining the bit position that corresponds to MPSI support in the capabilities field of the ApplicationProperties_t struct.
- #define APPLICATION_PROPERTIES_CAPABILITIES

  Macro defining the capabilities that this application has. This value is set in the capabilities field of the ApplicationProperties_t struct.

Functions

- BlBaseType halBootloaderGetType (void)

  Returns the bootloader base type the application was built for.
- BlExtendedType halBootloaderGetInstalledType (void)

  Returns the extended bootloader type of the bootloader that is present on the chip.
- uint16_t halGetBootloaderVersion (void)

  Returns the version of the installed bootloader, regardless of its type.
- void halGetExtendedBootloaderVersion (uint32_t *emberVersion, uint32_t *customerVersion)

  Return extended bootloader version information, if supported. This API is not supported for EM2XX chips and only returns extra information on bootloaders built on or after the 4.7 release.

Bootloader Numerical Definitions

These are numerical definitions for the possible bootloader types and a typedef of the bootloader base type.

- #define BL_TYPE_NULL (0)

  Numerical definition for a bootloader type.
- #define BL_TYPE_STANDALONE (1)

  Numerical definition for a bootloader type.
- #define BL_TYPE_APPLICATION (2)

  Numerical definition for a bootloader type.
- #define BL_TYPE_BOOTLOADER (3)

  Numerical definition for a bootloader type.
- #define BL_TYPE_SMALL_BOOTLOADER (4)

  Numerical definition for a bootloader type.

Bootloader type definitions

These are the type definitions for the bootloader.

- typedef uint8_t BlBaseType

  Define the bootloader base type.
- typedef uint16_t BlExtendedType

  Define the bootloader extended type.

6.49.1 Detailed Description

See bootloader-interface.h for source code.

6.49.2 Macro Definition Documentation

6.49.2.1 #define APPLICATION_PROPERTIES_CAPABILITIES

Value:

```
(MPSI_PLUGIN_SUPPORT <<
     APPLICATION_PROPERTIES_CAPABILITIES_MPSI_SUPPORT_BIT) \
  | (CUSTOMER_APPLICATION_CAPABILITIES & 0x7FFFFFFF)
```

6.49.2.2 #define APPLICATION_PROPERTIES_CAPABILITIES_MPSI_SUPPORT_BIT 31

6.49.2.3 #define BL_EXT_TYPE_APP_I2C ((BL_TYPE_APPLICATION << 8U) | 0x02U)

6.49.2.4 #define BL_EXT_TYPE_APP_LOCAL_STORAGE ((BL_TYPE_APPLICATION << 8U) | 0x03U)

6.49.2.5 #define BL_EXT_TYPE_APP_SPI ((BL_TYPE_APPLICATION << 8U) | 0x01U)

6.49.2.6 #define BL_EXT_TYPE_APP_UNKNOWN ((BL_TYPE_APPLICATION << 8U) | 0x00U)

6.49.2.7 #define BL_EXT_TYPE_EZSP_SPI ((BL_TYPE_STANDALONE << 8U) | 0x04U)

6.49.2.8 #define BL_EXT_TYPE_EZSP_SPI_OTA ((BL_TYPE_STANDALONE << 8U) | 0x06U)

6.49.2.9 #define BL_EXT_TYPE_NULL ((BL_TYPE_NULL << 8U) | 0x00U)

6.49.2.10 #define BL_EXT_TYPE_SERIAL_UART ((BL_TYPE_STANDALONE << 8U) | 0x01U)

6.49.2.11 #define BL_EXT_TYPE_SERIAL_UART_OTA ((BL_TYPE_STANDALONE << 8U) | 0x03U)

6.49.2.12 #define BL_EXT_TYPE_SERIAL_USB ((BL_TYPE_STANDALONE << 8U) | 0x07U)

6.49.2.13 #define BL_EXT_TYPE_SERIAL_USB_OTA ((BL_TYPE_STANDALONE << 8U) | 0x08U)

6.49.2.14 #define BL_EXT_TYPE_STANDALONE_UNKNOWN ((BL_TYPE_STANDALONE << 8U) | 0x00U)

6.49.2.15 #define BL_TYPE_APPLICATION (2)

6.49.2.16 #define BL_TYPE_BOOTLOADER (3)

6.49.2.17 #define BL_TYPE_NULL (0)

6.49.2.18 #define BL_TYPE_SMALL_BOOTLOADER (4)

6.49.2.19 #define BL_TYPE_STANDALONE (1)

6.49.2.20 #define BOOTLOADER_BASE_TYPE(*extendedType*) ((uint8_t)(((extendedType) >> 8U) & 0xFFU))

6.49.2.21 #define BOOTLOADER_INVALID_VERSION 0xFFFF

6.49.2.22 #define BOOTLOADER_MAKE_EXTENDED_TYPE(*baseType, extendedSpecifier*) ((uint16_t)(((uint16_t)baseType) << 8U) | (((uint16_t)extendedSpecifier) & 0xFFU))

6.49.2.23 #define CUSTOMER_APPLICATION_CAPABILITIES 0

Note

The capabilities field in the ApplicationProperties_t struct is shared with other values.

6.49.2.24 #define CUSTOMER_APPLICATION_PRODUCT_ID { 0 }

6.49.2.25 #define CUSTOMER_APPLICATION_VERSION 0

6.49.2.26 #define MPSI_PLUGIN_SUPPORT 0

6.49.3 Typedef Documentation

6.49.3.1 typedef uint8_t BlBaseType

6.49.3.2 typedef uint16_t BlExtendedType

6.49.4 Function Documentation

6.49.4.1 BlExtendedType halBootloaderGetInstalledType (void)

6.49.4.2 BlBaseType halBootloaderGetType (void)

Returns

BL_TYPE_NULL, BL_TYPE_STANDALONE, or BL_TYPE_APPLICATION

6.49.4.3 uint16_t halGetBootloaderVersion (void)

Returns

Version if bootloader installed, or BOOTLOADER_INVALID_VERSION. A returned version of 0x1234U would indicate version 1.2 build 34

6.49.4.4 void halGetExtendedBootloaderVersion (uint32_t * *emberVersion*, uint32_t * *customerVersion*)

Parameters

emberVersion	If specified, we will return the full 32bit ember version for this bootloader. Format is major, minor, patch, doc (4bit nibbles) followed by a 16bit build number.
customerVersion	This will return the 32bit value specified in CUSTOMER_BOOTLOADER_VERSION at build time.

6.50 Standalone

Definition of the standalone bootloader interface.

Macros

- #define NO_BOOTLOADER_MODE 0xFF

  Define a numerical value for NO BOOTLOADER mode. In other words, the bootloader should not be run.
- #define STANDALONE_BOOTLOADER_NORMAL_MODE 1

  Define a numerical value for the normal bootloader mode.
- #define STANDALONE_BOOTLOADER_RECOVERY_MODE 0

  Define a numerical value for the recovery bootloader mode.

Functions

- uint16_t halGetStandaloneBootloaderVersion (void)

  Detects if the standalone bootloader is installed, and if so returns the installed version.
- EmberStatus halLaunchStandaloneBootloader (uint8_t mode)

  Quits the current application and launches the standalone bootloader (if installed). The function returns an error if the standalone bootloader is not present.

6.50.1 Detailed Description

Some functions in this file return an EmberStatus value. See error-def.h for definitions of all EmberStatus return values.

See bootloader-interface-standalone.h for source code.

6.50.2 Macro Definition Documentation

6.50.2.1 #define NO_BOOTLOADER_MODE 0xFF

6.50.2.2 #define STANDALONE_BOOTLOADER_NORMAL_MODE 1

6.50.2.3 #define STANDALONE_BOOTLOADER_RECOVERY_MODE 0

6.50.3 Function Documentation

6.50.3.1 uint16_t halGetStandaloneBootloaderVersion (void)

A returned version of 0x1234 would indicate version 1.2 build 34

Returns

BOOTLOADER_INVALID_VERSION if the standalone bootloader is not present, or the version of the installed standalone bootloader.

6.50.3.2 EmberStatus halLaunchStandaloneBootloader (uint8_t *mode*)

Parameters

mode	Controls the mode in which the standalone bootloader will run. See the bootloader Application 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 via serial xmodem.

Returns

An EmberStatus error if the standalone bootloader is not present, or EMBER_SUCCESS.

6.51 Application

Defiunition of the application bootloader interface.

Macros

- #define BOOTLOADER_SEGMENT_SIZE_LOG2 6

  This is the working unit of data for the app bootloader. We want it as big as possible, but it must be a factor of the NVM page size and fit into a single Zigbee packet. We choose 2^6 = 64 bytes.
- #define BOOTLOADER_SEGMENT_SIZE (1 << BOOTLOADER_SEGMENT_SIZE_LOG2)

  This is the working unit of data for the app bootloader. We want it as big as possible, but it must be a factor of the NVM page size and fit into a single Zigbee packet. We choose 2^6 = 64 bytes.
- #define BL_IMAGE_IS_VALID_CONTINUE ((uint16_t)0xFFFF)

  Define a numerical value for checking image validity when calling the image interface functions.

Functions

- uint8_t halAppBootloaderInit (void)

  Call this function as part of your application initialization to ensure the storage mechanism is ready to use. Note: some earlier drivers may assert instead of returning an error if initialization fails.
- const HalEepromInformationType * halAppBootloaderInfo (void)

  Call this function to get information about the attached storage device and its capabilities.
- void halAppBootloaderShutdown (void)

  Call this function when you are done accessing the storage mechanism to ensure that it is returned to its lowest power state.
- void halAppBootloaderImageIsValidReset (void)

  Call this function once before checking for a valid image to reset the call flag.
- uint16_t halAppBootloaderImageIsValid (void)

  Reads the app image out of storage, calculates the total file CRC to verify the image is intact.
- EmberStatus halAppBootloaderInstallNewImage (void)

  Invokes the bootloader to install the application in storage. This function resets the device to start the bootloader code and does not return!
- uint8_t halAppBootloaderWriteRawStorage (uint32_t address, const uint8_t *data, uint16_t len)

  Writes data to the specified raw storage address and length without being restricted to any page size Note: Not all storage implementations support accesses that are not page aligned, refer to the HalEepromInformationType structure for more information. Note: Some storage devices require contents to be erased before new data can be written, and will return an EEPROM_ERR_ERASE_REQUIRED error if write is called on a location that is not already erased. Refer to the HalEepromInformationType structure to see if the attached storage device requires erasing.
- uint8_t halAppBootloaderReadRawStorage (uint32_t address, uint8_t *data, uint16_t len)

  Reads data from the specified raw storage address and length without being restricted to any page size Note: Not all storage implementations support accesses that are not page aligned, refer to the HalEepromInformationType structure for more information.
- uint8_t halAppBootloaderEraseRawStorage (uint32_t address, uint32_t len)

  Erases the specified region of the storage device. Note: Most devices require the specified region to be page aligned, and will return an error if an unaligned region is specified. Note: Many devices take an extremely long time to perform an erase operation. When erasing a large region, it may be preferable to make multiple calls to this API so that other application functionality can be performed while the erase is in progress. The halAppBootloaderStorageBusy() API may be used to determine when the last erase operation has completed. Erase timing information can be found in the HalEepromInformationType structure.
- bool halAppBootloaderStorageBusy (void)

  Determine if the attached storage device is still busy performing the last operation, such as a write or an erase.
- uint8_t halAppBootloaderReadDownloadSpace (uint16_t pageToBeRead, uint8_t *destRamBuffer)

Converts pageToBeRead to an address and the calls storage read function. Note: This function is deprecated. It has been replaced by halAppBootloaderReadRawStorage()

- uint8_t halAppBootloaderWriteDownloadSpace (uint16_t pageToBeWritten, uint8_t *RamPtr)

  Converts pageToBeWritten to an address and calls the storage write function. Note: This function is deprecated. It has been replaced by halAppBootloaderWriteRawStorage()
- uint8_t halAppBootloaderGetImageData (uint32_t *timestamp, uint8_t *userData)

  Read the application image data from storage.
- uint16_t halAppBootloaderGetVersion (void)

  Returns the application bootloader version.
- uint16_t halAppBootloaderGetRecoveryVersion (void)

  Returns the recovery image version.
- bool halAppBootloaderSupportsIbr (void)

  Return a value indicating whether the app bootloader supports IBRs.

6.51.1 Detailed Description

Some functions in this file return an EmberStatus value. See error-def.h for definitions of all EmberStatus return values.

See bootloader-interface-app.h for source code.

6.51.2 Macro Definition Documentation

6.51.2.1 #define BL_IMAGE_IS_VALID_CONTINUE ((uint16_t)0xFFFF)

6.51.2.2 #define BOOTLOADER_SEGMENT_SIZE (1 << BOOTLOADER_SEGMENT_SIZE_LOG2)

6.51.2.3 #define BOOTLOADER_SEGMENT_SIZE_LOG2 6

6.51.3 Function Documentation

6.51.3.1 uint8_t halAppBootloaderEraseRawStorage (uint32_t *address*, uint32_t *len*)

Parameters

address	Address to start erasing
len	Length of the region to be erased

Returns

EEPROM_SUCCESS or EEPROM_ERR.

6.51.3.2 uint8_t halAppBootloaderGetImageData (uint32_t * *timestamp*, uint8_t * *userData*)

Parameters

timestamp	write the image timestamp to this data pointer.
userData	write the user data field to this buffer.

Returns

EEPROM_SUCCESS or EEPROM_ERR

6.51.3.3 uint16_t halAppBootloaderGetRecoveryVersion (void)

6.51.3.4 uint16_t halAppBootloaderGetVersion (void)

6.51.3.5 uint16_t halAppBootloaderImageIsValid (void)

Caller should loop calling this function while it returns BL_IMAGE_IS_VALID_CONTINUE to get final result. This allows caller to service system needs during validation.

Call halAppBootloaderImageIsValidReset() before calling halAppBootloaderImageIsValid() to reset the call flag.

Here is an example application call:

```
1 halAppBootloaderImageIsValidReset();
2 while ( (pages = halAppBootloaderImageIsValid() ) == BL_IMAGE_IS_VALID_CONTINUE) {
3   // make app specific calls here, if any
4   emberTick();
5 }
```

Returns

One of the following:

- Number of pages in a valid image
- 0 for an invalid image
- BL_IMAGE_IS_VALID_CONTINUE (-1) to continue to iterate for the final result.

6.51.3.6 void halAppBootloaderImageIsValidReset (void)

6.51.3.7 const HalEepromInformationType* halAppBootloaderInfo (void)

Returns

A pointer to a HalEepromInformationType data structure, or NULL if the driver does not support this API

6.51.3.8 uint8_t halAppBootloaderInit (void)

Returns

EEPROM_SUCCESS or EEPROM_ERR_INVALID_CHIP

6.51.3.9 EmberStatus halAppBootloaderInstallNewImage (void)

6.51.3.10 uint8_t halAppBootloaderReadDownloadSpace (uint16_t *pageToBeRead*, uint8_t * *destRamBuffer*)

Parameters

pageToBeRead	pass in the page to be read. This will be converted to the appropriate address. Pages are EEPROM_PAGE_SIZE long.
destRamBuffer	a pointer to the buffer to write to.

Returns

EEPROM_SUCCESS or EEPROM_ERR.

6.51.3.11 uint8_t halAppBootloaderReadRawStorage (uint32_t *address*, uint8_t * *data*, uint16_t *len*)

Parameters

address	Address from which to start reading data
data	A pointer to a buffer where data should be read into
len	Length of the data to read

Returns

EEPROM_SUCCESS or EEPROM_ERR.

6.51.3.12 void halAppBootloaderShutdown (void)

6.51.3.13 bool halAppBootloaderStorageBusy (void)

Returns

true if still busy or false if not.

6.51.3.14 bool halAppBootloaderSupportsIbr (void)

Returns

true if the app bootloader supports IBRs, false otherwise

6.51.3.15 uint8_t halAppBootloaderWriteDownloadSpace (uint16_t *pageToBeWritten*, uint8_t * *RamPtr*)

Parameters

pageToBeWritten	pass in the page to be written. This will be converted to the appropriate address. Pages are EEPROM_PAGE_SIZE long.
RamPtr	a pointer to the data to be written.

Returns

EEPROM_SUCCESS or EEPROM_ERR

6.51.3.16 uint8_t halAppBootloaderWriteRawStorage (uint32_t *address*, const uint8_t * *data*, uint16_t *len*)

Parameters

address	Address to start writing data
data	A pointer to the buffer of data to write.
len	Length of the data to write

Returns

EEPROM_SUCCESS or EEPROM_ERR.

6.52 Custom Bootloader HAL

Modules

- Common

  Common bootloader definitions.
- Standalone

  EM35x standalone bootloader public definitions.
- Application

  Application bootloader and generic EEPROM Interface.

6.52.1 Detailed Description

6.53 Common

Common bootloader definitions.

Modules

- GPIO

  Bootloader GPIO definitions.
- Serial

  Common bootloader serial definitions.

Typedefs

- typedef uint8_t BL_Status

  Define the bootloader status type.

Enumerations

- enum {
  COMM_SERIAL = 0x01,
  COMM_RADIO = 0x02 }

Bootloader Status Definitions

These are numerical definitions for the possible bootloader status codes.

- #define BL_SUCCESS 0U

  Numerical definition for a bootloader status code: Success.
- #define BL_CRC_MATCH 2U

  Numerical definition for a bootloader status code: CRC match.
- #define BL_IMG_FLASHED 3U

  Numerical definition for a bootloader status code: Image flashed.
- #define BL_ERR 1U

  Numerical definition for a bootloader status code: serial error.
- #define BL_ERR_MASK 0x40U

  Numerical definition for a bootloader status code: Error mask.
- #define BL_ERR_HEADER_EXP 0x41U

  Numerical definition for a bootloader status code: Failed in header state. Header expected.
- #define BL_ERR_HEADER_WRITE_CRC 0x42U

  Numerical definition for a bootloader status code: Failed write/CRC of header.
- #define BL_ERR_CRC 0x43U

  Numerical definition for a bootloader status code: Failed file CRC.
- #define BL_ERR_UNKNOWN_TAG 0x44U

  Numerical definition for a bootloader status code: Unknown tag.
- #define BL_ERR_SIG 0x45U

  Numerical definition for a bootloader status code: EBL header error.
- #define BL_ERR_ODD_LEN 0x46U

Numerical definition for a bootloader status code: Trying to flash odd length bytes.

- #define BL_ERR_BLOCK_INDEX 0x47U

  Numerical definition for a bootloader status code: Indexed past end of block buffer.

- #define BL_ERR_OVWR_BL 0x48U

  Numerical definition for a bootloader status code: Attempt to overwrite bootloader flash.

- #define BL_ERR_OVWR_SIMEE 0x49U

  Numerical definition for a bootloader status code: Attempt to overwrite Simulated EEPROM flash.

- #define BL_ERR_ERASE_FAIL 0x4AU

  Numerical definition for a bootloader status code: Flash erase failed.

- #define BL_ERR_WRITE_FAIL 0x4BU

  Numerical definition for a bootloader status code: Flash write failed.

- #define BL_ERR_CRC_LEN 0x4CU

  Numerical definition for a bootloader status code: END tag CRC wrong length.

- #define BL_ERR_NO_QUERY 0X4DU

  Numerical definition for a bootloader status code: Received data before query request/response.

- #define BL_ERR_BAD_LEN 0x4EU

  Numerical definition for a bootloader status code: Invalid length detected.

- #define BL_ERR_TAGBUF 0x4FU

  Numerical definition for a bootloader status code: Problem with tagBuf detected.

- #define BL_EBL_CONTINUE 0x50U

  Numerical definition for a bootloader status code: processEbl deferred, call again to continue.

- #define BL_ERR_UNEXPECTED_TAG 0x51U

  Numerical definition for a bootloader status code: A known tag was found in an unexpected location (eg. header tag found after data)

- #define BL_ERR_UNK_ENC 0x52U

  Numerical definition for a bootloader status code: The specified encryption type is unknown to this bootloader.

- #define BL_ERR_INV_KEY 0x53U

  Numerical definition for a bootloader status code: No valid encryption key found on the device (ie. It's all 0xFF's). Bootloader will not function until this key is set.

- #define BL_ERR_ENC 0x54U

  Numerical definition for a bootloader status code: Generic error indicating that there was a problem with the encrypted file when decrypting.

- #define BL_IBR_ERR_CRC 0x55U

  Numerical definition for a bootloader status code: Failed IBR crc.

- #define BL_IBR_ERR_VERS 0x56U

  Numerical definition for a bootloader status code: Incorrect IBR version.

- #define BL_IBR_ERR_ADDR 0x57U

  Numerical definition for a bootloader status code: Invalid ebl address in IBR.

- #define BL_IBR_ERR_HDR 0x58U

  Numerical definition for a bootloader status code: Incorrect IBR header.

Bootloader State Flags

These are numerical flags for the possible bootloader states. These values are used in the bootloader code for making the current state more verbose.

Note

The flags do not start at 0 so that they can be output via the serial port during debug and easily screened out of normal xmodem traffic which depends only on ACK (0x06) and NAK (0x15).

- #define TIMEOUT 0x16

  Bootloader state flag.
- #define FILEDONE 0x17

  Bootloader state flag.
- #define FILEABORT 0x18

  Bootloader state flag.
- #define BLOCKOK 0x19

  Bootloader state flag.
- #define QUERYFOUND 0x1A

  Bootloader state flag.
- #define START_TIMEOUT 0x1B

  Bootloader state flag.
- #define BLOCK_TIMEOUT 0x1C

  Bootloader state flag.
- #define BLOCKERR_MASK 0x20

  Bootloader state flag.
- #define BLOCKERR_SOH 0x21

  Bootloader state flag: Start Of Header not received.
- #define BLOCKERR_CHK 0x22

  Bootloader state flag: Sequence of bytes don't match.
- #define BLOCKERR_CRCH 0x23

  Bootloader state flag: CRC High byte failure.
- #define BLOCKERR_CRCL 0x24

  Bootloader state flag: CRC Low byte failure.
- #define BLOCKERR_SEQUENCE 0x25

  Bootloader state flag: Block received out of sequence.
- #define BLOCKERR_PARTIAL 0x26

  Bootloader state flag: Partial block received.
- #define BLOCKERR_DUPLICATE 0x27

  Bootloader state flag: Duplicate of previous block.

6.53.1 Detailed Description

See bootloader-common.h for source code.

6.53.2 Macro Definition Documentation

6.53.2.1 #define BL_CRC_MATCH 2U

6.53.2.2 #define BL_EBL_CONTINUE 0x50U

6.53.2.3 #define BL_ERR 1U

6.53.2.4 #define BL_ERR_BAD_LEN 0x4EU

6.53.2.5 #define BL_ERR_BLOCK_INDEX 0x47U

6.53.2.6 #define BL_ERR_CRC 0x43U

6.53.2.7 #define BL_ERR_CRC_LEN 0x4CU

6.53.2.8 #define BL_ERR_ENC 0x54U

6.53.2.9 #define BL_ERR_ERASE_FAIL 0x4AU

6.53.2.10 #define BL_ERR_HEADER_EXP 0x41U

6.53.2.11 #define BL_ERR_HEADER_WRITE_CRC 0x42U

6.53.2.12 #define BL_ERR_INV_KEY 0x53U

6.53.2.13 #define BL_ERR_MASK 0x40U

6.53.2.14 #define BL_ERR_NO_QUERY 0X4DU

6.53.2.15 #define BL_ERR_ODD_LEN 0x46U

6.53.2.16 #define BL_ERR_OVWR_BL 0x48U

6.53.2.17 #define BL_ERR_OVWR_SIMEE 0x49U

6.53.2.18 #define BL_ERR_SIG 0x45U

6.53.2.19 #define BL_ERR_TAGBUF 0x4FU

6.53.2.20 #define BL_ERR_UNEXPECTED_TAG 0x51U

6.53.2.21 #define BL_ERR_UNK_ENC 0x52U

6.53.2.22 #define BL_ERR_UNKNOWN_TAG 0x44U

6.53.2.23 #define BL_ERR_WRITE_FAIL 0x4BU

6.53.2.24 #define BL_IBR_ERR_ADDR 0x57U

6.53.2.25 #define BL_IBR_ERR_CRC 0x55U

6.53.2.26 #define BL_IBR_ERR_HDR 0x58U

6.53.2.27 #define BL_IBR_ERR_VERS 0x56U

6.53.2.28 #define BL_IMG_FLASHED 3U

6.53.2.29 #define BL_SUCCESS 0U

6.53.2.30 #define BLOCK_TIMEOUT 0x1C

6.53.2.31 #define BLOCKERR_CHK 0x22

6.53.2.32 #define BLOCKERR_CRCH 0x23

6.53.2.33 #define BLOCKERR_CRCL 0x24

6.53.2.34 #define BLOCKERR_DUPLICATE 0x27

6.53.2.35 #define BLOCKERR_MASK 0x20

6.53.2.36 #define BLOCKERR_PARTIAL 0x26

6.53.2.37 #define BLOCKERR_SEQUENCE 0x25

6.53.2.38 #define BLOCKERR_SOH 0x21

6.53.2.39 #define BLOCKOK 0x19

6.53.2.40 #define FILEABORT 0x18

6.53.2.41 #define FILEDONE 0x17

6.53.2.42 #define QUERYFOUND 0x1A

6.53.2.43 #define START_TIMEOUT 0x1B

6.53.2.44 #define TIMEOUT 0x16

6.53.3 Typedef Documentation

6.53.3.1 typedef uint8_t BL_Status

6.53.4 Enumeration Type Documentation

6.53.4.1 anonymous enum

Enumerator

COMM_SERIAL

COMM_RADIO

6.54 GPIO

Bootloader GPIO definitions.

Enumerations

- enum blState_e {
  BL_ST_UP,
  BL_ST_DOWN,
  BL_ST_POLLING_LOOP,
  BL_ST_DOWNLOAD_LOOP,
  BL_ST_DOWNLOAD_FAILURE,
  BL_ST_DOWNLOAD_SUCCESS }

  Defines various bootloader states. Use in LED code to signal bootload activity.

Functions

- void bootloadGpioInit (void)

  Initialize GPIO.
- void bootloadStateIndicator (enum blState_e state)

  Helper function used for displaying bootloader state (for example: with LEDs).
- bool bootloadForceActivation (void)

  Force activation of bootloader.

State Indicator Macros

The bootloader indicates which state it is in by calling these // macros. Map them to the ::halBootloadStateIndicator function // (in bootloder-gpio.c) if you want to display that bootloader state. // Used to blink the LED's or otherwise signal bootloader activity.

- #define BL_STATE_UP() do { bootloadStateIndicator(BL_ST_UP); } while (0)

  Finished init sequence, ready for bootload.
- #define BL_STATE_DOWN() do { bootloadStateIndicator(BL_ST_DOWN); } while (0)

  Called right before bootloader resets to application. Use to cleanup and reset GPIO's to leave node in known state for app start, if necessary.
- #define BL_STATE_POLLING_LOOP() do { bootloadStateIndicator(BL_ST_POLLING_LOOP); } while (0)

  Standalone bootloader polling serial/radio interface.
- #define BL_STATE_DOWNLOAD_LOOP() do { bootloadStateIndicator(BL_ST_DOWNLOAD_LOOP); } while (0)

  Processing download image.
- #define BL_STATE_DOWNLOAD_SUCCESS() do { bootloadStateIndicator(BL_ST_DOWNLOAD_SUCC↩ ESS); } while (0)

  Download process was a success.
- #define BL_STATE_DOWNLOAD_FAILURE() do { bootloadStateIndicator(BL_ST_DOWNLOAD_FAILURE); } while (0)

  Download process failed.

6.54.1 Detailed Description

See bootloader-gpio.h for source code.

6.54.2 Macro Definition Documentation

6.54.2.1 #define BL_STATE_DOWN( ) do { bootloadStateIndicator(BL_ST_DOWN); } while (0)

6.54.2.2 #define BL_STATE_DOWNLOAD_FAILURE( ) do { bootloadStateIndicator(BL_ST_DOWNLOAD_FAILURE); } while (0)

6.54.2.3 #define BL_STATE_DOWNLOAD_LOOP( ) do { bootloadStateIndicator(BL_ST_DOWNLOAD_LOOP); } while (0)

6.54.2.4 #define BL_STATE_DOWNLOAD_SUCCESS( ) do { bootloadStateIndicator(BL_ST_DOWNLOAD_SUCCESS); } while (0)

6.54.2.5 #define BL_STATE_POLLING_LOOP( ) do { bootloadStateIndicator(BL_ST_POLLING_LOOP); } while (0)

6.54.2.6 #define BL_STATE_UP( ) do { bootloadStateIndicator(BL_ST_UP); } while (0)

6.54.3 Enumeration Type Documentation

6.54.3.1 enum blState_e

Enumerator

BL_ST_UP bootloader up

BL_ST_DOWN bootloader going down

BL_ST_POLLING_LOOP polling interfaces

BL_ST_DOWNLOAD_LOOP downloading

BL_ST_DOWNLOAD_FAILURE download failure

BL_ST_DOWNLOAD_SUCCESS download success

6.54.4 Function Documentation

6.54.4.1 bool bootloadForceActivation (void)

6.54.4.2 void bootloadGpioInit (void)

6.54.4.3 void bootloadStateIndicator (enum blState_e *state*)

6.55 Serial

Common bootloader serial definitions.

Functions

- void serInit (void)
  Initialize serial port.
- void serPutFlush (void)
  Flush the transmiter.
- void serPutChar (uint8_t ch)
  Transmit a character.
- void serPutStr (const char *str)
  Transmit a string.
- void serPutBuf (const uint8_t buf[ ], uint8_t size)
  Transmit a buffer.
- void serPutDecimal (uint16_t val)
  Transmit a 16bit value in decimal.
- void serPutHex (uint8_t byte)
  Transmit a byte as hex.
- void serPutHexInt (uint16_t word)
  Transmit a 16bit integer as hex.
- bool serCharAvailable (void)
  Determine if a character is available.
- uint8_t serGetChar (uint8_t *ch)
  Get a character if available, otherwise return an error.
- void serGetFlush (void)
  Flush the receiver.

6.55.1 Detailed Description

See bootloader-serial.h for source code.

6.55.2 Function Documentation

6.55.2.1 bool serCharAvailable (void)

Returns

true if a character is available, false otherwise.

6.55.2.2 uint8_t serGetChar (uint8_t * *ch*)

Parameters

ch	Pointer to a location where the received byte will be placed.

Returns

BL_SUCCESS if a character was obtained, BL_ERR otherwise.

6.55.2.3 void serGetFlush (void)

6.55.2.4 void serInit (void)

6.55.2.5 void serPutBuf (const uint8_t *buf[ ],* uint8_t *size*)

Parameters

buf	A buffer.
size	Length of buffer.

6.55.2.6 void serPutChar (uint8_t *ch*)

Parameters

ch	A character.

6.55.2.7 void serPutDecimal (uint16_t *val*)

Parameters

val	The data to print.

6.55.2.8 void serPutFlush (void)

6.55.2.9 void serPutHex (uint8_t *byte*)

Parameters

byte	A byte.

6.55.2.10 void serPutHexInt (uint16_t *word*)

Parameters

word	A 16bit integer.

6.55.2.11 void serPutStr (const char * *str*)

Parameters

str	A string.

6.56 Standalone

EM35x standalone bootloader public definitions.

Required Custom Functions

- void bootloaderMenu (void)

  This function must be implemented, providing a bootloader menu.

Available Bootloader Library Functions

Functions implemented by the bootloader library that may be used by custom functions.

- BL_Status receiveImage (uint8_t commState)

  Puts the bootloader into a mode where it will receive an image. commState indicates whether the image is received via serial (COMM_SERIAL) or over the air (COMM_RADIO)
- bool checkDebugMenuOption (uint8_t ch)

  A hook to the bootloader library for it to check for extra menu options. Only used for ember internal debug builds, not normally needed.
- BL_Status initOtaState (void)

  Initialize OTA Bootloader state.
- BL_Status checkOtaStart (void)

  Check to see if the bootloader has detected an OTA upload start.
- BL_Status receiveOtaImage (void)

  Puts the bootloader into a mode where it will receive an image over the air. The function checkOtaStart() should have been called first and it should have returned with a status of BL_SUCCESS before calling this function.
- bool paIsPresent (void)

  Uses the information in the PHY_CONFIG token to determine if a power amplifier is present in the node design.
- bool halCheckIntegrity (void)

  Validate application integrity by running AES-MMO hash and comparing to AAT.

6.56.1 Detailed Description

See standalone-bootloader.h for source code.

6.56.2 Function Documentation

6.56.2.1 void bootloaderMenu (void)

6.56.2.2 bool checkDebugMenuOption (uint8_t *ch*)

Returns

true if the option was handled, false if not.

6.56.2.3 BL_Status checkOtaStart (void)

Note

OTA support hooks are subject to change!

Returns

BL_Status of the success of the function.

6.56.2.4 bool halCheckIntegrity (void)

Returns

false if fails integrity check, true if pass

6.56.2.5 BL_Status initOtaState (void)

Note

OTA support hooks are subject to change!

Returns

BL_Status of the success of the function.

6.56.2.6 bool palsPresent (void)

Note

This function must not be called before emBootloaderRadioBoot().

Returns

true if a power amplifier is present, false otherwise.

6.56.2.7 BL_Status receiveImage (uint8_t *commState*)

6.56.2.8 BL_Status receiveOtaImage (void)

Note

OTA support hooks are subject to change!

Returns

BL_Status of the success of the function.

6.57 Application

Application bootloader and generic EEPROM Interface.

Data Structures

- struct HalEepromInformationType

  This structure defines a variety of information about the attached external EEPROM device.

Macros

- #define EEPROM_PAGE_SIZE (128ul)

  Definition of an EEPROM page size, in bytes. This definition is deprecated, and should no longer be used.
- #define EEPROM_FIRST_PAGE (0)

  Define the location of the first page in EEPROM. This definition is deprecated, and should no longer be used.
- #define EEPROM_IMAGE_START (EEPROM_FIRST_PAGE * EEPROM_PAGE_SIZE)

  Define the location of the image start in EEPROM as a function of the EEPROM_FIRST_PAGE and EEPROM_PAGE_SIZE. This definition is deprecated, and should no longer be used.
- #define EEPROM_SUCCESS 0U

  Define EEPROM success status.
- #define EEPROM_ERR 1U

  Define EEPROM error status.
- #define EEPROM_ERR_MASK 0x80U

  Define EEPROM error mask.
- #define EEPROM_ERR_PG_BOUNDARY 0x81U

  Define EEPROM page boundary error.
- #define EEPROM_ERR_PG_SZ 0x82U

  Define EEPROM page size error.
- #define EEPROM_ERR_WRT_DATA 0x83U

  Define EEPROM write data error.
- #define EEPROM_ERR_IMG_SZ 0x84U

  Define EEPROM image too large error.
- #define EEPROM_ERR_ADDR 0x85U

  Define EEPROM invalid address error.
- #define EEPROM_ERR_INVALID_CHIP 0x86U

  Define EEPROM chip initialization error.
- #define EEPROM_ERR_ERASE_REQUIRED 0x87U

  Define EEPROM erase required error.
- #define EEPROM_ERR_NO_ERASE_SUPPORT 0x88U

  Define EEPROM error for no erase support.

EEPROM interaction functions.

- uint8_t halEepromInit (void)

  Initialize EEPROM. Note: some earlier drivers may assert instead of returning an error if initialization fails.
- void halEepromShutdown (void)

  Shutdown the EEPROM to conserve power.
- const HalEepromInformationType * halEepromInfo (void)

  Call this function to get information about the external EEPROM and its capabilities.
- uint32_t halEepromSize (void)

  Return the size of the EEPROM.
- bool halEepromBusy (void)

  Determine if the exernal EEPROM is still busy performing the last operation, such as a write or an erase.
- uint8_t halEepromRead (uint32_t address, uint8_t *data, uint16_t len)

  Read from the external EEPROM.
- uint8_t halEepromWrite (uint32_t address, const uint8_t *data, uint16_t len)

  Write to the external EEPROM.
- uint8_t halEepromErase (uint32_t address, uint32_t totalLength)

  Erases the specified region of the external EEPROM.
- #define EEPROM_INFO_VERSION (0x0202)

  The current version of the HalEepromInformationType data structure.
- #define EEPROM_INFO_MAJOR_VERSION (0x0200)

  The current version of the HalEepromInformationType data structure.
- #define EEPROM_INFO_MAJOR_VERSION_MASK (0xFF00)

  The current version of the HalEepromInformationType data structure.
- #define EEPROM_INFO_MIN_VERSION_WITH_WORD_SIZE_SUPPORT 0x0102U

  The current version of the HalEepromInformationType data structure.
- #define EEPROM_CAPABILITIES_ERASE_SUPPORTED (0x0001U)

  Eeprom capabilites mask that indicates the erase API is supported.
- #define EEPROM_CAPABILITIES_PAGE_ERASE_REQD (0x0002U)

  Eeprom capabilites mask that indicates page erasing is required before new data can be written to a device.
- #define EEPROM_CAPABILITIES_BLOCKING_WRITE (0x0004U)

  Eeprom capabilites mask that indicates that the write routine is blocking on this device.
- #define EEPROM_CAPABILITIES_BLOCKING_ERASE (0x0008U)

  Eeprom capabilites mask that indicates that the erase routine is blocking on this device.
- #define EEPROM_CAPABILITIES_PART_ERASE_SECONDS (0x0010U)

  Eeprom capabilities mask that indicateds that the partEraseTime field of HalEepromInformationType is in seconds instead of the usual milliseconds.

Required Custom Functions

- void bootloaderInit ()

  Drives the app bootloader. If the ::runRecovery parameter is ::true, the recovery mode should be activated, otherwise it should attempt to install an image. This function should not return. It should always exit by resetting the the bootloader.
- void bootloaderInitCustom ()

  Drives the app bootloader. If the ::runRecovery parameter is ::true, the recovery mode should be activated, otherwise it should attempt to install an image. This function should not return. It should always exit by resetting the the bootloader.
- void bootloaderAction (bool runRecovery)

  Drives the app bootloader. If the ::runRecovery parameter is ::true, the recovery mode should be activated, otherwise it should attempt to install an image. This function should not return. It should always exit by resetting the the bootloader.

Available Bootloader Library Functions

Functions implemented by the bootloader library that may be used by custom functions.

- BL_Status recoveryMode (void)

  Activates recoveryMode to receive a new image over xmodem.
- BL_Status processImage (bool install)

  Processes an image in the external eeprom.

6.57.1 Detailed Description

The file bootloader-eeprom.h defines generic EEPROM parameters.

Changing EEPROM size will change the size of the application image space without changing the size or relative location of the recovery and reserved sections. See eeprom.c for more information on modifying EEPROM functionality.

See bootloader-eeprom.h for source code.

See app-bootloader.h for source code.

6.57.2 Macro Definition Documentation

6.57.2.1 #define EEPROM_CAPABILITIES_BLOCKING_ERASE (0x0008U)

6.57.2.2 #define EEPROM_CAPABILITIES_BLOCKING_WRITE (0x0004U)

6.57.2.3 #define EEPROM_CAPABILITIES_ERASE_SUPPORTED (0x0001U)

6.57.2.4 #define EEPROM_CAPABILITIES_PAGE_ERASE_REQD (0x0002U)

6.57.2.5 #define EEPROM_CAPABILITIES_PART_ERASE_SECONDS (0x0010U)

6.57.2.6 #define EEPROM_ERR 1U

6.57.2.7 #define EEPROM_ERR_ADDR 0x85U

6.57.2.8 #define EEPROM_ERR_ERASE_REQUIRED 0x87U

6.57.2.9 #define EEPROM_ERR_IMG_SZ 0x84U

6.57.2.10 #define EEPROM_ERR_INVALID_CHIP 0x86U

6.57.2.11 #define EEPROM_ERR_MASK 0x80U

6.57.2.12 #define EEPROM_ERR_NO_ERASE_SUPPORT 0x88U

6.57.2.13 #define EEPROM_ERR_PG_BOUNDARY 0x81U

6.57.2.14 #define EEPROM_ERR_PG_SZ 0x82U

6.57.2.15 #define EEPROM_ERR_WRT_DATA 0x83U

6.57.2.16 #define EEPROM_FIRST_PAGE (0)

6.57.2.17 #define EEPROM_IMAGE_START (EEPROM_FIRST_PAGE * EEPROM_PAGE_SIZE)

6.57.2.18 #define EEPROM_INFO_MAJOR_VERSION (0x0200)

6.57.2.19 #define EEPROM_INFO_MAJOR_VERSION_MASK (0xFF00)

6.57.2.20 #define EEPROM_INFO_MIN_VERSION_WITH_WORD_SIZE_SUPPORT 0x0102U

6.57.2.21 #define EEPROM_INFO_VERSION (0x0202)

6.57.2.22 #define EEPROM_PAGE_SIZE (128ul)

6.57.2.23 #define EEPROM_SUCCESS 0U

6.57.3 Function Documentation

6.57.3.1 void bootloaderAction (bool *runRecovery*)

Parameters

runRecovery	If ::true, recover mode is activated. Otherwise, normal image installation is activated.

6.57.3.2 void bootloaderInit ( )

Parameters

runRecovery	If ::true, recover mode is activated. Otherwise, normal image installation is activated.

6.57.3.3 void bootloaderInitCustom ( )

Parameters

runRecovery	If ::true, recover mode is activated. Otherwise, normal image installation is activated.

6.57.3.4 bool halEepromBusy (void)

The format of this call must not be altered. However, the content can be changed to work with a different device.

Returns

true if still busy or false if not.

6.57.3.5 uint8_t halEepromErase (uint32_t *address*, uint32_t *totalLength*)

The format of this call must not be altered. However, the content can be changed to work with a different device. Note: Most devices require the specified region to be page aligned, and will return an error if an unaligned region is specified. Note: Many devices take an extremely long time to perform an erase operation. When erasing a large region, it may be preferable to make multiple calls to this API so that other application functionality can be performed while the erase is in progress. The halEepromBusy() API may be used to determine when the last erase operation has completed. Erase timing information can be found in the HalEepromInformationType structure.

Parameters

address	Address to start erasing
len	Length of the region to be erased

Returns

EEPROM_SUCCESS or EEPROM_ERR.

6.57.3.6 const HalEepromInformationType* halEepromInfo (void)

The format of this call must not be altered. However, the content can be changed to work with a different device.

Returns

A pointer to a HalEepromInformationType data structure, or NULL if the driver does not support this API

6.57.3.7 uint8_t halEepromInit (void)

Returns

EEPROM_SUCCESS or EEPROM_ERR_INVALID_CHIP

6.57.3.8 uint8_t halEepromRead (uint32_t *address*, uint8_t * *data*, uint16_t *len*)

This is the standard external EEPROM read function. The format of this call must not be altered. However, the content can be changed to work with a different device. Note: Not all storage implementations support accesses that are not page aligned, refer to the HalEepromInformationType structure for more information.

Parameters

address	The address to start reading from.
data	A pointer to where read data is stored.
len	The length of data to read.

Returns

EEPROM_SUCCESS or EEPROM_ERR

6.57.3.9 void halEepromShutdown (void)

6.57.3.10 uint32_t halEepromSize (void)

The format of this call must not be altered. However, the content can be changed to work with a different device. Internal use only. No exposure to application

Returns

int32_t size

6.57.3.11 uint8_t halEepromWrite (uint32_t *address*, const uint8_t * *data*, uint16_t *len*)

This is the standard external EEPROM write function. The format of this call must not be altered. However, the content can be changed to work with a different device. Note: Not all storage implementations support accesses that are not page aligned, refer to the HalEepromInformationType structure for more information. Note: Some storage devices require contents to be erased before new data can be written, and will return an EEPROM_ERR_ERASE↩ _REQUIRED error if write is called on a location that is not already erased. Refer to the HalEepromInformationType structure to see if the attached storage device requires erasing.

Parameters

address	The address to start writing to.
data	A pointer to the data to write.
len	The length of data to write.

Returns

EEPROM_SUCCESS or EEPROM_ERR

6.57.3.12 BL_Status processImage (bool *install*)

Parameters

install	If ::false, it will simply validate the image without touching main flash. If ::true, the image will be programmed to main flash.

Returns

BL_SUCCESS if an image was successfully installed/validated

6.57.3.13 BL_Status recoveryMode (void)

Returns

BL_SUCCESS if an image was successfully received.

6.58 Application Framework API Reference

Modules

- ZCL over IP
- Callbacks

6.58.1 Detailed Description

6.59 ZCL over IP

Modules

- OTA Bootload
- Utilities
- ZCL Types
- Discovery
- Messages
- Addresses
- Endpoints
- Groups
- Clusters
- Attributes
- Bindings
- Commands
- Reporting
- Management

6.59.1 Detailed Description

The ZCL Core plugin provides the necessary foundation of APIs to interface with a ZCLIP-capable device.

The functionality contained in this plugin provides basic ZCLIP features including, but not limited to, the following:

- Attribute management
- Binding management
- Command handling and dispatching
- Endpoint management
- Group management
- Reporting configuration management
- Notification handling and dispatching
- Device discovery
- Application provisioning
- General ZCLIP utilities

This plugin uses the Silicon Labs Constrained Application Protocol (CoAP) implementation to communicate over the air with remote devices. More information about the Silicon Labs CoAP implementation can be found in Constrained Application Protocol API. This plugin also uses the Silicon Labs token system for storing non-volatile ZCLIP data. More information about the Silicon Labs token system can be found in Token Access.

This plugin also provides a list of command line interface (CLI) commands with which users can drive their applications. These CLI commands are documented in the README.zclip file included in this release.

See ZCL Core Callbacks for the ZCLIP application callback API.

6.60 Callbacks

Modules

- Framework Callbacks
- ASHv3 Callbacks
- Battery Monitor Callbacks
- Bulb PWM Driver Callbacks
- Button Callbacks
- Button Interface Callbacks
- Button-Press Callbacks
- Color Control Cluster Server Callbacks
- Connection Manager: In Band Joining Callbacks
- GPIO Sensor Interface Callbacks
- HAL Library Callbacks
- Identify Server Callbacks
- Idle/Sleep Callbacks
- Main Callbacks
- Microphone Codec MSADPCM Callbacks
- Microphone IMAADPCM Callbacks
- Occupancy PYD-1698 Callbacks
- Occupancy Sensor Server Cluster Callbacks
- OTA Bootload Client Callbacks
- OTA Bootload Server Callbacks
- Polling Callbacks
- SB1 Gesture Sensor Callbacks
- STM32F103RET Library Callbacks
- Tamper Switch Interface Callbacks
- Gateway MQTT Transport Callbacks
- ZCL Core Callbacks
- buffer-management API Callbacks
- Callbacks
- coap-diagnostic API Callbacks
- connection-manager API Callbacks
- host-mfglib API Callbacks
- icmp API Callbacks
- network-management API Callbacks
- power-meter API Callbacks
- sim-eeprom API Callbacks
- stack-info API Callbacks
- thread-debug API Callbacks
- udp API Callbacks

6.60.1 Detailed Description

6.61 Framework Callbacks

Functions

- int main (MAIN_FUNCTION_PARAMETERS)

  Main Application Entry Point.

6.61.1 Detailed Description

These callbacks are contributed by the framework.

6.61.2 Function Documentation

6.61.2.1 int main (MAIN_FUNCTION_PARAMETERS)

This is the main application entry point. All applications must implement this function.

6.62 ASHv3 Callbacks

Functions

- void emberAshStatusHandler (AshState state)

  Notification that ash has changed state.

6.62.1 Detailed Description

These callbacks are contributed by the ASHv3 plugin.

6.62.2 Function Documentation

6.62.2.1 void emberAshStatusHandler (AshState *state*)

Parameters

state	An AshState indicating one of the conditions described below.

This callback will report an AshState of ASH_STATE_RESET_TX_PRE whenever the RESET packet has been sent to the UART link.

This callback will report an AshState of ASH_STATE_RESET_TX_POST whenever the UART is done sending the RESET packet.

This callback will report an AshState of ASH_STATE_RUNNING when we've received an RSTACK to our RESET.

6.63 Battery Monitor Callbacks

Functions

- void emberAfPluginBatteryMonitorDataReadyCallback (uint16_t batteryVoltageMilliV)
  Data Ready.

6.63.1 Detailed Description

These callbacks are contributed by the Battery Monitor plugin.

6.63.2 Function Documentation

6.63.2.1 void emberAfPluginBatteryMonitorDataReadyCallback (uint16_t *batteryVoltageMilliV*)

This function is called whenever the battery monitor has generated a new valid battery level

Parameters

batteryVoltageMilliV	The battery voltage, in milli Volts Ver.: always

6.64 Bulb PWM Driver Callbacks

Functions

- uint16_t halBulbPwmDriverFrequencyCallback (void)

  A callback used to configure the frequency of the PWM driver. This is called by the bulb-pwm driver upon initialization to determine the frequency at which the PWM driver should be driven. It should return either the frequency, in Hz, or USE_DEFAULT_FREQUENCY to indicate that the plugin should use the default value. The default value is 1000 Hz, but can be overridden by a macro in the board header if a user wishes.
- void halBulbPwmDriverInitCompleteCallback (void)

  Function to indicate that the PWM driver has been initialized and the bulb should drive the initial LED PWM values at this time.
- void halBulbPwmDriverBlinkOnCallback (void)

  This callback is generated during blinking behavior when it is time to turn the bulb on. While the plugin will determine when to blink the bulb on or off, it is up to this callback to determine how to turn the bulb on.
- void halBulbPwmDriverBlinkOffCallback (void)

  This callback is generated during blinking behavior when it is time to turn the bulb off. While the plugin will determine when to blink the bulb on or off, it is up to this callback to determine how to turn the bulb off.
- void halBulbPwmDriverBlinkStartCallback (void)

  This callback is generated when the application layer makes a call to initiate blinking behavior. It warns the application layer PWM code to not attempt to drive the LEDs directly and interfere with the blinking behavior.
- void halBulbPwmDriverBlinkStopCallback (void)

  This callback is generated when the current blinking command finishes. The application layer PWM code must then determine what the bulb drive should be, based on the current appliation layer attributes (i.e. level, on/off, color XY, etc.)

6.64.1 Detailed Description

These callbacks are contributed by the Bulb PWM Driver plugin.

6.64.2 Function Documentation

6.64.2.1 void halBulbPwmDriverBlinkOffCallback (void)

Should be implemented by an application layer configuration plugin.

6.64.2.2 void halBulbPwmDriverBlinkOnCallback (void)

Should be implemented by an application layer configuration plugin.

6.64.2.3 void halBulbPwmDriverBlinkStartCallback (void)

Should be implemented by an application layer configuration plugin.

6.64.2.4 void halBulbPwmDriverBlinkStopCallback (void)

Should be implemented by an application layer configuration plugin.

6.64.2.5 uint16_t halBulbPwmDriverFrequencyCallback (void)

Should be implemented by an application layer configuration plugin.

6.64.2.6 void halBulbPwmDriverInitCompleteCallback (void)

Should be implemented by an application layer configuration plugin.

6.65 Button Callbacks

Functions

- void halButtonIsr (uint8_t button, uint8_t state)

  A callback called in interrupt context whenever a button changes its state.

6.65.1 Detailed Description

These callbacks are contributed by the Button plugin.

6.65.2 Function Documentation

6.65.2.1 void halButtonIsr (uint8_t *button,* uint8_t *state*)

Must be implemented by the application. This function should contain the functionality to be executed in response to changes of state in each of the buttons, or callbacks to the appropriate functionality.

Parameters

button	The button which has changed state, either BUTTON0 or BUTTON1 as defined in the appropriate BOARD_HEADER.
state	The new state of the button referenced by the button parameter, either BUTTON_PRESSED if the button has been pressed or BUTTON_RELEASED if the button has been released.

6.66 Button Interface Callbacks

Functions

- void emberAfPluginButtonInterfaceButton0PressedShortCallback (uint16_t timePressedMs)

  Button0 Pressed Short.
- void emberAfPluginButtonInterfaceButton1PressedShortCallback (uint16_t timePressedMs)

  Button1 Pressed Short.
- void emberAfPluginButtonInterfaceButton0PressedLongCallback (uint16_t timePressedMs, bool pressedAtReset)

  Button0 Pressed Long.
- void emberAfPluginButtonInterfaceButton1PressedLongCallback (uint16_t timePressedMs, bool pressedAtReset)

  Button1 Pressed Long.
- void emberAfPluginButtonInterfaceButton0PressingCallback (void)

  Button0 Pressing.
- void emberAfPluginButtonInterfaceButton1PressingCallback (void)

  Button1 Pressing.
- void emberAfPluginButtonInterfaceButton0LowCallback (void)

  Button0 Low.
- void emberAfPluginButtonInterfaceButton0HighCallback (void)

  Button0 High.
- void emberAfPluginButtonInterfaceButton1LowCallback (void)

  Button1 Low.
- void emberAfPluginButtonInterfaceButton1HighCallback (void)

  Button1 High.

6.66.1 Detailed Description

These callbacks are contributed by the Button Interface plugin.

6.66.2 Function Documentation

6.66.2.1 void emberAfPluginButtonInterfaceButton0HighCallback (void)

This function is called when the GPIO tied to button zero goes high

6.66.2.2 void emberAfPluginButtonInterfaceButton0LowCallback (void)

This function is called when the GPIO tied to button zero goes low

6.66.2.3 void emberAfPluginButtonInterfaceButton0PressedLongCallback (uint16_t *timePressedMs,* bool *pressedAtReset*)

This function returns the number of times a button was short pressed.

Parameters

timePressedMs	Amount of time button 0 was pressed. Ver.: always
pressedAtReset	Was the button pressed at startup. Ver.: always

6.66.2.4 void emberAfPluginButtonInterfaceButton0PressedShortCallback (uint16_t *timePressedMs*)

This function returns the number of times a button was short pressed.

Parameters

timePressedMs	Time (in ms) button 0 was pressed Ver.: always

6.66.2.5 void emberAfPluginButtonInterfaceButton0PressingCallback (void)

This function is periodically called when button 0 is being pressed.

6.66.2.6 void emberAfPluginButtonInterfaceButton1HighCallback (void)

This function is called when the GPIO tied to button one goes high

6.66.2.7 void emberAfPluginButtonInterfaceButton1LowCallback (void)

This function is called when the GPIO tied to button one goes low

6.66.2.8 void emberAfPluginButtonInterfaceButton1PressedLongCallback (uint16_t *timePressedMs,* bool *pressedAtReset*)

This function returns the number of times a button was short pressed.

Parameters

timePressedMs	Amount of time button 1 was pressed. Ver.: always
pressedAtReset	Was the button pressed at startup. Ver.: always

6.66.2.9 void emberAfPluginButtonInterfaceButton1PressedShortCallback (uint16_t *timePressedMs*)

This function returns the number of times a button was short pressed.

Parameters

timePressedMs	Time (in ms) button 1 was pressed Ver.: always

6.66.2.10 void emberAfPluginButtonInterfaceButton1PressingCallback (void)

This function is periodically called when button 1 is being pressed.

6.67 Button-Press Callbacks

Functions

- void emberButtonPressIsr (uint8_t button, EmberButtonPress press)

  A callback called when a button is pressed. It is sometimes called in ISR context.

6.67.1 Detailed Description

These callbacks are contributed by the Button-Press plugin.

6.67.2 Function Documentation

6.67.2.1 void emberButtonPressIsr (uint8_t *button,* EmberButtonPress *press*)

Must be implemented by the application. This function should contain the functionality to be executed in response to a button press, or callbacks to the appropriate functionality.

Parameters

button	The button which was pressed, either BUTTON0 or BUTTON1 as defined in the appropriate BOARD_HEADER.
press	Either EMBER_SINGLE_PRESS if it was a single press, or EMBER_DOUBLE_PRESS if it was a double press.

6.68 Color Control Cluster Server Callbacks

Functions

- void emberAfPluginColorControlServerComputePwmFromHsvCallback (uint8_t endpoint)

  Compute Pwm from HSV.
- void emberAfPluginColorControlServerComputePwmFromXyCallback (uint8_t endpoint)

  Compute Pwm from HSV.
- void emberAfPluginColorControlServerComputePwmFromTempCallback (uint8_t endpoint)

  Compute Pwm from HSV.

6.68.1 Detailed Description

These callbacks are contributed by the Color Control Cluster Server plugin.

6.68.2 Function Documentation

6.68.2.1 void emberAfPluginColorControlServerComputePwmFromHsvCallback (uint8_t *endpoint*)

This function is called from the color server when it is time for the PWMs to be driven with a new value from the HSV values.

Parameters

endpoint	The identifying endpoint Ver.: always

6.68.2.2 void emberAfPluginColorControlServerComputePwmFromTempCallback (uint8_t *endpoint*)

This function is called from the color server when it is time for the PWMs to be driven with a new value from the color temperature.

Parameters

endpoint	The identifying endpoint Ver.: always

6.68.2.3 void emberAfPluginColorControlServerComputePwmFromXyCallback (uint8_t *endpoint*)

This function is called from the color server when it is time for the PWMs to be driven with a new value from the color X and color Y values.

Parameters

endpoint	The identifying endpoint Ver.: always

6.69 Connection Manager: In Band Joining Callbacks

Functions

- uint8_t emberConnectionManagerJibGetJoinKeyCallback (uint8_t **joinKey)
  Get the fixed joining key.

6.69.1 Detailed Description

These callbacks are contributed by the Connection Manager: In Band Joining plugin.

6.69.2 Function Documentation

6.69.2.1 uint8_t emberConnectionManagerJibGetJoinKeyCallback (uint8_t ** *joinKey*)

This function will be called whenever the in band commissioning fixed joining key is needed by the connection manager. The key will be set by this function using the joinKey parameter, and the size will be relayed by the return value. A return value of 0 indicates that the key has not been set, which will cause the connection manager to halt its connection attempt.

Parameters

joinKey	A pointer to the fixed joining key

6.70 GPIO Sensor Interface Callbacks

Functions

- void emberAfPluginGpioSensorStateChangedCallback (uint8_t newSensorState)

  State Changed.

6.70.1 Detailed Description

These callbacks are contributed by the GPIO Sensor Interface plugin.

6.70.2 Function Documentation

6.70.2.1 void emberAfPluginGpioSensorStateChangedCallback (uint8_t *newSensorState*)

This function is called whenever the gpio sensor detects a change in state

Parameters

newSensorState	The new state of the sensor based alarm (EMBER_AF_PLUGIN_GPIO_SENSOR_ACTIVE or EMBER_AF_PLUGIN_GPIO_SENSOR_NOT_ACTIVE) Ver.: always

6.71 HAL Library Callbacks

Functions

- void halRadioPowerUpHandler (void)

  Handler called whenever the radio is powered on.
- void halRadioPowerDownHandler (void)

  Handler called whenever the radio is powered off.

6.71.1 Detailed Description

These callbacks are contributed by the HAL Library plugin.

6.71.2 Function Documentation

6.71.2.1 void halRadioPowerDownHandler (void)

Handler called in main context after radio has been powered off.

6.71.2.2 void halRadioPowerUpHandler (void)

Handler called in main context prior to radio being powered on.

6.72 Identify Server Callbacks

Functions

- void emberZclIdentifyServerStartIdentifyingCallback (EmberZclEndpointId_t endpointId, uint16_t identifyTimeS)

  Start Identifying.
- void emberZclIdentifyServerStopIdentifyingCallback (EmberZclEndpointId_t endpointId)

  Stop Identifying.

6.72.1 Detailed Description

These callbacks are contributed by the Identify Server plugin.

6.72.2 Function Documentation

6.72.2.1 void emberZclIdentifyServerStartIdentifyingCallback (EmberZclEndpointId_t *endpointId,* uint16_t *identifyTimeS*)

This function is called when the device should start identifying. The device should continue to identify until emberZclIdentifyServerStopIdentifyingCallback is called.

6.72.2.2 void emberZclIdentifyServerStopIdentifyingCallback (EmberZclEndpointId_t *endpointId*)

This function is called when the device should stop identifying.

6.73 Idle/Sleep Callbacks

Functions

- bool emberAfPluginIdleSleepOkToSleepCallback (uint32_t durationMs)

  Ok To Sleep.
- void emberAfPluginIdleSleepWakeUpCallback (uint32_t durationMs)

  Wake Up.
- bool emberAfPluginIdleSleepOkToIdleCallback (uint32_t durationMs)

  Ok To Idle.
- void emberAfPluginIdleSleepActiveCallback (uint32_t durationMs)

  Active.

6.73.1 Detailed Description

These callbacks are contributed by the Idle/Sleep plugin.

6.73.2 Function Documentation

6.73.2.1 void emberAfPluginIdleSleepActiveCallback (uint32_t *durationMs*)

This function is called by the Idle/Sleep plugin after idling.

Parameters

durationMs	The duration in milliseconds that the device idled.

6.73.2.2 bool emberAfPluginIdleSleepOkToIdleCallback (uint32_t *durationMs*)

This function is called by the Idle/Sleep plugin before idling. It is called with interrupts disabled. The application should return true if the device may idle or false otherwise.

Parameters

durationMs	The maximum duration in milliseconds that the device will idle.

6.73.2.3 bool emberAfPluginIdleSleepOkToSleepCallback (uint32_t *durationMs*)

This function is called by the Idle/Sleep plugin before sleeping. It is called with interrupts disabled. The application should return true if the device may sleep or false otherwise.

Parameters

durationMs	The maximum duration in milliseconds that the device will sleep.

6.73.2.4 void emberAfPluginIdleSleepWakeUpCallback (uint32_t *durationMs*)

This function is called by the Idle/Sleep plugin after sleeping.

Parameters

durationMs	The duration in milliseconds that the device slept.

6.74 Main Callbacks

Functions

- void emberAfMarkApplicationBuffersCallback (void)

  Mark Application Buffers.
- void emberAfNetworkStatusCallback (EmberNetworkStatus newNetworkStatus, EmberNetworkStatus oldNetworkStatus, EmberJoinFailureReason reason)

  Network Status.
- void emberAfMainCallback (MAIN_FUNCTION_PARAMETERS)

  Main.
- void emberAfInitCallback (void)

  Init.
- void emberAfTickCallback (void)

  Tick.

6.74.1 Detailed Description

These callbacks are contributed by the Main plugin.

These callbacks were contributed by the main API.

6.74.2 Function Documentation

6.74.2.1 void emberAfInitCallback (void)

This function is called after the stack initializes and can be used to perform any additional initialization required at stack startup. On SoCs, this will generally be called only once: at system startup. On hosts, this will be called when the NCP initializes, and may be called multiple times during the lifetime of the host application.

6.74.2.2 void emberAfMainCallback (MAIN_FUNCTION_PARAMETERS)

This function is called immediately after the application starts executing and can be used to perform initialization that should occur before any other components are initialized.

6.74.2.3 void emberAfMarkApplicationBuffersCallback (void)

This function is called when the application must mark its buffers. Buffers that are not marked will be reclaimed by the stack.

6.74.2.4 void emberAfNetworkStatusCallback (EmberNetworkStatus *newNetworkStatus,* EmberNetworkStatus *oldNetworkStatus,* EmberJoinFailureReason *reason*)

This function is called when the network status changes.

6.74.2.5 void emberAfTickCallback (void)

This function is called in each iteration of the main application loop and can be used to perform periodic functions. The frequency with which this function is called depends on how quickly the main loop runs. If the application blocks at any time during the main loop, this function will not be called until execution resumes. On SoC platforms, sleeping and idling will block. On Unix hosts, process yielding (e.g., via select) will block.

6.75 Microphone Codec MSADPCM Callbacks

Functions

- void halMicrophoneCodecMsadpcmDataReadyCallback (uint8_t *data, uint8_t length)

  A callback called when new microphone data is ready.

6.75.1 Detailed Description

These callbacks are contributed by the Microphone Codec MSADPCM plugin.

6.75.2 Function Documentation

6.75.2.1 void halMicrophoneCodecMsadpcmDataReadyCallback (uint8_t * *data*, uint8_t *length*)

This function is called by the plugin when new data has been processed and is ready to be processed by other parts of the system.

Parameters

data	Pointer to the data that is ready
length	Length of the data

6.76 Microphone IMAADPCM Callbacks

Functions

- void halMicrophoneImaadpcmDataReadyCallback (uint8_t *data, uint8_t length)

  A callback called when new microphone data is ready.

6.76.1 Detailed Description

These callbacks are contributed by the Microphone IMAADPCM plugin.

6.76.2 Function Documentation

6.76.2.1 void halMicrophoneImaadpcmDataReadyCallback (uint8_t * *data*, uint8_t *length*)

This function is called by the plugin when new data has been processed and is ready to be processed by other parts of the system.

Parameters

data	Pointer to the data that is ready
length	Length of the data

6.77 Occupancy PYD-1698 Callbacks

Functions

- void halOccupancyStateChangedCallback (HalOccupancyState occupancyState)

  Occupancy State Changed.

6.77.1 Detailed Description

These callbacks are contributed by the Occupancy PYD-1698 plugin.

6.77.2 Function Documentation

6.77.2.1 void halOccupancyStateChangedCallback (HalOccupancyState *occupancyState*)

This callback is called when the occupancy sensor state changes.

Parameters

occupancyState	The bitmap used to determine occupancy state. At present, only bit 0 is used, and will be set to either HAL_OCCUPANCY_STATE_OCCUPIED or HAL_OCCUPANCY_STATE_UNOCCUPIED.

6.78 Occupancy Sensor Server Cluster Callbacks

Functions

- void emberZclOccupancySensingServerOccupancyStateChangedCallback (HalOccupancyState occupancy↩ State)

  Occupancy state changed.

6.78.1 Detailed Description

These callbacks are contributed by the Occupancy Sensor Server Cluster plugin.

6.78.2 Function Documentation

6.78.2.1 void emberZclOccupancySensingServerOccupancyStateChangedCallback (HalOccupancyState *occupancyState*)

This callback is generated when the occupancy measurement server receives a new occupancy status.

Parameters

occupancyState	The bitmap used to determine occupancy state. At present, only bit 0 is used, and will be set to either HAL_OCCUPANCY_STATE_OCCUPIED or HAL_OCCUPANCY_STATE_UNOCCUPIED. Ver.: always

6.79 OTA Bootload Client Callbacks

Functions

- bool emberZclOtaBootloadClientSetVersionInfoCallback ()
- bool emberZclOtaBootloadClientServerHasStaticAddressCallback (EmberZclOtaBootloadClientServerInfo_t *serverInfo)
- bool emberZclOtaBootloadClientServerHasDnsNameCallback (EmberZclOtaBootloadClientServerInfo_t *serverInfo)
- bool emberZclOtaBootloadClientServerHasDiscByClusterId (const EmberZclClusterSpec_t *clusterSpec, EmberCoapResponseHandler responseHandler)
- bool emberZclOtaBootloadClientServerDiscoveredCallback (const EmberZclOtaBootloadClientServerInfo_t *serverInfo)
- bool emberZclOtaBootloadClientGetQueryNextImageParametersCallback (EmberZclOtaBootloadFileSpec_t *fileSpec, EmberZclOtaBootloadHardwareVersion_t *hardwareVersion)
- bool emberZclOtaBootloadClientStartDownloadCallback (const EmberZclOtaBootloadFileSpec_t *fileSpec, bool existingFile)
- EmberZclStatus_t emberZclOtaBootloadClientDownloadCompleteCallback (const EmberZclOtaBootloadFileSpec_t *fileSpec, EmberZclStatus_t status)
- void emberZclOtaBootloadClientPreBootloadCallback (const EmberZclOtaBootloadFileSpec_t *fileSpec)

6.79.1 Detailed Description

These callbacks are contributed by the OTA Bootload Client plugin.

6.79.2 Function Documentation

6.79.2.1 EmberZclStatus_t emberZclOtaBootloadClientDownloadCompleteCallback (const EmberZclOtaBootloadFileSpec_t * *fileSpec*, EmberZclStatus_t *status*)

An OTA file has been downloaded and verified, or failed to do one of the two.

Parameters

fileSpec	The specification for the OTA file in question.
status	The status of file download and verification.

Returns

A status based on the application's analysis of the downloaded file. This is an opportunity for the application to verify the downloaded image and return a related status.

See also

emberZclOtaBootloadStorageFind

6.79.2.2 bool emberZclOtaBootloadClientGetQueryNextImageParametersCallback (EmberZclOtaBootloadFileSpec_t * *fileSpec*, EmberZclOtaBootloadHardwareVersion_t * *hardwareVersion*)

Get the parameters for a QueryNextImage command.

Parameters

fileSpec	The OTA file specification to include in the query.
hardwareVersion	The current hardware version of the device, or EMBER_ZCL_OTA_BOOTLOAD_HARDWARE_VERSION_NULL if this information is to be ignored.

Returns

`true` if the device should continue to send a QueryNextImage command, or `false` if it should reschedule the command for a later time.

6.79.2.3 void emberZclOtaBootloadClientPreBootloadCallback (const EmberZclOtaBootloadFileSpec_t * *fileSpec*)

An OTA file is about to be installed.

Parameters

fileSpec	The specification for the OTA file that will be installed.

See also

emberZclOtaBootloadStorageFind

6.79.2.4 bool emberZclOtaBootloadClientServerDiscoveredCallback (const EmberZclOtaBootloadClientServerInfo_t * *serverInfo*)

A potential OTA Server has been discovered.

Parameters

serverInfo	Information about the discovered OTA Server device.

Returns

`true` if the device should be chosen as the OTA Server, `false` otherwise.

Note

If `true` is returned, the OTA Client state machine will move to the next state, which is querying the OTA Server device for a new OTA file to download. If `false` is returned, the OTA Client will continue to try to discover an OTA Server.

6.79.2.5 bool emberZclOtaBootloadClientServerHasDiscByClusterId (const EmberZclClusterSpec_t * *clusterSpec*, EmberCoapResponseHandler *responseHandler*)

Initiate a server discovery

Parameters

clusterSpec	server specifications
responseHandler	function to handle the response to the CoAP message

Returns

`true` if the specification is valid and the request has been sent

Note

If `true` is returned, the OTA Client state machine will wait until the responseHandler is called

References emberZclDiscByClusterId().

6.79.2.6 bool emberZclOtaBootloadClientServerHasDnsNameCallback (EmberZclOtaBootloadClientServerInfo_t * *serverInfo*)

Start a DNS hostname resolution

Parameters

serverInfo	Updated to the default server parameters

Returns

`true` if the device should use the DNS hostname for the OTA Server

Note

If `true` is returned, the OTA Client state machine will wait for the DNS hostname to be resolved.

6.79.2.7 bool emberZclOtaBootloadClientServerHasStaticAddressCallback (EmberZclOtaBootloadClientServerInfo_t * *serverInfo*)

Check to see if a static IP address should be used for the OTA server

Parameters

serverInfo	Structure to store the OTA server parameters

Returns

`true` if the device should use a static address for the OTA Server

Note

If `true` is returned, the OTA Client state machine will move to the next state, which is querying the OTA Server device for a new OTA file to download. If `false` is returned, the OTA Client will continue to try to discover an OTA Server.

6.79.2.8 bool emberZclOtaBootloadClientSetVersionInfoCallback ( )

This function attempts to set the cluster attribute values to those in the policy parameters. The attributes affected are 'OTA Current File Version', 'Manufacturer ID', and 'Image Type ID'.

Returns

`true` if there was an error setting one of the attributes

6.79.2.9 bool emberZclOtaBootloadClientStartDownloadCallback (const EmberZclOtaBootloadFileSpec_t * *fileSpec*, bool *existingFile*)

A download of an OTA file can be started.

Parameters

fileSpec	The specification for the OTA file to potentially be downloaded.
existingFile	A file with this same specification currently exists in OTA storage.

Returns

`true` if the device should kick off the download process for this file, or `false` if it should continue to query for new images.

See also

emberZclOtaBootloadStorageFind

6.80 OTA Bootload Server Callbacks

Functions

- bool emberZclOtaBootloadServerGetImageNotifyInfoCallback (EmberIpv6Address *address, EmberZcl↩ OtaBootloadFileSpec_t *fileSpec)
- EmberZclStatus_t emberZclOtaBootloadServerGetNextImageCallback (const EmberIpv6Address *source, const EmberZclOtaBootloadFileSpec_t *currentFileSpec, EmberZclOtaBootloadFileSpec_t *nextFileSpec)
- uint32_t emberZclOtaBootloadServerUpgradeEndRequestCallback (const EmberIpv6Address *source, const EmberZclOtaBootloadFileSpec_t *fileSpec, EmberZclStatus_t status)

6.80.1 Detailed Description

These callbacks are contributed by the OTA Bootload Server plugin.

6.80.2 Function Documentation

6.80.2.1 bool emberZclOtaBootloadServerGetImageNotifyInfoCallback (EmberIpv6Address * *address*, EmberZclOtaBootloadFileSpec_t * *fileSpec*)

Get the information needed to send an ImageNotify command.

Parameters

address	Address to which to send the ImageNotify command
fileSpec	The OTA file specification data contained in the ImageNotify command payload. See OTA specification for more details.

Returns

`true` if the ImageNotify command should be sent, `false` otherwise.

Note

If `false` is returned, then the ImageNotify command will be rescheduled to be sent at a later time.

6.80.2.2 EmberZclStatus_t emberZclOtaBootloadServerGetNextImageCallback (const EmberIpv6Address * *source*, const EmberZclOtaBootloadFileSpec_t * *currentFileSpec*, EmberZclOtaBootloadFileSpec_t * *nextFileSpec*)

Get the next OTA file to send in response to a QueryNextImage command.

Parameters

currentFileSpec	The current file spec from the QueryNextImage command
nextFileSpec	The next file spec to be downloaded by the client

Returns

One of the following EmberZclStatus_t values.

- EMBER_ZCL_STATUS_SUCCESS if the server should tell the client to start downloading the OTA file described by the nextFileSpec data
- EMBER_ZCL_STATUS_NO_IMAGE_AVAILABLE if there is no image available for the client to download
- EMBER_ZCL_STATUS_NOT_AUTHORIZED if the client is not authorized to download the next image

References EMBER_ZCL_STATUS_NO_IMAGE_AVAILABLE.

6.80.2.3 uint32_t emberZclOtaBootloadServerUpgradeEndRequestCallback (const EmberIpv6Address * *source*, const EmberZclOtaBootloadFileSpec_t * *fileSpec*, EmberZclStatus_t *status*)

Get the status to send to an OTA client after a download has completed.

Parameters

source	The source address of the OTA client that has completed a download
fileSpec	The file specification of the OTA file that the client has downloaded
status	The status reported by the client upon completing the download

Returns

The time at which the client should upgrade to the newly downloaded image.

Note

This callback is called regardless of whether or not the client completed the download successfully. If the status parameter is not equal to EMBER_ZCL_STATUS_SUCCESS, then the server will not tell the client to proceed with the upgrade.

6.81 Polling Callbacks

Functions

- bool emberAfPluginPollingOkToLongPollCallback (void)

  Ok To Long Poll.

6.81.1 Detailed Description

These callbacks are contributed by the Polling plugin.

6.81.2 Function Documentation

6.81.2.1 bool emberAfPluginPollingOkToLongPollCallback (void)

This function is called by the Polling plugin to determine if the node can wait an extended period of time between polls. Generally, a node can poll infrequently when it does not expect to receive data, via its parent, from other nodes in the network. When data is expected, the node must poll more frequently to avoid having its parent discard stale data due to the MAC indirect transmission timeout (EMBER_INDIRECT_TRANSMISSION_TIMEOUT). The application should return true if it is not expecting data or false otherwise.

6.82 SB1 Gesture Sensor Callbacks

Functions

- void emberAfPluginSb1GestureSensorGestureReceivedCallback (uint8_t gestureReceived, uint8_t switch↩ Number)

  Gesture Received.

6.82.1 Detailed Description

These callbacks are contributed by the SB1 Gesture Sensor plugin.

6.82.2 Function Documentation

6.82.2.1 void emberAfPluginSb1GestureSensorGestureReceivedCallback (uint8_t *gestureReceived,* uint8_t *switchNumber*)

This function is called whenever the sb1 receives a gesture

Parameters

gestureReceived	The (enumerated) gesture received Ver.: always
switchNumber	The switch that received the gesture Ver.: always

6.83 STM32F103RET Library Callbacks

Functions

- void halNcpIsAwakeIsr (bool isAwake)

  The SPI Protocol calls halNcpIsAwakeIsr() once the wakeup handshaking is complete and the NCP is ready to accept a command.

6.83.1 Detailed Description

These callbacks are contributed by the STM32F103RET Library plugin.

6.83.2 Function Documentation

6.83.2.1 void halNcpIsAwakeIsr (bool *isAwake*)

Parameters

isAwake	true if the wake handshake completed and the NCP is awake. false is the wake handshake failed and the NCP is unresponsive.

6.84 Tamper Switch Interface Callbacks

Functions

- void emberAfPluginTamperSwitchTamperActiveCallback (void)

  Tamper Active.
- void emberAfPluginTamperSwitchTamperAlarmCallback (void)

  Tamper Alarm.

6.84.1 Detailed Description

These callbacks are contributed by the Tamper Switch Interface plugin.

6.84.2 Function Documentation

6.84.2.1 void emberAfPluginTamperSwitchTamperActiveCallback (void)

This function is called whenever the tamper switch detects that it has entered the enclosure, thus activating tamper monitoring.

6.84.2.2 void emberAfPluginTamperSwitchTamperAlarmCallback (void)

This function is called when the plugin detects that the enclosure has been opened.

6.85 Gateway MQTT Transport Callbacks

Functions

- void emberAfPluginTransportMqttStateChangedCallback (EmberAfPluginTransportMqttState state)
  MQTT Client State Changed Callback.
- bool emberAfPluginTransportMqttMessageArrivedCallback (const char *topic, const char *payload)
  MQTT Message Arrived.

6.85.1 Detailed Description

These callbacks are contributed by the Gateway MQTT Transport plugin.

6.85.2 Function Documentation

6.85.2.1 bool emberAfPluginTransportMqttMessageArrivedCallback (const char * *topic*, const char * *payload*)

This function will be called when the MQTT client for the gateway receives an incoming message on a topic. If the message is processed by the application true should be returned, if the message is not processed return false. This function is called on a separate thread, so no stack calls should be made within the implementation of this function. Instead use a global variable in that function to communicate the message arrival to a stack event or timer running from the main loop.

Parameters

topic	String contains the topic for the message that arrived. While the underlying MQTT libraries allow NULL characters in a topic, NULL characters are not supported in this implementation so the `topic` parameter can be assumed to be NULL terminated.
payload	String contains the payload for the message that arrived

6.85.2.2 void emberAfPluginTransportMqttStateChangedCallback (EmberAfPluginTransportMqttState *state*)

This function will be called when the state of the MQTT client changes.

Parameters

state	Contains the new and current EmberAfPluginTransportMqttState state

6.86 ZCL Core Callbacks

Functions

- void emberZclGetPublicKeyCallback (const uint8_t **publicKey, uint16_t *publicKeySize)
- bool emberZclPreAttributeChangeCallback (EmberZclEndpointId_t endpointId, const EmberZclClusterSpec_t *clusterSpec, EmberZclAttributeId_t attributeId, const void *buffer, size_t bufferLength)
- void emberZclPostAttributeChangeCallback (EmberZclEndpointId_t endpointId, const EmberZclClusterSpec_t *clusterSpec, EmberZclAttributeId_t attributeId, const void *buffer, size_t bufferLength)
- EmberZclStatus_t emberZclReadExternalAttributeCallback (EmberZclEndpointId_t endpointId, const EmberZclClusterSpec_t *clusterSpec, EmberZclAttributeId_t attributeId, void *buffer, size_t bufferLength)
- EmberZclStatus_t emberZclWriteExternalAttributeCallback (EmberZclEndpointId_t endpointId, const EmberZclClusterSpec_t *clusterSpec, EmberZclAttributeId_t attributeId, const void *buffer, size_t bufferLength)
- void emberZclGetDefaultReportingConfigurationCallback (EmberZclEndpointId_t endpointId, const EmberZclClusterSpec_t *clusterSpec, EmberZclReportingConfiguration_t *configuration)
- void emberZclGetDefaultReportableChangeCallback (EmberZclEndpointId_t endpointId, const EmberZclClusterSpec_t *clusterSpec, EmberZclAttributeId_t attributeId, void *buffer, size_t bufferLength)
- void emberZclNotificationCallback (const EmberZclNotificationContext_t *context, const EmberZclClusterSpec_t *clusterSpec, EmberZclAttributeId_t attributeId, const void *buffer, size_t bufferLength)

6.86.1 Detailed Description

These callbacks are contributed by the ZCL Core plugin.

6.86.2 Function Documentation

6.86.2.1 void emberZclGetDefaultReportableChangeCallback (EmberZclEndpointId_t *endpointId,* const EmberZclClusterSpec_t * *clusterSpec,* EmberZclAttributeId_t *attributeId,* void * *buffer,* size_t *bufferLength*)

Get the default reportable change for an attribute.

Parameters

endpointId	The endpoint to which the reportable change applies
clusterSpec	The cluster to which the reportable change applies
attributeId	The attribute ID
buffer	The data buffer to populate with the reportable change value
bufferLength	The length of the data buffer

This callback gives the application an opportunity to define the reportable change value for each attribute that is being reported. This callback is called when a reporting configuration is reset to its defaults.

See also

emberZclGetDefaultReportingConfigurationCallback

6.86.2.2 void emberZclGetDefaultReportingConfigurationCallback (EmberZclEndpointId_t *endpointId,* const EmberZclClusterSpec_t * *clusterSpec,* EmberZclReportingConfiguration_t * *configuration*)

Get the default reporting configuration for a cluster on an endpoint.

Parameters

endpointId	The endpoint to which the reporting configuration applies
clusterSpec	The cluster to which the reporting configuration applies
configuration	The reporting configuration structure to populate with the default reporting configuration

This callback gives the application an opportunity to define the default reporting configuration for each cluster on each endpoint. The configuration parameter will be passed with default values and the application is expected to update this parameter with its desired reporting configuration. This callback is called when a reporting configuration is reset to its defaults.

See also

emberZclGetDefaultReportableChangeCallback

6.86.2.3 void emberZclGetPublicKeyCallback (const uint8_t ** *publicKey,* uint16_t * *publicKeySize*)

Get the public key used in the application.

Parameters

publicKey	The returned pointer to the public key data
publicKeySize	The returned size of the public key data

The public key data is used to generate this device's UID.

Note

Both the publicKey and publicKeySize parameters are meant to be assigned by the implementation of this call. The expectation is that the public key is a global value, so that the pointer provided by the implementation of this callback will point to constant public key data.

6.86.2.4 void emberZclNotificationCallback (const EmberZclNotificationContext_t * *context,* const EmberZclClusterSpec_t * *clusterSpec,* EmberZclAttributeId_t *attributeId,* const void * *buffer,* size_t *bufferLength*)

A notification has been received.

Parameters

context	Information about the notification
clusterSpec	The cluster to which the attribute applies
attributeId	The attribute ID to which the notification applies
buffer	The data buffer containing the reported attribute value
bufferLength	The length of the data buffer

6.86.2.5 void emberZclPostAttributeChangeCallback (EmberZclEndpointId_t *endpointId*, const EmberZclClusterSpec_t * *clusterSpec*, EmberZclAttributeId_t *attributeId*, const void * *buffer*, size_t *bufferLength*)

An attribute has changed value.

Parameters

endpointId	The endpoint to which the new attribute value applies
clusterSpec	The cluster to which the new attribute value applies
attributeId	The attribute ID
buffer	The data representing the new attribute value
bufferLength	The length of the new attribute value data

This callback gives the application an opportunity to react to an attribute changing value.

See also

emberZclPreAttributeChangeCallback

6.86.2.6 bool emberZclPreAttributeChangeCallback (EmberZclEndpointId_t *endpointId*, const EmberZclClusterSpec_t * *clusterSpec*, EmberZclAttributeId_t *attributeId*, const void * *buffer*, size_t *bufferLength*)

An attribute is about to change value.

Parameters

endpointId	The endpoint to which the new attribute value applies
clusterSpec	The cluster to which the new attribute value applies
attributeId	The attribute ID
buffer	The data representing the new attribute value
bufferLength	The length of the new attribute value data

Returns

`true` if the attribute should take this new value, `false` otherwise.

This callback gives the application an opportunity to prevent an attribute from changing value by returning `false`.

See also

emberZclPostAttributeChangeCallback

6.86.2.7 EmberZclStatus_t emberZclReadExternalAttributeCallback (EmberZclEndpointId_t *endpointId*, const EmberZclClusterSpec_t * *clusterSpec*, EmberZclAttributeId_t *attributeId*, void * *buffer*, size_t *bufferLength*)

An external attribute value needs to be read.

Parameters

endpointId	The endpoint to which the attribute value applies
clusterSpec	The cluster to which the attribute value applies
attributeId	The attribute ID
buffer	The data buffer into which the attribute value will be read
bufferLength	The length of the data buffer

Returns

An EmberZclStatus_t value representing the success or failure of the read operation.

This callback alerts the application that an externally stored attribute needs to be read. The application is expected to read the attribute value from its external storage, populate the buffer parameter with the attribute value, and return an EmberZclStatus_t value representing the success or failure of the read operation.

See also

emberZclWriteExternalAttributeCallback

References EMBER_ZCL_STATUS_UNSUPPORTED_ATTRIBUTE.

6.86.2.8 EmberZclStatus_t emberZclWriteExternalAttributeCallback (EmberZclEndpointId_t *endpointId*, const EmberZclClusterSpec_t * *clusterSpec*, EmberZclAttributeId_t *attributeId*, const void * *buffer*, size_t *bufferLength*)

An external attribute value needs to be written.

Parameters

endpointId	The endpoint to which the attribute value applies
clusterSpec	The cluster to which the attribute value applies
attributeId	The attribute ID
buffer	The data buffer holding the attribute value to be written
bufferLength	The length of the data buffer

Returns

An EmberZclStatus_t value representing the success or failure of the write operation.

This callback alerts the application that an externally stored attribute needs to be written. The application is expected to write the attribute value to its external storage and return an EmberZclStatus_t value representing the success or failure of the write operation.

See also

emberZclReadExternalAttributeCallback

References EMBER_ZCL_STATUS_UNSUPPORTED_ATTRIBUTE.

6.87 buffer-management API Callbacks

Functions

- void * emberAllocateMemoryForPacketHandler (uint32_t size, void **objectRef)

  This function can be used to hook an external memory allocator into the stack. It will be called when we need to allocate large packets.
- void emberFreeMemoryForPacketHandler (void *objectRef)

  This handler is called when freeing memory allocated with emberAllocateMemoryForPacketHandler.
- void emberMarkApplicationBuffersHandler (void)

  Applications that use buffers must mark them by defining this function. The stack uses this when reclaiming unused buffers.

6.87.1 Detailed Description

These callbacks were contributed by the buffer-management API.

6.87.2 Function Documentation

6.87.2.1 void* emberAllocateMemoryForPacketHandler (uint32_t *size*, void ** *objectRef*)

If a value other than NULL is returned that pointer will be used to store packet data. You must also fill in the objectRef parameter with whatever reference you would like passed to the emberFreeMemoryForPacketHandler() when we're done with this memory. If you set the objectRef to NULL then the free handler will not be called.

Parameters

size	size of packet data
objectRef	Reference of the memory to be used in emberFreeMemoryForPacketHandler

Returns

pointer that stores the packet data

References NULL.

6.87.2.2 void emberFreeMemoryForPacketHandler (void * *objectRef*)

Parameters

objectRef	Reference used in emberAllocateMemoryForPacketHandler

6.87.2.3 void emberMarkApplicationBuffersHandler (void)

6.88 coap-diagnostic API Callbacks

Functions

- void emberDiagnosticAnswerHandler (EmberStatus status, const EmberIpv6Address *remoteAddress, const uint8_t *payload, uint8_t payloadLength)

  Application callback for emberSendDiagnosticQuery().

6.88.1 Detailed Description

These callbacks were contributed by the coap-diagnostic API.

6.88.2 Function Documentation

6.88.2.1 void emberDiagnosticAnswerHandler (EmberStatus *status*, const EmberIpv6Address * *remoteAddress*, const uint8_t * *payload*, uint8_t *payloadLength*)

Application callback for emberSendDiagnosticQuery() and emberSendDiagnosticGet().

Parameters

status	The return status.
remoteAddress	The remote address that sent the answer.
payload	The raw payload.
payloadLength	payload's length.

6.89 connection-manager API Callbacks

Functions

- void emberConnectionManagerConnectCompleteCallback (EmberConnectionManagerConnectionStatus status)

  Connection attempt completed.

6.89.1 Detailed Description

These callbacks were contributed by the connection-manager API.

6.89.2 Function Documentation

6.89.2.1 void emberConnectionManagerConnectCompleteCallback (EmberConnectionManagerConnectionStatus *status*)

This function is called when an attempt to connect to a network has completed. It will convery the result of an attempt to join a network using the emberConnectionManagerStartConnect function. The status will be one of the following values:

EMBER_CONNECTION_MANAGER_STATUS_CONNECTED: The device succesfully attached to the network. EMBER_CONNECTION_MANAGER_STATUS_TIMED_OUT: The device was unable to join the network after attempting the number of times specified in the connection manager plugin options.

6.90 host-mfglib API Callbacks

Functions

- void mfglibEndReturn (EmberStatus status, uint32_t receiveCount)

  This function provides the result of a call to mfglibEnd().
- void mfglibGetChannelReturn (uint8_t channel)

  This function provides the result of a call to mfglibGetChannel().
- void mfglibGetOptionsReturn (uint8_t options)

  This function provides the result of a call to mfglibGetOptions().
- void mfglibGetPowerModeReturn (uint16_t txPowerMode)

  This function provides the result of a call to mfglibGetPowerMode().
- void mfglibGetPowerReturn (int8_t power)

  This function provides the result of a call to mfglibGetPower().
- void mfglibGetSynOffsetReturn (int8_t synthOffset)

  This function provides the result of a call to mfglibGetSynOffset().
- void mfglibRxHandler (uint8_t *packet, uint8_t linkQuality, int8_t rssi)

  RX Handler for the mfglib test library.
- void mfglibSendPacketReturn (EmberStatus status)

  This function provides the result of a call to mfglibSendPacket().
- void mfglibSetChannelReturn (EmberStatus status)

  This function provides the result of a call to mfglibSetChannel().
- void mfglibSetOptionsReturn (EmberStatus status)

  This function provides the result of a call to mfglibSetOptions().
- void mfglibSetPowerReturn (EmberStatus status)

  This function provides the result of a call to mfglibSetPower().
- void mfglibStartReturn (EmberStatus status)

  This function provides the result of a call to mfglibStart().
- void mfglibStartStreamReturn (EmberStatus status)

  This function provides the result of a call to mfglibStartStream().
- void mfglibStartToneReturn (EmberStatus status)

  This function provides the result of a call to mfglibStartTone().
- void mfglibStopStreamReturn (EmberStatus status)

  This function provides the result of a call to mfglibStopStream().
- void mfglibStopToneReturn (EmberStatus status)

  This function provides the result of a call to mfglibStopTone().

6.90.1 Detailed Description

These callbacks were contributed by the host-mfglib API.

6.90.2 Function Documentation

6.90.2.1 void mfglibEndReturn (EmberStatus *status*, uint32_t *receiveCount*)

Parameters

status	• EMBER_SUCCESS if the mfg test mode has been exited. • EMBER_ERR_FATAL if the mfg test mode cannot be exited.
receiveCount	The total number of packets received during the test.

6.90.2.2 void mfglibGetChannelReturn (uint8_t *channel*)

Parameters

channel	The current channel.

6.90.2.3 void mfglibGetOptionsReturn (uint8_t *options*)

Parameters

options	The current options based on the current test mode.

6.90.2.4 void mfglibGetPowerModeReturn (uint16_t *txPowerMode*)

Parameters

txPowerMode	The current power mode setting.

6.90.2.5 void mfglibGetPowerReturn (int8_t *power*)

Parameters

power	The current power setting.

6.90.2.6 void mfglibGetSynOffsetReturn (int8_t *synthOffset*)

Parameters

synthOffset	The synth offset in 11.7kHz steps.

6.90.2.7 void mfglibRxHandler (uint8_t * *packet*, uint8_t *linkQuality*, int8_t *rssi*)

Parameters

packet	incoming packet
linkQuality	link quality as a numeric value
rssi	RSSI in dBm

6.90.2.8 void mfglibSendPacketReturn (EmberStatus *status*)

Parameters

status	• EMBER_SUCCESS if the packet was sent. • EMBER_ERR_FATAL if the mfg test mode is not available or TONE or STREAM test is running.

6.90.2.9 void mfglibSetChannelReturn (EmberStatus *status*)

Parameters

status	• EMBER_SUCCESS if the channel has been set. • ::EMBER_ERROR_INVALID_CHANNEL if the channel requested is invalid. • EMBER_ERR_FATAL if the mfg test mode is not available or TONE or STREAM test is running.

6.90.2.10 void mfglibSetOptionsReturn (EmberStatus *status*)

Parameters

status	• EMBER_SUCCESS if the options have been set. • EMBER_BAD_ARGUMENT if any options are unavailable. • EMBER_ERR_FATAL if the mfg test mode is not available or TONE or STREAM test is running.

6.90.2.11 void mfglibSetPowerReturn (EmberStatus *status*)

Parameters

status	• EMBER_SUCCESS if the power has been set. • ::EMBER_ERROR_INVALID_POWER if the power requested is invalid. • EMBER_ERR_FATAL if the mfg test mode is not available or TONE or STREAM test is running.

6.90.2.12 void mfglibStartReturn (EmberStatus *status*)

Parameters

status	• EMBER_SUCCESS if the mfg test mode has been enabled. • EMBER_ERR_FATAL if the mfg test mode is not available.

6.90.2.13 void mfglibStartStreamReturn (EmberStatus *status*)

Parameters

status	• EMBER_SUCCESS if the transmit stream has started. • EMBER_ERR_FATAL if the stream cannot be started.

6.90.2.14 void mfglibStartToneReturn (EmberStatus *status*)

Parameters

status	• EMBER_SUCCESS if the transmit tone has started. • EMBER_ERR_FATAL if the tone cannot be started.

6.90.2.15 void mfglibStopStreamReturn (EmberStatus *status*)

Parameters

status	• EMBER_SUCCESS if the transmit stream has stopped. • EMBER_ERR_FATAL if the stream cannot be stopped.

6.90.2.16 void mfglibStopToneReturn (EmberStatus *status*)

Parameters

status	• EMBER_SUCCESS if the transmit tone has stopped. • EMBER_ERR_FATAL if the tone cannot be stopped.

6.91 icmp API Callbacks

Functions

- void emberIncomingIcmpHandler (Ipv6Header *ipHeader)
  Application callback for an incoming ICMP message.

6.91.1 Detailed Description

These callbacks were contributed by the icmp API.

6.91.2 Function Documentation

6.91.2.1 void emberIncomingIcmpHandler (Ipv6Header * *ipHeader*)

An application callback for an incoming ICMP message.

Parameters

ipHeader	Pointer to an IPV6 buffer

6.92 network-management API Callbacks

Functions

- void emberActiveScanHandler (const EmberMacBeaconData *beaconData)

  Reports an incoming beacon during an active scan.
- void emberAddressConfigurationChangeHandler (const EmberIpv6Address *address, uint32_t preferred↩ Lifetime, uint32_t validLifetime, uint8_t addressFlags)

  This is called when a new address is configured on the application.
- void emberAttachToNetworkReturn (EmberStatus status)

  A callback that indicates whether the attach process was successfully initiated via a prior call to emberAttachTo↩ Network(). The status argument is either EMBER_SUCCESS, or EMBER_INVALID_CALL if attach was called when the network status was not EMBER_JOINED_NETWORK_NO_PARENT, or while an attach was underway.
- void emberBecomeCommissionerReturn (EmberStatus status)

  Return call for emberBecomeCommissioner(). The status is EMBER_SUCCESS if a petition was sent or EMBER↩ _ERR_FATAL if some temporary resource shortage prevented doing so.
- void emberChangeNodeTypeReturn (EmberStatus status)

  Provides the result of a call to emberChangeNodeType(): either EMBER_SUCCESS, or EMBER_INVALID_CALL.
- void emberAllowNativeCommissionerReturn (EmberStatus status)

  Provides the result of a call to emberAllowNativeCommissioner(): either EMBER_SUCCESS or EMBER_INVALID↩ _CALL.
- void emberSetCommissionerKeyReturn (EmberStatus status)

  Provides the result of a call to emberSetCommissionerKey(): either EMBER_SUCCESS or EMBER_INVALID_CALL.
- void emberSetPskcHandler (const uint8_t *pskc)

  Handler to let application know that a PSKc TLV was successfully set.
- void emberSetJoinKeyReturn (EmberStatus status)

  Provides the result of a call to emberSetJoinKey(): either EMBER_SUCCESS or EMBER_BAD_ARGUMENT.
- void emberCommissionNetworkReturn (EmberStatus status)

  Provides the result of a call to emberCommissionNetwork.
- void emberCommissionerStatusHandler (uint16_t flags, const uint8_t *commissionerName, uint8_↩ t commissionerNameLength)

  Reports on the current commissioner state.
- void emberConfigureGatewayReturn (EmberStatus status)

  Provides the result of a call to emberConfigureGateway.
- void emberSetNdDataReturn (EmberStatus status, uint16_t length)

  Provides the result of a call to emberSetNdData.
- void emberSetLocalNetworkDataReturn (EmberStatus status, uint16_t length)

  Provides the result of a call to emberSetLocalNetworkData.
- void emberConfigureExternalRouteReturn (EmberStatus status)

  Provides the result of a call to emberConfigureExternalRoute.
- void emberCounterHandler (EmberCounterType type, uint16_t increment)

  A callback invoked to inform the application of the occurrence of an event defined by EmberCounterType, for example, transmissions and receptions at different layers of the stack.
- uint16_t emberCounterValueHandler (EmberCounterType type)

  A callback invoked to query the application for the countervalue of an event defined by EmberCounterType.
- void emberCustomHostToNcpMessageHandler (const uint8_t *message, uint8_t messageLength)

  NCP handler called to process a custom message from the Host.
- void emberCustomNcpToHostMessageHandler (const uint8_t *message, uint8_t messageLength)

  Host handler called to process a custom message from the NCP.
- void emberDeepSleepCompleteHandler (uint16_t sleepDuration)

  For a sleepy end device, report how long the chip went to deep sleep. In a NCP + host setup, the stack reports this to the host app.

- void emberDeepSleepReturn (EmberStatus status)

  Provides the result of a call to emberDeepSleep().
- void emberDhcpServerChangeHandler (const uint8_t *prefix, uint8_t prefixLengthInBits, bool available)

  This is called when the stack knows about a new dhcp server or if a dhcp server has become unavailable.
- void emberEnergyScanHandler (uint8_t channel, int8_t maxRssiValue)

  Reports the maximum RSSI value measured on the channel.
- void emberEventDelayUpdatedFromIsrHandler (Event *event)

  This method is called any time an event is scheduled from within an ISR context. It can be used to determine when to stop a long running sleep to see what application or stack events now need to be processed.
- void emberExternalRouteChangeHandler (const uint8_t *prefix, uint8_t prefixLengthInBits, bool available)

  This is called when the stack knows about a border router that has an external route to a prefix.
- void emberFormNetworkReturn (EmberStatus status)

  A callback that indicates whether a prior call to emberFormNetwork() successfully initiated the form process. The status argument is either EMBER_SUCCSS, or EMBER_INVALID_CALL if resume was called when the network status was not EMBER_NO_NETWORK, or a scan was underway.
- void emberGetAntennaModeReturn (EmberStatus status, uint8_t mode)

  Provides the result of a call to emberGetAntennaMode.
- void emberGetCcaThresholdReturn (int8_t threshold)

  Provides the result of a call to emberGetCcaThreshold().
- void emberGetChannelCalDataTokenReturn (uint8_t lna, int8_t tempAtLna, uint8_t modDac, int8_t tempAt↩ ModDac)

  Gets the token information for tokenId = EMBER_CHANNEL_CAL_DATA_TOKEN.
- void emberGetCounterReturn (EmberCounterType type, uint16_t value)

  Provides the result of a call to emberGetCounter().
- void emberGetCtuneReturn (uint16_t tune, EmberStatus status)

  Provides the result of a call to emberGetCtune.
- void emberGetGlobalAddressReturn (const EmberIpv6Address *address, uint32_t preferredLifetime, uint32_t validLifetime, uint8_t addressFlags)

  Provides the result of a call to emberGetGlobalAddresses.
- void emberGetGlobalPrefixReturn (uint8_t flags, bool isStable, const uint8_t *prefix, uint8_t prefixLengthIn↩ Bits, uint8_t domainId, uint32_t preferredLifetime, uint32_t validLifetime)

  Provides the result of a call to ::emberGetGlobalPrefix.
- void emberGetMfgTokenReturn (EmberMfgTokenId tokenId, EmberStatus status, const uint8_t *tokenData, uint8_t tokenDataLength)

  Provides the result of a call to emberGetMfgToken.
- void emberGetNetworkDataReturn (EmberStatus status, uint8_t *networkData, uint16_t bufferLength)

  Provides the result of a call to emberGetNetworkData.
- void emberGetNetworkDataTlvReturn (uint8_t typeByte, uint8_t index, uint8_t versionNumber, const uint8_t *tlv, uint8_t tlvLength)

  Provides the result of a call to emberGetNetworkDataTlv().
- void emberGetPtaEnableReturn (bool enabled)

  Provides the result of a call to emberGetPtaEnable.
- void emberGetPtaOptionsReturn (uint32_t options)

  Provides the result of a call to emberGetPtaOptions.
- void emberGetRadioPowerReturn (int8_t power)

  Provides the result of a call to emberGetRadioPower() on the host.
- void emberGetRipEntryReturn (uint8_t index, const EmberRipEntry *entry)

  Provides the result of a call to emberGetRipEntry().
- void emberGetRoutingLocatorReturn (const EmberIpv6Address *rloc)

  Provides the result of a call to emberGetRoutingLocator.
- void emberGetStandaloneBootloaderInfoReturn (uint16_t version, uint8_t platformId, uint8_t microId, uint8↩ _t phyId)

Provides the result of a call to emberGetStandaloneBootloaderInfo.

- void emberGetTxPowerModeReturn (uint16_t txPowerMode)

  Provides the result of a call to emberGetTxPowerMode() on the host.
- void emberGetVersionsReturn (const uint8_t *versionName, uint16_t managementVersionNumber, uint16_t stackVersionNumber, uint16_t stackBuildNumber, EmberVersionType versionType, const uint8_t *buildTimestamp)

  Provides the result of a call to emberGetVersions().
- void emberHostStateHandler (const EmberNetworkParameters *parameters, const EmberEui64 *localEui64, const EmberEui64 *macExtendedId, EmberNetworkStatus networkStatus)

  In a host/NCP setup, notifies the host to changes in the network parameters.
- void emberInitReturn (EmberStatus status)

  Provides the result of a call to emberInit().
- void emberJoinNetworkReturn (EmberStatus status)

  A callback that indicates whether the join process was successfully initiated via a prior call to emberJoinNetwork() or emberJoinCommissioned(). The possible EmberStatus values are: EMBER_SUCCESS, EMBER_BAD_ARGUMENT, or EMBER_INVALID_CALL (if join was called when the network status was something other than EMBER_NO_NETWORK).
- void emberLaunchStandaloneBootloaderReturn (EmberStatus status)

  Provides the result of a call to emberLaunchStandaloneBootloader.
- void emberLeaderDataHandler (const uint8_t *leaderData)

  A callback invoked when the leader data changes.
- bool emberMacPassthroughFilterHandler (uint8_t *macHeader)

  Application handler to define "passthrough" packets.
- void emberMacPassthroughMessageHandler (PacketHeader header)

  Application handler to intercept "passthrough" packets and handle them at the application.
- bool emberMacRssiFilterHandler (uint8_t *macHeader)

  Application handler to filter 802.15.4 packets to be observed for signal strength.
- void emberMacRssiHandler (int8_t currentRssi)

  Gets the received signal strength indication (RSSI) for the last 802.15.4 packet received by the stack.
- void emberNetworkDataChangeHandler (const uint8_t *networkData, uint16_t length)

  This is called when the stack receives new Thread Network Data.
- void emberNetworkStatusHandler (EmberNetworkStatus newNetworkStatus, EmberNetworkStatus oldNetworkStatus, EmberJoinFailureReason reason)

  Reports a change to the network status. For example, the network status changes while going through the joining process, or while reattaching to the network, which can happen for a variety of reasons. In particular, after issuing a form, join, resume, or attach command, the application knows that the device is on the network and ready to communicate when this handler is called with a newNetworkStatus of EMBER_JOINED_NETWORK_ATTACHED.
- void emberOkToNapReturn (uint8_t stateMask)

  If implementing event-driven sleep on an NCP host, this method will return the bitmask indicating the stack's current tasks. (see enum above)
- void emberPollForDataReturn (EmberStatus status)

  Provides the result of a call to emberPollForData().
- void emberRadioGetRandomNumbersReturn (EmberStatus status, const uint16_t *rn, uint8_t count)

  Provides the result of a call to emberRadioGetRandomNumbers.
- void emberRequestDhcpAddressReturn (EmberStatus status, const uint8_t *prefix, uint8_t prefixLengthInBits)

  Provides the result of a call to emberRequestDhcpAddress.
- void emberRequestSlaacAddressReturn (EmberStatus status, const uint8_t *prefix, uint8_t prefixLengthInBits)

  Provides the result of a call to emberRequestSlaacAddress.
- void emberResetMicroHandler (EmberResetCause cause)

  Notifies the application of a reset on the Ember chip due to the indicated cause.
- void emberResetNetworkStateReturn (EmberStatus status)

Provides the result of a call to emberResetNetworkState().

- void emberResignGlobalAddressReturn (EmberStatus status)

  Provides the result of a call to emberResignGlobalAddress().
- void emberResumeNetworkReturn (EmberStatus status)

  A callback that indicates whether a prior call to emberResumeNetwork() successfully initiated the resume process. The status argument is either EMBER_SUCCESS, or EMBER_INVALID_CALL if resume was called when the network status was not EMBER_SAVED_NETWORK, or while a scan was underway.
- void emberScanReturn (EmberStatus status)

  Provides the status upon completion of a scan.
- void emberSendSteeringDataReturn (EmberStatus status)

  Provides the result of a call to emberSendSteeringData().
- void emberSetAntennaModeReturn (EmberStatus status)

  Provides the result of a call to emberSetAntennaMode.
- void emberSetCcaThresholdReturn (EmberStatus status)

  Provides the result of a call to emberSetCcaThreshold().
- void emberSetCtuneReturn (EmberStatus status)

  Provides the result of a call to emberSetCtune.
- void emberSetMfgTokenReturn (EmberMfgTokenId tokenId, EmberStatus status)

  Provides the result of a call to emberSetMfgToken.
- void emberSetPtaEnableReturn (EmberStatus status)

  Provides the result of a call to emberSetPtaEnable.
- void emberSetPtaOptionsReturn (EmberStatus status)

  Provides the result of a call to emberSetPtaOptions.
- void emberSetRadioHoldOffReturn (EmberStatus status)

  Provides the result of a call to emberSetRadioHoldOff.
- void emberSetRadioPowerReturn (EmberStatus status)

  Provides the result of a call to emberSetRadioPower() on the host.
- void emberSetSecurityParametersReturn (EmberStatus status)

  Provides the result of a call to emberSetSecurityParameters().
- void emberSetTxPowerModeReturn (EmberStatus status)

  Provides the result of a call to emberSetTxPowerMode() on the host.
- void emberSlaacServerChangeHandler (const uint8_t *prefix, uint8_t prefixLengthInBits, bool available)

  This is called when the stack knows about a new SLAAC prefix or if a SLAAC server has become unavailable.
- void emberStackPollForDataReturn (EmberStatus status)

  Provides the result of a call to emberStackPollForData().
- void emberStartHostJoinClientHandler (const uint8_t *parentAddress)

  Callback to tell the host to start security commissioning.
- void emberStateReturn (const EmberNetworkParameters *parameters, const EmberEui64 *localEui64, const EmberEui64 *macExtendedId, EmberNetworkStatus networkStatus)

  In a host/NCP setup, provides the result of a call to emberState() on the host.
- void emberSwitchToNextNetworkKeyHandler (EmberStatus status)

  This can be stubbed out on the SoC and host app. It is used by the NCP to update security on the driver when it is instructed to switch the network key by an over the air update.
- void emberSwitchToNextNetworkKeyReturn (EmberStatus status)

  Provides the result of a call to emberSwitchToNextNetworkKey().
- void emberSetDtlsDeviceCertificateReturn (uint32_t result)

  Provides the result of a call to emberSetDtlsDeviceCertificate().
- void emberSetDtlsPresharedKeyReturn (EmberStatus status)

  Provides the result of a call to emberSetDtlsPresharedKey().
- void emberOpenDtlsConnectionReturn (uint32_t result, const EmberIpv6Address *remoteAddress, uint16_t localPort, uint16_t remotePort)

Provides the result of a call to emberOpenDtlsConnection().

- void emberDtlsSecureSessionEstablished (uint8_t flags, uint8_t sessionId, const EmberIpv6Address *local↩ Address, const EmberIpv6Address *remoteAddress, uint16_t localPort, uint16_t remotePort)

  Indicates to the application that a secure connection was successfully established.
- void emberGetSecureDtlsSessionIdReturn (uint8_t sessionId, const EmberIpv6Address *remoteAddress, uint16_t localPort, uint16_t remotePort)

  Provides the result of a call to emberGetSecureDtlsSessionId().
- void emberCloseDtlsConnectionReturn (uint8_t sessionId, EmberStatus status)

  Provides the result of a call to emberCloseDtlsConnection(), or indicates that the connection was closed on the other end.
- void emberProcessCoap (const uint8_t *message, uint16_t messageLength, EmberCoapRequestInfo *info)

  Process a CoAP message received over an alternate transport.
- void emberMicroBusyHandler (bool busy)

  Callback informing the application running on the micro of interruptions to normal processing. If ::busy is true, the micro will be busy processing and unavailable for an indefinite period of time. If ::busy is false, the micro has resumed normal operation. The main use case is jpake crypto on EM3xx processors. This gives the application a chance to prepare for the pause in regular processing.

6.92.1 Detailed Description

These callbacks were contributed by the network-management API.

6.92.2 Function Documentation

6.92.2.1 void emberActiveScanHandler (const EmberMacBeaconData * *beaconData*)

This function reports an incoming beacon during an active scan.

6.92.2.2 void emberAddressConfigurationChangeHandler (const EmberIpv6Address * *address*, uint32_t *preferredLifetime*, uint32_t *validLifetime*, uint8_t *addressFlags*)

This function is called when a new address is configured on the application.

If addressFlags is EMBER_LOCAL_ADDRESS, it means that the address configured is a Thread-local address.

Otherwise, it means that the address assigned is a global address (DHCP or SLAAC).

In either case, if the valid lifetime is zero, then the address is no longer available.

Parameters

address	the address
preferredLifetime	the preferred lifetime of the address (in seconds)
validLifetime	the valid lifetime of the address (in seconds)
addressFlags	address configuration flags (see LocalServerFlag_e)

6.92.2.3 void emberAllowNativeCommissionerReturn (EmberStatus *status*)

This function provides the result of a call to emberAllowNativeCommissioner(): either EMBER_SUCCESS or EM↩ BER_INVALID_CALL.

6.92.2.4 void emberAttachToNetworkReturn (EmberStatus *status*)

6.92.2.5 void emberBecomeCommissionerReturn (EmberStatus *status*)

6.92.2.6 void emberChangeNodeTypeReturn (EmberStatus *status*)

This function provides the result of a call to emberChangeNodeType(): either EMBER_SUCCESS, or EMBER_I↩ NVALID_CALL.

6.92.2.7 void emberCloseDtlsConnectionReturn (uint8_t *sessionId,* EmberStatus *status*)

Parameters

session↩ Id	sessionId used for secure CoAP transport.
status	• EMBER_SUCCESS - Successfully closed the connection • EMBER_INVALID_CALL - Invalid session ID • EMBER_ERR_FATAL - Fatal error closing the connection

6.92.2.8 void emberCommissionerStatusHandler (uint16_t *flags,* const uint8_t * *commissionerName,* uint8_t *commissionerNameLength*)

This function reports on the current commissioner state.

Parameters

flags	A combination of zero or more of the following: • EMBER_HAVE_COMMISSIONER a commissioner is active in the network • EMBER_AM_COMMISSIONER this device is the active commissioner if emberStopCommissioning is called, then this flag is not returned as we are open to commissioner petitions • EMBER_JOINING_ENABLED joining is enabled • EMBER_JOINING_WITH_EUI_STEERING steering data restricts which devices can join. if not set, no restriction, any device can join (significant only when EMBER_JOINING_ENABLED is set)
commissionerName	The name of the active commissioner, or NULL if there is none or the name is not known.
commissionerNameLength	The length of commissonerName.

6.92.2.9 void emberCommissionNetworkReturn (EmberStatus *status*)

This function provides the result of a call to emberCommissionNetwork.

Returns EMBER_SUCCESS if successful EMBER_BAD_ARGUMENT if any of the options are wrong EMBER_I↩ NVALID_CALL if the node is already on a network

Parameters

status	Whether the call to emberCommissionNetwork was successful

6.92.2.10 void emberConfigureExternalRouteReturn (EmberStatus *status*)

This function provides the result of a call to emberConfigureExternalRoute.

6.92.2.11 void emberConfigureGatewayReturn (EmberStatus *status*)

This function provides the result of a call to emberConfigureGateway.

6.92.2.12 void emberCounterHandler (EmberCounterType *type*, uint16_t *increment*)

The application must define EMBER_APPLICATION_HAS_COUNTER_HANDLER in its CONFIGURATION_HE↩ ADER to use this. This function may be called in ISR context, so processing should be kept to a minimum.

Parameters

type	The type of the event.
increment	Specify the increase in the counter's tally.

6.92.2.13 uint16_t emberCounterValueHandler (EmberCounterType *type*)

The application must define EMBER_APPLICATION_HAS_COUNTER_VALUE_HANDLER in its CONFIGURAT↩ ION_HEADER to use this.

Parameters

type	The type of the event.

Returns

The counter's tally.

6.92.2.14 void emberCustomHostToNcpMessageHandler (const uint8_t * *message*, uint8_t *messageLength*)

Parameters

message	message received
messageLength	length of message

6.92.2.15 void emberCustomNcpToHostMessageHandler (const uint8_t * *message*, uint8_t *messageLength*)

Parameters

message	message received
messageLength	length of message

6.92.2.16 void emberDeepSleepCompleteHandler (uint16_t *sleepDuration*)

6.92.2.17 void emberDeepSleepReturn (EmberStatus *status*)

This function provides the result of a call to emberDeepSleep().

6.92.2.18 void emberDhcpServerChangeHandler (const uint8_t * *prefix*, uint8_t *prefixLengthInBits*, bool *available*)

This function is called when the stack knows about a new dhcp server or if a dhcp server has become unavailable.

"available" means the DHCP server can offer us an address if requested.

Parameters

prefix	dhcp server prefix
prefixLengthInBits	length in bits of the prefix
available	whether this dhcp server is available

6.92.2.19 void emberDtlsSecureSessionEstablished (uint8_t *flags*, uint8_t *sessionId*, const EmberIpv6Address * *localAddress*, const EmberIpv6Address * *remoteAddress*, uint16_t *localPort*, uint16_t *remotePort*)

Parameters

flags	1 = server, 0 = client (possibly other info later)
sessionId	sessionId used for secure CoAP transport
localAddress	local IPv6 address
remoteAddress	remote IPv6 address
localPort	local port
remotePort	remote port

6.92.2.20 void emberEnergyScanHandler (uint8_t *channel*, int8_t *maxRssiValue*)

This function reports the maximum RSSI value measured on the channel.

Parameters

channel	The 802.15.4 channel on which the RSSI value was measured.
maxRssiValue	The maximum RSSI value measured (in units of dBm).

6.92.2.21 void emberEventDelayUpdatedFromIsrHandler (Event * *event*)

Parameters

event	The event that was scheduled by the ISR.

6.92.2.22 void emberExternalRouteChangeHandler (const uint8_t * *prefix*, uint8_t *prefixLengthInBits*, bool *available*)

This function is called when the stack knows about a border router that has an external route to a prefix.

Parameters

prefix	external route prefix
prefixLengthInBits	length in bits of the prefix
available	whether this external route is available.

6.92.2.23 void emberFormNetworkReturn (EmberStatus *status*)

6.92.2.24 void emberGetAntennaModeReturn (EmberStatus *status*, uint8_t *mode*)

This function provides the result of a call to emberGetAntennaMode.

Parameters

status	An EmberStatus value indicating success or the reason for failure.
mode	the current antenna mode, 0-primary, 1-secondary, 2-toggle on tx ack fail

6.92.2.25 void emberGetCcaThresholdReturn (int8_t *threshold*)

This function provides the result of a call to emberGetCcaThreshold().

6.92.2.26 void emberGetChannelCalDataTokenReturn (uint8_t *lna*, int8_t *tempAtLna*, uint8_t *modDac*, int8_t *tempAtModDac*)

This function gets the token information for tokenId = EMBER_CHANNEL_CAL_DATA_TOKEN.

Parameters

lna	[msb: cal needed? | bit 0-5: lna tune value]
tempAtLna	[the temp (degC) when the LNA was calibrated] #param modDac [msb: cal needed? | bit
	0-5: modulation DAC tune value]
tempAtModDac	[the temp (degC) when the mod DAC was calibrated]

6.92.2.27 void emberGetCounterReturn (EmberCounterType *type*, uint16_t *value*)

This function provides the result of a call to emberGetCounter().

6.92.2.28 void emberGetCtuneReturn (uint16_t *tune*, EmberStatus *status*)

This function provides the result of a call to emberGetCtune.

Parameters

tune	The current CTUNE value.
status	An EmberStatus value indicating success or the reason for failure.

6.92.2.29 void emberGetGlobalAddressReturn (const EmberIpv6Address * *address*, uint32_t *preferredLifetime*, uint32_t *validLifetime*, uint8_t *addressFlags*)

This function provides the result of a call to emberGetGlobalAddresses.

Parameters

address	IPv6 global address
preferredLifetime	Preferred lifetime (in seconds)
validLifetime	Valid lifetime (in seconds)
addressFlags	Address configuration flags (EMBER_GLOBAL_ADDRESS_*)

6.92.2.30 void emberGetGlobalPrefixReturn (uint8_t *flags*, bool *isStable*, const uint8_t * *prefix*, uint8_t *prefixLengthInBits*, uint8_t *domainId*, uint32_t *preferredLifetime*, uint32_t *validLifetime*)

This function provides the result of a call to ::emberGetGlobalPrefix.

Parameters

flags	Please ignore this param, it is currently unused. (returns 0)
isStable	Stable or temporary prefix
prefix	Border router prefix
prefixLengthInBits	Prefix length in bits
domainId	Provisioning domain ID
preferredLifetime	Preferred lifetime (in seconds)
validLifetime	Valid lifetime (in seconds)

6.92.2.31 void emberGetMfgTokenReturn (EmberMfgTokenId *tokenId*, EmberStatus *status*, const uint8_t * *tokenData*, uint8_t *tokenDataLength*)

This function provides the result of a call to emberGetMfgToken.

Parameters

tokenId	Which manufacturing token read.
status	An EmberStatus value indicating success or the
tokenData	The manufacturing token data.
tokenDataLength	The length of the *tokenData* parameter in bytes.

6.92.2.32 void emberGetNetworkDataReturn (EmberStatus *status*, uint8_t * *networkData*, uint16_t *bufferLength*)

This function provides the result of a call to emberGetNetworkData.

The status value is one of:

- EMBER_SUCCESS
- EMBER_NETWORK_DOWN
- EMBER_BAD_ARGUMENT (the supplied buffer was too small)

Parameters

status	
networkData	location of the Network Data
dataLength	length in bytes of the Network Data

6.92.2.33 void emberGetNetworkDataTlvReturn (uint8_t *typeByte*, uint8_t *index*, uint8_t *versionNumber*, const uint8_t * *tlv*, uint8_t *tlvLength*)

This function provides the result of a call to emberGetNetworkDataTlv().

Parameters

type	the type of TLV returned. This is the same value as the value specified in the emberGetNetworkDataTlv() call.
index	the instance number of the TLV. This is the same value as the value specified in the emberGetNetworkDataTlv() call.
versionNumber	the network data version
tlv	the TLV corresponding to type or NULL.
tlvLength	length of tlv

6.92.2.34 void emberGetPtaEnableReturn (bool *enabled*)

This function provides the result of a call to emberGetPtaEnable.

Parameters

enabled	When true, indicates packet traffic arbitration is enabed. When false, indicates packet traffic arbitration is disabled.

6.92.2.35 void emberGetPtaOptionsReturn (uint32_t *options*)

This function provides the result of a call to emberGetPtaOptions.

Parameters

indicates	packet traffic arbitration options bit field. Field Bit Position Size(bits) RX retry timeout ms 0 8 Enable ack radio holdoff 8 1 Abort mid TX if grant is lost 9 1 TX request is high priority 10 1 RX request is high prioirity 11 1 RX retry request is high priority 12 1 RX retry request is enabled 13 1 Radio holdoff is enabled 14 1 Toggle request on mac retransmit 15 1 Reserved 16 15 Hold request across CCA failures 31 1

6.92.2.36 void emberGetRadioPowerReturn (int8_t *power*)

This function provides the result of a call to emberGetRadioPower() on the host.

6.92.2.37 void emberGetRipEntryReturn (uint8_t *index*, const **EmberRipEntry** * *entry*)

This function provides the result of a call to emberGetRipEntry().

6.92.2.38 void emberGetRoutingLocatorReturn (const **EmberIpv6Address** * *rloc*)

This function provides the result of a call to emberGetRoutingLocator.

Parameters

rloc	The Routing Locator as a full IPv6 address.

6.92.2.39 void emberGetSecureDtlsSessionIdReturn (uint8_t *sessionId*, const **EmberIpv6Address** * *remoteAddress*, uint16_t *localPort*, uint16_t *remotePort*)

Parameters

sessionId	sessionId used for secure CoAP transport
remoteAddress	remote IPv6 address
localPort	local port
remotePort	remote port

6.92.2.40 void emberGetStandaloneBootloaderInfoReturn (uint16_t *version*, uint8_t *platformId*, uint8_t *microId*, uint8_t *phyId*)

This function provides the result of a call to emberGetStandaloneBootloaderInfo.

Parameters

version	BOOTLOADER_INVALID_VERSION if the standalone bootloader is not present, or the version of the installed standalone bootloader.
platformId	The value of PLAT on the node.
microId	The value of MICRO on the node.
phyId	The value of PHY on the node.

6.92.2.41 void emberGetTxPowerModeReturn (uint16_t *txPowerMode*)

This function provides the result of a call to emberGetTxPowerMode() on the host.

Returns

the current tx power mode.

6.92.2.42 void emberGetVersionsReturn (const uint8_t * *versionName*, uint16_t *managementVersionNumber*, uint16_t *stackVersionNumber*, uint16_t *stackBuildNumber*, EmberVersionType *versionType*, const uint8_t * *buildTimestamp*)

6.92.2.43 void emberHostStateHandler (const EmberNetworkParameters * *parameters*, const EmberEui64 * *localEui64*, const EmberEui64 * *macExtendedId*, EmberNetworkStatus *networkStatus*)

Parameters

parameters	Current network parameters
localEui64	The EUI64 of the Ember chip
macExtendedId	The extended MAC ID of the Ember chip
networkStatus	The current status of the network

6.92.2.44 void emberInitReturn (EmberStatus *status*)

This function provides the result of a call to emberInit().

6.92.2.45 void emberJoinNetworkReturn (EmberStatus *status*)

6.92.2.46 void emberLaunchStandaloneBootloaderReturn (EmberStatus *status*)

This function provides the result of a call to emberLaunchStandaloneBootloader.

Parameters

status	An EmberStatus value indicating success or the reason for failure.

6.92.2.47 void emberLeaderDataHandler (const uint8_t * *leaderData*)

Parameters

leaderData	the leader data

6.92.2.48 bool emberMacPassthroughFilterHandler (uint8_t * *macHeader*)

Note

This API is for SoCs only.
The application must define EMBER_APPLICATION_HAS_MAC_PASSTHROUGH_FILTER_HANDLER

Parameters

macHeader	A pointer to the initial portion of the incoming MAC header, in the standard 802.15.4 format. The first two bytes comprise the frame control, which dictates source / destination PAN and addressing formats. (See the MAC sublayer definition in the standards definition 802.15.4e/2012)

The relevant bytes of the header are:

| **octets:** | **2** | **1** | **0/2** | **0/2/8** | **0/2** | **0/2/8** |

| | **ctl** | **seq** | **dst.pan** | **dst.addr** | **src.pan** | **src.addr** | **...**

Note that subsequent MAC fields, and the MAC payload, may not yet be present at this point.

Returns

true if the message is an application MAC passthrough message.

6.92.2.49 void emberMacPassthroughMessageHandler (PacketHeader *header*)

Note

This API is for SoCs only.
The application must define EMBER_APPLICATION_HAS_MAC_PASSTHROUGH_MESSAGE_HANDLER

Parameters

header	The message buffer pointing to the full 802.15.4 frame to be handled by the application.

6.92.2.50 bool emberMacRssiFilterHandler (uint8_t * *macHeader*)

Note

This API is for SoCs only.
The application must define EMBER_APPLICATION_HAS_RSSI_FILTER_HANDLER

Parameters

macHeader	A pointer to the initial portion of the incoming MAC header, in the standard 802.15.4 format. The first two bytes comprise the frame control, which dictates source / destination PAN and addressing formats. (See the MAC sublayer definition in the standards definition 802.15.4e/2012)

The relevant bytes of the header are:

| octets: | 2 | 1 | 0/2 | 0/2/8 | 0/2 | 0/2/8 |

| | ctl | seq | dst.pan | dst.addr | src.pan | src.addr | ...

Note that subsequent MAC fields, and the MAC payload, may not yet be present at this point.

Returns

true if the application wants to peek at the RSSI for this message.

6.92.2.51 void emberMacRssiHandler (int8_t *currentRssi*)

Note

This is called on the application for all packets that match the rule defined in emberMacRssiFilterHandler()

The quantity referenced by currentRssi will contain the energy level (in units of dBm) observed during the last 802.15.4 packet received in that handler.

Note

This API is for SoCs only.
The application must define EMBER_APPLICATION_HAS_RSSI_FILTER_HANDLER
This functionality is not available for packets such as 802.15.4 data requests or acknowledgements. Data requests must be handled quickly due to strict 15.4 timing requirements, and so the RSSI information is not recorded. Similarly, 802.15.4 ACKs are handled by the hardware and the information does not make it up to the stack.

Parameters

currentRssi	The RSSI for the last incoming message processed.

6.92.2.52 void emberMicroBusyHandler (bool *busy*)

This callback is not available on a host processor. Note that if ::busy is true, the micro may become busy as soon as this handler exits. In a host/ncp setup, one solution for informing the host is to implement this handler in your own xNCP image and use it to toggle the serial CTS line.

6.92.2.53 void emberNetworkDataChangeHandler (const uint8_t * *networkData,* uint16_t *length*)

This function is called when the stack receives new Thread Network Data. The networkData argument may be NULL, in which case emberGetNetworkData can be used to obtain the new Thread Network Data.

Parameters

networkData	the Network Data
length	length in bytes of the Network Data

6.92.2.54 void emberNetworkStatusHandler (EmberNetworkStatus *newNetworkStatus,* EmberNetworkStatus *oldNetworkStatus,* EmberJoinFailureReason *reason*)

This function reports a change to the network status. For example, the network status changes while going through the joining process, or while reattaching to the network, which can happen for a variety of reasons. In particular, after issuing a form, join, resume, or attach command, the application knows that the device is on the network and ready to communicate when this handler is called with a newNetworkStatus of EMBER_JOINED_NETWORK_A↩ TTACHED.

If the status handler is reporting a join failure, then the newNetworkStatus argument will have a value of EMBER↩ _NO_NETWORK and the reason argument will contain an appropriate value. For other network status reports, the reason argument does not apply and is set to EMBER_JOIN_FAILURE_REASON_NONE.

6.92.2.55 void emberOkToNapReturn (uint8_t *stateMask*)

The mask EMBER_HIGH_PRIORITY_TASKS defines which tasks are high priority. Devices should not sleep if any high priority tasks are active. Active tasks that are not high priority are waiting for messages to arrive from other devices. If there are active tasks, but no high priority ones, the device may sleep but should periodically wake up and call emberPollForData() in order to receive messages. Parents will hold messages for EMBER_INDIRECT_↩ TRANSMISSION_TIMEOUT (in quarter seconds) before discarding them.

Returns

A bitmask of the stack's active tasks.

6.92.2.56 void emberOpenDtlsConnectionReturn (uint32_t *result,* const EmberIpv6Address * *remoteAddress,* uint16_t *localPort,* uint16_t *remotePort*)

Parameters

result	error code • an EmberStatus value if using Silicon Labs TLS • an mbed TLS error code if using mbed TLS library (see mbedtls:include/mbedtls/ssl.h)
remoteAddress	IPv6 address of the server
localPort	local port
remotePort	remote port

6.92.2.57 void emberPollForDataReturn (EmberStatus *status*)

This function provides the result of a call to emberPollForData().

Parameters

An	EmberStatus value: • EMBER_SUCCESS - The poll message has been submitted for transmission • EMBER_INVALID_CALL - The node is not a sleepy end device. • EMBER_NOT_JOINED - The node is not part of a network.

6.92.2.58 void emberProcessCoap (const uint8_t * *message*, uint16_t *messageLength*, EmberCoapRequestInfo * *info*)

This function processes a CoAP message received over an alternate transport.

Called to process a CoAP message that arrived via DTLS or other alternative transport. Only the address, port and transmit handler fields of `info` are used. The token and ackData fields are ignored.

6.92.2.59 void emberRadioGetRandomNumbersReturn (EmberStatus *status*, const uint16_t * *rn*, uint8_t *count*)

This function provides the result of a call to emberRadioGetRandomNumbers.

Parameters

status	An EmberStatus value indicating success or the reason for failure. When EMBER_SUCCESS is returned ::rn and ::count will contain valid data. ::rn and ::count are undefined when EMBER_SUCCESS is not returned.
rn	the uint16_t random values
count	- the count of uint16_t values located at ::rn

6.92.2.60 void emberRequestDhcpAddressReturn (EmberStatus *status*, const uint8_t * *prefix*, uint8_t *prefixLengthInBits*)

This function provides the result of a call to emberRequestDhcpAddress.

This call only indicates the status of the request (EMBER_ERR_FATAL if no DHCP server is found, and EMBER↩ _SUCCESS otherwise). The assigned IPv6 address is returned via emberAddressConfigurationChangeHandler

Parameters

status	Status of DHCP Address Request
prefix	Prefix requested in emberRequestDhcpAddress
prefixLengthInBits	Prefix length in bits requested in emberRequestDhcpAddress

6.92.2.61 void emberRequestSlaacAddressReturn (EmberStatus *status*, const uint8_t * *prefix*, uint8_t *prefixLengthInBits*)

This function provides the result of a call to emberRequestSlaacAddress.

This call only indicates the status of the request (EMBER_ERR_FATAL if no SLAAC server is found, and EMBE↩ R_SUCCESS otherwise). The assigned IPv6 address is returned via emberAddressConfigurationChangeHandler

Parameters

status	Status of SLAAC Address Request
prefix	Prefix requested in emberRequestSlaacAddress
prefixLengthInBits	Prefix length in bits requested in emberRequestSlaacAddress

6.92.2.62 void emberResetMicroHandler (EmberResetCause *cause*)

This function notifies the application of a reset on the Ember chip due to the indicated cause.

6.92.2.63 void emberResetNetworkStateReturn (EmberStatus *status*)

This function provides the result of a call to emberResetNetworkState().

6.92.2.64 void emberResignGlobalAddressReturn (EmberStatus *status*)

This function provides the result of a call to emberResignGlobalAddress().

6.92.2.65 void emberResumeNetworkReturn (EmberStatus *status*)

6.92.2.66 void emberScanReturn (EmberStatus *status*)

This function provides the status upon completion of a scan.

6.92.2.67 void emberSendSteeringDataReturn (EmberStatus *status*)

This function provides the result of a call to emberSendSteeringData().

6.92.2.68 void emberSetAntennaModeReturn (EmberStatus *status*)

This function provides the result of a call to emberSetAntennaMode.

Parameters

EMBER_SUCCESS	if antenna mode is configured as desired or EMBER_BAD_ARGUMENT if antenna mode is not supported.

6.92.2.69 void emberSetCcaThresholdReturn (EmberStatus *status*)

This function provides the result of a call to emberSetCcaThreshold().

6.92.2.70 void emberSetCommissionerKeyReturn (EmberStatus *status*)

This function provides the result of a call to emberSetCommissionerKey(): either EMBER_SUCCESS or EMBER↩ _INVALID_CALL.

6.92.2.71 void emberSetCtuneReturn (EmberStatus *status*)

This function provides the result of a call to emberSetCtune.

Parameters

status	An EmberStatus value indicating success or the reason for failure.

6.92.2.72 void emberSetDtlsDeviceCertificateReturn (uint32_t *result*)

Parameters

result	• ::0 The certificate was set successfully. • ::result error code – an EmberStatus value if using Silicon Labs TLS – an mbed TLS error code if using mbed TLS library (see mbedtls:include/mbedtls/ssl.h)

6.92.2.73 void emberSetDtlsPresharedKeyReturn (EmberStatus *status*)

Parameters

result	• ::status An EmberStatus value

6.92.2.74 void emberSetJoinKeyReturn (EmberStatus *status*)

This function provides the result of a call to emberSetJoinKey().

6.92.2.75 void emberSetLocalNetworkDataReturn (EmberStatus *status,* uint16_t *length*)

Provides the result of a call to ::emberSetServerNetworkData.

6.92.2.76 void emberSetMfgTokenReturn (EmberMfgTokenId *tokenId,* EmberStatus *status*)

This function provides the result of a call to emberSetMfgToken.

Parameters

tokenId	Which manufacturing token set.
status	An EmberStatus value indicating success or the reason for failure.

6.92.2.77 void emberSetNdDataReturn (EmberStatus *status,* uint16_t *length*)

This function provides the result of a call to emberSetNdData.

6.92.2.78 void emberSetPskcHandler (const uint8_t * *pskc*)

Parameters

pskc	PSKc: 16 bytes in length

6.92.2.79 void emberSetPtaEnableReturn (EmberStatus *status*)

This function provides the result of a call to emberSetPtaEnable.

Parameters

status	An EmberStatus value indicating success or the reason for failure.

6.92.2.80 void emberSetPtaOptionsReturn (EmberStatus *status*)

This function provides the result of a call to emberSetPtaOptions.

Parameters

status	An EmberStatus value indicating success or the reason for failure.

6.92.2.81 void emberSetRadioHoldOffReturn (EmberStatus *status*)

This function provides the result of a call to emberSetRadioHoldOff.

Parameters

status	An EmberStatus value indicating success or the reason for failure. EMBER_SUCCESS if Radio HoldOff was configured as desired or EMBER_BAD_ARGUMENT if requesting it be enabled but RHO has not been configured by the BOARD_HEADER.

6.92.2.82 void emberSetRadioPowerReturn (EmberStatus *status*)

This function provides the result of a call to emberSetRadioPower() on the host.

6.92.2.83 void emberSetSecurityParametersReturn (EmberStatus *status*)

This function provides the result of a call to emberSetSecurityParameters().

6.92.2.84 void emberSetTxPowerModeReturn (EmberStatus *status*)

This function provides the result of a call to emberSetTxPowerMode() on the host.

6.92.2.85 void emberSlaacServerChangeHandler (const uint8_t * *prefix,* uint8_t *prefixLengthInBits,* bool *available*)

This function is called when the stack knows about a new SLAAC prefix or if a SLAAC server has become unavailable.

"available" means we can configure a SLAAC address.

Parameters

prefix	SLAAC prefix
prefixLengthInBits	length in bits of the prefix
available	whether we can configure an address

6.92.2.86 void emberStackPollForDataReturn (EmberStatus *status*)

This function provides the result of a call to emberStackPollForData().

6.92.2.87 void emberStartHostJoinClientHandler (const uint8_t * *parentAddress*)

A callback to tell the host to start security commissioning.

Parameters

address	parent IP address, 16 bytes

6.92.2.88 void emberStateReturn (const EmberNetworkParameters * *parameters*, const EmberEui64 * *localEui64*, const EmberEui64 * *macExtendedId*, EmberNetworkStatus *networkStatus*)

Parameters

parameters	Current network parameters
localEui64	The EUI64 of the Ember chip
macExtendedId	The extended MAC ID of the Ember chip
networkStatus	The current status of the network

6.92.2.89 void emberSwitchToNextNetworkKeyHandler (EmberStatus *status*)

This function can be stubbed out on the SoC and host app. It is used by the NCP to update security on the driver when it is instructed to switch the network key by an over the air update.

6.92.2.90 void emberSwitchToNextNetworkKeyReturn (EmberStatus *status*)

This function provides the result of a call to emberSwitchToNextNetworkKey().

6.93 power-meter API Callbacks

Functions

- void halPowerMeterOverHeatStatusChangeCallback (uint8_t status)
  Over Heat Callback.
- void halPowerMeterOverCurrentStatusChangeCallback (uint8_t status)
  Over Current Callback.
- void halPowerMeterCalibrationFinishedCallback (uint16_t gainSetting)
  Calibration Finished Callback.

6.93.1 Detailed Description

These callbacks were contributed by the power-meter API.

6.93.2 Function Documentation

6.93.2.1 void halPowerMeterCalibrationFinishedCallback (uint16_t *gainSetting*)

This function is called upon a calibration procedure is done

Parameters

gainSetting	gain setting

6.93.2.2 void halPowerMeterOverCurrentStatusChangeCallback (uint8_t *status*)

This function is called upon the status change of over current condition.

Parameters

status	OVER_CURRENT_TO_NORMAL (0):changed from over current to normal; NORMAL_TO_OVER_CURRENT (1):over current occured.

6.93.2.3 void halPowerMeterOverHeatStatusChangeCallback (uint8_t *status*)

This function is called upon the status change of over heat condition.

Parameters

status	OVER_HEAT_TO_NORMAL (0):changed from over heat to normal; NORMAL_TO_OVER_CURRENT (1):over heat occured.

6.94 sim-eeprom API Callbacks

Functions

- void halSimEepromCallback (EmberStatus status)

  The Simulated EEPROM callback function, implemented by the application.

6.94.1 Detailed Description

These callbacks were contributed by the sim-eeprom API.

6.94.2 Function Documentation

6.94.2.1 void halSimEepromCallback (EmberStatus *status*)

Parameters

status	An EmberStatus error code indicating one of the conditions described below.

This callback will report an EmberStatus of EMBER_SIM_EEPROM_ERASE_PAGE_GREEN whenever a token is set and a page needs to be erased. If the main application loop does not periodically call halSimEepromErase↩ Page(), it is best to then erase a page in response to EMBER_SIM_EEPROM_ERASE_PAGE_GREEN.

This callback will report an EmberStatus of EMBER_SIM_EEPROM_ERASE_PAGE_RED when the pages *must* be erased to prevent data loss. halSimEepromErasePage() needs to be called until it returns 0 to indicate there are no more pages that need to be erased. Ignoring this indication and not erasing the pages will cause dropping the new data trying to be written.

This callback will report an EmberStatus of EMBER_SIM_EEPROM_FULL when the new data cannot be written due to unerased pages. **Not erasing pages regularly, not erasing in response to EMBER_SIM_EEPROM_E↩ RASE_PAGE_GREEN, or not erasing in response to EMBER_SIM_EEPROM_ERASE_PAGE_RED will cause EMBER_SIM_EEPROM_FULL and the new data will be lost!.** Any future write attempts will be lost as well.

This callback will report an EmberStatus of EMBER_SIM_EEPROM_REPAIRING when the Simulated EEPROM needs to repair itself. While there's nothing for an app to do when the SimEE is going to repair itself (SimEE has to be fully functional for the rest of the system to work), alert the application to the fact that repairing is occuring. There are debugging scenarios where an app might want to know that repairing is happening; such as monitoring frequency.

Note

Common situations will trigger an expected repair, such as using a new chip or changing token definitions.

If the callback ever reports the status EMBER_ERR_FLASH_WRITE_INHIBITED or EMBER_ERR_FLASH_V↩ ERIFY_FAILED, this indicates a catastrophic failure in flash writing, meaning either the address being written is not empty or the write itself has failed. If EMBER_ERR_FLASH_WRITE_INHIBITED is encountered, the function ::halInternalSimEeRepair(false) should be called and the chip should then be reset to allow proper initialization to recover. If EMBER_ERR_FLASH_VERIFY_FAILED is encountered the Simulated EEPROM (and tokens) on the specific chip with this error should not be trusted anymore.

References assert, EMBER_ERR_FLASH_VERIFY_FAILED, EMBER_ERR_FLASH_WRITE_INHIBITED, EM↩ BER_SIM_EEPROM_ERASE_PAGE_GREEN, EMBER_SIM_EEPROM_ERASE_PAGE_RED, EMBER_SIM_E↩ EPROM_FULL, EMBER_SIM_EEPROM_REPAIRING, halInternalSysReset(), halSimEepromErasePage(), and halSimEepromPagesRemainingToBeErased().

6.95 stack-info API Callbacks

Functions

- void emberRadioNeedsCalibratingHandler (void)

  The radio calibration callback function.

6.95.1 Detailed Description

These callbacks were contributed by the stack-info API.

6.95.2 Function Documentation

6.95.2.1 void emberRadioNeedsCalibratingHandler (void)

This function enables boost power mode and/or the alternate transmit path.

The Voltage Controlled Oscillator (VCO) can drift with temperature changes. During every call to emberTick(), the stack will check to see if the VCO has drifted. If the VCO has drifted, the stack will call emberRadioNeeds↩CalibratingHandler() to inform the application that it should perform calibration of the current channel as soon as possible. Calibration can take up to 150ms. The default callback function implementation provided here performs calibration immediately. If the application wishes, it can define its own callback by defining ::EMBER_APPLICAT↩ION_HAS_CUSTOM_RADIO_CALIBRATION_CALLBACK in its CONFIGURATION_HEADER. It can then failsafe any critical processes or peripherals before calling emberCalibrateCurrentChannel(). The application must call emberCalibrateCurrentChannel() in response to this callback to maintain expected radio performance.

References emberCalibrateCurrentChannel().

6.96 thread-debug API Callbacks

Functions

- void emberAddAddressDataReturn (uint16_t shortId)

  Callback for a debug command. Provides the result of ::emberAddAddressData.
- void emberAssertInfoReturn (const uint8_t *fileName, uint32_t lineNumber)

  Callback for a debug command. Provides the result of ::emberAssertInfo.
- void emberClearAddressCacheReturn (void)

  Callback for a debug command. Provides the result of ::emberClearAddressCache.
- void emberConfigUartReturn (void)

  Callback for a debug command. Provides the result of ::emberConfigUart.
- void emberEchoReturn (const uint8_t *data, uint8_t length)

  Callback for a debug command. Provides the result of emberEcho.
- void emberGetMulticastEntryReturn (uint8_t lastSequence, uint8_t windowBitmask, uint8_t dwellQs, const uint8_t *seed)

  Callback for a debug command. Provides the result of ::emberGetMulticastEntry.
- void emberGetNetworkKeyInfoReturn (EmberStatus status, uint32_t sequence, uint8_t state)

  Callback for a debug command. Provides the result of ::emberGetNetworkKeyInfo.
- void emberGetNodeStatusReturn (EmberStatus status, uint8_t ripId, EmberNodeId nodeId, uint8_t parent↩ RipId, EmberNodeId parentId, const uint8_t *networkFragmentIdentifier, uint32_t networkFrameCounter)

  Callback for a debug command. Provides the result of ::emberGetNodeStatus.
- void emberLookupAddressDataReturn (uint16_t shortId)

  Callback for a debug command. Provides the result of ::emberLookupAddressData.
- void emberNcpUdpStormCompleteHandler (void)

  Callback for a debug command. Provides the result of ::emberNcpUdpStormComplete.
- void emberNcpUdpStormReturn (EmberStatus status)

  Callback for a debug command. Provides the result of ::emberNcpUdpStorm.
- void emberResetNcpAshReturn (void)

  Callback for a debug command. Provides the result of ::emberResetNcpAsh.
- void emberSendDoneReturn (void)

  Callback for a debug command. Provides the result of ::emberSendDone.
- void emberSetRandomizeMacExtendedIdReturn (void)

  Callback for a debug command. Provides the result of ::emberSetRandomizeMacExtendedId.
- void emberSetWakeupSequenceNumberReturn (void)

  Callback for a debug command. Provides the result of ::emberSetWakeupSequenceNumber.
- void emberStartUartStormReturn (void)

  Callback for a debug command. Provides the result of ::emberStartUartStorm.
- void emberStopUartStormReturn (void)

  Callback for a debug command. Provides the result of ::emberStopUartStorm.
- void emberUartSpeedTestReturn (uint32_t totalBytesSent, uint32_t payloadBytesSent, uint32_t timeout)

  Callback for a debug command. Provides the result of ::emberUartSpeedTest.

6.96.1 Detailed Description

These callbacks were contributed by the thread-debug API.

6.96.2 Function Documentation

6.96.2.1 void emberAddAddressDataReturn (uint16_t *shortId*)

6.96.2.2 void emberAssertInfoReturn (const uint8_t * *fileName,* uint32_t *lineNumber*)

Sent from the NCP to the host when an assert occurs.

6.96.2.3 void emberClearAddressCacheReturn (void)

6.96.2.4 void emberConfigUartReturn (void)

6.96.2.5 void emberEchoReturn (const uint8_t * *data,* uint8_t *length*)

6.96.2.6 void emberGetMulticastEntryReturn (uint8_t *lastSequence,* uint8_t *windowBitmask,* uint8_t *dwellQs,* const uint8_t * *seed*)

6.96.2.7 void emberGetNetworkKeyInfoReturn (EmberStatus *status,* uint32_t *sequence,* uint8_t *state*)

6.96.2.8 void emberGetNodeStatusReturn (EmberStatus *status,* uint8_t *ripId,* EmberNodeId *nodeId,* uint8_t *parentRipId,* EmberNodeId *parentId,* const uint8_t * *networkFragmentIdentifier,* uint32_t *networkFrameCounter*)

6.96.2.9 void emberLookupAddressDataReturn (uint16_t *shortId*)

6.96.2.10 void emberNcpUdpStormCompleteHandler (void)

6.96.2.11 void emberNcpUdpStormReturn (EmberStatus *status*)

6.96.2.12 void emberResetNcpAshReturn (void)

6.96.2.13 void emberSendDoneReturn (void)

6.96.2.14 void emberSetRandomizeMacExtendedIdReturn (void)

6.96.2.15 void emberSetWakeupSequenceNumberReturn (void)

6.96.2.16 void emberStartUartStormReturn (void)

6.96.2.17 void emberStopUartStormReturn (void)

6.96.2.18 void emberUartSpeedTestReturn (uint32_t *totalBytesSent,* uint32_t *payloadBytesSent,* uint32_t *timeout*)

6.97 udp API Callbacks

Functions

- void emberUdpHandler (const uint8_t *destination, const uint8_t *source, uint16_t localPort, uint16_t remotePort, const uint8_t *payload, uint16_t payloadLength)

  Application callback for an incoming UDP message.
- void emberUdpMulticastHandler (const uint8_t *destination, const uint8_t *source, uint16_t localPort, uint16_t remotePort, const uint8_t *payload, uint16_t payloadLength)

  Application callback for an incoming UDP multicast.

6.97.1 Detailed Description

These callbacks were contributed by the udp API.

6.97.2 Function Documentation

6.97.2.1 void emberUdpHandler (const uint8_t * *destination*, const uint8_t * *source*, uint16_t *localPort*, uint16_t *remotePort*, const uint8_t * *payload*, uint16_t *payloadLength*)

An application callback for an incoming UDP message.

Parameters

destination	IPV6 destination address
source	IPV6 source address
localPort	UDP source port
remotePort	UDP destination port
payload	UDP transport payload
payloadLength	payload length

Referenced by emberUdpMulticastHandler().

6.97.2.2 void emberUdpMulticastHandler (const uint8_t * *destination*, const uint8_t * *source*, uint16_t *localPort*, uint16_t *remotePort*, const uint8_t * *payload*, uint16_t *payloadLength*)

An application callback for an incoming UDP multicast.

Parameters

destination	IPV6 destination address
source	IPV6 source address
localPort	UDP source port
remotePort	UDP destination port
payload	UDP transport payload
payloadLength	payload length

References emberUdpHandler().

6.98 OTA Bootload

Modules

- OTA Bootload Types
- OTA Bootload API

6.98.1 Detailed Description

The following image shows bootloading accomplished using Over-The-Air (OTA), that is through the wireless network.

6.99 OTA Bootload Types

Data Structures

- struct EmberZclOtaBootloadClientServerInfo_t
- struct EmberZclOtaBootloadHardwareVersionRange_t
- struct EmberZclOtaBootloadFileSpec_t
- struct EmberZclOtaBootloadFileHeaderInfo_t
- struct EmberZclOtaBootloadStorageInfo_t
- struct EmberZclOtaBootloadStorageFileInfo_t

Macros

- #define EMBER_ZCL_OTA_BOOTLOAD_FILE_MAGIC_NUMBER 0x0BEEF11E
- #define EMBER_ZCL_OTA_BOOTLOAD_FILE_MAGIC_NUMBER_SIZE 4
- #define EMBER_ZCL_OTA_BOOTLOAD_FILE_VERSION 0x2000
- #define EMBER_ZCL_OTA_BOOTLOAD_HEADER_STRING_SIZE 32
- #define EMBER_ZCL_OTA_BOOTLOAD_HEADER_MAX_SIZE 93
- #define EMBER_ZCL_OTA_BOOTLOAD_FILE_VERSION_NULL ((EmberZclOtaBootloadFileVersion_t)-1)
- #define EMBER_ZCL_OTA_BOOTLOAD_HARDWARE_VERSION_NULL ((EmberZclOtaBootloadHardwareVersion_t)-1)

Typedefs

- typedef uint32_t EmberZclOtaBootloadFileVersion_t
- typedef uint16_t EmberZclOtaBootloadHardwareVersion_t
- typedef void(* EmberZclOtaBootloadStorageDeleteCallback) (EmberZclOtaBootloadStorageStatus_t)

Enumerations

- enum EmberZclOtaBootloadFileHeaderFieldControl_t {
  EMBER_ZCL_OTA_BOOTLOAD_FILE_HEADER_FIELD_CONTROL_DESTINATION = 0x0002,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_HEADER_FIELD_CONTROL_HARDWARE_VERSION = 0x0004,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_HEADER_FIELD_CONTROL_NULL = ((EmberZclOtaBootloadFileHeaderFieldControl_t)-1) }
- enum EmberZclOtaBootloadFileType_t {
  EMBER_ZCL_OTA_BOOTLOAD_FILE_TYPE_MANUFACTURER_SPECIFIC_MAXIMUM = 0xFFBF,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_TYPE_SECURITY_CREDENTIALS = 0xFFC0,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_TYPE_CONFIGURATION = 0xFFC1,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_TYPE_LOG = 0xFFC2,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_TYPE_PICTURE = 0xFFC3,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_TYPE_WILDCARD = 0xFFFF }
- enum EmberZclOtaBootloadStackVersion_t {
  EMBER_ZCL_OTA_BOOTLOAD_STACK_VERSION_IP = 0x0004,
  EMBER_ZCL_OTA_BOOTLOAD_STACK_VERSION_NONE = ((EmberZclOtaBootloadStackVersion_t)-1) }
- enum EmberZclOtaBootloadSecurityCredentialVersion_t {
  EMBER_ZCL_OTA_BOOTLOAD_SECURITY_CREDENTIAL_VERSION_IP = 0x03,
  EMBER_ZCL_OTA_BOOTLOAD_SECURITY_CREDENTIAL_VERSION_NULL = ((EmberZclOtaBootloadSecurityCredentialVersion_t)-1) }

- enum EmberZclOtaBootloadFileStatus_t {
  EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_VALID = 0x00,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_INVALID_MAGIC_NUMBER = 0x01,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_INVALID_VERSION = 0x02,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_INVALID_HEADER_SIZE = 0x03,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_INVALID_STACK_VERSION = 0x04,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_INVALID_SECURITY_CREDENTIAL_VERSION = 0x05,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_NULL = 0xFF }
- enum EmberZclOtaBootloadStorageStatus_t {
  EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_SUCCESS = 0x00,
  EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_FAILED = 0x01,
  EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_OUT_OF_RANGE = 0x02,
  EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_INVALID_FILE = 0x03,
  EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_OUT_OF_SPACE = 0x04,
  EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_NULL = 0xFF }

Variables

- const EmberZclOtaBootloadFileSpec_t emberZclOtaBootloadFileSpecNull

6.99.1 Detailed Description

6.99.2 Macro Definition Documentation

6.99.2.1 #define EMBER_ZCL_OTA_BOOTLOAD_FILE_MAGIC_NUMBER 0x0BEEF11E

This is the magic 32-bit number that appears at the beginning of every OTA file.

See also

EMBER_ZCL_OTA_BOOTLOAD_FILE_MAGIC_NUMBER_SIZE
EmberZclOtaBootloadFileHeaderInfo_t

6.99.2.2 #define EMBER_ZCL_OTA_BOOTLOAD_FILE_MAGIC_NUMBER_SIZE 4

This is the size of the magic 32-bit number that appears at the beginning of every OTA file.

See also

EMBER_ZCL_OTA_BOOTLOAD_FILE_MAGIC_NUMBER
EmberZclOtaBootloadFileHeaderInfo_t

6.99.2.3 #define EMBER_ZCL_OTA_BOOTLOAD_FILE_VERSION 0x2000

This is the version of OTA files that work with ZCLIP.

See also

EmberZclOtaBootloadFileHeaderInfo_t

6.99.2.4 #define EMBER_ZCL_OTA_BOOTLOAD_FILE_VERSION_NULL ((EmberZclOtaBootloadFileVersion_t)-1)

Distinguished value that represents a null (invalid) OTA file version.

6.99.2.5 #define EMBER_ZCL_OTA_BOOTLOAD_HARDWARE_VERSION_NULL ((EmberZclOtaBootloadHardware↩ Version_t)-1)

Distinguished value that represents a null (invalid) OTA file hardware version.

6.99.2.6 #define EMBER_ZCL_OTA_BOOTLOAD_HEADER_MAX_SIZE 93

This is the maximum number of bytes contained in an OTA file header.

These fields are required.

- 4-byte file identifier (see EMBER_ZCL_OTA_BOOTLOAD_FILE_MAGIC_NUMBER)
- 2-byte header version
- 2-byte header length
- 2-byte header field control
- 2-byte manufacturer code
- 2-byte file type
- 4-byte file version
- 2-byte communication stack version
- 32-byte header string (see EMBER_ZCL_OTA_BOOTLOAD_HEADER_STRING_SIZE)
- 4-byte file size
- 1-byte security credential version

These fields are optional.

- 32-byte destination UID
- 2-byte minimum hardware version
- 2-byte maximum hardware version

See also

EmberZclOtaBootloadFileHeaderInfo_t

6.99.2.7 #define EMBER_ZCL_OTA_BOOTLOAD_HEADER_STRING_SIZE 32

This is the size of the OTA file header string in bytes. This size includes the byte for the NUL-terminator, which must be included in a header string.

See also

EmberZclOtaBootloadFileHeaderInfo_t

6.99.3 Typedef Documentation

6.99.3.1 typedef uint32_t EmberZclOtaBootloadFileVersion_t

OTA file version.

6.99.3.2 typedef uint16_t EmberZclOtaBootloadHardwareVersion_t

OTA file hardware version.

6.99.3.3 typedef void(* EmberZclOtaBootloadStorageDeleteCallback) (EmberZclOtaBootloadStorageStatus_t)

6.99.4 Enumeration Type Documentation

6.99.4.1 enum EmberZclOtaBootloadFileHeaderFieldControl_t

OTA file header field control.

Enumerator

EMBER_ZCL_OTA_BOOTLOAD_FILE_HEADER_FIELD_CONTROL_DESTINATION OTA file header contains destination field.

EMBER_ZCL_OTA_BOOTLOAD_FILE_HEADER_FIELD_CONTROL_HARDWARE_VERSION OTA file header contains minimum and maximum valid hardware versions.

EMBER_ZCL_OTA_BOOTLOAD_FILE_HEADER_FIELD_CONTROL_NULL Distinguished value that represents a null (invalid) OTA file header field control.

6.99.4.2 enum EmberZclOtaBootloadFileStatus_t

OTA file status.

Enumerator

EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_VALID OTA file is valid.

EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_INVALID_MAGIC_NUMBER OTA file has invalid magic number.

EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_INVALID_VERSION OTA file has invalid version.

EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_INVALID_HEADER_SIZE OTA file has invalid header size.

EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_INVALID_STACK_VERSION OTA file has invalid stack version.

EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_INVALID_SECURITY_CREDENTIAL_VERSION OTA file has invalid security credential version.

EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_NULL Distinguished value that represents a null (invalid) OTA file status.

6.99.4.3 enum EmberZclOtaBootloadFileType_t

OTA file type.

Enumerator

EMBER_ZCL_OTA_BOOTLOAD_FILE_TYPE_MANUFACTURER_SPECIFIC_MAXIMUM This is the maximum value for a manufacturer-specific file type.

EMBER_ZCL_OTA_BOOTLOAD_FILE_TYPE_SECURITY_CREDENTIALS OTA file is security credentials.

EMBER_ZCL_OTA_BOOTLOAD_FILE_TYPE_CONFIGURATION OTA file is a configuration.

EMBER_ZCL_OTA_BOOTLOAD_FILE_TYPE_LOG OTA file is a log.

EMBER_ZCL_OTA_BOOTLOAD_FILE_TYPE_PICTURE OTA file is a picture.

EMBER_ZCL_OTA_BOOTLOAD_FILE_TYPE_WILDCARD OTA file is unspecified.

6.99.4.4 enum EmberZclOtaBootloadSecurityCredentialVersion_t

OTA file security credential version.

Enumerator

EMBER_ZCL_OTA_BOOTLOAD_SECURITY_CREDENTIAL_VERSION_IP OTA file uses IP security credentials.

EMBER_ZCL_OTA_BOOTLOAD_SECURITY_CREDENTIAL_VERSION_NULL Distinguished value that represents a null (invalid) OTA file security credential version.

6.99.4.5 enum EmberZclOtaBootloadStackVersion_t

OTA file stack version.

Enumerator

EMBER_ZCL_OTA_BOOTLOAD_STACK_VERSION_IP OTA file is for an IP stack.

EMBER_ZCL_OTA_BOOTLOAD_STACK_VERSION_NONE Distinguished value that represents a null (invalid) OTA file stack version.

6.99.4.6 enum EmberZclOtaBootloadStorageStatus_t

OTA storage status.

Enumerator

EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_SUCCESS An operation has succeeded.

EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_FAILED An operation has failed.

EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_OUT_OF_RANGE An operation is outside a valid range.

EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_INVALID_FILE An operation is specified on a nonexistent file.

EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_OUT_OF_SPACE An operation is outside valid space constraints.

EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_NULL Distinguished value that represents a null (invalid) status value.

6.99.5 Variable Documentation

6.99.5.1 const EmberZclOtaBootloadFileSpec_t emberZclOtaBootloadFileSpecNull

This is a distinguished value that represents a null (invalid) OTA file specification.

This is provided as a utility to applications that wish to use a file specification value that is *uninitialized* or *invalid*.

6.100 OTA Bootload API

Functions

- void emberZclOtaBootloadInitFileHeaderInfo (EmberZclOtaBootloadFileHeaderInfo_t *headerInfo)
- bool emberZclOtaBootloadFileSpecsAreEqual (const EmberZclOtaBootloadFileSpec_t *s1, const EmberZclOtaBootloadFileSpec_t *s2)
- size_t emberZclOtaBootloadFetchFileSpec (const uint8_t *data, EmberZclOtaBootloadFileSpec_t *fileSpec)
- size_t emberZclOtaBootloadStoreFileSpec (const EmberZclOtaBootloadFileSpec_t *fileSpec, uint8_t *data)
- EmberZclOtaBootloadFileStatus_t emberZclOtaBootloadFetchFileHeaderInfo (const uint8_t *data, EmberZclOtaBootloadFileHeaderInfo_t *fileHeaderInfo)
- EmberZclOtaBootloadFileStatus_t emberZclOtaBootloadStoreFileHeaderInfo (uint8_t *data, EmberZclOtaBootloadFileHeaderInfo_t *fileHeaderInfo, size_t imageDataSize)
- void emberZclOtaBootloadStorageGetInfo (EmberZclOtaBootloadStorageInfo_t *info, EmberZclOtaBootloadFileSpec_t *returnedFiles, size_t returnedFilesMaxCount)
- EmberZclOtaBootloadStorageStatus_t emberZclOtaBootloadStorageFind (const EmberZclOtaBootloadFileSpec_t *fileSpec, EmberZclOtaBootloadStorageFileInfo_t *fileInfo)
- EmberZclOtaBootloadStorageStatus_t emberZclOtaBootloadStorageCreate (const EmberZclOtaBootloadFileSpec_t *fileSpec)
- EmberZclOtaBootloadStorageStatus_t emberZclOtaBootloadStorageRead (const EmberZclOtaBootloadFileSpec_t *fileSpec, size_t offset, void *data, size_t dataLength)
- EmberZclOtaBootloadStorageStatus_t emberZclOtaBootloadStorageWrite (const EmberZclOtaBootloadFileSpec_t *fileSpec, size_t offset, const void *data, size_t dataLength)
- EmberZclOtaBootloadStorageStatus_t emberZclOtaBootloadStorageDelete (const EmberZclOtaBootloadFileSpec_t *fileSpec, EmberZclOtaBootloadStorageDeleteCallback callback)

6.100.1 Detailed Description

See OTA Bootload Client Callbacks for the application callbacks associated with the OTA Bootload Client plugin.

6.100.2 Function Documentation

6.100.2.1 EmberZclOtaBootloadFileStatus_t emberZclOtaBootloadFetchFileHeaderInfo (const uint8_t * *data*, EmberZclOtaBootloadFileHeaderInfo_t * *fileHeaderInfo*)

This function reads an EmberZclOtaBootloadFileHeaderInfo_t from a flat buffer (little-endian).

Parameters

data	Flat buffer from which to read a file header info structure
fileHeaderInfo	File header info struct to be populated from a flat buffer

Returns

An EmberZclOtaBootloadFileStatus_t value describing if the data is a valid or invalid dotdot OTA data.

6.100.2.2 size_t emberZclOtaBootloadFetchFileSpec (const uint8_t * *data*, EmberZclOtaBootloadFileSpec_t * *fileSpec*)

This function reads an EmberZclOtaBootloadFileSpec_t from a flat buffer (little-endian).

Parameters

data	Flat buffer from which to read a file specification struct
fileSpec	File specification struct to be populated from a flat buffer

Returns

The number of bytes read from the flat buffer.

6.100.2.3 bool emberZclOtaBootloadFileSpecsAreEqual (const EmberZclOtaBootloadFileSpec_t * *s1*, const EmberZclOtaBootloadFileSpec_t * *s2*)

This function compares two OTA file specifications.

Parameters

s1	OTA file specification to be compared
s2	OTA file specification to be compared

Returns

`true` if both OTA file specifications are equal, `false` otherwise.

6.100.2.4 void emberZclOtaBootloadInitFileHeaderInfo (EmberZclOtaBootloadFileHeaderInfo_t * *headerInfo*)

This function initializes the EmberZclOtaBootloadFileHeaderInfo_t structure.

Parameters

headerInfo	Structure to be initialized

6.100.2.5 EmberZclOtaBootloadStorageStatus_t emberZclOtaBootloadStorageCreate (const EmberZclOtaBootloadFileSpec_t * *fileSpec*)

This function creates a file in the storage module.

Parameters

fileSpec	A file specification for the file to be created

Returns

One of the following status values.

- EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_SUCCESS if the file was successfully created
- EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_INVALID_FILE if the file already exists

- EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_FAILED if any other failure occured
- EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_OUT_OF_SPACE if there is not enough space to create this file

Note

The implementation of this function must adhere to the following specifications:

- The implementation of this function must be atomic.
- The implementation of this function does not need to be thread-safe.

6.100.2.6 EmberZclOtaBootloadStorageStatus_t emberZclOtaBootloadStorageDelete (const EmberZclOtaBootloadFileSpec_t * *fileSpec,* EmberZclOtaBootloadStorageDeleteCallback *callback*)

This function deletes one or all files in the storage module asynchronously.

Parameters

fileSpec	A file specification for the file to be deleted, or emberZclOtaBootloadFileSpecNull for all files to be deleted
callback	A callback to be called upon completion of this deletion operation

Returns

One of the following status values.

- EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_SUCCESS if deletion was successfully started on the indicated file/s
- EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_INVALID_FILE if the file does not exist in the storage module
- EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_FAILED if any other failure occured

Note

The implementation of this function must adhere to the following specification:

- The implementation of this function does not need to be thread-safe.

6.100.2.7 EmberZclOtaBootloadStorageStatus_t emberZclOtaBootloadStorageFind (const EmberZclOtaBootloadFileSpec_t * *fileSpec,* EmberZclOtaBootloadStorageFileInfo_t * *fileInfo*)

This function finds a file in the storage module.

Parameters

fileSpec	A file specification describing the file to be found
fileInfo	Returned information about the file to be found; only valid if the function returns EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_SUCCESS

Returns

One of the following status values.

- EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_SUCCESS if the file was successfully found
- EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_INVALID_FILE if the file does not exist
- EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_FAILED if any other failure occured

Note

The implementation of this function must adhere to the following specifications:

- The implementation of this function must be safe and idempotent, i.e., contiguous calls to this function with equal fileSpec parameters (as determined by emberZclOtaBootloadFileSpecsAreEqual) must result in the same returned information and no change in the state of the storage module.
- The implementation of this function does not need to be thread-safe.

6.100.2.8 void emberZclOtaBootloadStorageGetInfo (EmberZclOtaBootloadStorageInfo_t * *info*, EmberZclOtaBootloadFileSpec_t * *returnedFiles*, size_t *returnedFilesMaxCount*)

This function gets information about the current storage module.

Parameters

info	Returned information about the current storage module.
returnedFiles	The returned list of files that exist in this module. This parameter can be set to NULL to be ignored by the implementation.
returnedFilesMaxCount	The maximum number of entries in the provided returnedFiles parameter.

Note

The implementation of this function must adhere to the following specifications:

- The implementation of this function must return successfully. If, for any reason, the storage module cannot successfully return the necessary information in the implementation of this function, it should raise the error by calling assert(false).
- The implementation of this function must be safe and idempotent, i.e., contiguous calls to this function must result in the same returned information and no change in the state of the storage module.
- The implementation of this function does not need to be thread-safe.

6.100.2.9 EmberZclOtaBootloadStorageStatus_t emberZclOtaBootloadStorageRead (const EmberZclOtaBootloadFileSpec_t * *fileSpec*, size_t *offset*, void * *data*, size_t *dataLength*)

This function reads contiguous bytes from a file in the storage module.

Parameters

fileSpec	A file specification for the file to be read
offset	The offset into the file at which to start reading bytes
data	The buffer into which the bytes will be read
dataLength	The number of bytes to read from the file

Returns

One of the following status values.

- EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_SUCCESS if dataLength number of bytes were successfully read
- EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_OUT_OF_RANGE if reading dataLength number of bytes starting at offset would result in reading past the end of the file
- EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_INVALID_FILE if the file does not exist in the storage module
- EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_FAILED if any other failure occured

Note

The implementation of this function must adhere to the following specifications:

- The implementation of this function must be atomic.
- The implementation of this function must support random read access.
- The implementation of this function must allow for random access reading, i.e., reading from any valid offset must be supported.
- The implementation of this function does not need to be thread-safe.

6.100.2.10 EmberZclOtaBootloadStorageStatus_t emberZclOtaBootloadStorageWrite (const EmberZclOtaBootloadFileSpec_t * *fileSpec,* size_t *offset,* const void * *data,* size_t *dataLength*)

This function writes contiguous bytes to a file in the storage module.

Parameters

fileSpec	A file specification for the file to be written
offset	The offset into the file at which to start writing bytes
data	The bytes to be written
dataLength	The number of bytes to written to the file

Returns

One of the following status values.

- EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_SUCCESS if dataLength number of bytes were successfully written
- EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_OUT_OF_RANGE if writing dataLength number of bytes starting at offset would result in writing past the end of the file
- EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_INVALID_FILE if the file does not exist in the storage module
- EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_FAILED if any other failure occured

Note

The implementation of this function must adhere to the following specifications:

- The implementation of this function must be atomic.
- The implementation of this function must support sequential write access, but does not need to support random write access.
- The implementation of this function does not need to be thread-safe.

6.100.2.11 EmberZclOtaBootloadFileStatus_t emberZclOtaBootloadStoreFileHeaderInfo (uint8_t * *data*, EmberZclOtaBootloadFileHeaderInfo_t * *fileHeaderInfo*, size_t *imageDataSize*)

This function writes an EmberZclOtaBootloadFileHeaderInfo_t to a flat buffer (little-endian). The fileHeaderInfo->headerSize and fileHeaderInfo->fileSize variables are updated to reflect the stored sizes.

Parameters

data	Flat buffer to which to write a file header info structure
fileHeaderInfo	File header info struct to be written

Returns

An EmberZclOtaBootloadFileStatus_t value describing if the data is a valid or invalid dotdot OTA data.

6.100.2.12 size_t emberZclOtaBootloadStoreFileSpec (const EmberZclOtaBootloadFileSpec_t * *fileSpec*, uint8_t * *data*)

This function writes an EmberZclOtaBootloadFileSpec_t to a flat buffer (little-endian).

Parameters

fileSpec	File specification struct to be written to a flat buffer
data	Flat buffer to which the file specification struct will be written

Returns

The number of bytes written to the flat buffer.

6.101 Utilities

Macros

- #define EMBER_ZCL_URI_MAX_LENGTH 120
- #define EMBER_ZCL_URI_PATH_MAX_LENGTH 29
- #define EMBER_ZCL_URI_PATH_CLUSTER_ID_MAX_LENGTH 11
- #define EMBER_ZCL_URI_PATH_MANUFACTURER_CODE_CLUSTER_ID_SEPARATOR '_'
- #define EMBER_ZCL_STRING_OVERHEAD 1
- #define EMBER_ZCL_STRING_LENGTH_MAX 0xFE
- #define EMBER_ZCL_STRING_LENGTH_INVALID 0xFF
- #define EMBER_ZCL_LONG_STRING_OVERHEAD 2
- #define EMBER_ZCL_LONG_STRING_LENGTH_MAX 0xFFFE
- #define EMBER_ZCL_LONG_STRING_LENGTH_INVALID 0xFFFF

Functions

- uint8_t emberZclStringLength (const uint8_t *buffer)
- uint8_t emberZclStringSize (const uint8_t *buffer)
- uint16_t emberZclLongStringLength (const uint8_t *buffer)
- uint16_t emberZclLongStringSize (const uint8_t *buffer)

6.101.1 Detailed Description

See zcl-core-types.h for source code.

6.101.2 Macro Definition Documentation

6.101.2.1 #define EMBER_ZCL_LONG_STRING_LENGTH_INVALID 0xFFFF

Invalid long string length.

6.101.2.2 #define EMBER_ZCL_LONG_STRING_LENGTH_MAX 0xFFFE

Maximum long string length.

6.101.2.3 #define EMBER_ZCL_LONG_STRING_OVERHEAD 2

Long string overhead.

6.101.2.4 #define EMBER_ZCL_STRING_LENGTH_INVALID 0xFF

Invalid string length.

6.101.2.5 #define EMBER_ZCL_STRING_LENGTH_MAX 0xFE

Maximum string length.

6.101.2.6 #define EMBER_ZCL_STRING_OVERHEAD 1

String overhead.

6.101.2.7 #define EMBER_ZCL_URI_MAX_LENGTH 120

The longest ZCL/IP URI is: coaps://nih:sha-256;<uid>:PPPPP/zcl/g/GGGG/RMMMM_CCCC/a/AAAA where <uid> is a 256-bit UID represented as 64 hexadecimal characters, PPPPP is a 16-bit UDP port in decimal, GGGG is the 16-bit group ID in hexadecimal, R is c or s for client or server, MMMM is the 16-bit manufacturer code in hexadecimal, CCCC is the 16-bit cluster ID in hexadecimal, and AAAA is the 16-bit attribute ID in hexadecimal. An extra byte is reserved for a null terminator.

6.101.2.8 #define EMBER_ZCL_URI_PATH_CLUSTER_ID_MAX_LENGTH 11

The longest cluster ID in a ZCL/IP URI path is manufacturer-specific: RMMMM_CCCC where R is c or s for client or server, MMMM is the 16-bit manufacturer code, and CCCC is the 16-bit cluster ID. An extra byte is reserved for a null terminator.

6.101.2.9 #define EMBER_ZCL_URI_PATH_MANUFACTURER_CODE_CLUSTER_ID_SEPARATOR '_'

Manufacturer codes, if present, are separated from the cluster ID by an underscore.

6.101.2.10 #define EMBER_ZCL_URI_PATH_MAX_LENGTH 29

The longest ZCL/IP URI path is a manufacturer-specific attribute request sent to a group: zcl/g/GGGG/RMMMM_↩ CCCC/a/AAAA where GGGG is the 16-bit group ID, R is c or s for client or server, MMMM is the 16-bit manufacturer code, CCCC is the 16-bit cluster ID, and AAAA is the 16-bit attribute ID. An extra byte is reserved for a null terminator.

6.101.3 Function Documentation

6.101.3.1 uint16_t emberZclLongStringLength (const uint8_t * *buffer*)

This function returns the length of the octet or character data in a given long string.

Parameters

buffer	string pointer

Returns

length of string

6.101.3.2 uint16_t emberZclLongStringSize (const uint8_t * *buffer*)

This function returns the size of a given long string including overhead and data.

Parameters

buffer	string pointer

Returns

size of string

6.101.3.3 uint8_t emberZclStringLength (const uint8_t * *buffer*)

This function returns the length of the octet or character data in a given string.

Parameters

buffer	string pointer

Returns

length of string

6.101.3.4 uint8_t emberZclStringSize (const uint8_t * *buffer*)

This function returns the size of a given string including overhead and data.

Parameters

buffer	string pointer

Returns

size of string

6.102 ZCL Types

Data Structures

- struct EmberZclStringType_t

Typedefs

- typedef uint8_t data8_t
- typedef uint16_t data16_t
- typedef uint32_t data32_t
- typedef uint64_t data64_t
- typedef uint8_t bitmap8_t
- typedef uint16_t bitmap16_t
- typedef uint32_t bitmap32_t
- typedef uint64_t bitmap64_t
- typedef uint8_t enum8_t
- typedef uint16_t enum16_t
- typedef uint32_t utc_time_t

Enumerations

- enum EmberZclStatus_t {
  EMBER_ZCL_STATUS_SUCCESS = 0x00,
  EMBER_ZCL_STATUS_FAILURE = 0x01,
  EMBER_ZCL_STATUS_NOT_AUTHORIZED = 0x7E,
  EMBER_ZCL_STATUS_RESERVED_FIELD_NOT_ZERO = 0x7F,
  EMBER_ZCL_STATUS_MALFORMED_COMMAND = 0x80,
  EMBER_ZCL_STATUS_UNSUP_CLUSTER_COMMAND = 0x81,
  EMBER_ZCL_STATUS_UNSUP_GENERAL_COMMAND = 0x82,
  EMBER_ZCL_STATUS_UNSUP_MANUF_CLUSTER_COMMAND = 0x83,
  EMBER_ZCL_STATUS_UNSUP_MANUF_GENERAL_COMMAND = 0x84,
  EMBER_ZCL_STATUS_INVALID_FIELD = 0x85,
  EMBER_ZCL_STATUS_UNSUPPORTED_ATTRIBUTE = 0x86,
  EMBER_ZCL_STATUS_INVALID_VALUE = 0x87,
  EMBER_ZCL_STATUS_READ_ONLY = 0x88,
  EMBER_ZCL_STATUS_INSUFFICIENT_SPACE = 0x89,
  EMBER_ZCL_STATUS_DUPLICATE_EXISTS = 0x8A,
  EMBER_ZCL_STATUS_NOT_FOUND = 0x8B,
  EMBER_ZCL_STATUS_UNREPORTABLE_ATTRIBUTE = 0x8C,
  EMBER_ZCL_STATUS_INVALID_DATA_TYPE = 0x8D,
  EMBER_ZCL_STATUS_INVALID_SELECTOR = 0x8E,
  EMBER_ZCL_STATUS_WRITE_ONLY = 0x8F,
  EMBER_ZCL_STATUS_INCONSISTENT_STARTUP_STATE = 0x90,
  EMBER_ZCL_STATUS_DEFINED_OUT_OF_BAND = 0x91,
  EMBER_ZCL_STATUS_INCONSISTENT = 0x92,
  EMBER_ZCL_STATUS_ACTION_DENIED = 0x93,
  EMBER_ZCL_STATUS_TIMEOUT = 0x94,
  EMBER_ZCL_STATUS_ABORT = 0x95,
  EMBER_ZCL_STATUS_INVALID_IMAGE = 0x96,
  EMBER_ZCL_STATUS_WAIT_FOR_DATA = 0x97,
  EMBER_ZCL_STATUS_NO_IMAGE_AVAILABLE = 0x98,
  EMBER_ZCL_STATUS_REQUIRE_MORE_IMAGE = 0x99,
  EMBER_ZCL_STATUS_NOTIFICATION_PENDING = 0x9A,
  EMBER_ZCL_STATUS_HARDWARE_FAILURE = 0xC0,
  EMBER_ZCL_STATUS_SOFTWARE_FAILURE = 0xC1,
  EMBER_ZCL_STATUS_CALIBRATION_ERROR = 0xC2,
  EMBER_ZCL_STATUS_NULL = 0xFF }

6.102.1 Detailed Description

See zcl-core-types.h for source code.

6.102.2 Typedef Documentation

6.102.2.1 typedef uint16_t bitmap16_t

6.102.2.2 typedef uint32_t bitmap32_t

6.102.2.3 typedef uint64_t bitmap64_t

6.102.2.4 typedef uint8_t bitmap8_t

6.102.2.5 typedef uint16_t data16_t

6.102.2.6 typedef uint32_t data32_t

6.102.2.7 typedef uint64_t data64_t

6.102.2.8 typedef uint8_t data8_t

6.102.2.9 typedef uint16_t enum16_t

6.102.2.10 typedef uint8_t enum8_t

6.102.2.11 typedef uint32_t utc_time_t

6.102.3 Enumeration Type Documentation

6.102.3.1 enum EmberZclStatus_t

A success or failure status, used as a system-wide return type for functions.

Enumerator

EMBER_ZCL_STATUS_SUCCESS The operation was successful.

EMBER_ZCL_STATUS_FAILURE The operation was not successful.

EMBER_ZCL_STATUS_NOT_AUTHORIZED The sender of the command does not have authorization to carry out this command.

EMBER_ZCL_STATUS_RESERVED_FIELD_NOT_ZERO A reserved field/subfield/bit contains a non-zero value.

EMBER_ZCL_STATUS_MALFORMED_COMMAND The command appears to contain the wrong fields, as detected either by the presence of one or more invalid field entries or by missing fields. Command not carried out.

EMBER_ZCL_STATUS_UNSUP_CLUSTER_COMMAND The specified cluster command is not supported on the device. The command is not carried out.

EMBER_ZCL_STATUS_UNSUP_GENERAL_COMMAND The specified general ZCL command is not supported on the device.

EMBER_ZCL_STATUS_UNSUP_MANUF_CLUSTER_COMMAND A manufacturer-specific unicast, cluster specific command was received with an unknown manufacturer code, or the manufacturer code was recognized but the command is not supported.

EMBER_ZCL_STATUS_UNSUP_MANUF_GENERAL_COMMAND A manufacturer-specific unicast, ZCL specific command was received with an unknown manufacturer code, or the manufacturer code was recognized but the command is not supported.

EMBER_ZCL_STATUS_INVALID_FIELD At least one field of the command contains an incorrect value, according to the specification the device is implemented to.

EMBER_ZCL_STATUS_UNSUPPORTED_ATTRIBUTE The specified attribute does not exist on the device.

EMBER_ZCL_STATUS_INVALID_VALUE An out of range error, or set to a reserved value. An attribute keeps its old value. Note that an attribute value may be out of range if the attribute is related to another, e.g., with minimum and maximum attributes. See the individual attribute descriptions in ZCL specification for specific details.

EMBER_ZCL_STATUS_READ_ONLY Attempt to write a read only attribute.

EMBER_ZCL_STATUS_INSUFFICIENT_SPACE An operation (e.g., an attempt to create an entry in a table) failed due to an insufficient amount of free space available.

EMBER_ZCL_STATUS_DUPLICATE_EXISTS An attempt to create an entry in a table failed due to a duplicate entry already present in the table.

EMBER_ZCL_STATUS_NOT_FOUND The requested information (e.g., table entry) could not be found.

EMBER_ZCL_STATUS_UNREPORTABLE_ATTRIBUTE Periodic reports cannot be issued for this attribute.

EMBER_ZCL_STATUS_INVALID_DATA_TYPE The data type given for an attribute is incorrect. The command is not carried out.

EMBER_ZCL_STATUS_INVALID_SELECTOR The selector for an attribute is incorrect.

EMBER_ZCL_STATUS_WRITE_ONLY A request has been made to read an attribute that the requestor is not authorized to read. No action taken.

EMBER_ZCL_STATUS_INCONSISTENT_STARTUP_STATE Setting the requested values puts the device in an inconsistent state on startup. No action taken.

EMBER_ZCL_STATUS_DEFINED_OUT_OF_BAND An attempt has been made to write an attribute that is present but is defined using an out-of-band method and not over the air.

EMBER_ZCL_STATUS_INCONSISTENT The supplied values (e.g., contents of table cells) are inconsistent.

EMBER_ZCL_STATUS_ACTION_DENIED The credentials presented by the device sending the command are not sufficient to perform this action.

EMBER_ZCL_STATUS_TIMEOUT The exchange was aborted due to excessive response time.

EMBER_ZCL_STATUS_ABORT Failed case when a client or a server decides to abort the upgrade process.

EMBER_ZCL_STATUS_INVALID_IMAGE Invalid OTA upgrade image (ex. failed signature validation or signer information check or CRC check).

EMBER_ZCL_STATUS_WAIT_FOR_DATA Server does not have the data block available yet.

EMBER_ZCL_STATUS_NO_IMAGE_AVAILABLE No OTA upgrade image available for a particular client.

EMBER_ZCL_STATUS_REQUIRE_MORE_IMAGE The client still requires more OTA upgrade image files to successfully upgrade.

EMBER_ZCL_STATUS_NOTIFICATION_PENDING The command has been received and is being processed.

EMBER_ZCL_STATUS_HARDWARE_FAILURE An operation was unsuccessful due to a hardware failure.

EMBER_ZCL_STATUS_SOFTWARE_FAILURE An operation was unsuccessful due to a software failure.

EMBER_ZCL_STATUS_CALIBRATION_ERROR An error occurred during calibration.

EMBER_ZCL_STATUS_NULL Distinguished value that represents a null (invalid) status.

6.103 Discovery

Enumerations

- enum EmberZclDiscoveryRequestMode {
  EMBER_ZCL_DISCOVERY_REQUEST_SINGLE_QUERY = 0,
  EMBER_ZCL_DISCOVERY_REQUEST_MULTIPLE_QUERY = 1,
  EMBER_ZCL_DISCOVERY_REQUEST_MODE_MAX = 2 }

Functions

- void emberZclDiscInit (void)
- bool emberZclDiscSetMode (EmberZclDiscoveryRequestMode mode)
- bool emberZclDiscSend (EmberCoapResponseHandler responseHandler)
- bool emberZclDiscByClusterId (const EmberZclClusterSpec_t *clusterSpec, EmberCoapResponseHandler responseHandler)
- bool emberZclDiscByEndpoint (EmberZclEndpointId_t endpointId, EmberZclDeviceId_t deviceId, EmberCoapResponseHandler responseHandler)
- bool emberZclDiscByUid (const EmberZclUid_t *uid, uint16_t uidBits, EmberCoapResponseHandler responseHandler)
- bool emberZclDiscByClusterRev (EmberZclClusterRevision_t version, EmberCoapResponseHandler responseHandler)
- bool emberZclDiscByDeviceId (EmberZclDeviceId_t deviceId, EmberCoapResponseHandler responseHandler)
- bool emberZclDiscByResourceVersion (EmberZclClusterRevision_t version, EmberCoapResponseHandler responseHandler)

6.103.1 Detailed Description

See zcl-core-types.h for source code.

See zcl-core-well-known.h for source code.

6.103.2 Enumeration Type Documentation

6.103.2.1 enum EmberZclDiscoveryRequestMode

Defines possible request modes.

Enumerator

EMBER_ZCL_DISCOVERY_REQUEST_SINGLE_QUERY Discovery request is allowed a single query.

EMBER_ZCL_DISCOVERY_REQUEST_MULTIPLE_QUERY Discovery request is allowed multiple queries.

EMBER_ZCL_DISCOVERY_REQUEST_MODE_MAX Maximum discovery request mode.

6.103.3 Function Documentation

6.103.3.1 bool emberZclDiscByClusterId (const EmberZclClusterSpec_t * *clusterSpec,* EmberCoapResponseHandler *responseHandler*)

This function appends a cluster ID query to the discovery request string.

Parameters

clusterSpec	A structure for cluster ID / role / manufacture code
responseHandler	The response handler

Returns

in EMBER_ZCL_DISCOVERY_REQUEST_SINGLE_QUERY mode: True if the command was sent or false otherwise. in EMBER_ZCL_DISCOVERY_REQUEST_MULTIPLE_QUERY mode: True if the command was appended or false otherwise.

Referenced by emberZclOtaBootloadClientServerHasDiscByClusterId().

6.103.3.2 bool emberZclDiscByClusterRev (EmberZclClusterRevision_t *version,* EmberCoapResponseHandler *responseHandler*)

This function appends a cluster revision query to the discovery request string.

Parameters

version	The version
responseHandler	The response handler

Returns

in EMBER_ZCL_DISCOVERY_REQUEST_SINGLE_QUERY mode: True if the command was sent or false otherwise. in EMBER_ZCL_DISCOVERY_REQUEST_MULTIPLE_QUERY mode: True if the command was appended or false otherwise.

6.103.3.3 bool emberZclDiscByDeviceId (EmberZclDeviceId_t *deviceId,* EmberCoapResponseHandler *responseHandler*)

This function appends a device ID query to the discovery request string.

Parameters

deviceId	The device identifier
responseHandler	The response handler

Returns

in EMBER_ZCL_DISCOVERY_REQUEST_SINGLE_QUERY mode: True if the command was sent or false otherwise. in EMBER_ZCL_DISCOVERY_REQUEST_MULTIPLE_QUERY mode: True if the command was appended or false otherwise.

6.103.3.4 bool emberZclDiscByEndpoint (EmberZclEndpointId_t *endpointId,* EmberZclDeviceId_t *deviceId,* EmberCoapResponseHandler *responseHandler*)

This function appends an endpoint query to the Discovery request string.

Parameters

endpointId	The endpoint identifier
deviceId	The device identifier
responseHandler	The response handler

Returns

in EMBER_ZCL_DISCOVERY_REQUEST_SINGLE_QUERY mode: True if the command was sent or false otherwise. in EMBER_ZCL_DISCOVERY_REQUEST_MULTIPLE_QUERY mode: True if the command was appended or false otherwise.

6.103.3.5 bool emberZclDiscByResourceVersion (EmberZclClusterRevision_t *version,* EmberCoapResponseHandler *responseHandler*)

This function appends a resource version query to the discovery request string.

Parameters

version	The version
responseHandler	The response handler

Returns

in EMBER_ZCL_DISCOVERY_REQUEST_SINGLE_QUERY mode: True if the command was sent or false otherwise. in EMBER_ZCL_DISCOVERY_REQUEST_MULTIPLE_QUERY mode: True if the command was appended or false otherwise.

6.103.3.6 bool emberZclDiscByUid (const EmberZclUid_t * *uid,* uint16_t *uidBits,* EmberCoapResponseHandler *responseHandler*)

This function appends a UID query to the discovery request string.

Parameters

uid	The uid
uidBits	The uid bits
responseHandler	The response handler

Returns

in EMBER_ZCL_DISCOVERY_REQUEST_SINGLE_QUERY mode: True if the command was sent or false otherwise. in EMBER_ZCL_DISCOVERY_REQUEST_MULTIPLE_QUERY mode: True if the command was appended or false otherwise.

6.103.3.7 void emberZclDiscInit (void)

Initialization for sending Discovery command.

6.103.3.8 bool emberZclDiscSend (EmberCoapResponseHandler *responseHandler*)

This function broadcasts a GET using the Discovery request string.

Parameters

responseHandler	The response handler

Returns

True if the message was sent or false otherwise.

6.103.3.9 bool emberZclDiscSetMode (EmberZclDiscoveryRequestMode *mode*)

This function sets mode to create a query.

Parameters

mode	EMBER_ZCL_DISCOVERY_REQUEST_SINGLE_QUERY - single query EMBER_ZCL_DISCOVERY_REQUEST_MULTIPLE_QUERY - multiple queries

Returns

True if mode was set or false otherwise.

Under EMBER_ZCL_DISCOVERY_REQUEST_SINGLE_QUERY mode, appending one query string automatically triggers the Discovery command to be broadcast. Under EMBER_ZCL_DISCOVERY_REQUEST_MULTIPLE_↩ QUERY mode, appended query strings is accumulated. The accumulated query string will not be broadcast until emberZclDiscSend() is called.

6.104 Messages

Enumerations

- enum EmberZclMessageStatus_t {
  EMBER_ZCL_MESSAGE_STATUS_COAP_TIMEOUT = EMBER_COAP_MESSAGE_TIMED_OUT,
  EMBER_ZCL_MESSAGE_STATUS_COAP_ACK = EMBER_COAP_MESSAGE_ACKED,
  EMBER_ZCL_MESSAGE_STATUS_COAP_RESET = EMBER_COAP_MESSAGE_RESET,
  EMBER_ZCL_MESSAGE_STATUS_COAP_RESPONSE = EMBER_COAP_MESSAGE_RESPONSE,
  EMBER_ZCL_MESSAGE_STATUS_DISCOVERY_TIMEOUT,
  EMBER_ZCL_MESSAGE_STATUS_NULL = 0xFF }

6.104.1 Detailed Description

See zcl-core-types.h for source code.

6.104.2 Enumeration Type Documentation

6.104.2.1 enum EmberZclMessageStatus_t

Defines possible message statuses.

Enumerator

EMBER_ZCL_MESSAGE_STATUS_COAP_TIMEOUT CoAP EMBER_COAP_MESSAGE_TIMED_OUT status recevied.

EMBER_ZCL_MESSAGE_STATUS_COAP_ACK CoAP EMBER_COAP_MESSAGE_ACKED status received.

EMBER_ZCL_MESSAGE_STATUS_COAP_RESET CoAP EMBER_COAP_MESSAGE_RESET status received.

EMBER_ZCL_MESSAGE_STATUS_COAP_RESPONSE CoAP EMBER_COAP_MESSAGE_RESPONSE status received.

EMBER_ZCL_MESSAGE_STATUS_DISCOVERY_TIMEOUT Discovery timed out.

EMBER_ZCL_MESSAGE_STATUS_NULL CoAP unknown status recevied.

6.105 Addresses

Data Structures

- struct EmberZclUid_t
- struct EmberZclCoapEndpoint_t
- struct EmberZclApplicationDestination_t
- struct EmberZclDestination_t

Macros

- #define EMBER_ZCL_UID_BITS 256
- #define EMBER_ZCL_UID_SIZE EMBER_BITS_TO_BYTES(EMBER_ZCL_UID_BITS)
- #define EMBER_ZCL_UID_STRING_LENGTH (EMBER_ZCL_UID_BITS / 4)
- #define EMBER_ZCL_UID_STRING_SIZE (EMBER_ZCL_UID_STRING_LENGTH + 1)
- #define EMBER_ZCL_UID_BASE64URL_LENGTH (((EMBER_ZCL_UID_SIZE ∗ 8) + 5) / 6)
- #define EMBER_ZCL_UID_BASE64URL_SIZE (EMBER_ZCL_UID_BASE64URL_LENGTH + 1)

Enumerations

- enum {
  EMBER_ZCL_NO_FLAGS = 0x00,
  EMBER_ZCL_USE_COAPS_FLAG = 0x01,
  EMBER_ZCL_HAVE_IPV6_ADDRESS_FLAG = 0x02,
  EMBER_ZCL_HAVE_UID_FLAG = 0x04 }
- enum EmberZclApplicationDestinationType_t {
  EMBER_ZCL_APPLICATION_DESTINATION_TYPE_ENDPOINT = 0x00,
  EMBER_ZCL_APPLICATION_DESTINATION_TYPE_GROUP = 0x01 }

6.105.1 Detailed Description

See zcl-core-types.h for source code.

6.105.2 Macro Definition Documentation

6.105.2.1 #define EMBER_ZCL_UID_BASE64URL_LENGTH (((EMBER_ZCL_UID_SIZE ∗ 8) + 5) / 6)

Text string length to represent UID length (base64url characters).

6.105.2.2 #define EMBER_ZCL_UID_BASE64URL_SIZE (EMBER_ZCL_UID_BASE64URL_LENGTH + 1)

Text string length to represent UID length (base64url characters), plus trailing NUL.

6.105.2.3 #define EMBER_ZCL_UID_BITS 256

UID size in bits.

6.105.2.4 #define EMBER_ZCL_UID_SIZE EMBER_BITS_TO_BYTES(EMBER_ZCL_UID_BITS)

UID size in bytes.

6.105.2.5 #define EMBER_ZCL_UID_STRING_LENGTH (EMBER_ZCL_UID_BITS / 4)

Text string length to represent a UID (hexadecimal characters).

6.105.2.6 #define EMBER_ZCL_UID_STRING_SIZE (EMBER_ZCL_UID_STRING_LENGTH + 1)

Text string length to represent a UID (hexadecimal characters), plus trailing NUL.

6.105.3 Enumeration Type Documentation

6.105.3.1 anonymous enum

Enumerator

EMBER_ZCL_NO_FLAGS

EMBER_ZCL_USE_COAPS_FLAG

EMBER_ZCL_HAVE_IPV6_ADDRESS_FLAG

EMBER_ZCL_HAVE_UID_FLAG

6.105.3.2 enum EmberZclApplicationDestinationType_t

Defines possible types for an application destination.

Enumerator

EMBER_ZCL_APPLICATION_DESTINATION_TYPE_ENDPOINT An application destination uses an endpoint type.

EMBER_ZCL_APPLICATION_DESTINATION_TYPE_GROUP An application destination uses a group type.

6.106 Endpoints

Macros

- #define EMBER_ZCL_ENDPOINT_MIN 0x01
- #define EMBER_ZCL_ENDPOINT_MAX 0xF0
- #define EMBER_ZCL_ENDPOINT_NULL ((EmberZclEndpointId_t)-1)
- #define EMBER_ZCL_ENDPOINT_INDEX_NULL ((EmberZclEndpointIndex_t)-1)
- #define EMBER_ZCL_DEVICE_ID_NULL ((EmberZclDeviceId_t)-1)

Typedefs

- typedef uint8_t EmberZclEndpointId_t
- typedef uint8_t EmberZclEndpointIndex_t
- typedef uint16_t EmberZclDeviceId_t

Functions

- EmberZclEndpointIndex_t emberZclEndpointIdToIndex (EmberZclEndpointId_t endpointId, const EmberZcl↩ ClusterSpec_t *clusterSpec)
- EmberZclEndpointId_t emberZclEndpointIndexToId (EmberZclEndpointIndex_t index, const EmberZcl↩ ClusterSpec_t *clusterSpec)

6.106.1 Detailed Description

See zcl-core-types.h for source code.

See zcl-core.h for source code.

6.106.2 Macro Definition Documentation

6.106.2.1 #define EMBER_ZCL_DEVICE_ID_NULL ((EmberZclDeviceId_t)-1)

A distinguished value that represents a null (invalid) device identifer.

6.106.2.2 #define EMBER_ZCL_ENDPOINT_INDEX_NULL ((EmberZclEndpointIndex_t)-1)

A distinguished value that represents a null (invalid) endpoint index.

6.106.2.3 #define EMBER_ZCL_ENDPOINT_MAX 0xF0

A maximum endpoint identifer value.

6.106.2.4 #define EMBER_ZCL_ENDPOINT_MIN 0x01

A minimum endpoint identifer value.

6.106.2.5 #define EMBER_ZCL_ENDPOINT_NULL ((EmberZclEndpointId_t)-1)

A distinguished value that represents a null (invalid) endpoint identifer.

6.106.3 Typedef Documentation

6.106.3.1 typedef uint16_t EmberZclDeviceId_t

A device identifier.

6.106.3.2 typedef uint8_t EmberZclEndpointId_t

An endpoint identifier.

6.106.3.3 typedef uint8_t EmberZclEndpointIndex_t

An endpoint index.

6.106.4 Function Documentation

6.106.4.1 EmberZclEndpointIndex_t emberZclEndpointIdToIndex (EmberZclEndpointId_t *endpointId,* const EmberZclClusterSpec_t * *clusterSpec*)

This function finds the endpoint index for the specified endpoint identifier and cluster specification.

Parameters

endpointId	An endpoint identifier
clusterSpec	A cluster specification or NULL

Returns

An endpoint index or EMBER_ZCL_ENDPOINT_INDEX_NULL if no match is found.

This function searches the endpoint table and returns the endpoint index for the entry that matches the specified endpoint identifier and cluster specification. If clusterSpec is NULL, match on endpointId only.

Note

The endpoint index is the zero-based relative position of an endpoint among the subset of endpoints that support the specified cluster. For example, an index of 3 refers to the fourth endpoint that supports the specified cluster. The value of endpoint index for a given endpoint identifier may be different for different clusters.

See also

emberZclEndpointIndexToId()

6.106.4.2 EmberZclEndpointId_t emberZclEndpointIndexToId (EmberZclEndpointIndex_t *index,* const EmberZclClusterSpec_t * *clusterSpec*)

This function finds the endpoint identifier for the specified endpoint index and cluster specification.

Parameters

index	An endpoint index
clusterSpec	A cluster specification or NULL

Returns

An endpoint identifier or EMBER_ZCL_ENDPOINT_NULL if no match is found.

This function searches the endpoint table and returns the endpoint identifier for the entry that matches the specified endpoint index and cluster specification. If clusterSpec is NULL, match on index only.

Note

The endpoint index is the zero-based relative position of an endpoint among the subset of endpoints that support the specified cluster. For example, an index of 3 refers to the fourth endpoint that supports the specified cluster. The value of endpoint index for a given endpoint identifier may be different for different clusters.

See also

emberZclEndpointIdToIndex()

6.107 Groups

Data Structures

- struct EmberZclGroupEntry_t

Macros

- #define EMBER_ZCL_GROUP_ALL_ENDPOINTS 0xFFFF
- #define EMBER_ZCL_GROUP_MIN 0x0001
- #define EMBER_ZCL_GROUP_MAX 0xFFF7
- #define EMBER_ZCL_GROUP_NULL 0x0000
- #define EMBER_ZCL_MAX_GROUP_NAME_LENGTH 0

Typedefs

- typedef uint16_t EmberZclGroupId_t

Functions

- bool emberZclIsEndpointInGroup (EmberZclEndpointId_t endpointId, EmberZclGroupId_t groupId)
- bool emberZclGetGroupName (EmberZclEndpointId_t endpointId, EmberZclGroupId_t groupId, uint8_t *groupName, uint8_t *groupNameLength)
- EmberZclStatus_t emberZclAddEndpointToGroup (EmberZclEndpointId_t endpointId, EmberZclGroupId_t groupId, const uint8_t *groupName, uint8_t groupNameLength)
- EmberZclStatus_t emberZclRemoveEndpointFromGroup (EmberZclEndpointId_t endpointId, EmberZclGroupId_t groupId)
- EmberZclStatus_t emberZclRemoveEndpointFromAllGroups (EmberZclEndpointId_t endpointId)
- EmberZclStatus_t emberZclRemoveGroup (EmberZclGroupId_t groupId)
- EmberZclStatus_t emberZclRemoveAllGroups (void)

6.107.1 Detailed Description

See zcl-core-types.h for source code.

See zcl-core.h for source code.

6.107.2 Macro Definition Documentation

6.107.2.1 #define EMBER_ZCL_GROUP_ALL_ENDPOINTS 0xFFFF

A group identifier for the all-endpoints (endpoint broadcast) group.

All endpoints are always members of this group. This group cannot be removed and no endpoint can be removed from it.

6.107.2.2 #define EMBER_ZCL_GROUP_MAX 0xFFF7

A maximum group identifier value.

6.107.2.3 #define EMBER_ZCL_GROUP_MIN 0x0001

A minimum group identifier value.

6.107.2.4 #define EMBER_ZCL_GROUP_NULL 0x0000

A distinguished value that represents a null (invalid) group identifier.

6.107.2.5 #define EMBER_ZCL_MAX_GROUP_NAME_LENGTH 0

6.107.3 Typedef Documentation

6.107.3.1 typedef uint16_t EmberZclGroupId_t

A group identifier.

6.107.4 Function Documentation

6.107.4.1 EmberZclStatus_t emberZclAddEndpointToGroup (EmberZclEndpointId_t *endpointId,* EmberZclGroupId_t *groupId,* const uint8_t * *groupName,* uint8_t *groupNameLength*)

This function adds an endpoint to a group.

Parameters

endpointId	An endpoint identifier
groupId	A group identifier
groupName	A pointer to an array containing name of group
groupNameLength	Length of group name array (groupName is ignored if this length is 0)

Returns

- EMBER_ZCL_STATUS_SUCCESS if the endpoint was added to the group
- EMBER_ZCL_STATUS_DUPLICATE_EXISTS if the endpoint is already a member of the group
- EMBER_ZCL_STATUS_INSUFFICIENT_SPACE if there is no capacity to store the endpoint/group association
- EMBER_ZCL_STATUS_FAILURE if groupId is EMBER_ZCL_GROUP_ALL_ENDPOINTS, or if groupId is not a value between EMBER_ZCL_GROUP_MIN and EMBER_ZCL_GROUP_MAX inclusive, or if groupNameLength is greater than EMBER_ZCL_MAX_GROUP_NAME_LENGTH, or if groupName↩ Length is non-zero and groupName is NULL

6.107.4.2 bool emberZclGetGroupName (EmberZclEndpointId_t *endpointId,* EmberZclGroupId_t *groupId,* uint8_t * *groupName,* uint8_t * *groupNameLength*)

This function gets a group name and its length.

Parameters

endpointId	An endpoint identifier
groupId	A group identifier
groupName	An array pointer which will contain the group name
groupNameLength	A pointer which will contain the group name length

Returns

`true` if group name was retrieved successfully, `false` otherwise.

6.107.4.3 bool emberZclIsEndpointInGroup (EmberZclEndpointId_t *endpointId,* EmberZclGroupId_t *groupId*)

This function determines if an endpoint is a member of a group.

Parameters

endpoint↩ Id	An endpoint identifier
groupId	A group identifier

Returns

`true` if the endpoint is a member of the group, `false` otherwise.

6.107.4.4 EmberZclStatus_t emberZclRemoveAllGroups (void)

This function removes all groups.

Returns

- EMBER_ZCL_STATUS_SUCCESS if all groups were removed (other than the all-endpoints group)
- EMBER_ZCL_STATUS_NOT_FOUND if no groups exist (other than the all-endpoints group)

6.107.4.5 EmberZclStatus_t emberZclRemoveEndpointFromAllGroups (EmberZclEndpointId_t *endpointId*)

This function removes an endpoint from all groups to which it belongs.

Parameters

endpoint↩ Id	An endpoint identifier

Returns

- EMBER_ZCL_STATUS_SUCCESS if the endpoint was removed from one or more groups
- EMBER_ZCL_STATUS_NOT_FOUND if the endpoint is not a member of any group (other than the all-endpoints group)

6.107.4.6 EmberZclStatus_t emberZclRemoveEndpointFromGroup (EmberZclEndpointId_t *endpointId,* EmberZclGroupId_t *groupId*)

This function removes an endpoint from a group.

Parameters

endpoint↩ Id	An endpoint identifier
groupId	A group identifier

Returns

- EMBER_ZCL_STATUS_SUCCESS if the endpoint was removed from the group
- EMBER_ZCL_STATUS_NOT_FOUND if the endpoint is not a member of the group
- EMBER_ZCL_STATUS_INVALID_FIELD if the groupId is not a value between EMBER_ZCL_GROU↩ P_MIN and EMBER_ZCL_GROUP_MAX inclusive

6.107.4.7 EmberZclStatus_t emberZclRemoveGroup (EmberZclGroupId_t *groupId*)

This function removes a group.

Parameters

group↩ Id	A group identifier

Returns

- EMBER_ZCL_STATUS_SUCCESS if the group was removed
- EMBER_ZCL_STATUS_NOT_FOUND if the group does not exist
- EMBER_ZCL_STATUS_INVALID_FIELD if the groupId is not a value between EMBER_ZCL_GROU↩ P_MIN and EMBER_ZCL_GROUP_MAX inclusive

6.108 Clusters

Data Structures

- struct EmberZclClusterSpec_t

Macros

- #define EMBER_ZCL_MANUFACTURER_CODE_NULL 0x0000
- #define EMBER_ZCL_CLUSTER_NULL ((EmberZclClusterId_t)-1)

Typedefs

- typedef uint16_t EmberZclManufacturerCode_t
- typedef uint16_t EmberZclClusterId_t

Enumerations

- enum EmberZclRole_t {
  EMBER_ZCL_ROLE_CLIENT = 0,
  EMBER_ZCL_ROLE_SERVER = 1 }

Functions

- int32_t emberZclCompareClusterSpec (const EmberZclClusterSpec_t *s1, const EmberZclClusterSpec_t *s2)
- bool emberZclAreClusterSpecsEqual (const EmberZclClusterSpec_t *s1, const EmberZclClusterSpec_t *s2)
- void emberZclReverseClusterSpec (const EmberZclClusterSpec_t *s1, EmberZclClusterSpec_t *s2)

6.108.1 Detailed Description

See zcl-core-types.h for source code.

See zcl-core.h for source code.

6.108.2 Macro Definition Documentation

6.108.2.1 #define EMBER_ZCL_CLUSTER_NULL ((EmberZclClusterId_t)-1)

A distinguished value that represents a null (invalid) cluster identifier.

6.108.2.2 #define EMBER_ZCL_MANUFACTURER_CODE_NULL 0x0000

A distinguished value that represents a null (invalid) manufacturer code.

6.108.3 Typedef Documentation

6.108.3.1 typedef uint16_t EmberZclClusterId_t

A cluster identifier.

6.108.3.2 typedef uint16_t EmberZclManufacturerCode_t

A manufacturer code.

6.108.4 Enumeration Type Documentation

6.108.4.1 enum EmberZclRole_t

Defines possible roles of a cluster.

Enumerator

EMBER_ZCL_ROLE_CLIENT Cluster is a client.

EMBER_ZCL_ROLE_SERVER Cluster is a server.

6.108.5 Function Documentation

6.108.5.1 bool emberZclAreClusterSpecsEqual (const EmberZclClusterSpec_t * *s1*, const EmberZclClusterSpec_t * *s2*)

This function compares two cluster specifications.

Parameters

s1	A cluster specification to be compared
s2	A cluster specification to be compared

Returns

`true` if both cluster specifications are equal, `false` otherwise.

6.108.5.2 int32_t emberZclCompareClusterSpec (const EmberZclClusterSpec_t * *s1*, const EmberZclClusterSpec_t * *s2*)

This function compares two cluster specifications.

Parameters

s1	A cluster specification to be compared
s2	A cluster specification to be compared

Returns

- `0` if both cluster specifications are equal
- `-1` if `s1`'s EmberZclRole_t is not equal to `s2`'s EmberZclRole_t and `s1`'s role is EMBER_ZCL_ROLE_CLIENT
- `1` if `s1`'s EmberZclRole_t is not equal to `s2`'s EmberZclRole_t and `s1`'s role is EMBER_ZCL_ROLE_SERVER
- difference between EmberZclManufacturerCode_t of `s1` and `s2`, or difference between EmberZclClusterId_t of `s1` and `s2`

6.108.5.3 void emberZclReverseClusterSpec (const EmberZclClusterSpec_t * *s1*, EmberZclClusterSpec_t * *s2*)

This function reverses cluster specifications.

Parameters

s1	A cluster specification used for reversing
s2	A cluster specification to be reversed

This function changes EmberZclRole_t of `s2` to be opposite of `s1`. It also sets EmberZclManufacturerCode_t and EmberZclClusterId_t of `s2` to be the same as `s1`.

6.109 Attributes

Data Structures

- struct EmberZclAttributeContext_t
- struct EmberZclAttributeWriteData_t

Macros

- #define EMBER_ZCL_ATTRIBUTE_CLUSTER_REVISION 0xFFFD
- #define EMBER_ZCL_ATTRIBUTE_REPORTING_STATUS 0xFFFE
- #define EMBER_ZCL_ATTRIBUTE_NULL ((EmberZclAttributeId_t)-1)
- #define EMBER_ZCL_CLUSTER_REVISION_PRE_ZCL6 0
- #define EMBER_ZCL_CLUSTER_REVISION_ZCL6 1
- #define EMBER_ZCL_CLUSTER_REVISION_NULL ((EmberZclClusterRevision_t)-1)

Typedefs

- typedef uint16_t EmberZclAttributeId_t
- typedef uint16_t EmberZclClusterRevision_t
- typedef void(∗ EmberZclReadAttributeResponseHandler) (EmberZclMessageStatus_t status, const EmberZclAttributeContext_t ∗context, const void ∗buffer, size_t bufferLength)
- typedef void(∗ EmberZclWriteAttributeResponseHandler) (EmberZclMessageStatus_t status, const EmberZclAttributeContext_t ∗context)

Functions

- void emberZclResetAttributes (EmberZclEndpointId_t endpointId)
- EmberZclStatus_t emberZclReadAttribute (EmberZclEndpointId_t endpointId, const EmberZclClusterSpec_t ∗clusterSpec, EmberZclAttributeId_t attributeId, void ∗buffer, size_t bufferLength)
- EmberZclStatus_t emberZclWriteAttribute (EmberZclEndpointId_t endpointId, const EmberZclClusterSpec_t ∗clusterSpec, EmberZclAttributeId_t attributeId, const void ∗buffer, size_t bufferLength)
- EmberZclStatus_t emberZclExternalAttributeChanged (EmberZclEndpointId_t endpointId, const EmberZclClusterSpec_t ∗clusterSpec, EmberZclAttributeId_t attributeId, const void ∗buffer, size_t bufferLength)
- EmberStatus emberZclSendAttributeRead (const EmberZclDestination_t ∗destination, const EmberZclClusterSpec_t ∗clusterSpec, const EmberZclAttributeId_t ∗attributeIds, size_t attributeIdsCount, const EmberZclReadAttributeResponseHandler handler)
- EmberStatus emberZclSendAttributeWrite (const EmberZclDestination_t ∗destination, const EmberZclClusterSpec_t ∗clusterSpec, const EmberZclAttributeWriteData_t ∗attributeWriteData, size_t attributeWriteDataCount, const EmberZclWriteAttributeResponseHandler handler)

6.109.1 Detailed Description

See zcl-core-types.h for source code.

See zcl-core.h for source code.

6.109.2 Macro Definition Documentation

6.109.2.1 #define EMBER_ZCL_ATTRIBUTE_CLUSTER_REVISION 0xFFFD

An attribute identifier for the Cluster revision.

6.109.2.2 #define EMBER_ZCL_ATTRIBUTE_NULL ((EmberZclAttributeId_t)-1)

A distinguished value that represents a null (invalid) attribute identifier.

6.109.2.3 #define EMBER_ZCL_ATTRIBUTE_REPORTING_STATUS 0xFFFE

An attribute identifier for a Reporting status.

6.109.2.4 #define EMBER_ZCL_CLUSTER_REVISION_NULL ((EmberZclClusterRevision_t)-1)

A distinguished value that represents a null (invalid) cluster revision.

6.109.2.5 #define EMBER_ZCL_CLUSTER_REVISION_PRE_ZCL6 0

A cluster revision for Pre-ZCL 6 specification.

6.109.2.6 #define EMBER_ZCL_CLUSTER_REVISION_ZCL6 1

A cluster revision for ZCL 6 specification.

6.109.3 Typedef Documentation

6.109.3.1 typedef uint16_t EmberZclAttributeId_t

An attribute identifier.

6.109.3.2 typedef uint16_t EmberZclClusterRevision_t

A cluster revision.

6.109.3.3 typedef void(* EmberZclReadAttributeResponseHandler) (EmberZclMessageStatus_t status, const EmberZclAttributeContext_t *context, const void *buffer, size_t bufferLength)

A handler fired when reading attributes.

Parameters

status	A message status
context	A context of a read attribute
buffer	A content of a read attribute
bufferLength	Content length

Note

`context->status` shows whether attribute was read successfully or if an error occured. If successful, a buffer contains an attribute value. If unsuccessful, the buffer is irrelevant.

See also

emberZclSendAttributeRead()

6.109.3.4 typedef void(∗ EmberZclWriteAttributeResponseHandler) (EmberZclMessageStatus_t status, const EmberZclAttributeContext_t ∗context)

A handler fired when writing attributes.

Parameters

status	A message status
context	A context of a written attribute

Note

`context->status` shows whether attribute was written successfully or if an error occured.

See also

emberZclSendAttributeWrite()

6.109.4 Function Documentation

6.109.4.1 EmberZclStatus_t emberZclExternalAttributeChanged (EmberZclEndpointId_t *endpointId,* const EmberZclClusterSpec_t ∗ *clusterSpec,* EmberZclAttributeId_t *attributeId,* const void ∗ *buffer,* size_t *bufferLength*)

This function notifies the core code that an externally stored attribute has changed.

Parameters

endpointId	An endpoint identifier of the attribute
clusterSpec	A cluster specification of the attribute
attributeId	An attribute identifier that is changed
buffer	A buffer to write the attribute from
bufferLength	Length of the write buffer

Returns

- EMBER_ZCL_STATUS_SUCCESS if the function call was successful
- EMBER_ZCL_STATUS_UNSUPPORTED_ATTRIBUTE if the attribute is not external or if the attribute is not supported on a specified endpoint
- EmberStatus with failure reason otherwise

Note

emberZclPostAttributeChangeCallback is triggered for a successful call to this function.

6.109.4.2 EmberZclStatus_t emberZclReadAttribute (EmberZclEndpointId_t *endpointId,* const EmberZclClusterSpec_t * *clusterSpec,* EmberZclAttributeId_t *attributeId,* void * *buffer,* size_t *bufferLength*)

This function reads the value of an attibute.

Parameters

endpointId	An endpoint identifier of the attribute
clusterSpec	A cluster specification of the attribute
attributeId	An attribute identifier to read
buffer	A buffer to read the attribute into
bufferLength	Length of the read buffer

Returns

- EMBER_ZCL_STATUS_SUCCESS if the attribute was read successfully
- EMBER_ZCL_STATUS_UNSUPPORTED_ATTRIBUTE if the attribute is remote or if the attribute is not supported on a specified endpoint
- EMBER_ZCL_STATUS_INSUFFICIENT_SPACE if not enough space is available in the passed buffer to store the attribute value
- EmberStatus with failure reason otherwise

Note

emberZclReadExternalAttributeCallback is triggered if attribute is externally stored. If so, the result of that call is returned.

6.109.4.3 void emberZclResetAttributes (EmberZclEndpointId_t *endpointId*)

This function resets all local attributes on given endpoint.

Parameters

endpoint↩ Id	An endpoint identifier

Note

emberZclPostAttributeChangeCallback is triggered after each attribute is changed.

6.109.4.4 EmberStatus emberZclSendAttributeRead (const EmberZclDestination_t * *destination*, const EmberZclClusterSpec_t * *clusterSpec*, const EmberZclAttributeId_t * *attributeIds*, size_t *attributeIdsCount*, const EmberZclReadAttributeResponseHandler *handler*)

This function sends an attribute read command to a specified destination.

Parameters

destination	A location to read the attribute from
clusterSpec	A cluster specification of the attribute
attributeIds	An array of attribute IDs to read
attributeIdsCount	A total count of EmberZclAttributeId_t elements in a passed array
handler	callback that is triggered for a response

Returns

- EMBER_ZCL_STATUS_SUCCESS if function call was successful
- EmberStatus with failure reason otherwise

See also

EmberZclReadAttributeResponseHandler()

6.109.4.5 EmberStatus emberZclSendAttributeWrite (const EmberZclDestination_t * *destination*, const EmberZclClusterSpec_t * *clusterSpec*, const EmberZclAttributeWriteData_t * *attributeWriteData*, size_t *attributeWriteDataCount*, const EmberZclWriteAttributeResponseHandler *handler*)

This function sends an attribute write command to a specified destination.

Parameters

destination	A location to write the attribute to
clusterSpec	A cluster specification of the attribute
attributeWriteData	An array containing write data for attributes
attributeWriteDataCount	A total count of EmberZclAttributeWriteData_t elements in a passed array
handler	A callback that is triggered for a response

Returns

- EMBER_ZCL_STATUS_SUCCESS if the function call was successful
- EmberStatus with failure reason otherwise

See also

EmberZclWriteAttributeResponseHandler()

6.109.4.6 EmberZclStatus_t emberZclWriteAttribute (EmberZclEndpointId_t *endpointId*, const EmberZclClusterSpec_t * *clusterSpec*, EmberZclAttributeId_t *attributeId*, const void * *buffer*, size_t *bufferLength*)

This function writes the value of an attibute.

Parameters

endpointId	An endpoint identifier of the attribute
clusterSpec	A cluster specification of the attribute
attributeId	An attribute identifier to write
buffer	A buffer to write the attribute from
bufferLength	Length of the write buffer

Returns

- EMBER_ZCL_STATUS_SUCCESS if the attribute was written successfully
- EMBER_ZCL_STATUS_UNSUPPORTED_ATTRIBUTE if the attribute is remote or if the attribute is not supported on a specified endpoint
- EMBER_ZCL_STATUS_INSUFFICIENT_SPACE if not enough space is available in the attribute table to store the attribute value
- EMBER_ZCL_STATUS_INVALID_VALUE if the attribute value is invalid
- EMBER_ZCL_STATUS_FAILURE if emberZclPreAttributeChangeCallback returned `false`
- EmberStatus with failure reason otherwise

Note

emberZclWriteExternalAttributeCallback is triggered if the attribute is externally stored. If so, the result of that call is returned.
emberZclPostAttributeChangeCallback is triggered after the attribute is successfully changed.

6.110 Bindings

Data Structures

- struct EmberZclBindingContext_t
- struct EmberZclBindingEntry_t

Macros

- #define EMBER_ZCL_BINDING_NULL ((EmberZclBindingId_t)-1)

Typedefs

- typedef uint8_t EmberZclBindingId_t
- typedef void(∗ EmberZclBindingResponseHandler) (EmberZclMessageStatus_t status, const EmberZcl↩ BindingContext_t ∗context, const EmberZclBindingEntry_t ∗entry)

Enumerations

- enum EmberZclScheme_t {
  EMBER_ZCL_SCHEME_COAP = 0x00,
  EMBER_ZCL_SCHEME_COAPS = 0x01 }
- enum EmberZclNetworkDestinationType_t {
  EMBER_ZCL_NETWORK_DESTINATION_TYPE_ADDRESS = 0x00,
  EMBER_ZCL_NETWORK_DESTINATION_TYPE_UID = 0x01 }

Functions

- bool emberZclHasBinding (EmberZclBindingId_t bindingId)
- bool emberZclGetBinding (EmberZclBindingId_t bindingId, EmberZclBindingEntry_t ∗entry)
- bool emberZclSetBinding (EmberZclBindingId_t bindingId, const EmberZclBindingEntry_t ∗entry)
- EmberZclBindingId_t emberZclAddBinding (const EmberZclBindingEntry_t ∗entry)
- bool emberZclRemoveBinding (EmberZclBindingId_t bindingId)
- bool emberZclRemoveAllBindings (void)
- EmberStatus emberZclSendAddBinding (const EmberZclDestination_t ∗destination, const EmberZcl↩ BindingEntry_t ∗entry, const EmberZclBindingResponseHandler handler)
- EmberStatus emberZclSendUpdateBinding (const EmberZclDestination_t ∗destination, const EmberZcl↩ BindingEntry_t ∗entry, EmberZclBindingId_t bindingId, const EmberZclBindingResponseHandler handler)
- EmberStatus emberZclSendRemoveBinding (const EmberZclDestination_t ∗destination, const EmberZcl↩ ClusterSpec_t ∗clusterSpec, EmberZclBindingId_t bindingId, const EmberZclBindingResponseHandler han- dler)
- bool emberZclGetDestinationFromBinding (const EmberZclClusterSpec_t ∗clusterSpec, EmberZclBinding↩ Id_t ∗bindingIdx, EmberZclDestination_t ∗destination)

6.110.1 Detailed Description

See zcl-core-types.h for source code.

See zcl-core.h for source code.

6.110.2 Macro Definition Documentation

6.110.2.1 #define EMBER_ZCL_BINDING_NULL ((EmberZclBindingId_t)-1)

A distinguished value that represents a null (invalid) binding identifier.

6.110.3 Typedef Documentation

6.110.3.1 typedef uint8_t EmberZclBindingId_t

A binding identifier.

6.110.3.2 typedef void(* EmberZclBindingResponseHandler) (EmberZclMessageStatus_t status, const EmberZclBindingContext_t *context, const EmberZclBindingEntry_t *entry)

A handler fired when adding, updating, or removing a binding.

Parameters

status	A message status
context	A context of binding to add, update, or remove
entry	An entry of binding to add, update, or remove

See also

emberZclSendAddBinding()
emberZclSendUpdateBinding()
emberZclSendRemoveBinding()

6.110.4 Enumeration Type Documentation

6.110.4.1 enum EmberZclNetworkDestinationType_t

Defines possible types for a network destination.

Enumerator

EMBER_ZCL_NETWORK_DESTINATION_TYPE_ADDRESS A network destination uses an address type.

EMBER_ZCL_NETWORK_DESTINATION_TYPE_UID A network destination uses a unique identifier type.

6.110.4.2 enum EmberZclScheme_t

Defines possible schemes for a network destination.

Enumerator

EMBER_ZCL_SCHEME_COAP Network destination uses standard CoAP scheme.

EMBER_ZCL_SCHEME_COAPS Network destination uses secure CoAP scheme.

6.110.5 Function Documentation

6.110.5.1 EmberZclBindingId_t emberZclAddBinding (const EmberZclBindingEntry_t * *entry*)

This function adds a given entry to the binding table.

Parameters

entry	A binding entry to add to the table

Returns

- A binding identifier of entry if it was added successfully
- EMBER_ZCL_BINDING_NULL if entry was not added successfully

This function checks the binding table for duplicates. If a duplicate is found, a binding identifier of the previous entry is used. Otherwise, a new one is allocated. This function also validates contents of the given binding entry.

6.110.5.2 bool emberZclGetBinding (EmberZclBindingId_t *bindingId,* EmberZclBindingEntry_t * *entry*)

This function gets a specified binding.

Parameters

bindingId	A binding identifier of an entry to get
entry	A binding entry to put the retrieved binding into

Returns

- `true` if binding was retrieved successfully
- `false` if binding was not retrieved successfully

6.110.5.3 bool emberZclGetDestinationFromBinding (const EmberZclClusterSpec_t * *clusterSpec,* EmberZclBindingId_t * *bindingIdx,* EmberZclDestination_t * *destination*)

This function gets destination from specified matching binding

Parameters

clusterSpec	cluster specification of binding entry to remove
bindingIdx	index to start searching the binding table (index value is incremented on return if a matching binding is found)
destination	the destination of the matching binding entry

Returns

- `true` if matching binding was found
- `false` if matching binding was not found

See also

EmberZclGetDestinationFromBinding()

6.110.5.4 bool emberZclHasBinding (EmberZclBindingId_t *bindingId*)

This function checks whether a specified binding exists.

Parameters

bindingId	A binding identifier to check

Returns

- `true` if binding exists
- `false` if binding does not exist

6.110.5.5 bool emberZclRemoveAllBindings (void)

This function removes all entries from the binding table.

Returns

- `true` if all entries were removed successfully
- `false` if all entries were not removed successfully

6.110.5.6 bool emberZclRemoveBinding (EmberZclBindingId_t *bindingId*)

This function removes a specified entry from the binding table.

Parameters

bindingId	A binding identifier of an entry to be removed from the table

Returns

- `true` if an entry was removed successfully
- `false` if an entry was not removed successfully

6.110.5.7 EmberStatus emberZclSendAddBinding (const EmberZclDestination_t * *destination*, const EmberZclBindingEntry_t * *entry*, const EmberZclBindingResponseHandler *handler*)

This function sends a command to a specified destination to add a binding.

Parameters

destination	A location to send the command to
entry	A binding entry to add to destination's binding table
handler	A callback that is triggered for a response

Returns

- EMBER_ZCL_STATUS_SUCCESS if the function call was successful
- EmberStatus with failure reason otherwise

See also

EmberZclBindingResponseHandler()

6.110.5.8 EmberStatus emberZclSendRemoveBinding (const EmberZclDestination_t * *destination*, const EmberZclClusterSpec_t * *clusterSpec*, EmberZclBindingId_t *bindingId*, const EmberZclBindingResponseHandler *handler*)

This function sends a command to a specified destination to remove a binding.

Parameters

destination	location to send command to
clusterSpec	cluster specification of binding entry to remove
bindingId	Binding identifier to remove from destination's binding table
handler	callback that is triggered for response

Returns

- EMBER_ZCL_STATUS_SUCCESS if function call was successful
- EmberStatus with failure reason otherwise

See also

EmberZclBindingResponseHandler()

6.110.5.9 EmberStatus emberZclSendUpdateBinding (const EmberZclDestination_t * *destination*, const EmberZclBindingEntry_t * *entry*, EmberZclBindingId_t *bindingId*, const EmberZclBindingResponseHandler *handler*)

This function sends a command to a specified destination to update a binding.

Parameters

destination	A location to send a command to
entry	A new binding entry to use for an update
bindingId	A binding identifier to update in destination's binding table
handler	A callback that is triggered for a response

Returns

- EMBER_ZCL_STATUS_SUCCESS if function call was successful
- EmberStatus with failure reason otherwise

See also

EmberZclBindingResponseHandler()

6.110.5.10 bool emberZclSetBinding (EmberZclBindingId_t *bindingId,* const EmberZclBindingEntry_t * *entry*)

This function sets a specified binding.

Parameters

bindingId	A binding identifier of entry to set
entry	A new entry to set the binding to

Returns

- `true` if binding was set successfully
- `false` if binding was not set successfully

6.111 Commands

Data Structures

- struct EmberZclCommandContext_t

Macros

- #define EMBER_ZCL_COMMAND_NULL ((EmberZclCommandId_t)-1)

Typedefs

- typedef uint8_t EmberZclCommandId_t

Functions

- EmberStatus emberZclSendDefaultResponse (const EmberZclCommandContext_t *context, EmberZcl↩ Status_t status)

6.111.1 Detailed Description

See zcl-core-types.h for source code.

See zcl-core.h for source code.

6.111.2 Macro Definition Documentation

6.111.2.1 #define EMBER_ZCL_COMMAND_NULL ((EmberZclCommandId_t)-1)

A distinguished value that represents a null (invalid) command identifier.

6.111.3 Typedef Documentation

6.111.3.1 typedef uint8_t EmberZclCommandId_t

A command identifier.

6.111.4 Function Documentation

6.111.4.1 EmberStatus emberZclSendDefaultResponse (const EmberZclCommandContext_t * *context*, EmberZclStatus_t *status*)

This function sends a default response to a command.

Parameters

context	A command context for the response
status	A status to respond with

Returns

- EMBER_ZCL_STATUS_SUCCESS if response was sent successfully
- EmberStatus with failure reason otherwise

6.112 Reporting

Data Structures

- struct EmberZclNotificationContext_t
- struct EmberZclReportingConfiguration_t

Macros

- #define EMBER_ZCL_REPORTING_CONFIGURATION_DEFAULT 0
- #define EMBER_ZCL_REPORTING_CONFIGURATION_NULL ((EmberZclReportingConfigurationId_t)-1)

Typedefs

- typedef uint8_t EmberZclReportingConfigurationId_t

Functions

- void emberZclReportingConfigurationsFactoryReset (EmberZclEndpointId_t endpointId)

  This function performs a factory reset of the reporting configurations:-.

6.112.1 Detailed Description

See zcl-core-types.h for source code.

See zcl-core.h for source code.

6.112.2 Macro Definition Documentation

6.112.2.1 #define EMBER_ZCL_REPORTING_CONFIGURATION_DEFAULT 0

A distinguished value that represents a default reporting configuration identifier.

6.112.2.2 #define EMBER_ZCL_REPORTING_CONFIGURATION_NULL ((EmberZclReportingConfigurationId_t)-1)

A distinguished value that represents a null (invalid) reporting configuration identifier.

6.112.3 Typedef Documentation

6.112.3.1 typedef uint8_t EmberZclReportingConfigurationId_t

A reporting configuration identifier.

6.112.4 Function Documentation

6.112.4.1 void emberZclReportingConfigurationsFactoryReset (EmberZclEndpointId_t *endpointId*)

1. All entries in nv reporting configurations table are erased
2. Default configurations for each endpoint/clusterSpec are restored to their initial values and saved to nv.

6.113 Management

Functions

- EmberStatus emberZclStartEzMode (void)
- void emberZclStopEzMode (void)
- bool emberZclEzModeIsActive (void)

6.113.1 Detailed Description

See zcl-core.h for source code.

6.113.2 Function Documentation

6.113.2.1 bool emberZclEzModeIsActive (void)

This function checks whether EZ-Mode is currently active.

Returns

`true` if EZ-Mode is active, `false` otherwise.

6.113.2.2 EmberStatus emberZclStartEzMode (void)

This function puts a device in EZ-Mode for a fixed-duration.

Returns

- EMBER_ZCL_STATUS_SUCCESS if EZ-Mode started successfully
- EMBER_ERR_FATAL if the multicast address is invalid
- EmberStatus with failure reason otherwise

Each time EZ-Mode is invoked, the device extends the window by the same fixed duration. During the window, devices perform EZ-Mode finding and binding with other devices also in EZ-Mode. Multicast messages advertise capabilities of the device to other nodes in the network. Unicast messages communicate binding targets to specific devices. While the timer is active and not expired, including when the window is extended due to subsequent invocations, the device listens on the EZ-Mode multicast address and processes EZ-Mode requests.

6.113.2.3 void emberZclStopEzMode (void)

This function stops EZ-Mode.

The device ignores all EZ-Mode requests and stops listening on the EZ-Mode multicast address.

Chapter 7

Data Structure Documentation

7.1 AshRxState Struct Reference

`#include <ash-v3.h>`

Data Fields

- uint8_t payload [MAX_ASH_PAYLOAD_SIZE]
- uint8_t payloadIndex
- uint8_t escapedPayloadIndex
- uint8_t payloadLength
- uint8_t controlByte
- uint8_t headerEscapeByte
- uint16_t computedCrc
- uint8_t highCrcByte
- uint8_t inBetweenCrcByte
- AshRxFrameState frameState
- bool escapeNextByte

The documentation for this struct was generated from the following file:

- ash-v3.h

7.2 AshTxDmaBuffer Struct Reference

`#include <ash-v3.h>`

Data Fields

- uint8_t data [MAX_ASH_PACKET_SIZE]
- uint8_t * finger
- AshTxDmaBufferState state
- bool resend
- uint8_t resendCount
- bool isCorrupt

The documentation for this struct was generated from the following file:

- ash-v3.h

7.3 AshTxState Struct Reference

```
#include <ash-v3.h>
```

Data Fields

- AshTxDmaBuffer dmaBufferA
- AshTxDmaBuffer dmaBufferB
- AshTxDmaBuffer * dmaBuffer
- uint8_t outgoingFrameCounter
- uint8_t ackNackFrameCounter
- bool serialLayerReplied

The documentation for this struct was generated from the following file:

- ash-v3.h

7.4 Bytes16 Struct Reference

Defines a data type of size 16 bytes.

```
#include <ember-types.h>
```

Data Fields

- uint8_t contents [16]

The documentation for this struct was generated from the following file:

- ember-types.h

7.5 Bytes8 Struct Reference

Defines a data type of size 8 bytes.

`#include <ember-types.h>`

Data Fields

- uint8_t contents [8]

The documentation for this struct was generated from the following file:

- ember-types.h

7.6 CertificateAuthority Struct Reference

Defines a certificate authority structure.

`#include <ember-types.h>`

Data Fields

- const uint8_t * name
- uint16_t nameLength
- uint8_t * publicKey
- uint8_t maxPathLength

The documentation for this struct was generated from the following file:

- ember-types.h

7.7 DeviceCertificate Struct Reference

Defines a device certificate structure.

`#include <ember-types.h>`

Data Fields

- const uint8_t * privateKey
- const uint8_t * certificate
- const uint16_t certificateSize

The documentation for this struct was generated from the following file:

- ember-types.h

7.8 EmberCoapBlockOption Struct Reference

`#include <coap.h>`

Data Fields

- bool more
- uint8_t logSize
- uint32_t number

7.8.1 Field Documentation

7.8.1.1 uint8_t EmberCoapBlockOption::logSize

7.8.1.2 bool EmberCoapBlockOption::more

7.8.1.3 uint32_t EmberCoapBlockOption::number

The documentation for this struct was generated from the following file:

- coap.h

7.9 EmberCoapOption Struct Reference

Structure that includes options in outgoing requests and responses.

`#include <coap.h>`

Data Fields

- EmberCoapOptionType type
- const uint8_t * value
- uint16_t valueLength
- uint32_t intValue

7.9.1 Detailed Description

Calls that send messages can be passed a pointer to any array of `EmberCoapOption`s.

7.9.2 Field Documentation

7.9.2.1 uint32_t EmberCoapOption::intValue

The value of the option interpreted as a uint32_t.

7.9.2.2 EmberCoapOptionType EmberCoapOption::type

The type of the option.

7.9.2.3 const uint8_t* EmberCoapOption::value

A pointer to the option's value.

7.9.2.4 uint16_t EmberCoapOption::valueLength

Number of bytes in the option's value.

The documentation for this struct was generated from the following file:

- coap.h

7.10 EmberCoapRequestInfo Struct Reference

Additional information about an incoming request.

`#include <coap.h>`

Data Fields

- EmberIpv6Address localAddress
- EmberIpv6Address remoteAddress
- uint16_t localPort
- uint16_t remotePort
- EmberCoapTransmitHandler transmitHandler
- void * transmitHandlerData
- uint8_t token [EMBER_COAP_MAX_TOKEN_LENGTH]
- uint8_t tokenLength
- void * ackData

7.10.1 Detailed Description

`transmitHandler` is non-NULL if the request was passed to `emberProcessCoap()`, and will be called to deliver any response. If it is NULL, the request was an ordinary UDP message and any response is sent using UDP.

7.10.2 Field Documentation

7.10.2.1 void* EmberCoapRequestInfo::ackData

must be NULL when sending a delayed response

7.10.2.2 EmberIpv6Address EmberCoapRequestInfo::localAddress

7.10.2.3 uint16_t EmberCoapRequestInfo::localPort

7.10.2.4 EmberIpv6Address EmberCoapRequestInfo::remoteAddress

7.10.2.5 uint16_t EmberCoapRequestInfo::remotePort

7.10.2.6 uint8_t EmberCoapRequestInfo::token[EMBER_COAP_MAX_TOKEN_LENGTH]

7.10.2.7 uint8_t EmberCoapRequestInfo::tokenLength

7.10.2.8 EmberCoapTransmitHandler EmberCoapRequestInfo::transmitHandler

7.10.2.9 void* EmberCoapRequestInfo::transmitHandlerData

The documentation for this struct was generated from the following file:

- coap.h

7.11 EmberCoapResponseInfo Struct Reference

Additional information about an incoming response.

```
#include <coap.h>
```

Data Fields

- EmberIpv6Address localAddress
- EmberIpv6Address remoteAddress
- uint16_t localPort
- uint16_t remotePort
- void * applicationData
- uint16_t applicationDataLength

7.11.1 Field Documentation

7.11.1.1 void* EmberCoapResponseInfo::applicationData

The value passed to `emberCoapSend()`.

7.11.1.2 uint16_t EmberCoapResponseInfo::applicationDataLength

The value passed to `emberCoapSend()`.

7.11.1.3 EmberIpv6Address EmberCoapResponseInfo::localAddress

7.11.1.4 uint16_t EmberCoapResponseInfo::localPort

7.11.1.5 EmberIpv6Address EmberCoapResponseInfo::remoteAddress

7.11.1.6 uint16_t EmberCoapResponseInfo::remotePort

The documentation for this struct was generated from the following file:

- coap.h

7.12 EmberCoapSendInfo Struct Reference

Optional information when sending a message.

```
#include <coap.h>
```

Data Fields

- bool nonConfirmed: 1
- bool multicastLoopback: 1
- EmberIpv6Address localAddress
- uint16_t localPort
- uint16_t remotePort
- const EmberCoapOption * options
- uint8_t numberOfOptions
- uint32_t responseTimeoutMs
- const uint8_t * responseAppData
- uint16_t responseAppDataLength
- EmberCoapTransmitHandler transmitHandler
- void * transmitHandlerData

7.12.1 Detailed Description

For all fields a value of 0 or NULL means that the default will be used.

Multicast are always sent as unconfirmed.

7.12.2 Field Documentation

7.12.2.1 EmberIpv6Address EmberCoapSendInfo::localAddress

Default is to let the IP stack choose

7.12.2.2 uint16_t EmberCoapSendInfo::localPort

Defaults to the CoAP port (5683)

7.12.2.3 bool EmberCoapSendInfo::multicastLoopback

Defaults to not looping back

7.12.2.4 bool EmberCoapSendInfo::nonConfirmed

Defaults to confirmed

7.12.2.5 uint8_t EmberCoapSendInfo::numberOfOptions

Defaults to zero

7.12.2.6 const EmberCoapOption* EmberCoapSendInfo::options

Defaults to NULL

7.12.2.7 uint16_t EmberCoapSendInfo::remotePort

Defaults to the CoAP port (5683)

7.12.2.8 const uint8_t* EmberCoapSendInfo::responseAppData

Defaults to NULL

7.12.2.9 uint16_t EmberCoapSendInfo::responseAppDataLength

Defaults to zero

7.12.2.10 uint32_t EmberCoapSendInfo::responseTimeoutMs

Default is 'EMBER_COAP_DEFAULT_TIMEOUT_MS'

7.12.2.11 EmberCoapTransmitHandler EmberCoapSendInfo::transmitHandler

Defaults to using UDP

7.12.2.12 void* EmberCoapSendInfo::transmitHandlerData

Defaults to NULL

The documentation for this struct was generated from the following file:

- coap.h

7.13 EmberCommandEntry Struct Reference

Command entry for a command table.

`#include <command-interpreter2.h>`

Data Fields

- union {
  PGM_P name
  uint16_t id
  } identifier

- CommandAction action
- PGM_P argumentTypes
- PGM_P description

7.13.1 Field Documentation

7.13.1.1 CommandAction EmberCommandEntry::action

A reference to a function in the application that implements the command. If this entry refers to a nested command, then action field has to be set to NULL.

7.13.1.2 PGM_P EmberCommandEntry::argumentTypes

In case of normal (non-nested) commands, argumentTypes is a string that specifies the number and types of arguments the command accepts. The argument specifiers are:

- u: one-byte unsigned integer.
- v: two-byte unsigned integer
- w: four-byte unsigned integer
- s: one-byte signed integer
- r: two-byte signed integer
- q: four-byte signed integer
- b: string. The argument can be entered in ascii by using quotes, for example: "foo". Or it may be entered in hex by using curly braces, for example: { 08 A1 f2 }. There must be an even number of hex digits, and spaces are ignored.
- *: zero or more of the previous type. If used, this must be the last specifier.
- ?: Unknown number of arguments. If used this must be the only character. This means, that command interpreter will not perform any validation of arguments, and will call the action directly, trusting it that it will handle with whatever arguments are passed in. Integer arguments can be either decimal or hexadecimal. A 0x prefix indicates a hexadecimal integer. Example: 0x3ed.

In case of a nested command (action is NULL), then argumentTypes will contain a pointer to the nested commands.

7.13.1.3 PGM_P EmberCommandEntry::description

A description of the command.

7.13.1.4 uint16_t EmberCommandEntry::id

A two-byte identifier for serial communication

7.13.1.5 union { ... } EmberCommandEntry::identifier

7.13.1.6 PGM_P EmberCommandEntry::name

Use letters, digits, and underscores, '_', for the command name. Command names are case-sensitive.

The documentation for this struct was generated from the following file:

- command-interpreter2.h

7.14 EmberCommandState Struct Reference

For use when declaring a separate command streams. The fields are not accessed directly by the application.

```
#include <command-interpreter2.h>
```

Data Fields

- uint8_t state
- uint8_t buffer [EMBER_COMMAND_BUFFER_LENGTH]
- uint16_t tokenIndices [MAX_TOKEN_COUNT+1]
- uint8_t tokenCount
- uint16_t index
- uint8_t error
- uint8_t hexHighNibble
- uint8_t argOffset
- uint8_t previousCharacter
- uint8_t defaultBase
- EmberCommandEntry * currentCommand

7.14.1 Field Documentation

7.14.1.1 uint8_t EmberCommandState::argOffset

7.14.1.2 uint8_t EmberCommandState::buffer[EMBER_COMMAND_BUFFER_LENGTH]

7.14.1.3 EmberCommandEntry* EmberCommandState::currentCommand

7.14.1.4 uint8_t EmberCommandState::defaultBase

7.14.1.5 uint8_t EmberCommandState::error

7.14.1.6 uint8_t EmberCommandState::hexHighNibble

7.14.1.7 uint16_t EmberCommandState::index

7.14.1.8 uint8_t EmberCommandState::previousCharacter

7.14.1.9 uint8_t EmberCommandState::state

7.14.1.10 uint8_t EmberCommandState::tokenCount

7.14.1.11 uint16_t EmberCommandState::tokenIndices[MAX_TOKEN_COUNT+1]

The documentation for this struct was generated from the following file:

- command-interpreter2.h

7.15 EmberDiagnosticData Struct Reference

```
#include <coap-diagnostic.h>
```

Data Fields

- uint32_t tlvMask
- const uint8_t * macExtendedAddress
- uint16_t address16
- uint8_t mode
- uint16_t timeout
- const uint8_t * connectivity
- const uint8_t * routingTable
- const uint8_t * leaderData
- const uint8_t * networkData
- const uint8_t * ipv6AddressList
- const uint8_t * macCounters
- uint8_t batteryLevel
- uint16_t voltage
- const uint8_t * childTable
- const uint8_t * channelPages

7.15.1 Field Documentation

7.15.1.1 uint16_t EmberDiagnosticData::address16

7.15.1.2 uint8_t EmberDiagnosticData::batteryLevel

7.15.1.3 const uint8_t* EmberDiagnosticData::channelPages

7.15.1.4 const uint8_t* EmberDiagnosticData::childTable

7.15.1.5 const uint8_t* EmberDiagnosticData::connectivity

7.15.1.6 const uint8_t* EmberDiagnosticData::ipv6AddressList

7.15.1.7 const uint8_t* EmberDiagnosticData::leaderData

7.15.1.8 const uint8_t* EmberDiagnosticData::macCounters

7.15.1.9 const uint8_t* EmberDiagnosticData::macExtendedAddress

7.15.1.10 uint8_t EmberDiagnosticData::mode

7.15.1.11 const uint8_t* EmberDiagnosticData::networkData

7.15.1.12 const uint8_t* EmberDiagnosticData::routingTable

7.15.1.13 uint16_t EmberDiagnosticData::timeout

7.15.1.14 uint32_t EmberDiagnosticData::tlvMask

7.15.1.15 uint16_t EmberDiagnosticData::voltage

The documentation for this struct was generated from the following file:

- coap-diagnostic.h

7.16 EmberDnsResponse Struct Reference

Structure for returning information from a DNS lookup. A structure is used to make it easier to add additional values.

`#include <network-management.h>`

Data Fields

- EmberIpv6Address ipAddress

7.16.1 Field Documentation

7.16.1.1 EmberIpv6Address EmberDnsResponse::ipAddress

The returned address

The documentation for this struct was generated from the following file:

- network-management.h

7.17 EmberEui64 Struct Reference

EUI 64-bit ID (an IEEE address).

```
#include <ember-types.h>
```

Data Fields

- uint8_t bytes [EUI64_SIZE]

7.17.1 Detailed Description

Due to an unfortunate choice by the IEEE, EUI64s are stored in reverse order in 802.15.4 headers. As a consequence, they are stored in reverse order in the EmberEui64 type as well.

The documentation for this struct was generated from the following file:

- ember-types.h

7.18 EmberEventControl Struct Reference

Control structure for events.

```
#include <ember-types.h>
```

Data Fields

- EmberEventUnits status
- EmberTaskId taskid
- uint32_t timeToExecute

7.18.1 Detailed Description

This structure should not be accessed directly. It holds the event status (one of the *EMBER_EVENT_* values) and the time left before the event fires.

The documentation for this struct was generated from the following file:

- ember-types.h

7.19 EmberIpv6Address Struct Reference

An IPv6 Address structure.

`#include <ember-types.h>`

Data Fields

- uint8_t bytes [16]

The documentation for this struct was generated from the following file:

- ember-types.h

7.20 EmberIpv6Prefix Struct Reference

An IPv6 Prefix structure.

`#include <ember-types.h>`

Data Fields

- uint8_t bytes [8]

The documentation for this struct was generated from the following file:

- ember-types.h

7.21 EmberKeyData Struct Reference

This data structure contains the key data that is passed into various other functions.

`#include <ember-types.h>`

Data Fields

- uint8_t contents [EMBER_ENCRYPTION_KEY_SIZE]

The documentation for this struct was generated from the following file:

- ember-types.h

7.22 EmberMacBeaconData Struct Reference

Structure to hold information about an 802.15.4 beacon for use on the application.

```
#include <network-management.h>
```

Data Fields

- uint8_t networkId [16]
- uint8_t extendedPanId [8]
- uint8_t longId [8]
- uint16_t panId
- uint8_t protocolId
- uint8_t channel
- bool allowingJoin
- uint8_t lqi
- int8_t rssi
- uint8_t version
- uint16_t shortId
- uint8_t steeringData [16]
- uint8_t steeringDataLength

7.22.1 Field Documentation

7.22.1.1 bool EmberMacBeaconData::allowingJoin

7.22.1.2 uint8_t EmberMacBeaconData::channel

7.22.1.3 uint8_t EmberMacBeaconData::extendedPanId[8]

7.22.1.4 uint8_t EmberMacBeaconData::longId[8]

7.22.1.5 uint8_t EmberMacBeaconData::lqi

7.22.1.6 uint8_t EmberMacBeaconData::networkId[16]

7.22.1.7 uint16_t EmberMacBeaconData::panId

7.22.1.8 uint8_t EmberMacBeaconData::protocolId

7.22.1.9 int8_t EmberMacBeaconData::rssi

7.22.1.10 uint16_t EmberMacBeaconData::shortId

7.22.1.11 uint8_t EmberMacBeaconData::steeringData[16]

7.22.1.12 uint8_t EmberMacBeaconData::steeringDataLength

7.22.1.13 uint8_t EmberMacBeaconData::version

The documentation for this struct was generated from the following file:

- network-management.h

7.23 EmberNetworkParameters Struct Reference

An application structure to hold useful network parameters.

```
#include <network-management.h>
```

Data Fields

- uint8_t networkId [EMBER_NETWORK_ID_SIZE]

  This will only be NUL terminated if the length of the network id is less than EMBER_NETWORK_ID_SIZE.
- EmberIpv6Prefix ulaPrefix
- EmberIpv6Prefix legacyUla
- uint8_t extendedPanId [EXTENDED_PAN_ID_SIZE]
- uint16_t panId
- uint8_t channel
- EmberNodeType nodeType
- int8_t radioTxPower
- EmberKeyData masterKey
- uint8_t joinKey [EMBER_JOIN_KEY_MAX_SIZE]

  This will only be NUL terminated if the length of the key is less than EMBER_JOIN_KEY_MAX_SIZE.
- uint8_t joinKeyLength

7.23.1 Field Documentation

7.23.1.1 uint8_t EmberNetworkParameters::channel

7.23.1.2 uint8_t EmberNetworkParameters::extendedPanId[EXTENDED_PAN_ID_SIZE]

7.23.1.3 uint8_t EmberNetworkParameters::joinKey[EMBER_JOIN_KEY_MAX_SIZE]

7.23.1.4 uint8_t EmberNetworkParameters::joinKeyLength

7.23.1.5 EmberIpv6Prefix EmberNetworkParameters::legacyUla

7.23.1.6 EmberKeyData EmberNetworkParameters::masterKey

7.23.1.7 uint8_t EmberNetworkParameters::networkId[EMBER_NETWORK_ID_SIZE]

7.23.1.8 EmberNodeType EmberNetworkParameters::nodeType

7.23.1.9 uint16_t EmberNetworkParameters::panId

7.23.1.10 int8_t EmberNetworkParameters::radioTxPower

7.23.1.11 EmberIpv6Prefix EmberNetworkParameters::ulaPrefix

The documentation for this struct was generated from the following file:

- network-management.h

7.24 EmberRipEntry Struct Reference

Structure that holds information about a routing table entry for use on the application. See emberGetRipEntry.

`#include <network-management.h>`

Data Fields

- uint8_t longId [8]
- EmberNodeType type
- int8_t rollingRssi
- uint8_t nextHopIndex
- uint8_t ripMetric
- uint8_t incomingLinkQuality
- uint8_t outgoingLinkQuality
- bool mleSync
- uint8_t age
- uint8_t routeDelta

7.24.1 Field Documentation

7.24.1.1 uint8_t EmberRipEntry::age

7.24.1.2 uint8_t EmberRipEntry::incomingLinkQuality

7.24.1.3 uint8_t EmberRipEntry::longId[8]

7.24.1.4 bool EmberRipEntry::mleSync

7.24.1.5 uint8_t EmberRipEntry::nextHopIndex

7.24.1.6 uint8_t EmberRipEntry::outgoingLinkQuality

7.24.1.7 uint8_t EmberRipEntry::ripMetric

7.24.1.8 int8_t EmberRipEntry::rollingRssi

7.24.1.9 uint8_t EmberRipEntry::routeDelta

7.24.1.10 EmberNodeType EmberRipEntry::type

The documentation for this struct was generated from the following file:

- network-management.h

7.25 EmberSecurityParameters Struct Reference

Values of security parameters for use in forming or joining a network.

```
#include <network-management.h>
```

Data Fields

- EmberKeyData * networkKey
- uint8_t * presharedKey
- uint8_t presharedKeyLength

7.25.1 Detailed Description

Structure to hold information about pre-shared network security parameters.

7.25.2 Field Documentation

7.25.2.1 EmberKeyData* EmberSecurityParameters::networkKey

7.25.2.2 uint8_t* EmberSecurityParameters::presharedKey

7.25.2.3 uint8_t EmberSecurityParameters::presharedKeyLength

The documentation for this struct was generated from the following file:

- network-management.h

7.26 EmberTaskControl Struct Reference

Control structure for tasks.

```
#include <ember-types.h>
```

Data Fields

- uint32_t nextEventTime
- EmberEventData * events
- bool busy

7.26.1 Detailed Description

This structure should not be accessed directly.

The documentation for this struct was generated from the following file:

- ember-types.h

7.27 EmberUdpConnectionData Struct Reference

Data stored for each connection.

```
#include <udp-peer.h>
```

Data Fields

- EmberUdpConnectionHandle connection
- uint16_t flags
- uint16_t internal
- uint8_t localAddress [16]
- uint8_t remoteAddress [16]
- uint16_t localPort
- uint16_t remotePort

7.27.1 Field Documentation

7.27.1.1 EmberUdpConnectionHandle EmberUdpConnectionData::connection

7.27.1.2 uint16_t EmberUdpConnectionData::flags

7.27.1.3 uint16_t EmberUdpConnectionData::internal

7.27.1.4 uint8_t EmberUdpConnectionData::localAddress[16]

7.27.1.5 uint16_t EmberUdpConnectionData::localPort

7.27.1.6 uint8_t EmberUdpConnectionData::remoteAddress[16]

7.27.1.7 uint16_t EmberUdpConnectionData::remotePort

The documentation for this struct was generated from the following file:

- udp-peer.h

7.28 EmberVersion Struct Reference

For use when declaring data that holds the Ember software version type.

```
#include <ember-types.h>
```

Data Fields

- uint8_t major
- uint8_t minor
- uint8_t patch
- EmberVersionType type
- uint16_t build
- uint32_t change

The documentation for this struct was generated from the following file:

- ember-types.h

7.29 EmberZclApplicationDestination_t Struct Reference

```
#include <zcl-core-types.h>
```

Data Fields

- union {
  EmberZclEndpointId_t endpointId
  EmberZclGroupId_t groupId
  } data

- EmberZclApplicationDestinationType_t type

7.29.1 Detailed Description

This structure holds an application destination.

7.29.2 Field Documentation

7.29.2.1 union { ... } EmberZclApplicationDestination_t::data

Data holds an endpoint identifier or a group identifier for an application destination.

7.29.2.2 EmberZclEndpointId_t EmberZclApplicationDestination_t::endpointId

An endpoint identifier of an application destination.

7.29.2.3 EmberZclGroupId_t EmberZclApplicationDestination_t::groupId

A group identifier of an application destination.

7.29.2.4 EmberZclApplicationDestinationType_t EmberZclApplicationDestination_t::type

Type of an application destination.

The documentation for this struct was generated from the following file:

- zcl-core-types.h

7.30 EmberZclAttributeContext_t Struct Reference

```
#include <zcl-core-types.h>
```

Data Fields

- EmberCoapCode code
- EmberZclGroupId_t groupId
- EmberZclEndpointId_t endpointId
- const EmberZclClusterSpec_t * clusterSpec
- EmberZclAttributeId_t attributeId
- EmberZclStatus_t status

7.30.1 Detailed Description

This structure holds an attribute specification.

7.30.2 Field Documentation

7.30.2.1 EmberZclAttributeId_t EmberZclAttributeContext_t::attributeId

An attribute identifier.

7.30.2.2 const EmberZclClusterSpec_t* EmberZclAttributeContext_t::clusterSpec

A cluster specification of an attribute.

7.30.2.3 EmberCoapCode EmberZclAttributeContext_t::code

CoAP code of an attribute.

7.30.2.4 EmberZclEndpointId_t EmberZclAttributeContext_t::endpointId

An endpoint identifier of an attribute.

7.30.2.5 EmberZclGroupId_t EmberZclAttributeContext_t::groupId

A group identifier of an attribute.

7.30.2.6 EmberZclStatus_t EmberZclAttributeContext_t::status

A status of an attribute used when reading and writing.

The documentation for this struct was generated from the following file:

- zcl-core-types.h

7.31 EmberZclAttributeWriteData_t Struct Reference

`#include <zcl-core-types.h>`

Data Fields

- EmberZclAttributeId_t attributeId
- const void * buffer
- size_t bufferLength

7.31.1 Detailed Description

This structure holds write data for an attribute.

7.31.2 Field Documentation

7.31.2.1 EmberZclAttributeId_t EmberZclAttributeWriteData_t::attributeId

An attribute identifier to write to.

7.31.2.2 const void* EmberZclAttributeWriteData_t::buffer

A buffer containing data to be written.

7.31.2.3 size_t EmberZclAttributeWriteData_t::bufferLength

Length of data to be written.

The documentation for this struct was generated from the following file:

- zcl-core-types.h

7.32 EmberZclBindingContext_t Struct Reference

```
#include <zcl-core-types.h>
```

Data Fields

- EmberCoapCode code
- EmberZclGroupId_t groupId
- EmberZclEndpointId_t endpointId
- const EmberZclClusterSpec_t * clusterSpec
- EmberZclBindingId_t bindingId

7.32.1 Detailed Description

This structure holds a binding context.

7.32.2 Field Documentation

7.32.2.1 EmberZclBindingId_t EmberZclBindingContext_t::bindingId

A binding identifier.

7.32.2.2 const EmberZclClusterSpec_t* EmberZclBindingContext_t::clusterSpec

A cluster specification of binding.

7.32.2.3 EmberCoapCode EmberZclBindingContext_t::code

CoAP code of binding.

7.32.2.4 EmberZclEndpointId_t EmberZclBindingContext_t::endpointId

An endpoint identifier of binding.

7.32.2.5 EmberZclGroupId_t EmberZclBindingContext_t::groupId

A group identifier of binding.

The documentation for this struct was generated from the following file:

- zcl-core-types.h

7.33 EmberZclBindingEntry_t Struct Reference

```
#include <zcl-core-types.h>
```

Data Fields

- EmberZclEndpointId_t endpointId
- EmberZclClusterSpec_t clusterSpec
- struct {
  struct {
  EmberZclScheme_t scheme
  union {
  EmberIpv6Address address
  EmberZclUid_t uid
  } data
  EmberZclNetworkDestinationType_t type
  uint16_t port
  } network
  EmberZclApplicationDestination_t application
  } destination
- EmberZclReportingConfigurationId_t reportingConfigurationId

7.33.1 Detailed Description

This structure holds a binding entry.

7.33.2 Field Documentation

7.33.2.1 EmberIpv6Address EmberZclBindingEntry_t::address

7.33.2.2 EmberZclApplicationDestination_t EmberZclBindingEntry_t::application

7.33.2.3 EmberZclClusterSpec_t EmberZclBindingEntry_t::clusterSpec

A cluster specification of binding.

7.33.2.4 union { ... } EmberZclBindingEntry_t::data

7.33.2.5 struct { ... } EmberZclBindingEntry_t::destination

7.33.2.6 EmberZclEndpointId_t EmberZclBindingEntry_t::endpointId

An endpoint identifier of binding.

7.33.2.7 struct { ... } EmberZclBindingEntry_t::network

7.33.2.8 uint16_t EmberZclBindingEntry_t::port

7.33.2.9 EmberZclReportingConfigurationId_t EmberZclBindingEntry_t::reportingConfigurationId

A reporting configuration of binding.

7.33.2.10 EmberZclScheme_t EmberZclBindingEntry_t::scheme

7.33.2.11 EmberZclNetworkDestinationType_t EmberZclBindingEntry_t::type

7.33.2.12 EmberZclUid_t EmberZclBindingEntry_t::uid

The documentation for this struct was generated from the following file:

- zcl-core-types.h

7.34 EmberZclClusterSpec_t Struct Reference

`#include <zcl-core-types.h>`

Data Fields

- EmberZclRole_t role
- EmberZclManufacturerCode_t manufacturerCode
- EmberZclClusterId_t id

7.34.1 Detailed Description

This structure holds a cluster specification.

7.34.2 Field Documentation

7.34.2.1 EmberZclClusterId_t EmberZclClusterSpec_t::id

Identifier of a cluster.

7.34.2.2 EmberZclManufacturerCode_t EmberZclClusterSpec_t::manufacturerCode

Manufacturer code of a cluster.

7.34.2.3 EmberZclRole_t EmberZclClusterSpec_t::role

Role of a cluster.

The documentation for this struct was generated from the following file:

- zcl-core-types.h

7.35 EmberZclCoapEndpoint_t Struct Reference

```
#include <zcl-core-types.h>
```

Data Fields

- uint16_t flags
- EmberIpv6Address address
- EmberZclUid_t uid
- uint16_t port

7.35.1 Field Documentation

7.35.1.1 EmberIpv6Address EmberZclCoapEndpoint_t::address

7.35.1.2 uint16_t EmberZclCoapEndpoint_t::flags

7.35.1.3 uint16_t EmberZclCoapEndpoint_t::port

7.35.1.4 EmberZclUid_t EmberZclCoapEndpoint_t::uid

The documentation for this struct was generated from the following file:

- zcl-core-types.h

7.36 EmberZclCommandContext_t Struct Reference

```
#include <zcl-core-types.h>
```

Data Fields

- EmberIpv6Address remoteAddress
- EmberCoapCode code
- const uint8_t * payload
- uint16_t payloadLength
- EmberZclGroupId_t groupId
- EmberZclEndpointId_t endpointId
- const EmberZclClusterSpec_t * clusterSpec
- EmberZclCommandId_t commandId

7.36.1 Detailed Description

This structure holds a command context.

7.36.2 Field Documentation

7.36.2.1 const EmberZclClusterSpec_t* EmberZclCommandContext_t::clusterSpec

A cluster specification of a command.

7.36.2.2 EmberCoapCode EmberZclCommandContext_t::code

CoAP code of a command.

7.36.2.3 EmberZclCommandId_t EmberZclCommandContext_t::commandId

A command identifier.

7.36.2.4 EmberZclEndpointId_t EmberZclCommandContext_t::endpointId

An endpoint identifier of a command.

7.36.2.5 EmberZclGroupId_t EmberZclCommandContext_t::groupId

A group identifier of a command.

7.36.2.6 const uint8_t* EmberZclCommandContext_t::payload

Payload of a command.

7.36.2.7 uint16_t EmberZclCommandContext_t::payloadLength

Payload length of a command.

7.36.2.8 EmberIpv6Address EmberZclCommandContext_t::remoteAddress

A remote address of a command.

The documentation for this struct was generated from the following file:

- zcl-core-types.h

7.37 EmberZclDestination_t Struct Reference

`#include <zcl-core-types.h>`

Data Fields

- EmberZclCoapEndpoint_t network
- EmberZclApplicationDestination_t application

7.37.1 Detailed Description

This structure holds a destination.

7.37.2 Field Documentation

7.37.2.1 EmberZclApplicationDestination_t EmberZclDestination_t::application

A destination of an application.

7.37.2.2 EmberZclCoapEndpoint_t EmberZclDestination_t::network

A destination of a network.

The documentation for this struct was generated from the following file:

- zcl-core-types.h

7.38 EmberZclGroupEntry_t Struct Reference

```
#include <zcl-core-types.h>
```

Data Fields

- EmberZclGroupId_t groupId
- EmberZclEndpointId_t endpointId
- uint8_t groupNameLength
- uint8_t groupName [EMBER_ZCL_MAX_GROUP_NAME_LENGTH]

7.38.1 Detailed Description

This structure holds a group entry that represents membership of an endpoint in a group.

7.38.2 Field Documentation

7.38.2.1 EmberZclEndpointId_t EmberZclGroupEntry_t::endpointId

An endpoint identifier of a group entry.

7.38.2.2 EmberZclGroupId_t EmberZclGroupEntry_t::groupId

A group identifier of a group entry.

7.38.2.3 uint8_t EmberZclGroupEntry_t::groupName[EMBER_ZCL_MAX_GROUP_NAME_LENGTH]

An array containing group name.

7.38.2.4 uint8_t EmberZclGroupEntry_t::groupNameLength

Length of group name.

The documentation for this struct was generated from the following file:

- zcl-core-types.h

7.39 EmberZclNotificationContext_t Struct Reference

`#include <zcl-core-types.h>`

Data Fields

- EmberIpv6Address remoteAddress
- EmberZclEndpointId_t sourceEndpointId
- EmberZclReportingConfigurationId_t sourceReportingConfigurationId
- uint32_t sourceTimestamp
- EmberZclGroupId_t groupId
- EmberZclEndpointId_t endpointId

7.39.1 Detailed Description

This structure holds a notification context.

7.39.2 Field Documentation

7.39.2.1 EmberZclEndpointId_t EmberZclNotificationContext_t::endpointId

An endpoint identifier of a notification.

7.39.2.2 EmberZclGroupId_t EmberZclNotificationContext_t::groupId

A group identifier of a notification.

7.39.2.3 EmberIpv6Address EmberZclNotificationContext_t::remoteAddress

A remote address of a notification.

7.39.2.4 EmberZclEndpointId_t EmberZclNotificationContext_t::sourceEndpointId

A source endpoint identifier of a notification.

7.39.2.5 EmberZclReportingConfigurationId_t EmberZclNotificationContext_t::sourceReportingConfigurationId

A source reporting configuration identifier of a notification.

7.39.2.6 uint32_t EmberZclNotificationContext_t::sourceTimestamp

A source timestamp of a notification.

The documentation for this struct was generated from the following file:

- zcl-core-types.h

7.40 EmberZclOtaBootloadClientServerInfo_t Struct Reference

`#include <ota-bootload-client.h>`

Data Fields

- EmberZclScheme_t scheme
- uint8_t const * name
- uint8_t nameLength
- EmberIpv6Address address
- uint16_t port
- EmberZclUid_t uid
- EmberZclEndpointId_t endpointId

7.40.1 Detailed Description

This structure holds information about an OTA Server device.

7.40.2 Field Documentation

7.40.2.1 EmberIpv6Address EmberZclOtaBootloadClientServerInfo_t::address

IPv6 address of the server.

7.40.2.2 EmberZclEndpointId_t EmberZclOtaBootloadClientServerInfo_t::endpointId

Endpoint ID of the server.

7.40.2.3 uint8_t const* EmberZclOtaBootloadClientServerInfo_t::name

7.40.2.4 uint8_t EmberZclOtaBootloadClientServerInfo_t::nameLength

7.40.2.5 uint16_t EmberZclOtaBootloadClientServerInfo_t::port

UDP port of the server.

7.40.2.6 EmberZclScheme_t EmberZclOtaBootloadClientServerInfo_t::scheme

Protocol used to communicate with the server.

7.40.2.7 EmberZclUid_t EmberZclOtaBootloadClientServerInfo_t::uid

UID of the server.

The documentation for this struct was generated from the following file:

- ota-bootload-client.h

7.41 EmberZclOtaBootloadFileHeaderInfo_t Struct Reference

`#include <ota-bootload-core.h>`

Data Fields

- uint16_t version
- uint16_t headerSize
- EmberZclOtaBootloadFileSpec_t spec
- EmberZclOtaBootloadStackVersion_t stackVersion
- uint8_t string [EMBER_ZCL_OTA_BOOTLOAD_HEADER_STRING_SIZE]
- uint32_t fileSize
- EmberZclOtaBootloadSecurityCredentialVersion_t securityCredentialVersion
- EmberZclUid_t destination
- EmberZclOtaBootloadHardwareVersionRange_t hardwareVersionRange

7.41.1 Detailed Description

This structure holds information about an OTA file header.

This type contains all of the same information in the header of an actual OTA file, except for the magic number (EMBER_ZCL_OTA_BOOTLOAD_FILE_MAGIC_NUMBER), which is constant, and the field control. The field control is replaced by the use of invalid values for the destination and hardwareVersionRange members.

Member	Invalid Value
destination	All 0xFF bytes
hardwareVersionRange.minimum	EMBER_ZCL_OTA_BOOTLOAD_HARDWARE_VERSION_NULL
hardwareVersionRange.maximum	EMBER_ZCL_OTA_BOOTLOAD_HARDWARE_VERSION_NULL

See also

EmberZclOtaBootloadFileHeaderFieldControl_t

7.41.2 Field Documentation

7.41.2.1 EmberZclUid_t EmberZclOtaBootloadFileHeaderInfo_t::destination

OTA file destination identifier.

7.41.2.2 uint32_t EmberZclOtaBootloadFileHeaderInfo_t::fileSize

OTA file size, in bytes.

7.41.2.3 EmberZclOtaBootloadHardwareVersionRange_t EmberZclOtaBootloadFileHeaderInfo_t::hardwareVersionRange

OTA file earliest and latest valid hardware versions.

7.41.2.4 uint16_t EmberZclOtaBootloadFileHeaderInfo_t::headerSize

OTA file header size, in bytes.

7.41.2.5 EmberZclOtaBootloadSecurityCredentialVersion_t EmberZclOtaBootloadFileHeaderInfo_t::securityCredentialVersion

OTA file security credential version.

7.41.2.6 EmberZclOtaBootloadFileSpec_t EmberZclOtaBootloadFileHeaderInfo_t::spec

OTA file specification.

7.41.2.7 EmberZclOtaBootloadStackVersion_t EmberZclOtaBootloadFileHeaderInfo_t::stackVersion

OTA file intended stack version.

7.41.2.8 uint8_t EmberZclOtaBootloadFileHeaderInfo_t::string[EMBER_ZCL_OTA_BOOTLOAD_HEADER_STRING_SIZE]

OTA file header string (must be NUL-terminated).

7.41.2.9 uint16_t EmberZclOtaBootloadFileHeaderInfo_t::version

OTA file metadata version.

The documentation for this struct was generated from the following file:

- ota-bootload-core.h

7.42 EmberZclOtaBootloadFileSpec_t Struct Reference

```
#include <ota-bootload-core.h>
```

Data Fields

- EmberZclManufacturerCode_t manufacturerCode
- EmberZclOtaBootloadFileType_t type
- EmberZclOtaBootloadFileVersion_t version

7.42.1 Detailed Description

This structure holds an OTA file specification.

This file specification identifies a single OTA file, distinguishing it from another OTA file.

See also

emberZclOtaBootloadFileSpecNull

7.42.2 Field Documentation

7.42.2.1 EmberZclManufacturerCode_t EmberZclOtaBootloadFileSpec_t::manufacturerCode

Manufacturer code.

7.42.2.2 EmberZclOtaBootloadFileType_t EmberZclOtaBootloadFileSpec_t::type

OTA file type.

7.42.2.3 EmberZclOtaBootloadFileVersion_t EmberZclOtaBootloadFileSpec_t::version

OTA file version.

The documentation for this struct was generated from the following file:

- ota-bootload-core.h

7.43 EmberZclOtaBootloadHardwareVersionRange_t Struct Reference

```
#include <ota-bootload-core.h>
```

Data Fields

- EmberZclOtaBootloadHardwareVersion_t minimum
- EmberZclOtaBootloadHardwareVersion_t maximum

7.43.1 Detailed Description

This structure holds an OTA file hardware version range.

7.43.2 Field Documentation

7.43.2.1 EmberZclOtaBootloadHardwareVersion_t EmberZclOtaBootloadHardwareVersionRange_t::maximum

Maximum OTA file hardware version.

7.43.2.2 EmberZclOtaBootloadHardwareVersion_t EmberZclOtaBootloadHardwareVersionRange_t::minimum

Minimum OTA file hardware version.

The documentation for this struct was generated from the following file:

- ota-bootload-core.h

7.44 EmberZclOtaBootloadStorageFileInfo_t Struct Reference

```
#include <ota-bootload-storage-core.h>
```

Data Fields

- size_t size

7.44.1 Detailed Description

OTA file information.

This is the information about an OTA file in the storage module.

7.44.2 Field Documentation

7.44.2.1 size_t EmberZclOtaBootloadStorageFileInfo_t::size

The size of the OTA file, in bytes.

The documentation for this struct was generated from the following file:

- ota-bootload-storage-core.h

7.45 EmberZclOtaBootloadStorageInfo_t Struct Reference

```
#include <ota-bootload-storage-core.h>
```

Data Fields

- size_t maximumFileSize
- size_t fileCount

7.45.1 Detailed Description

OTA storage module information.

This is the current information about the OTA storage module.

7.45.2 Field Documentation

7.45.2.1 size_t EmberZclOtaBootloadStorageInfo_t::fileCount

The number of OTA files in the storage module.

7.45.2.2 size_t EmberZclOtaBootloadStorageInfo_t::maximumFileSize

The maximum size of an OTA file allowed by a storage module.

The documentation for this struct was generated from the following file:

- ota-bootload-storage-core.h

7.46 EmberZclReportingConfiguration_t Struct Reference

`#include <zcl-core-types.h>`

Data Fields

- uint16_t minimumIntervalS
- uint16_t maximumIntervalS

7.46.1 Detailed Description

This structure holds a reporting configuration.

7.46.2 Field Documentation

7.46.2.1 uint16_t EmberZclReportingConfiguration_t::maximumIntervalS

A maximum interval in seconds.

7.46.2.2 uint16_t EmberZclReportingConfiguration_t::minimumIntervalS

A minimum interval in seconds.

The documentation for this struct was generated from the following file:

- zcl-core-types.h

7.47 EmberZclStringType_t Struct Reference

`#include <zcl-core-types.h>`

Data Fields

- uint32_t length
- uint8_t * ptr

7.47.1 Detailed Description

Representation (length and pointer) of a text or binary string value.

7.47.2 Field Documentation

7.47.2.1 uint32_t EmberZclStringType_t::length

Length of the string.

7.47.2.2 uint8_t* EmberZclStringType_t::ptr

Pointer to the string.

The documentation for this struct was generated from the following file:

- zcl-core-types.h

7.48 EmberZclUid_t Struct Reference

```
#include <zcl-core-types.h>
```

Data Fields

- uint8_t bytes [EMBER_ZCL_UID_SIZE]

7.48.1 Detailed Description

UID (Unique Identifier).

7.48.2 Field Documentation

7.48.2.1 uint8_t EmberZclUid_t::bytes[EMBER_ZCL_UID_SIZE]

The documentation for this struct was generated from the following file:

- zcl-core-types.h

7.49 Event_s Struct Reference

```
#include <ember-types.h>
```

Data Fields

- EventActions * actions
- struct Event_s * next
- uint32_t timeToExecute

The documentation for this struct was generated from the following file:

- ember-types.h

7.50 EventActions_s Struct Reference

The static part of an event. Each event can be used with only one event queue.

```
#include <ember-types.h>
```

Data Fields

- struct EventQueue_s * queue
- void(* handler)(struct Event_s *)
- void(* marker)(struct Event_s *)
- const char * name

The documentation for this struct was generated from the following file:

- ember-types.h

7.51 EventQueue_s Struct Reference

An event queue is currently a list of events ordered by execution time.

```
#include <ember-types.h>
```

Data Fields

- Event * isrEvents
- Event * events
- uint32_t runTime
- bool running

The documentation for this struct was generated from the following file:

- ember-types.h

7.52 HalEepromInformationType Struct Reference

This structure defines a variety of information about the attached external EEPROM device.

`#include <bootloader-eeprom.h>`

Data Fields

- uint16_t version
- uint16_t capabilitiesMask
- uint16_t pageEraseMs
- uint16_t partEraseTime
- uint32_t pageSize
- uint32_t partSize
- const char *const partDescription
- uint8_t wordSizeBytes

7.52.1 Field Documentation

7.52.1.1 uint16_t HalEepromInformationType::capabilitiesMask

A bitmask describing the capabilites of this particular external EEPROM

7.52.1.2 uint16_t HalEepromInformationType::pageEraseMs

Maximum time it takes to erase a page. (in 1024Hz Milliseconds)

7.52.1.3 uint32_t HalEepromInformationType::pageSize

The size of a single erasable page in bytes

7.52.1.4 const char* const HalEepromInformationType::partDescription

Pointer to a string describing the attached external EEPROM

7.52.1.5 uint16_t HalEepromInformationType::partEraseTime

Maximum time it takes to erase the entire part. (in 1024Hz Milliseconds). Can be changed to be in seconds using EEPROM_CAPABILITIES_PART_ERASE_SECONDS

7.52.1.6 uint32_t HalEepromInformationType::partSize

The total size of the external EEPROM in bytes

7.52.1.7 uint16_t HalEepromInformationType::version

The version of this data structure

7.52.1.8 uint8_t HalEepromInformationType::wordSizeBytes

The number of bytes in a word for the external EEPROM

The documentation for this struct was generated from the following file:

- bootloader-eeprom.h

7.53 ipModemThreadParamStruct Struct Reference

```
#include <ip-modem-library.h>
```

Data Fields

- bool defaultRouteToUart
- bool incomingForMetoUart
- bool securityToUart

7.53.1 Field Documentation

7.53.1.1 bool ipModemThreadParamStruct::defaultRouteToUart

7.53.1.2 bool ipModemThreadParamStruct::incomingForMetoUart

7.53.1.3 bool ipModemThreadParamStruct::securityToUart

The documentation for this struct was generated from the following file:

- ip-modem-library.h

7.54 Ipv6Header Struct Reference

A structure that holds an IPv6 header. All values are in their local byte order (as opposed to network byte order, which might be different).

```
#include <ember-types.h>
```

Data Fields

- uint16_t ipPayloadLength
- uint32_t flowLabel
- uint8_t trafficClass
- uint8_t nextHeader
- uint8_t hopLimit
- uint8_t source [16]
- uint8_t destination [16]
- uint8_t * ipPayload
- uint8_t transportProtocol
- uint8_t * transportHeader
- uint16_t transportHeaderLength
- uint8_t * transportPayload
- uint16_t transportPayloadLength
- uint16_t sourcePort
- uint16_t destinationPort
- uint8_t icmpType
- uint8_t icmpCode

7.54.1 Detailed Description

The order has been rearranged to avoid the need for padding. The version is known to be 6 so it is not included.

The documentation for this struct was generated from the following file:

- ember-types.h

7.55 MacCountersData Struct Reference

`#include <coap-diagnostic.h>`

Data Fields

- uint16_t packetsSent
- uint16_t packetsReceived
- uint16_t packetsDroppedOnTransmit
- uint16_t packetsDroppedOnReceive
- uint16_t securityErrors
- uint16_t numberOfRetries

7.55.1 Field Documentation

7.55.1.1 uint16_t MacCountersData::numberOfRetries

7.55.1.2 uint16_t MacCountersData::packetsDroppedOnReceive

7.55.1.3 uint16_t MacCountersData::packetsDroppedOnTransmit

7.55.1.4 uint16_t MacCountersData::packetsReceived

7.55.1.5 uint16_t MacCountersData::packetsSent

7.55.1.6 uint16_t MacCountersData::securityErrors

The documentation for this struct was generated from the following file:

- coap-diagnostic.h

7.56 RTCCRamData Struct Reference

```
#include <micro-common.h>
```

Data Fields

- uint32_t outgoingNwkFrameCounter
- uint32_t incomingParentNwkFrameCounter
- uint32_t outgoingLinkKeyFrameCounter
- uint32_t incomingLinkKeyFrameCounter

7.56.1 Field Documentation

7.56.1.1 uint32_t RTCCRamData::incomingLinkKeyFrameCounter

7.56.1.2 uint32_t RTCCRamData::incomingParentNwkFrameCounter

7.56.1.3 uint32_t RTCCRamData::outgoingLinkKeyFrameCounter

7.56.1.4 uint32_t RTCCRamData::outgoingNwkFrameCounter

The documentation for this struct was generated from the following file:

- micro-common.h

7.57 TlsSessionState Struct Reference

Defines a TLS session state.

`#include <ember-types.h>`

Data Fields

- uint8_t idLength
- uint16_t id [(TLS_SESSION_ID_SIZE+1)/2]
- uint8_t master [TLS_MASTER_SECRET_SIZE]

The documentation for this struct was generated from the following file:

- ember-types.h

7.58 USB_ConfigurationDescriptor_TypeDef Struct Reference

USB Configuration Descriptor.

`#include <em_usb.h>`

Data Fields

- uint8_t bLength
- uint8_t bDescriptorType
- uint16_t wTotalLength
- uint8_t bNumInterfaces
- uint8_t bConfigurationValue
- uint8_t iConfiguration
- uint8_t bmAttributes
- uint8_t bMaxPower

7.58.1 Field Documentation

7.58.1.1 uint8_t USB_ConfigurationDescriptor_TypeDef::bConfigurationValue

Value to use as an argument to the SetConfiguration request to select this configuration.

7.58.1.2 uint8_t USB_ConfigurationDescriptor_TypeDef::bDescriptorType

Constant CONFIGURATION Descriptor Type

7.58.1.3 uint8_t USB_ConfigurationDescriptor_TypeDef::bLength

Size of this descriptor in bytes

7.58.1.4 uint8_t USB_ConfigurationDescriptor_TypeDef::bmAttributes

Configuration characteristics.
D7: Reserved (set to one)
D6: Self-powered
D5: Remote Wakeup
D4...0: Reserved (reset to zero)

7.58.1.5 uint8_t USB_ConfigurationDescriptor_TypeDef::bMaxPower

Maximum power consumption of the USB device, unit is 2mA per LSB

7.58.1.6 uint8_t USB_ConfigurationDescriptor_TypeDef::bNumInterfaces

Number of interfaces supported by this configuration

7.58.1.7 uint8_t USB_ConfigurationDescriptor_TypeDef::iConfiguration

Index of string descriptor describing this configuration.

7.58.1.8 uint16_t USB_ConfigurationDescriptor_TypeDef::wTotalLength

Total length of data returned for this configuration. Includes the combined length of all descriptors (configuration, interface, endpoint, and class- or vendor-specific) returned for this configuration.

The documentation for this struct was generated from the following file:

- em_usb.h

7.59 USB_DeviceDescriptor_TypeDef Struct Reference

USB Device Descriptor.

```
#include <em_usb.h>
```

Data Fields

- uint8_t bLength
- uint8_t bDescriptorType
- uint16_t bcdUSB
- uint8_t bDeviceClass
- uint8_t bDeviceSubClass
- uint8_t bDeviceProtocol
- uint8_t bMaxPacketSize0
- uint16_t idVendor
- uint16_t idProduct
- uint16_t bcdDevice
- uint8_t iManufacturer
- uint8_t iProduct
- uint8_t iSerialNumber
- uint8_t bNumConfigurations

7.59.1 Field Documentation

7.59.1.1 uint16_t USB_DeviceDescriptor_TypeDef::bcdDevice

Device release number in binary-coded decimal

7.59.1.2 uint16_t USB_DeviceDescriptor_TypeDef::bcdUSB

USB Specification Release Number in Binary-Coded Decimal

7.59.1.3 uint8_t USB_DeviceDescriptor_TypeDef::bDescriptorType

Constant DEVICE Descriptor Type

7.59.1.4 uint8_t USB_DeviceDescriptor_TypeDef::bDeviceClass

Class code (assigned by the USB-IF)

7.59.1.5 uint8_t USB_DeviceDescriptor_TypeDef::bDeviceProtocol

Protocol code (assigned by the USB-IF)

7.59.1.6 uint8_t USB_DeviceDescriptor_TypeDef::bDeviceSubClass

Subclass code (assigned by the USB-IF)

7.59.1.7 uint8_t USB_DeviceDescriptor_TypeDef::bLength

Size of this descriptor in bytes

7.59.1.8 uint8_t USB_DeviceDescriptor_TypeDef::bMaxPacketSize0

Maximum packet size for endpoint zero

7.59.1.9 uint8_t USB_DeviceDescriptor_TypeDef::bNumConfigurations

Number of possible configurations

7.59.1.10 uint16_t USB_DeviceDescriptor_TypeDef::idProduct

Product ID (assigned by the manufacturer)

7.59.1.11 uint16_t USB_DeviceDescriptor_TypeDef::idVendor

Vendor ID (assigned by the USB-IF)

7.59.1.12 uint8_t USB_DeviceDescriptor_TypeDef::iManufacturer

Index of string descriptor describing manufacturer

7.59.1.13 uint8_t USB_DeviceDescriptor_TypeDef::iProduct

Index of string descriptor describing product

7.59.1.14 uint8_t USB_DeviceDescriptor_TypeDef::iSerialNumber

Index of string descriptor describing the device serialnumber

The documentation for this struct was generated from the following file:

- em_usb.h

7.60 USB_EndpointDescriptor_TypeDef Struct Reference

USB Endpoint Descriptor.

```
#include <em_usb.h>
```

Data Fields

- uint8_t bLength
- uint8_t bDescriptorType
- uint8_t bEndpointAddress
- uint8_t bmAttributes
- uint16_t wMaxPacketSize
- uint8_t bInterval

7.60.1 Field Documentation

7.60.1.1 uint8_t USB_EndpointDescriptor_TypeDef::bDescriptorType

Constant ENDPOINT Descriptor Type

7.60.1.2 uint8_t USB_EndpointDescriptor_TypeDef::bEndpointAddress

The address of the endpoint

Referenced by USBD_Init().

7.60.1.3 uint8_t USB_EndpointDescriptor_TypeDef::bInterval

Interval for polling EP for data transfers

7.60.1.4 uint8_t USB_EndpointDescriptor_TypeDef::bLength

Size of this descriptor in bytes

7.60.1.5 uint8_t USB_EndpointDescriptor_TypeDef::bmAttributes

This field describes the endpoint attributes

Referenced by USBD_Init().

7.60.1.6 uint16_t USB_EndpointDescriptor_TypeDef::wMaxPacketSize

Maximum packet size for the endpoint

Referenced by USBD_Init().

The documentation for this struct was generated from the following file:

- em_usb.h

7.61 USB_InterfaceDescriptor_TypeDef Struct Reference

USB Interface Descriptor.

`#include <em_usb.h>`

Data Fields

- uint8_t bLength
- uint8_t bDescriptorType
- uint8_t bInterfaceNumber
- uint8_t bAlternateSetting
- uint8_t bNumEndpoints
- uint8_t bInterfaceClass
- uint8_t bInterfaceSubClass
- uint8_t bInterfaceProtocol
- uint8_t iInterface

7.61.1 Field Documentation

7.61.1.1 uint8_t USB_InterfaceDescriptor_TypeDef::bAlternateSetting

Value used to select this alternate setting for the interface identified in the prior field.

7.61.1.2 uint8_t USB_InterfaceDescriptor_TypeDef::bDescriptorType

Constant INTERFACE Descriptor Type.

7.61.1.3 uint8_t USB_InterfaceDescriptor_TypeDef::bInterfaceClass

Class code (assigned by the USB-IF). A value of zero is reserved for future standardization. If this field is set to FFH, the interface class is vendor-specific. All other values are reserved for assignment by the USB-IF.

7.61.1.4 uint8_t USB_InterfaceDescriptor_TypeDef::bInterfaceNumber

Number of this interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.

7.61.1.5 uint8_t USB_InterfaceDescriptor_TypeDef::bInterfaceProtocol

Protocol code (assigned by the USB). These codes are qualified by the value of the bInterfaceClass and the b↩ InterfaceSubClass fields. If an interface supports class-specific requests, this code identifies the protocols that the device uses as defined by the specification of the device class. If this field is reset to zero, the device does not use a class-specific protocol on this interface. If this field is set to FFH, the device uses a vendor-specific protocol for this interface

7.61.1.6 uint8_t USB_InterfaceDescriptor_TypeDef::bInterfaceSubClass

Subclass code (assigned by the USB-IF). These codes are qualified by the value of the bInterfaceClass field. If the bInterfaceClass field is reset to zero, this field must also be reset to zero. If the bInterfaceClass field is not set to FFH, all values are reserved forassignment by the USB-IF.

7.61.1.7 uint8_t USB_InterfaceDescriptor_TypeDef::bLength

Size of this descriptor in bytes.

7.61.1.8 uint8_t USB_InterfaceDescriptor_TypeDef::bNumEndpoints

Number of endpoints used by this interface (excluding endpoint zero). If this value is zero, this interface only uses the Default Control Pipe.

7.61.1.9 uint8_t USB_InterfaceDescriptor_TypeDef::iInterface

Index of string descriptor describing this interface.

The documentation for this struct was generated from the following file:

- em_usb.h

7.62 USB_Setup_TypeDef Struct Reference

USB Setup request package.

```
#include <em_usb.h>
```

Data Fields

- union {
  struct {
  union {
  struct {
  uint8_t Recipient: 5
  uint8_t Type: 2
  uint8_t Direction: 1
  }
  uint8_t bmRequestType
  }
  uint8_t bRequest
  uint16_t wValue
  uint16_t wIndex
  uint16_t wLength
  }
  uint32_t dw [2]
  };

7.62.1 Field Documentation

7.62.1.1 union { ... }

7.62.1.2 uint8_t USB_Setup_TypeDef::bmRequestType

Request characteristics.

7.62.1.3 uint8_t USB_Setup_TypeDef::bRequest

Request code.

7.62.1.4 uint8_t USB_Setup_TypeDef::Direction

Transfer direction of SETUP data phase.

7.62.1.5 uint32_t USB_Setup_TypeDef::dw[2]

7.62.1.6 uint8_t USB_Setup_TypeDef::Recipient

Request recipient (device, interface, endpoint or other).

7.62.1.7 uint8_t USB_Setup_TypeDef::Type

Request type (standard, class or vendor).

7.62.1.8 uint16_t USB_Setup_TypeDef::wIndex

Index or offset, varies according to request.

7.62.1.9 uint16_t USB_Setup_TypeDef::wLength

Number of bytes to transfer if there is a data stage.

7.62.1.10 uint16_t USB_Setup_TypeDef::wValue

Varies according to request.

The documentation for this struct was generated from the following file:

- em_usb.h

7.63 USB_StringDescriptor_TypeDef Struct Reference

USB String Descriptor.

`#include <em_usb.h>`

Data Fields

- uint8_t len
- uint8_t type
- char16_t name [ ]

7.63.1 Field Documentation

7.63.1.1 uint8_t USB_StringDescriptor_TypeDef::len

Size of this descriptor in bytes.

7.63.1.2 char16_t USB_StringDescriptor_TypeDef::name[ ]

The string encoded with UTF-16LE UNICODE charset.

7.63.1.3 uint8_t USB_StringDescriptor_TypeDef::type

Constant STRING Descriptor Type.

The documentation for this struct was generated from the following file:

- em_usb.h

7.64 USBD_Callbacks_TypeDef Struct Reference

USB Device stack callback structure.

`#include <em_usb.h>`

Data Fields

- const USBD_UsbResetCb_TypeDef usbReset
- const USBD_DeviceStateChangeCb_TypeDef usbStateChange
- const USBD_SetupCmdCb_TypeDef setupCmd
- const USBD_IsSelfPoweredCb_TypeDef isSelfPowered
- const USBD_SofIntCb_TypeDef sofInt

7.64.1 Detailed Description

Callback functions used by the device stack to signal events or query status to/from the application. See USBD_Init_TypeDef. Assign members to NULL if your application don't need a specific callback.

7.64.2 Field Documentation

7.64.2.1 const USBD_IsSelfPoweredCb_TypeDef USBD_Callbacks_TypeDef::isSelfPowered

Called whenever the device stack needs to query if the device is currently self- or bus-powered. Applies to devices which can operate in both modes.

7.64.2.2 const USBD_SetupCmdCb_TypeDef USBD_Callbacks_TypeDef::setupCmd

Called on each setup request received from host.

7.64.2.3 const USBD_SofIntCb_TypeDef USBD_Callbacks_TypeDef::sofInt

Called at each SOF interrupt. If NULL, the device stack will not enable the SOF interrupt.

7.64.2.4 const USBD_UsbResetCb_TypeDef USBD_Callbacks_TypeDef::usbReset

Called whenever USB reset signalling is detected on the USB port.

7.64.2.5 const USBD_DeviceStateChangeCb_TypeDef USBD_Callbacks_TypeDef::usbStateChange

Called whenever the device change state.

The documentation for this struct was generated from the following file:

- em_usb.h

7.65 USBD_Init_TypeDef Struct Reference

USB Device stack initialization structure.

```
#include <em_usb.h>
```

Data Fields

- const USB_DeviceDescriptor_TypeDef * deviceDescriptor
- const uint8_t * configDescriptor
- const void *const * stringDescriptors
- const uint8_t numberOfStrings
- const uint8_t * bufferingMultiplier
- USBD_Callbacks_TypeDef_Pointer callbacks
- const uint32_t reserved

7.65.1 Detailed Description

This structure is passed to USBD_Init() when starting up the device.

7.65.2 Field Documentation

7.65.2.1 const uint8_t* USBD_Init_TypeDef::bufferingMultiplier

Pointer to an array defining the size of the endpoint buffers. The size is given in multiples of endpoint size. Generally a value of 1 (single) or 2 (double) buffering should be used.

Referenced by USBD_Init().

7.65.2.2 USBD_Callbacks_TypeDef_Pointer USBD_Init_TypeDef::callbacks

Pointer to struct with callbacks (USBD_Callbacks_TypeDef). These callbacks are used by the device stack to signal events to or query the application.

Referenced by USBD_Init().

7.65.2.3 const uint8_t* USBD_Init_TypeDef::configDescriptor

Pointer to a configuration descriptor.

Referenced by USBD_Init().

7.65.2.4 const USB_DeviceDescriptor_TypeDef* USBD_Init_TypeDef::deviceDescriptor

Pointer to a device descriptor.

Referenced by USBD_Init().

7.65.2.5 const uint8_t USBD_Init_TypeDef::numberOfStrings

Number of strings in string descriptor array.

Referenced by USBD_Init().

7.65.2.6 const uint32_t USBD_Init_TypeDef::reserved

Reserved for future use.

7.65.2.7 const void* const* USBD_Init_TypeDef::stringDescriptors

Pointer to an array of string descriptor pointers.

Referenced by USBD_Init().

The documentation for this struct was generated from the following file:

- em_usb.h

Chapter 8

File Documentation

8.1 aes.h File Reference

AES crypto routines.

Macros

- #define EMBER_AES_BLOCK_SIZE_BYTES 16

  The number of bytes in a 128-bit AES block.

Functions

- void emberAesEcbEncryptBlock (uint8_t *block, const uint8_t *key, bool sameKey)

  This function performs a standalone-mode "electronic code book" (ECB) AES-128 encryption of the 16-byte plaintext `block` *using the 128-bit (16-byte)* `key`*. The resulting 16 byte ciphertext overwrites the plaintext* `block`*.*
- void emberAesCtrCryptData (uint8_t *nonce, const uint8_t *key, uint8_t *data, uint32_t dataLen, uint32_t dataDid)

  This function performs a counter-mode (CTR) AES-128 encrypt/decrypt of the `data` *for* `dataLen` *bytes, using the 128-bit (16-byte)* `key` *and 128-bit (16-byte)* `nonce`*. The resulting encrypted/decrypted data overwrites the* `data` *passed in.*

8.2 app-bootloader.h File Reference

Application bootloader.

Functions

Required Custom Functions

- void bootloaderInit ()

  Drives the app bootloader. If the ::runRecovery parameter is ::true, the recovery mode should be activated, otherwise it should attempt to install an image. This function should not return. It should always exit by resetting the the bootloader.
- void bootloaderInitCustom ()

  Drives the app bootloader. If the ::runRecovery parameter is ::true, the recovery mode should be activated, otherwise it should attempt to install an image. This function should not return. It should always exit by resetting the the bootloader.
- void bootloaderAction (bool runRecovery)

  Drives the app bootloader. If the ::runRecovery parameter is ::true, the recovery mode should be activated, otherwise it should attempt to install an image. This function should not return. It should always exit by resetting the the bootloader.

Available Bootloader Library Functions

Functions implemented by the bootloader library that may be used by custom functions.

- BL_Status recoveryMode (void)

  Activates recoveryMode to receive a new image over xmodem.
- BL_Status processImage (bool install)

  Processes an image in the external eeprom.

8.2.1 Detailed Description

Version

3.20.2 See Application for detailed documentation.

8.2.2 License

8.3 ash-v3.h File Reference

Data Structures

- struct AshTxDmaBuffer
- struct AshTxState
- struct AshRxState

Macros

- #define ASH_FLAG 0x7E
- #define ASH_ESC 0x7D
- #define ASH_XON 0x11
- #define ASH_XOFF 0x13
- #define ASH_COBRA_FORCE_BOOT 0xF8
- #define ASH_WAKEUP 0xFF
- #define ASH_ESCAPE_BYTE 0x20
- #define ASH_ACK_TIMEOUT 500
- #define ASH_BIG_HEADER_LENGTH 7
- #define ASH_CONTROL_BYTE_ESCAPED (1 << 7)
- #define ASH_PAYLOAD_LENGTH_BYTE_ESCAPED (1 << 6)
- #define AshHeaderEscapeType uint8_t
- #define ASH_CRC_SIZE 3
- #define MAX_ASH_PACKET_SIZE 64
- #define MAX_ASH_PAYLOAD_SIZE (MAX_ASH_PACKET_SIZE - ASH_BIG_HEADER_LENGTH)
- #define MAX_ASH_RESEND_COUNT 10
- #define ASH_STATE_STRINGS
- #define ASH_FRAME_COUNTER_ROLLOVER 8
- #define NEXT_ASH_OUTGOING_FRAME_COUNTER(x)

Typedefs

- typedef uint16_t(* SerialRxHandler) (const uint8_t *data, uint16_t length)

Enumerations

- enum AshHeaderBytesLocation {
  ASH_FLAG_INDEX = 0,
  ASH_HEADER_ESCAPE_BYTE_INDEX = 1,
  ASH_CONTROL_BYTE_INDEX = 2,
  ASH_PAYLOAD_LENGTH_INDEX = 3,
  ASH_HEADER_LENGTH = 4 }
- enum AshState {
  ASH_STATE_RESET_TX_PRE,
  ASH_STATE_RESET_TX_POST,
  ASH_STATE_RUNNING,
  ASH_STATE_LAST }
- enum AshMessageType {
  ASH_RESET = 0,
  ASH_RESET_ACK = 1,
  ASH_ACK = 2,
  ASH_NACK = 3,
  LAST_ASH_MESSAGE_TYPE }

- enum AshTxDmaBufferState {
  ASH_TX_INACTIVE = 0,
  ASH_TX_ACTIVE = 1,
  ASH_TX_FLUSH = 2,
  ASH_TX_EN_ROUTE_PRE = 3,
  ASH_TX_EN_ROUTE_POST = 4,
  ASH_TX_RESEND_ACKED = 5,
  LAST_TX_STATE }
- enum AshRxFrameState {
  ASH_INACTIVE = 0,
  ASH_NEED_HEADER_ESCAPE_BYTE,
  ASH_NEED_CONTROL_BYTE,
  ASH_NEED_PAYLOAD_LENGTH,
  ASH_NEED_PAYLOAD,
  ASH_NEED_HIGH_CRC,
  ASH_NEED_IN_BETWEEN_CRC,
  ASH_NEED_LOW_CRC,
  ASH_DONE,
  ASH_LAST_FRAME_STATE }

Functions

- bool emAshByteShouldBeEscaped (uint8_t byte)
- uint8_t emProcessAshRxInput (const uint8_t *input, uint8_t inputLength)
- uint8_t emProcessAshRxInputWithCallback (const uint8_t *data, uint8_t dataLength, SerialRxHandler serialRxHandler)
- bool isAshActive (void)
- void emPrintAshState (void)
- void emPrintAshRxState (void)
- void emPrintAshTxState (void)
- void emPrintAshPacketInformation (const uint8_t *packet)
- void emMakeAshPacket (AshTxDmaBuffer *target, AshMessageType type, uint8_t outgoingFrameCounter, uint8_t ackNackFrameCounter, const uint8_t *payload, uint8_t payloadLength, bool flush)
- void emCopyAshDmaBuffer (AshTxDmaBuffer *target, AshTxDmaBuffer *source, AshMessageType type, uint8_t outgoingFrameCounter, uint8_t ackNackFrameCounter)
- uint8_t emAshGetOutgoingFrameCounter (AshTxDmaBuffer *buffer)
- uint8_t emAshGetAckNackFrameCounter (AshTxDmaBuffer *buffer)
- AshMessageType emAshGetType (const AshTxDmaBuffer *buffer)
- void emPrintBytes (const uint8_t *bytes, uint16_t legnth, uint8_t indent)
- void emAshSetType (AshTxDmaBuffer *buffer, AshMessageType type)
- uint16_t emAddAshTxData (const uint8_t *payloadData, uint16_t payloadDataLength)
- bool uartTxSpaceAvailable (void)
- void emResetAshState (void)
- void emResetAshTxState (void)
- void emResetAshRxState (void)
- void emInitializeAshTxDmaBuffer (AshTxDmaBuffer *target)
- bool emGetAshTxPacket (uint8_t **target, uint16_t *length)
- uint16_t halHostReallyEnqueueTx (const uint8_t *data, uint16_t dataLength, AshTxDmaBuffer *target, bool forceFlush)
- void uartLinkReset (void)
- void uartFlushTx (void)
- void emAshConfigUart (uint8_t dropPercentage, uint8_t corruptPercentage)
- void emAshNotifyTxComplete (void)
- void emAshReallyNotifyTxComplete (bool loadTx)

- uint16_t emCreateAshHeader (AshTxDmaBuffer *target, AshMessageType type, uint8_t outgoingFrameCounter, uint8_t ackNackFrameCounter, uint8_t payloadLength)
- void emEraseAndPrepareDmaBuffer (AshTxDmaBuffer *buffer)
- uint8_t emAshTxDmaBufferPayloadLength (const AshTxDmaBuffer *buffer)
- void emAssertAshTxState (uint8_t outgoingFrameCounter, uint8_t ackNackFrameCounter, uint8_t dmaBuffersAvailable)
- void emSetAshTxState (uint8_t outgoingFrameCounter, uint8_t ackNackFrameCounter)
- bool emAshPreparingForPowerDown (void)
- void emExtractAshPacketInformation (const uint8_t *packet, uint8_t *headerEscapeByteLoc, uint8_t *outgoingFrameCounterLoc, uint8_t *ackNackFrameCounterLoc, AshMessageType *typeLoc, uint8_t *payloadLengthLoc)
- uint32_t emGetAshCrc (const uint8_t *data, uint16_t length)
- uint8_t * emStoreAshCrc (uint8_t *location, uint16_t crc, uint8_t *escapeByteLocation)
- void emberAshStatusHandler (AshState state)

  Notification that ash has changed state.
- void emAshHandleNack (uint8_t frameCounter)
- void emAshHandleAck (uint8_t frameCounter)
- void emResetSerialState (bool external)

Application Functions

Implement these functions in your application

The following functions are only for builds that support software flow control and that are compiled with EMBER_APPLICATION_SUPPORTS_SOFTWARE_FLOW_CONTROL

- void emberXOnHandler (void)

  Tell the application that we received an XON.
- void emberXOffHandler (void)

  Tell the application that we received an XOFF.

Variables

- AshTxState ashTxState
- AshRxState ashRxState

8.3.1 Macro Definition Documentation

8.3.1.1 #define ASH_ACK_TIMEOUT 500

8.3.1.2 #define ASH_BIG_HEADER_LENGTH 7

8.3.1.3 #define ASH_COBRA_FORCE_BOOT 0xF8

8.3.1.4 #define ASH_CONTROL_BYTE_ESCAPED (1 << 7)

8.3.1.5 #define ASH_CRC_SIZE 3

8.3.1.6 #define ASH_ESC 0x7D

8.3.1.7 #define ASH_ESCAPE_BYTE 0x20

8.3.1.8 #define ASH_FLAG 0x7E

8.3.1.9 #define ASH_FRAME_COUNTER_ROLLOVER 8

8.3.1.10 #define ASH_PAYLOAD_LENGTH_BYTE_ESCAPED (1 << 6)

8.3.1.11 #define ASH_STATE_STRINGS

Value:

```
"ASH_STATE_RESET_TX_PRE",  \
  "ASH_STATE_RESET_TX_POST", \
  "ASH_STATE_RUNNING",       \
  "ASH_STATE_LAST"
```

8.3.1.12 #define ASH_WAKEUP 0xFF

8.3.1.13 #define ASH_XOFF 0x13

8.3.1.14 #define ASH_XON 0x11

8.3.1.15 #define AshHeaderEscapeType uint8_t

8.3.1.16 #define MAX_ASH_PACKET_SIZE 64

8.3.1.17 #define MAX_ASH_PAYLOAD_SIZE (MAX_ASH_PACKET_SIZE - ASH_BIG_HEADER_LENGTH)

8.3.1.18 #define MAX_ASH_RESEND_COUNT 10

8.3.1.19 #define NEXT_ASH_OUTGOING_FRAME_COUNTER(*x*)

Value:

```
(x < ASH_FRAME_COUNTER_ROLLOVER - 1       \
   ? x + 1                                  \
   : 1)
```

8.3.2 Typedef Documentation

8.3.2.1 typedef uint16_t(* SerialRxHandler) (const uint8_t *data, uint16_t length)

8.3.3 Enumeration Type Documentation

8.3.3.1 enum AshHeaderBytesLocation

Enumerator

ASH_FLAG_INDEX

ASH_HEADER_ESCAPE_BYTE_INDEX

ASH_CONTROL_BYTE_INDEX

ASH_PAYLOAD_LENGTH_INDEX

ASH_HEADER_LENGTH

8.3.3.2 enum AshMessageType

Enumerator

ASH_RESET

ASH_RESET_ACK

ASH_ACK

ASH_NACK

LAST_ASH_MESSAGE_TYPE

8.3.3.3 enum AshRxFrameState

Enumerator

ASH_INACTIVE

ASH_NEED_HEADER_ESCAPE_BYTE

ASH_NEED_CONTROL_BYTE

ASH_NEED_PAYLOAD_LENGTH

ASH_NEED_PAYLOAD

ASH_NEED_HIGH_CRC

ASH_NEED_IN_BETWEEN_CRC

ASH_NEED_LOW_CRC

ASH_DONE

ASH_LAST_FRAME_STATE

8.3.3.4 enum AshState

Enumerator

ASH_STATE_RESET_TX_PRE

ASH_STATE_RESET_TX_POST

ASH_STATE_RUNNING

ASH_STATE_LAST

8.3.3.5 enum AshTxDmaBufferState

Enumerator

ASH_TX_INACTIVE

ASH_TX_ACTIVE

ASH_TX_FLUSH

ASH_TX_EN_ROUTE_PRE

ASH_TX_EN_ROUTE_POST

ASH_TX_RESEND_ACKED

LAST_TX_STATE

8.3.4 Function Documentation

8.3.4.1 uint16_t emAddAshTxData (const uint8_t * *payloadData,* uint16_t *payloadDataLength*)

8.3.4.2 bool emAshByteShouldBeEscaped (uint8_t *byte*)

8.3.4.3 void emAshConfigUart (uint8_t *dropPercentage,* uint8_t *corruptPercentage*)

8.3.4.4 uint8_t emAshGetAckNackFrameCounter (**AshTxDmaBuffer** * *buffer*)

8.3.4.5 uint8_t emAshGetOutgoingFrameCounter (**AshTxDmaBuffer** * *buffer*)

8.3.4.6 **AshMessageType** emAshGetType (const **AshTxDmaBuffer** * *buffer*)

8.3.4.7 void emAshHandleAck (uint8_t *frameCounter*)

8.3.4.8 void emAshHandleNack (uint8_t *frameCounter*)

8.3.4.9 void emAshNotifyTxComplete (void)

8.3.4.10 bool emAshPreparingForPowerDown (void)

8.3.4.11 void emAshReallyNotifyTxComplete (bool *loadTx*)

8.3.4.12 void emAshSetType (**AshTxDmaBuffer** * *buffer,* **AshMessageType** *type*)

8.3.4.13 uint8_t emAshTxDmaBufferPayloadLength (const **AshTxDmaBuffer** * *buffer*)

8.3.4.14 void emAssertAshTxState (uint8_t *outgoingFrameCounter,* uint8_t *ackNackFrameCounter,* uint8_t *dmaBuffersAvailable*)

8.3.4.15 void emCopyAshDmaBuffer (**AshTxDmaBuffer** * *target,* **AshTxDmaBuffer** * *source,* **AshMessageType** *type,* uint8_t *outgoingFrameCounter,* uint8_t *ackNackFrameCounter*)

8.3.4.16 uint16_t emCreateAshHeader (**AshTxDmaBuffer** * *target,* **AshMessageType** *type,* uint8_t *outgoingFrameCounter,* uint8_t *ackNackFrameCounter,* uint8_t *payloadLength*)

8.3.4.17 void emEraseAndPrepareDmaBuffer (**AshTxDmaBuffer** * *buffer*)

8.3.4.18 void emExtractAshPacketInformation (const uint8_t * *packet,* uint8_t * *headerEscapeByteLoc,* uint8_t * *outgoingFrameCounterLoc,* uint8_t * *ackNackFrameCounterLoc,* **AshMessageType** * *typeLoc,* uint8_t * *payloadLengthLoc*)

8.3.4.19 uint32_t emGetAshCrc (const uint8_t * *data,* uint16_t *length*)

8.3.4.20 bool emGetAshTxPacket (uint8_t ** *target,* uint16_t * *length*)

8.3.4.21 void emInitializeAshTxDmaBuffer (AshTxDmaBuffer * *target*)

8.3.4.22 void emMakeAshPacket (AshTxDmaBuffer * *target,* AshMessageType *type,* uint8_t *outgoingFrameCounter,* uint8_t *ackNackFrameCounter,* const uint8_t * *payload,* uint8_t *payloadLength,* bool *flush*)

8.3.4.23 void emPrintAshPacketInformation (const uint8_t * *packet*)

8.3.4.24 void emPrintAshRxState (void)

8.3.4.25 void emPrintAshState (void)

8.3.4.26 void emPrintAshTxState (void)

8.3.4.27 void emPrintBytes (const uint8_t * *bytes,* uint16_t *legnth,* uint8_t *indent*)

8.3.4.28 uint8_t emProcessAshRxInput (const uint8_t * *input,* uint8_t *inputLength*)

8.3.4.29 uint8_t emProcessAshRxInputWithCallback (const uint8_t * *data,* uint8_t *dataLength,* SerialRxHandler *serialRxHandler*)

8.3.4.30 void emResetAshRxState (void)

8.3.4.31 void emResetAshState (void)

8.3.4.32 void emResetAshTxState (void)

8.3.4.33 void emResetSerialState (bool *external*)

8.3.4.34 void emSetAshTxState (uint8_t *outgoingFrameCounter,* uint8_t *ackNackFrameCounter*)

8.3.4.35 uint8_t* emStoreAshCrc (uint8_t * *location,* uint16_t *crc,* uint8_t * *escapeByteLocation*)

8.3.4.36 uint16_t halHostReallyEnqueueTx (const uint8_t * *data,* uint16_t *dataLength,* AshTxDmaBuffer * *target,* bool *forceFlush*)

8.3.4.37 bool isAshActive (void)

8.3.4.38 void uartFlushTx (void)

8.3.4.39 void uartLinkReset (void)

8.3.4.40 bool uartTxSpaceAvailable (void)

8.3.5 Variable Documentation

8.3.5.1 AshRxState ashRxState

8.3.5.2 AshTxState ashTxState

8.4 bootloader-common.h File Reference

Macros

Bootloader Status Definitions

These are numerical definitions for the possible bootloader status codes.

- #define BL_SUCCESS 0U
  Numerical definition for a bootloader status code: Success.
- #define BL_CRC_MATCH 2U
  Numerical definition for a bootloader status code: CRC match.
- #define BL_IMG_FLASHED 3U
  Numerical definition for a bootloader status code: Image flashed.
- #define BL_ERR 1U
  Numerical definition for a bootloader status code: serial error.
- #define BL_ERR_MASK 0x40U
  Numerical definition for a bootloader status code: Error mask.
- #define BL_ERR_HEADER_EXP 0x41U
  Numerical definition for a bootloader status code: Failed in header state. Header expected.
- #define BL_ERR_HEADER_WRITE_CRC 0x42U
  Numerical definition for a bootloader status code: Failed write/CRC of header.
- #define BL_ERR_CRC 0x43U
  Numerical definition for a bootloader status code: Failed file CRC.
- #define BL_ERR_UNKNOWN_TAG 0x44U
  Numerical definition for a bootloader status code: Unknown tag.
- #define BL_ERR_SIG 0x45U
  Numerical definition for a bootloader status code: EBL header error.
- #define BL_ERR_ODD_LEN 0x46U
  Numerical definition for a bootloader status code: Trying to flash odd length bytes.
- #define BL_ERR_BLOCK_INDEX 0x47U
  Numerical definition for a bootloader status code: Indexed past end of block buffer.
- #define BL_ERR_OVWR_BL 0x48U
  Numerical definition for a bootloader status code: Attempt to overwrite bootloader flash.
- #define BL_ERR_OVWR_SIMEE 0x49U
  Numerical definition for a bootloader status code: Attempt to overwrite Simulated EEPROM flash.
- #define BL_ERR_ERASE_FAIL 0x4AU
  Numerical definition for a bootloader status code: Flash erase failed.
- #define BL_ERR_WRITE_FAIL 0x4BU
  Numerical definition for a bootloader status code: Flash write failed.
- #define BL_ERR_CRC_LEN 0x4CU
  Numerical definition for a bootloader status code: END tag CRC wrong length.
- #define BL_ERR_NO_QUERY 0X4DU
  Numerical definition for a bootloader status code: Received data before query request/response.
- #define BL_ERR_BAD_LEN 0x4EU
  Numerical definition for a bootloader status code: Invalid length detected.
- #define BL_ERR_TAGBUF 0x4FU
  Numerical definition for a bootloader status code: Problem with tagBuf detected.
- #define BL_EBL_CONTINUE 0x50U
  Numerical definition for a bootloader status code: processEbl deferred, call again to continue.
- #define BL_ERR_UNEXPECTED_TAG 0x51U
  Numerical definition for a bootloader status code: A known tag was found in an unexpected location (eg. header tag found after data)
- #define BL_ERR_UNK_ENC 0x52U
  Numerical definition for a bootloader status code: The specified encryption type is unknown to this bootloader.
- #define BL_ERR_INV_KEY 0x53U
  Numerical definition for a bootloader status code: No valid encryption key found on the device (ie. It's all 0xFF's). Bootloader will not function until this key is set.
- #define BL_ERR_ENC 0x54U
  Numerical definition for a bootloader status code: Generic error indicating that there was a problem with the encrypted file when decrypting.
- #define BL_IBR_ERR_CRC 0x55U
  Numerical definition for a bootloader status code: Failed IBR crc.
- #define BL_IBR_ERR_VERS 0x56U
  Numerical definition for a bootloader status code: Incorrect IBR version.
- #define BL_IBR_ERR_ADDR 0x57U
  Numerical definition for a bootloader status code: Invalid ebl address in IBR.

- #define BL_IBR_ERR_HDR 0x58U
  Numerical definition for a bootloader status code: Incorrect IBR header.

Bootloader State Flags

These are numerical flags for the possible bootloader states. These values are used in the bootloader code for making the current state more verbose.

Note

The flags do not start at 0 so that they can be output via the serial port during debug and easily screened out of normal xmodem traffic which depends only on ACK (0x06) and NAK (0x15).

- #define TIMEOUT 0x16
  Bootloader state flag.
- #define FILEDONE 0x17
  Bootloader state flag.
- #define FILEABORT 0x18
  Bootloader state flag.
- #define BLOCKOK 0x19
  Bootloader state flag.
- #define QUERYFOUND 0x1A
  Bootloader state flag.
- #define START_TIMEOUT 0x1B
  Bootloader state flag.
- #define BLOCK_TIMEOUT 0x1C
  Bootloader state flag.
- #define BLOCKERR_MASK 0x20
  Bootloader state flag.
- #define BLOCKERR_SOH 0x21
  Bootloader state flag: Start Of Header not received.
- #define BLOCKERR_CHK 0x22
  Bootloader state flag: Sequence of bytes don't match.
- #define BLOCKERR_CRCH 0x23
  Bootloader state flag: CRC High byte failure.
- #define BLOCKERR_CRCL 0x24
  Bootloader state flag: CRC Low byte failure.
- #define BLOCKERR_SEQUENCE 0x25
  Bootloader state flag: Block received out of sequence.
- #define BLOCKERR_PARTIAL 0x26
  Bootloader state flag: Partial block received.
- #define BLOCKERR_DUPLICATE 0x27
  Bootloader state flag: Duplicate of previous block.

Typedefs

- typedef uint8_t BL_Status
  Define the bootloader status type.

Enumerations

- enum {
  COMM_SERIAL = 0x01,
  COMM_RADIO = 0x02 }

8.5 bootloader-eeprom.h File Reference

Data Structures

- struct HalEepromInformationType

  This structure defines a variety of information about the attached external EEPROM device.

Macros

- #define EEPROM_PAGE_SIZE (128ul)

  Definition of an EEPROM page size, in bytes. This definition is deprecated, and should no longer be used.
- #define EEPROM_FIRST_PAGE (0)

  Define the location of the first page in EEPROM. This definition is deprecated, and should no longer be used.
- #define EEPROM_IMAGE_START (EEPROM_FIRST_PAGE * EEPROM_PAGE_SIZE)

  Define the location of the image start in EEPROM as a function of the EEPROM_FIRST_PAGE and EEPROM_PAGE_SIZE. This definition is deprecated, and should no longer be used.
- #define EEPROM_SUCCESS 0U

  Define EEPROM success status.
- #define EEPROM_ERR 1U

  Define EEPROM error status.
- #define EEPROM_ERR_MASK 0x80U

  Define EEPROM error mask.
- #define EEPROM_ERR_PG_BOUNDARY 0x81U

  Define EEPROM page boundary error.
- #define EEPROM_ERR_PG_SZ 0x82U

  Define EEPROM page size error.
- #define EEPROM_ERR_WRT_DATA 0x83U

  Define EEPROM write data error.
- #define EEPROM_ERR_IMG_SZ 0x84U

  Define EEPROM image too large error.
- #define EEPROM_ERR_ADDR 0x85U

  Define EEPROM invalid address error.
- #define EEPROM_ERR_INVALID_CHIP 0x86U

  Define EEPROM chip initialization error.
- #define EEPROM_ERR_ERASE_REQUIRED 0x87U

  Define EEPROM erase required error.
- #define EEPROM_ERR_NO_ERASE_SUPPORT 0x88U

  Define EEPROM error for no erase support.

EEPROM interaction functions.

- #define EEPROM_INFO_VERSION (0x0202)

  The current version of the HalEepromInformationType data structure.
- #define EEPROM_INFO_MAJOR_VERSION (0x0200)

  The current version of the HalEepromInformationType data structure.
- #define EEPROM_INFO_MAJOR_VERSION_MASK (0xFF00)

  The current version of the HalEepromInformationType data structure.
- #define EEPROM_INFO_MIN_VERSION_WITH_WORD_SIZE_SUPPORT 0x0102U

  The current version of the HalEepromInformationType data structure.

- #define EEPROM_CAPABILITIES_ERASE_SUPPORTED (0x0001U)

  Eeprom capabilites mask that indicates the erase API is supported.
- #define EEPROM_CAPABILITIES_PAGE_ERASE_REQD (0x0002U)

  Eeprom capabilites mask that indicates page erasing is required before new data can be written to a device.
- #define EEPROM_CAPABILITIES_BLOCKING_WRITE (0x0004U)

  Eeprom capabilites mask that indicates that the write routine is blocking on this device.
- #define EEPROM_CAPABILITIES_BLOCKING_ERASE (0x0008U)

  Eeprom capabilites mask that indicates that the erase routine is blocking on this device.
- #define EEPROM_CAPABILITIES_PART_ERASE_SECONDS (0x0010U)

  Eeprom capabilities mask that indicateds that the partEraseTime field of HalEepromInformationType is in seconds instead of the usual milliseconds.
- uint8_t halEepromInit (void)

  Initialize EEPROM. Note: some earlier drivers may assert instead of returning an error if initialization fails.
- void halEepromShutdown (void)

  Shutdown the EEPROM to conserve power.
- const HalEepromInformationType * halEepromInfo (void)

  Call this function to get information about the external EEPROM and its capabilities.
- uint32_t halEepromSize (void)

  Return the size of the EEPROM.
- bool halEepromBusy (void)

  Determine if the exernal EEPROM is still busy performing the last operation, such as a write or an erase.
- uint8_t halEepromRead (uint32_t address, uint8_t *data, uint16_t len)

  Read from the external EEPROM.
- uint8_t halEepromWrite (uint32_t address, const uint8_t *data, uint16_t len)

  Write to the external EEPROM.
- uint8_t halEepromErase (uint32_t address, uint32_t totalLength)

  Erases the specified region of the external EEPROM.

8.5.1 Detailed Description

See Application for detailed documentation.

8.6 bootloader-gpio.h File Reference

Bootloader GPIO definitions. See GPIO for detailed documentation.

Macros

State Indicator Macros

The bootloader indicates which state it is in by calling these // macros. Map them to the ::halBootloadState↩ Indicator function // (in bootloder-gpio.c) if you want to display that bootloader state. // Used to blink the LED's or otherwise signal bootloader activity.

- #define BL_STATE_UP() do { bootloadStateIndicator(BL_ST_UP); } while (0)

  Finished init sequence, ready for bootload.
- #define BL_STATE_DOWN() do { bootloadStateIndicator(BL_ST_DOWN); } while (0)

  Called right before bootloader resets to application. Use to cleanup and reset GPIO's to leave node in known state for app start, if necessary.
- #define BL_STATE_POLLING_LOOP() do { bootloadStateIndicator(BL_ST_POLLING_LOOP); } while (0)

Standalone bootloader polling serial/radio interface.

- #define BL_STATE_DOWNLOAD_LOOP() do { bootloadStateIndicator(BL_ST_DOWNLOAD_LOOP); } while (0)

  Processing download image.
- #define BL_STATE_DOWNLOAD_SUCCESS() do { bootloadStateIndicator(BL_ST_DOWNLOAD_SUCCESS); } while (0)

  Download process was a success.
- #define BL_STATE_DOWNLOAD_FAILURE() do { bootloadStateIndicator(BL_ST_DOWNLOAD_FAILURE); } while (0)

  Download process failed.

Enumerations

- enum blState_e {
  BL_ST_UP,
  BL_ST_DOWN,
  BL_ST_POLLING_LOOP,
  BL_ST_DOWNLOAD_LOOP,
  BL_ST_DOWNLOAD_FAILURE,
  BL_ST_DOWNLOAD_SUCCESS }

  Defines various bootloader states. Use in LED code to signal bootload activity.

Functions

- void bootloadGpioInit (void)

  Initialize GPIO.
- void bootloadStateIndicator (enum blState_e state)

  Helper function used for displaying bootloader state (for example: with LEDs).
- bool bootloadForceActivation (void)

  Force activation of bootloader.

8.6.1 Detailed Description

Version

3.20.2

8.6.2 License

(C) Copyright 2016 Silicon Labs, www.silabs.com

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

DISCLAIMER OF WARRANTY/LIMITATION OF REMEDIES: Silicon Labs has no obligation to support this Software. Silicon Labs is providing the Software "AS IS", with no express or implied warranties of any kind, including, but not limited to, any implied warranties of merchantability or fitness for any particular purpose or warranties against infringement of any proprietary rights of a third party.

Silicon Labs will not be liable for any consequential, incidental, or special damages, or any other relief, or for any claim by any third party, arising from your use of this Software.

8.7 bootloader-interface-app.h File Reference

```
#include "hal/micro/bootloader-eeprom.h"
```

Macros

- #define BOOTLOADER_SEGMENT_SIZE_LOG2 6

  This is the working unit of data for the app bootloader. We want it as big as possible, but it must be a factor of the NVM page size and fit into a single Zigbee packet. We choose 2^6 = 64 bytes.
- #define BOOTLOADER_SEGMENT_SIZE (1 << BOOTLOADER_SEGMENT_SIZE_LOG2)

  This is the working unit of data for the app bootloader. We want it as big as possible, but it must be a factor of the NVM page size and fit into a single Zigbee packet. We choose 2^6 = 64 bytes.
- #define BL_IMAGE_IS_VALID_CONTINUE ((uint16_t)0xFFFF)

  Define a numerical value for checking image validity when calling the image interface functions.

Functions

- uint8_t halAppBootloaderInit (void)

  Call this function as part of your application initialization to ensure the storage mechanism is ready to use. Note: some earlier drivers may assert instead of returning an error if initialization fails.
- const HalEepromInformationType * halAppBootloaderInfo (void)

  Call this function to get information about the attached storage device and its capabilities.
- void halAppBootloaderShutdown (void)

  Call this function when you are done accessing the storage mechanism to ensure that it is returned to its lowest power state.
- void halAppBootloaderImageIsValidReset (void)

  Call this function once before checking for a valid image to reset the call flag.
- uint16_t halAppBootloaderImageIsValid (void)

  Reads the app image out of storage, calculates the total file CRC to verify the image is intact.
- EmberStatus halAppBootloaderInstallNewImage (void)

  Invokes the bootloader to install the application in storage. This function resets the device to start the bootloader code and does not return!
- uint8_t halAppBootloaderWriteRawStorage (uint32_t address, const uint8_t *data, uint16_t len)

  Writes data to the specified raw storage address and length without being restricted to any page size Note: Not all storage implementations support accesses that are not page aligned, refer to the HalEepromInformationType structure for more information. Note: Some storage devices require contents to be erased before new data can be written, and will return an EEPROM_ERR_ERASE_REQUIRED error if write is called on a location that is not already erased. Refer to the HalEepromInformationType structure to see if the attached storage device requires erasing.
- uint8_t halAppBootloaderReadRawStorage (uint32_t address, uint8_t *data, uint16_t len)

  Reads data from the specified raw storage address and length without being restricted to any page size Note: Not all storage implementations support accesses that are not page aligned, refer to the HalEepromInformationType structure for more information.
- uint8_t halAppBootloaderEraseRawStorage (uint32_t address, uint32_t len)

  Erases the specified region of the storage device. Note: Most devices require the specified region to be page aligned, and will return an error if an unaligned region is specified. Note: Many devices take an extremely long time to perform an erase operation. When erasing a large region, it may be preferable to make multiple calls to this API so that other application functionality can be performed while the erase is in progress. The halAppBootloaderStorageBusy() API may be used to determine when the last erase operation has completed. Erase timing information can be found in the HalEepromInformationType structure.
- bool halAppBootloaderStorageBusy (void)

  Determine if the attached storage device is still busy performing the last operation, such as a write or an erase.

- uint8_t halAppBootloaderReadDownloadSpace (uint16_t pageToBeRead, uint8_t *destRamBuffer)

  Converts pageToBeRead to an address and the calls storage read function. Note: This function is deprecated. It has been replaced by halAppBootloaderReadRawStorage()
- uint8_t halAppBootloaderWriteDownloadSpace (uint16_t pageToBeWritten, uint8_t *RamPtr)

  Converts pageToBeWritten to an address and calls the storage write function. Note: This function is deprecated. It has been replaced by halAppBootloaderWriteRawStorage()
- uint8_t halAppBootloaderGetImageData (uint32_t *timestamp, uint8_t *userData)

  Read the application image data from storage.
- uint16_t halAppBootloaderGetVersion (void)

  Returns the application bootloader version.
- uint16_t halAppBootloaderGetRecoveryVersion (void)

  Returns the recovery image version.
- bool halAppBootloaderSupportsIbr (void)

  Return a value indicating whether the app bootloader supports IBRs.

8.7.1 Detailed Description

See Application for documentation.

8.8 bootloader-interface-standalone.h File Reference

Macros

- #define NO_BOOTLOADER_MODE 0xFF

  Define a numerical value for NO BOOTLOADER mode. In other words, the bootloader should not be run.
- #define STANDALONE_BOOTLOADER_NORMAL_MODE 1

  Define a numerical value for the normal bootloader mode.
- #define STANDALONE_BOOTLOADER_RECOVERY_MODE 0

  Define a numerical value for the recovery bootloader mode.

Functions

- uint16_t halGetStandaloneBootloaderVersion (void)

  Detects if the standalone bootloader is installed, and if so returns the installed version.
- EmberStatus halLaunchStandaloneBootloader (uint8_t mode)

  Quits the current application and launches the standalone bootloader (if installed). The function returns an error if the standalone bootloader is not present.

8.8.1 Detailed Description

See Standalone for documentation.

8.9 bootloader-interface.h File Reference

```
#include "bootloader-interface-app.h"
#include "bootloader-interface-standalone.h"
```

Macros

- #define BOOTLOADER_BASE_TYPE(extendedType) ((uint8_t)(((extendedType) >> 8U) & 0xFFU))

  Macro returning the base type of a bootloader when given an extended type.
- #define BOOTLOADER_MAKE_EXTENDED_TYPE(baseType, extendedSpecifier) ((uint16_t)(((uint16_t)baseType) << 8U) | (((uint16_t)extendedSpecifier) & 0xFFU))

  Macro returning the extended type of a bootloader when given a base type and extendedSpecifier.
- #define BL_EXT_TYPE_NULL ((BL_TYPE_NULL << 8U) | 0x00U)

  Macro defining the extended NULL bootloader type.
- #define BL_EXT_TYPE_STANDALONE_UNKNOWN ((BL_TYPE_STANDALONE << 8U) | 0x00U)

  Macro defining the extended standalone unknown bootloader type.
- #define BL_EXT_TYPE_SERIAL_UART ((BL_TYPE_STANDALONE << 8U) | 0x01U)

  Macro defining the extended standalone UART bootloader type.
- #define BL_EXT_TYPE_SERIAL_UART_OTA ((BL_TYPE_STANDALONE << 8U) | 0x03U)

  Macro defining the extended standalone OTA and UART bootloader type.
- #define BL_EXT_TYPE_EZSP_SPI ((BL_TYPE_STANDALONE << 8U) | 0x04U)
- #define BL_EXT_TYPE_EZSP_SPI_OTA ((BL_TYPE_STANDALONE << 8U) | 0x06U)
- #define BL_EXT_TYPE_SERIAL_USB ((BL_TYPE_STANDALONE << 8U) | 0x07U)

  Macro defining the extended standalone USB bootloader type.
- #define BL_EXT_TYPE_SERIAL_USB_OTA ((BL_TYPE_STANDALONE << 8U) | 0x08U)

  Macro defining the extended standalone OTA and USB bootloader type.
- #define BL_EXT_TYPE_APP_UNKNOWN ((BL_TYPE_APPLICATION << 8U) | 0x00U)

  Macro defining the extended application unknown bootloader type.
- #define BL_EXT_TYPE_APP_SPI ((BL_TYPE_APPLICATION << 8U) | 0x01U)

  Macro defining the extended application SPI bootloader type.
- #define BL_EXT_TYPE_APP_I2C ((BL_TYPE_APPLICATION << 8U) | 0x02U)

  Macro defining the extended application I2C bootloader type.
- #define BL_EXT_TYPE_APP_LOCAL_STORAGE ((BL_TYPE_APPLICATION << 8U) | 0x03U)

  Macro defining a type for the local storage app bootloader.
- #define BOOTLOADER_INVALID_VERSION 0xFFFF

  Define an invalid bootloader version.
- #define CUSTOMER_APPLICATION_VERSION 0

  Macro defining the customer application version stored in the ApplicationProperties_t struct.
- #define CUSTOMER_APPLICATION_CAPABILITIES 0

  Macro defining the customer application capabilities stored in the ApplicationProperties_t struct.
- #define CUSTOMER_APPLICATION_PRODUCT_ID { 0 }

  Macro defining the customer application product ID stored in the ApplicationProperties_t struct.
- #define MPSI_PLUGIN_SUPPORT 0

  Macro defining the support for the MPSI protocol stored in the capabilities field of the ApplicationProperties_t struct.
- #define APPLICATION_PROPERTIES_CAPABILITIES_MPSI_SUPPORT_BIT 31

  Macro defining the bit position that corresponds to MPSI support in the capabilities field of the ApplicationProperties_t struct.
- #define APPLICATION_PROPERTIES_CAPABILITIES

  Macro defining the capabilities that this application has. This value is set in the capabilities field of the ApplicationProperties_t struct.

Bootloader Numerical Definitions

These are numerical definitions for the possible bootloader types and a typedef of the bootloader base type.

- #define BL_TYPE_NULL (0)

  Numerical definition for a bootloader type.
- #define BL_TYPE_STANDALONE (1)

Numerical definition for a bootloader type.

- #define BL_TYPE_APPLICATION (2)

  Numerical definition for a bootloader type.
- #define BL_TYPE_BOOTLOADER (3)

  Numerical definition for a bootloader type.
- #define BL_TYPE_SMALL_BOOTLOADER (4)

  Numerical definition for a bootloader type.

Typedefs

Bootloader type definitions

These are the type definitions for the bootloader.

- typedef uint8_t BlBaseType

  Define the bootloader base type.
- typedef uint16_t BlExtendedType

  Define the bootloader extended type.

Functions

- BlBaseType halBootloaderGetType (void)

  Returns the bootloader base type the application was built for.
- BlExtendedType halBootloaderGetInstalledType (void)

  Returns the extended bootloader type of the bootloader that is present on the chip.
- uint16_t halGetBootloaderVersion (void)

  Returns the version of the installed bootloader, regardless of its type.
- void halGetExtendedBootloaderVersion (uint32_t *emberVersion, uint32_t *customerVersion)

  Return extended bootloader version information, if supported. This API is not supported for EM2XX chips and only returns extra information on bootloaders built on or after the 4.7 release.

8.9.1 Detailed Description

See Common for detailed documentation.

8.10 bootloader-serial.h File Reference

Common bootloader serial definitions. See Serial for detailed documentation.

Functions

- void serInit (void)
  Initialize serial port.
- void serPutFlush (void)
  Flush the transmiter.
- void serPutChar (uint8_t ch)
  Transmit a character.
- void serPutStr (const char *str)
  Transmit a string.
- void serPutBuf (const uint8_t buf[ ], uint8_t size)
  Transmit a buffer.
- void serPutDecimal (uint16_t val)
  Transmit a 16bit value in decimal.
- void serPutHex (uint8_t byte)
  Transmit a byte as hex.
- void serPutHexInt (uint16_t word)
  Transmit a 16bit integer as hex.
- bool serCharAvailable (void)
  Determine if a character is available.
- uint8_t serGetChar (uint8_t *ch)
  Get a character if available, otherwise return an error.
- void serGetFlush (void)
  Flush the receiver.

8.10.1 Detailed Description

Version

3.20.2

8.10.2 License

(C) Copyright 2015 Silicon Labs, www.silabs.com

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

DISCLAIMER OF WARRANTY/LIMITATION OF REMEDIES: Silicon Labs has no obligation to support this Software. Silicon Labs is providing the Software "AS IS", with no express or implied warranties of any kind, including, but not limited to, any implied warranties of merchantability or fitness for any particular purpose or warranties against infringement of any proprietary rights of a third party.

Silicon Labs will not be liable for any consequential, incidental, or special damages, or any other relief, or for any claim by any third party, arising from your use of this Software.

8.11 button.h File Reference

Macros

Button State Definitions

A set of numerical definitions for use with the button APIs indicating the state of a button.

- #define BUTTON_PRESSED 1
  Button state is pressed.
- #define BUTTON_RELEASED 0
  Button state is released.

Functions

- void halInternalInitButton (void)
  Initializes the buttons. This function is automatically called by halInit().
- uint8_t halButtonState (uint8_t button)
  Returns the current state (pressed or released) of a button.
- uint8_t halButtonPinState (uint8_t button)
  Returns the current state (pressed or released) of the pin associated with a button.
- void halButtonIsr (uint8_t button, uint8_t state)
  A callback called in interrupt context whenever a button changes its state.

8.11.1 Detailed Description

See Button Control for documentation.

8.12 buzzer.h File Reference

Macros

Note Definitions

Flats are used instead of sharps because # is a special character.

- #define NOTE_C3 119
  A note which can be used in tune structure definitions.
- #define NOTE_Db3 112
  A note which can be used in tune structure definitions.
- #define NOTE_D3 106
  A note which can be used in tune structure definitions.
- #define NOTE_Eb3 100
  A note which can be used in tune structure definitions.
- #define NOTE_E3 94
  A note which can be used in tune structure definitions.
- #define NOTE_F3 89
  A note which can be used in tune structure definitions.
- #define NOTE_Gb3 84
  A note which can be used in tune structure definitions.
- #define NOTE_G3 79
  A note which can be used in tune structure definitions.

- #define NOTE_Ab3 74

  A note which can be used in tune structure definitions.
- #define NOTE_A3 70

  A note which can be used in tune structure definitions.
- #define NOTE_Bb3 66

  A note which can be used in tune structure definitions.
- #define NOTE_B3 63

  A note which can be used in tune structure definitions.
- #define NOTE_C4 59

  A note which can be used in tune structure definitions.
- #define NOTE_Db4 55

  A note which can be used in tune structure definitions.
- #define NOTE_D4 52

  A note which can be used in tune structure definitions.
- #define NOTE_Eb4 49

  A note which can be used in tune structure definitions.
- #define NOTE_E4 46

  A note which can be used in tune structure definitions.
- #define NOTE_F4 44

  A note which can be used in tune structure definitions.
- #define NOTE_Gb4 41

  A note which can be used in tune structure definitions.
- #define NOTE_G4 39

  A note which can be used in tune structure definitions.
- #define NOTE_Ab4 37

  A note which can be used in tune structure definitions.
- #define NOTE_A4 35

  A note which can be used in tune structure definitions.
- #define NOTE_Bb4 33

  A note which can be used in tune structure definitions.
- #define NOTE_B4 31

  A note which can be used in tune structure definitions.
- #define NOTE_C5 29

  A note which can be used in tune structure definitions.
- #define NOTE_Db5 27

  A note which can be used in tune structure definitions.
- #define NOTE_D5 26

  A note which can be used in tune structure definitions.
- #define NOTE_Eb5 24

  A note which can be used in tune structure definitions.
- #define NOTE_E5 23

  A note which can be used in tune structure definitions.
- #define NOTE_F5 21

  A note which can be used in tune structure definitions.
- #define NOTE_Gb5 20

  A note which can be used in tune structure definitions.
- #define NOTE_G5 19

  A note which can be used in tune structure definitions.
- #define NOTE_Ab5 18

  A note which can be used in tune structure definitions.
- #define NOTE_A5 17

  A note which can be used in tune structure definitions.
- #define NOTE_Bb5 16

  A note which can be used in tune structure definitions.
- #define NOTE_B5 15

  A note which can be used in tune structure definitions.

Functions

- void halPlayTune_P (uint8_t PGM *tune, bool bkg)

  Plays a tune on the piezo buzzer.
- void halStackIndicatePresence (void)

  Causes something to happen on a node (such as playing a tune on the buzzer) that can be used to indicate where it physically is.

8.12.1 Detailed Description

See Buzzer Control for documentation.

8.13 byte-utilities.h File Reference

Data store and fetch routines.

Macros

- #define EMBER_BITS_TO_BYTES(bits) (((bits) + 7) >> 3)

  This function converts from a number of bits to the equivalent number of bytes.

Functions

- void emberReverseMemCopy (uint8_t *dest, const uint8_t *src, uint8_t length)

  This function copies an array of bytes and reverses the order before writing the data to the destination.
- uint16_t emberFetchLowHighInt16u (const uint8_t *contents)

  This function returns the value built from the two `uint8_t` values `contents[0]` and `contents[1]`. `contents[0]` is the low byte.
- uint16_t emberFetchHighLowInt16u (const uint8_t *contents)

  This function returns the value built from the two `uint8_t` values `contents[0]` and `contents[1]`. `contents[1]` is the low byte.
- void emberStoreLowHighInt16u (uint8_t *contents, uint16_t value)

  This function stores `value` in `contents[0]` and `contents[1]`. `contents[0]` is the low byte.
- void emberStoreHighLowInt16u (uint8_t *contents, uint16_t value)

  This function stores `value` in `contents[0]` and `contents[1]`. `contents[1]` is the low byte.
- uint32_t emberFetchLowHighInt32u (uint8_t *contents)

  This function returns the value built from the four `uint8_t` values `contents[0]`, `contents[1]`, `contents[2]` and `contents[3]`. `contents[0]` is the low byte.
- uint32_t emberFetchHighLowInt32u (uint8_t *contents)
- void emberStoreLowHighInt32u (uint8_t *contents, uint32_t value)

  This function stores `value` in `contents[0]`, `contents[1]`, `contents[2]` and `contents[3]`. `contents[0]` is the low byte.
- void emberStoreHighLowInt32u (uint8_t *contents, uint32_t value)
- void emberStoreHighLowInt48u (uint8_t *contents, uint32_t value)
- uint64_t emberFetchHighLowInt48u (uint8_t *contents)
- uint16_t emStrlen (const uint8_t *const string)
- int8_t emStrcmp (const uint8_t *s1, const uint8_t *s2)
- uint16_t emMatchingPrefixBitLength (const uint8_t *x, uint16_t xLength, const uint8_t *y, uint16_t yLength)

This function returns the length in bits of the matching prefix of `x` *and* `y`*.*

- void emBitCopy (uint8_t *to, const uint8_t *from, uint16_t count)

  This function copies `count` *bits from* `from` *to* `to`*.*

- uint8_t emBitCountInt32u (uint32_t num)

  This function returns the number of set bits in `num`*.*

- uint8_t emberHexToInt (uint8_t ch)

  This function returns the value of hexadecimal digit ch (0 - 15). This function returns a value > 15 if ch is not a hexadecimal digit.

8.13.1 Macro Definition Documentation

8.13.1.1 #define EMBER_BITS_TO_BYTES(*bits*) (((bits) + 7) >> 3)

Parameters

bits	The number of bits.

8.13.2 Function Documentation

8.13.2.1 uint16_t emberFetchHighLowInt16u (const uint8_t * *contents*)

8.13.2.2 uint32_t emberFetchHighLowInt32u (uint8_t * *contents*)

This function returns the value built from the four `uint8_t` values `contents`[0], `contents`[1], `contents`[2] and `contents`[3]. `contents`[3] is the low byte.

8.13.2.3 uint64_t emberFetchHighLowInt48u (uint8_t * *contents*)

This function returns the value built from the six `uint8_t` values `contents`[0] thru `contents`[5]. `contents`[5] is the low bytes.

8.13.2.4 uint16_t emberFetchLowHighInt16u (const uint8_t * *contents*)

8.13.2.5 uint32_t emberFetchLowHighInt32u (uint8_t * *contents*)

8.13.2.6 uint8_t emberHexToInt (uint8_t *ch*)

8.13.2.7 void emberReverseMemCopy (uint8_t * *dest*, const uint8_t * *src*, uint8_t *length*)

Parameters

dest	A pointer to the location where the data will be copied to.
src	A pointer to the location where the data will be copied from.
length	The length (in bytes) of the data to be copied.

8.13.2.8 void emberStoreHighLowInt16u (uint8_t * *contents*, uint16_t *value*)

8.13.2.9 void emberStoreHighLowInt32u (uint8_t * *contents*, uint32_t *value*)

This function stores `value` in `contents`[0], `contents`[1], `contents`[2] and `contents`[3]. `contents`[3] is the low byte.

8.13.2.10 void emberStoreHighLowInt48u (uint8_t * *contents*, uint32_t *value*)

This function stores `value` in `contents`[0] thru `contents`[5]. `contents`[5] is the low byte.

8.13.2.11 void emberStoreLowHighInt16u (uint8_t * *contents*, uint16_t *value*)

8.13.2.12 void emberStoreLowHighInt32u (uint8_t * *contents*, uint32_t *value*)

8.13.2.13 void emBitCopy (uint8_t * *to*, const uint8_t * *from*, uint16_t *count*)

8.13.2.14 uint8_t emBitCountInt32u (uint32_t *num*)

8.13.2.15 uint16_t emMatchingPrefixBitLength (const uint8_t * *x*, uint16_t *xLength*, const uint8_t * *y*, uint16_t *yLength*)

8.13.2.16 int8_t emStrcmp (const uint8_t * *s1*, const uint8_t * *s2*)

8.13.2.17 uint16_t emStrlen (const uint8_t *const *string*)

8.14 callback.doc File Reference

Functions

- int main (MAIN_FUNCTION_PARAMETERS)

  Main Application Entry Point.
- void emberAshStatusHandler (AshState state)

  Notification that ash has changed state.
- void emberAfPluginBatteryMonitorDataReadyCallback (uint16_t batteryVoltageMilliV)

  Data Ready.
- uint16_t halBulbPwmDriverFrequencyCallback (void)

  A callback used to configure the frequency of the PWM driver. This is called by the bulb-pwm driver upon initialization to determine the frequency at which the PWM driver should be driven. It should return either the frequency, in Hz, or USE_DEFAULT_FREQUENCY to indicate that the plugin should use the default value. The default value is 1000 Hz, but can be overridden by a macro in the board header if a user wishes.
- void halBulbPwmDriverInitCompleteCallback (void)

  Function to indicate that the PWM driver has been initialized and the bulb should drive the initial LED PWM values at this time.
- void halBulbPwmDriverBlinkOnCallback (void)

  This callback is generated during blinking behavior when it is time to turn the bulb on. While the plugin will determine when to blink the bulb on or off, it is up to this callback to determine how to turn the bulb on.
- void halBulbPwmDriverBlinkOffCallback (void)

This callback is generated during blinking behavior when it is time to turn the bulb off. While the plugin will determine when to blink the bulb on or off, it is up to this callback to determine how to turn the bulb off.

- void halBulbPwmDriverBlinkStartCallback (void)

  This callback is generated when the application layer makes a call to initiate blinking behavior. It warns the application layer PWM code to not attempt to drive the LEDs directly and interfere with the blinking behavior.

- void halBulbPwmDriverBlinkStopCallback (void)

  This callback is generated when the current blinking command finishes. The application layer PWM code must then determine what the bulb drive should be, based on the current appliation layer attributes (i.e. level, on/off, color XY, etc.)

- void halButtonIsr (uint8_t button, uint8_t state)

  A callback called in interrupt context whenever a button changes its state.

- void emberAfPluginButtonInterfaceButton0PressedShortCallback (uint16_t timePressedMs)

  Button0 Pressed Short.

- void emberAfPluginButtonInterfaceButton1PressedShortCallback (uint16_t timePressedMs)

  Button1 Pressed Short.

- void emberAfPluginButtonInterfaceButton0PressedLongCallback (uint16_t timePressedMs, bool pressedAtReset)

  Button0 Pressed Long.

- void emberAfPluginButtonInterfaceButton1PressedLongCallback (uint16_t timePressedMs, bool pressedAtReset)

  Button1 Pressed Long.

- void emberAfPluginButtonInterfaceButton0PressingCallback (void)

  Button0 Pressing.

- void emberAfPluginButtonInterfaceButton1PressingCallback (void)

  Button1 Pressing.

- void emberAfPluginButtonInterfaceButton0LowCallback (void)

  Button0 Low.

- void emberAfPluginButtonInterfaceButton0HighCallback (void)

  Button0 High.

- void emberAfPluginButtonInterfaceButton1LowCallback (void)

  Button1 Low.

- void emberAfPluginButtonInterfaceButton1HighCallback (void)

  Button1 High.

- void emberButtonPressIsr (uint8_t button, EmberButtonPress press)

  A callback called when a button is pressed. It is sometimes called in ISR context.

- void emberAfPluginColorControlServerComputePwmFromHsvCallback (uint8_t endpoint)

  Compute Pwm from HSV.

- void emberAfPluginColorControlServerComputePwmFromXyCallback (uint8_t endpoint)

  Compute Pwm from HSV.

- void emberAfPluginColorControlServerComputePwmFromTempCallback (uint8_t endpoint)

  Compute Pwm from HSV.

- uint8_t emberConnectionManagerJibGetJoinKeyCallback (uint8_t **joinKey)

  Get the fixed joining key.

- void emberAfPluginGpioSensorStateChangedCallback (uint8_t newSensorState)

  State Changed.

- void halRadioPowerUpHandler (void)

  Handler called whenever the radio is powered on.

- void halRadioPowerDownHandler (void)

  Handler called whenever the radio is powered off.

- void emberZclIdentifyServerStartIdentifyingCallback (EmberZclEndpointId_t endpointId, uint16_t identifyTimeS)

  Start Identifying.

- void emberZclIdentifyServerStopIdentifyingCallback (EmberZclEndpointId_t endpointId)

  Stop Identifying.
- bool emberAfPluginIdleSleepOkToSleepCallback (uint32_t durationMs)

  Ok To Sleep.
- void emberAfPluginIdleSleepWakeUpCallback (uint32_t durationMs)

  Wake Up.
- bool emberAfPluginIdleSleepOkToIdleCallback (uint32_t durationMs)

  Ok To Idle.
- void emberAfPluginIdleSleepActiveCallback (uint32_t durationMs)

  Active.
- void emberAfMarkApplicationBuffersCallback (void)

  Mark Application Buffers.
- void emberAfNetworkStatusCallback (EmberNetworkStatus newNetworkStatus, EmberNetworkStatus old↩ NetworkStatus, EmberJoinFailureReason reason)

  Network Status.
- void halMicrophoneCodecMsadpcmDataReadyCallback (uint8_t ∗data, uint8_t length)

  A callback called when new microphone data is ready.
- void halMicrophoneImaadpcmDataReadyCallback (uint8_t ∗data, uint8_t length)

  A callback called when new microphone data is ready.
- void halOccupancyStateChangedCallback (HalOccupancyState occupancyState)

  Occupancy State Changed.
- void emberZclOccupancySensingServerOccupancyStateChangedCallback (HalOccupancyState occupancy↩ State)

  Occupancy state changed.
- bool emberZclOtaBootloadClientSetVersionInfoCallback ()
- bool emberZclOtaBootloadClientServerHasStaticAddressCallback (EmberZclOtaBootloadClientServerInfo↩ _t ∗serverInfo)
- bool emberZclOtaBootloadClientServerHasDnsNameCallback (EmberZclOtaBootloadClientServerInfo_↩ t ∗serverInfo)
- bool emberZclOtaBootloadClientServerHasDiscByClusterId (const EmberZclClusterSpec_t ∗clusterSpec, EmberCoapResponseHandler responseHandler)
- bool emberZclOtaBootloadClientServerDiscoveredCallback (const EmberZclOtaBootloadClientServerInfo↩ _t ∗serverInfo)
- bool emberZclOtaBootloadClientGetQueryNextImageParametersCallback (EmberZclOtaBootloadFileSpec↩ _t ∗fileSpec, EmberZclOtaBootloadHardwareVersion_t ∗hardwareVersion)
- bool emberZclOtaBootloadClientStartDownloadCallback (const EmberZclOtaBootloadFileSpec_t ∗fileSpec, bool existingFile)
- EmberZclStatus_t emberZclOtaBootloadClientDownloadCompleteCallback (const EmberZclOtaBootload↩ FileSpec_t ∗fileSpec, EmberZclStatus_t status)
- void emberZclOtaBootloadClientPreBootloadCallback (const EmberZclOtaBootloadFileSpec_t ∗fileSpec)
- bool emberZclOtaBootloadServerGetImageNotifyInfoCallback (EmberIpv6Address ∗address, EmberZcl↩ OtaBootloadFileSpec_t ∗fileSpec)
- EmberZclStatus_t emberZclOtaBootloadServerGetNextImageCallback (const EmberIpv6Address ∗source, const EmberZclOtaBootloadFileSpec_t ∗currentFileSpec, EmberZclOtaBootloadFileSpec_t ∗nextFileSpec)
- uint32_t emberZclOtaBootloadServerUpgradeEndRequestCallback (const EmberIpv6Address ∗source, const EmberZclOtaBootloadFileSpec_t ∗fileSpec, EmberZclStatus_t status)
- bool emberAfPluginPollingOkToLongPollCallback (void)

  Ok To Long Poll.
- void emberAfPluginSb1GestureSensorGestureReceivedCallback (uint8_t gestureReceived, uint8_t switch↩ Number)

  Gesture Received.
- void halNcpIsAwakeIsr (bool isAwake)

  The SPI Protocol calls halNcpIsAwakeIsr() once the wakeup handshaking is complete and the NCP is ready to accept a command.

- void emberAfPluginTamperSwitchTamperActiveCallback (void)

  Tamper Active.
- void emberAfPluginTamperSwitchTamperAlarmCallback (void)

  Tamper Alarm.
- void emberAfPluginTransportMqttStateChangedCallback (EmberAfPluginTransportMqttState state)

  MQTT Client State Changed Callback.
- bool emberAfPluginTransportMqttMessageArrivedCallback (const char *topic, const char *payload)

  MQTT Message Arrived.
- void emberZclGetPublicKeyCallback (const uint8_t **publicKey, uint16_t *publicKeySize)
- bool emberZclPreAttributeChangeCallback (EmberZclEndpointId_t endpointId, const EmberZclClusterSpec_t *clusterSpec, EmberZclAttributeId_t attributeId, const void *buffer, size_t bufferLength)
- void emberZclPostAttributeChangeCallback (EmberZclEndpointId_t endpointId, const EmberZclClusterSpec_t *clusterSpec, EmberZclAttributeId_t attributeId, const void *buffer, size_t bufferLength)
- EmberZclStatus_t emberZclReadExternalAttributeCallback (EmberZclEndpointId_t endpointId, const EmberZclClusterSpec_t *clusterSpec, EmberZclAttributeId_t attributeId, void *buffer, size_t bufferLength)
- EmberZclStatus_t emberZclWriteExternalAttributeCallback (EmberZclEndpointId_t endpointId, const EmberZclClusterSpec_t *clusterSpec, EmberZclAttributeId_t attributeId, const void *buffer, size_t bufferLength)
- void emberZclGetDefaultReportingConfigurationCallback (EmberZclEndpointId_t endpointId, const EmberZclClusterSpec_t *clusterSpec, EmberZclReportingConfiguration_t *configuration)
- void emberZclGetDefaultReportableChangeCallback (EmberZclEndpointId_t endpointId, const EmberZclClusterSpec_t *clusterSpec, EmberZclAttributeId_t attributeId, void *buffer, size_t bufferLength)
- void emberZclNotificationCallback (const EmberZclNotificationContext_t *context, const EmberZclClusterSpec_t *clusterSpec, EmberZclAttributeId_t attributeId, const void *buffer, size_t bufferLength)
- void * emberAllocateMemoryForPacketHandler (uint32_t size, void **objectRef)

  This function can be used to hook an external memory allocator into the stack. It will be called when we need to allocate large packets.
- void emberFreeMemoryForPacketHandler (void *objectRef)

  This handler is called when freeing memory allocated with emberAllocateMemoryForPacketHandler.
- void emberMarkApplicationBuffersHandler (void)

  Applications that use buffers must mark them by defining this function. The stack uses this when reclaiming unused buffers.
- void emberCoapRequestHandler (EmberCoapCode code, uint8_t *uri, EmberCoapReadOptions *options, const uint8_t *payload, uint16_t payloadLength, const EmberCoapRequestInfo *info)

  Callback for incoming requests.
- void emberDiagnosticAnswerHandler (EmberStatus status, const EmberIpv6Address *remoteAddress, const uint8_t *payload, uint8_t payloadLength)

  Application callback for emberSendDiagnosticQuery().
- void emberConnectionManagerConnectCompleteCallback (EmberConnectionManagerConnectionStatus status)

  Connection attempt completed.
- void mfglibEndReturn (EmberStatus status, uint32_t receiveCount)

  This function provides the result of a call to mfglibEnd().
- void mfglibGetChannelReturn (uint8_t channel)

  This function provides the result of a call to mfglibGetChannel().
- void mfglibGetOptionsReturn (uint8_t options)

  This function provides the result of a call to mfglibGetOptions().
- void mfglibGetPowerModeReturn (uint16_t txPowerMode)

  This function provides the result of a call to mfglibGetPowerMode().
- void mfglibGetPowerReturn (int8_t power)

  This function provides the result of a call to mfglibGetPower().
- void mfglibGetSynOffsetReturn (int8_t synthOffset)

  This function provides the result of a call to mfglibGetSynOffset().

- void mfglibRxHandler (uint8_t *packet, uint8_t linkQuality, int8_t rssi)

  RX Handler for the mfglib test library.
- void mfglibSendPacketReturn (EmberStatus status)

  This function provides the result of a call to mfglibSendPacket().
- void mfglibSetChannelReturn (EmberStatus status)

  This function provides the result of a call to mfglibSetChannel().
- void mfglibSetOptionsReturn (EmberStatus status)

  This function provides the result of a call to mfglibSetOptions().
- void mfglibSetPowerReturn (EmberStatus status)

  This function provides the result of a call to mfglibSetPower().
- void mfglibStartReturn (EmberStatus status)

  This function provides the result of a call to mfglibStart().
- void mfglibStartStreamReturn (EmberStatus status)

  This function provides the result of a call to mfglibStartStream().
- void mfglibStartToneReturn (EmberStatus status)

  This function provides the result of a call to mfglibStartTone().
- void mfglibStopStreamReturn (EmberStatus status)

  This function provides the result of a call to mfglibStopStream().
- void mfglibStopToneReturn (EmberStatus status)

  This function provides the result of a call to mfglibStopTone().
- void emberIncomingIcmpHandler (Ipv6Header *ipHeader)

  Application callback for an incoming ICMP message.
- void emberAfMainCallback (MAIN_FUNCTION_PARAMETERS)

  Main.
- void emberAfInitCallback (void)

  Init.
- void emberAfTickCallback (void)

  Tick.
- void emberActiveScanHandler (const EmberMacBeaconData *beaconData)

  Reports an incoming beacon during an active scan.
- void emberAddressConfigurationChangeHandler (const EmberIpv6Address *address, uint32_t preferred↩ Lifetime, uint32_t validLifetime, uint8_t addressFlags)

  This is called when a new address is configured on the application.
- void emberAttachToNetworkReturn (EmberStatus status)

  A callback that indicates whether the attach process was successfully initiated via a prior call to emberAttachTo↩ Network(). The status argument is either EMBER_SUCCESS, or EMBER_INVALID_CALL if attach was called when the network status was not EMBER_JOINED_NETWORK_NO_PARENT, or while an attach was underway.
- void emberBecomeCommissionerReturn (EmberStatus status)

  Return call for emberBecomeCommissioner(). The status is EMBER_SUCCESS if a petition was sent or EMBER↩ _ERR_FATAL if some temporary resource shortage prevented doing so.
- void emberChangeNodeTypeReturn (EmberStatus status)

  Provides the result of a call to emberChangeNodeType(): either EMBER_SUCCESS, or EMBER_INVALID_CALL.
- void emberAllowNativeCommissionerReturn (EmberStatus status)

  Provides the result of a call to emberAllowNativeCommissioner(): either EMBER_SUCCESS or EMBER_INVALID↩ _CALL.
- void emberSetCommissionerKeyReturn (EmberStatus status)

  Provides the result of a call to emberSetCommissionerKey(): either EMBER_SUCCESS or EMBER_INVALID_CALL.
- void emberSetPskcHandler (const uint8_t *pskc)

  Handler to let application know that a PSKc TLV was successfully set.
- void emberSetJoinKeyReturn (EmberStatus status)

  Provides the result of a call to emberSetJoinKey(): either EMBER_SUCCESS or EMBER_BAD_ARGUMENT.
- void emberCommissionNetworkReturn (EmberStatus status)

Provides the result of a call to emberCommissionNetwork.

- void emberCommissionerStatusHandler (uint16_t flags, const uint8_t *commissionerName, uint8_t commissionerNameLength)

  Reports on the current commissioner state.
- void emberConfigureGatewayReturn (EmberStatus status)

  Provides the result of a call to emberConfigureGateway.
- void emberSetNdDataReturn (EmberStatus status, uint16_t length)

  Provides the result of a call to emberSetNdData.
- void emberSetLocalNetworkDataReturn (EmberStatus status, uint16_t length)

  Provides the result of a call to emberSetLocalNetworkData.
- void emberConfigureExternalRouteReturn (EmberStatus status)

  Provides the result of a call to emberConfigureExternalRoute.
- void emberCounterHandler (EmberCounterType type, uint16_t increment)

  A callback invoked to inform the application of the occurrence of an event defined by EmberCounterType, for example, transmissions and receptions at different layers of the stack.
- uint16_t emberCounterValueHandler (EmberCounterType type)

  A callback invoked to query the application for the countervalue of an event defined by EmberCounterType.
- void emberCustomHostToNcpMessageHandler (const uint8_t *message, uint8_t messageLength)

  NCP handler called to process a custom message from the Host.
- void emberCustomNcpToHostMessageHandler (const uint8_t *message, uint8_t messageLength)

  Host handler called to process a custom message from the NCP.
- void emberDeepSleepCompleteHandler (uint16_t sleepDuration)

  For a sleepy end device, report how long the chip went to deep sleep. In a NCP + host setup, the stack reports this to the host app.
- void emberDeepSleepReturn (EmberStatus status)

  Provides the result of a call to emberDeepSleep().
- void emberDhcpServerChangeHandler (const uint8_t *prefix, uint8_t prefixLengthInBits, bool available)

  This is called when the stack knows about a new dhcp server or if a dhcp server has become unavailable.
- void emberEnergyScanHandler (uint8_t channel, int8_t maxRssiValue)

  Reports the maximum RSSI value measured on the channel.
- void emberEventDelayUpdatedFromIsrHandler (Event *event)

  This method is called any time an event is scheduled from within an ISR context. It can be used to determine when to stop a long running sleep to see what application or stack events now need to be processed.
- void emberExternalRouteChangeHandler (const uint8_t *prefix, uint8_t prefixLengthInBits, bool available)

  This is called when the stack knows about a border router that has an external route to a prefix.
- void emberFormNetworkReturn (EmberStatus status)

  A callback that indicates whether a prior call to emberFormNetwork() successfully initiated the form process. The status argument is either EMBER_SUCCSS, or EMBER_INVALID_CALL if resume was called when the network status was not EMBER_NO_NETWORK, or a scan was underway.
- void emberGetAntennaModeReturn (EmberStatus status, uint8_t mode)

  Provides the result of a call to emberGetAntennaMode.
- void emberGetCcaThresholdReturn (int8_t threshold)

  Provides the result of a call to emberGetCcaThreshold().
- void emberGetChannelCalDataTokenReturn (uint8_t lna, int8_t tempAtLna, uint8_t modDac, int8_t tempAtModDac)

  Gets the token information for tokenId = EMBER_CHANNEL_CAL_DATA_TOKEN.
- void emberGetCounterReturn (EmberCounterType type, uint16_t value)

  Provides the result of a call to emberGetCounter().
- void emberGetCtuneReturn (uint16_t tune, EmberStatus status)

  Provides the result of a call to emberGetCtune.
- void emberGetGlobalAddressReturn (const EmberIpv6Address *address, uint32_t preferredLifetime, uint32_t validLifetime, uint8_t addressFlags)

Provides the result of a call to emberGetGlobalAddresses.

- void emberGetGlobalPrefixReturn (uint8_t flags, bool isStable, const uint8_t *prefix, uint8_t prefixLengthInBits, uint8_t domainId, uint32_t preferredLifetime, uint32_t validLifetime)

  Provides the result of a call to ::emberGetGlobalPrefix.
- void emberGetMfgTokenReturn (EmberMfgTokenId tokenId, EmberStatus status, const uint8_t *tokenData, uint8_t tokenDataLength)

  Provides the result of a call to emberGetMfgToken.
- void emberGetNetworkDataReturn (EmberStatus status, uint8_t *networkData, uint16_t bufferLength)

  Provides the result of a call to emberGetNetworkData.
- void emberGetNetworkDataTlvReturn (uint8_t typeByte, uint8_t index, uint8_t versionNumber, const uint8_t *tlv, uint8_t tlvLength)

  Provides the result of a call to emberGetNetworkDataTlv().
- void emberGetPtaEnableReturn (bool enabled)

  Provides the result of a call to emberGetPtaEnable.
- void emberGetPtaOptionsReturn (uint32_t options)

  Provides the result of a call to emberGetPtaOptions.
- void emberGetRadioPowerReturn (int8_t power)

  Provides the result of a call to emberGetRadioPower() on the host.
- void emberGetRipEntryReturn (uint8_t index, const EmberRipEntry *entry)

  Provides the result of a call to emberGetRipEntry().
- void emberGetRoutingLocatorReturn (const EmberIpv6Address *rloc)

  Provides the result of a call to emberGetRoutingLocator.
- void emberGetStandaloneBootloaderInfoReturn (uint16_t version, uint8_t platformId, uint8_t microId, uint8_t phyId)

  Provides the result of a call to emberGetStandaloneBootloaderInfo.
- void emberGetTxPowerModeReturn (uint16_t txPowerMode)

  Provides the result of a call to emberGetTxPowerMode() on the host.
- void emberGetVersionsReturn (const uint8_t *versionName, uint16_t managementVersionNumber, uint16_t stackVersionNumber, uint16_t stackBuildNumber, EmberVersionType versionType, const uint8_t *buildTimestamp)

  Provides the result of a call to emberGetVersions().
- void emberHostStateHandler (const EmberNetworkParameters *parameters, const EmberEui64 *localEui64, const EmberEui64 *macExtendedId, EmberNetworkStatus networkStatus)

  In a host/NCP setup, notifies the host to changes in the network parameters.
- void emberInitReturn (EmberStatus status)

  Provides the result of a call to emberInit().
- void emberJoinNetworkReturn (EmberStatus status)

  A callback that indicates whether the join process was successfully initiated via a prior call to emberJoinNetwork() or emberJoinCommissioned(). The possible EmberStatus values are: EMBER_SUCCESS, EMBER_BAD_ARGUMENT, or EMBER_INVALID_CALL (if join was called when the network status was something other than EMBER_NO_NETWORK).
- void emberLaunchStandaloneBootloaderReturn (EmberStatus status)

  Provides the result of a call to emberLaunchStandaloneBootloader.
- void emberLeaderDataHandler (const uint8_t *leaderData)

  A callback invoked when the leader data changes.
- bool emberMacPassthroughFilterHandler (uint8_t *macHeader)

  Application handler to define "passthrough" packets.
- void emberMacPassthroughMessageHandler (PacketHeader header)

  Application handler to intercept "passthrough" packets and handle them at the application.
- bool emberMacRssiFilterHandler (uint8_t *macHeader)

  Application handler to filter 802.15.4 packets to be observed for signal strength.
- void emberMacRssiHandler (int8_t currentRssi)

Gets the received signal strength indication (RSSI) for the last 802.15.4 packet received by the stack.

- void emberNetworkDataChangeHandler (const uint8_t *networkData, uint16_t length)

  This is called when the stack receives new Thread Network Data.

- void emberNetworkStatusHandler (EmberNetworkStatus newNetworkStatus, EmberNetworkStatus oldNetworkStatus, EmberJoinFailureReason reason)

  Reports a change to the network status. For example, the network status changes while going through the joining process, or while reattaching to the network, which can happen for a variety of reasons. In particular, after issuing a form, join, resume, or attach command, the application knows that the device is on the network and ready to communicate when this handler is called with a newNetworkStatus of EMBER_JOINED_NETWORK_ATTACHED.

- void emberOkToNapReturn (uint8_t stateMask)

  If implementing event-driven sleep on an NCP host, this method will return the bitmask indicating the stack's current tasks. (see enum above)

- void emberPollForDataReturn (EmberStatus status)

  Provides the result of a call to emberPollForData().

- void emberRadioGetRandomNumbersReturn (EmberStatus status, const uint16_t *rn, uint8_t count)

  Provides the result of a call to emberRadioGetRandomNumbers.

- void emberRequestDhcpAddressReturn (EmberStatus status, const uint8_t *prefix, uint8_t prefixLengthInBits)

  Provides the result of a call to emberRequestDhcpAddress.

- void emberRequestSlaacAddressReturn (EmberStatus status, const uint8_t *prefix, uint8_t prefixLengthInBits)

  Provides the result of a call to emberRequestSlaacAddress.

- void emberResetMicroHandler (EmberResetCause cause)

  Notifies the application of a reset on the Ember chip due to the indicated cause.

- void emberResetNetworkStateReturn (EmberStatus status)

  Provides the result of a call to emberResetNetworkState().

- void emberResignGlobalAddressReturn (EmberStatus status)

  Provides the result of a call to emberResignGlobalAddress().

- void emberResumeNetworkReturn (EmberStatus status)

  A callback that indicates whether a prior call to emberResumeNetwork() successfully initiated the resume process. The status argument is either EMBER_SUCCESS, or EMBER_INVALID_CALL if resume was called when the network status was not EMBER_SAVED_NETWORK, or while a scan was underway.

- void emberScanReturn (EmberStatus status)

  Provides the status upon completion of a scan.

- void emberSendSteeringDataReturn (EmberStatus status)

  Provides the result of a call to emberSendSteeringData().

- void emberSetAntennaModeReturn (EmberStatus status)

  Provides the result of a call to emberSetAntennaMode.

- void emberSetCcaThresholdReturn (EmberStatus status)

  Provides the result of a call to emberSetCcaThreshold().

- void emberSetCtuneReturn (EmberStatus status)

  Provides the result of a call to emberSetCtune.

- void emberSetMfgTokenReturn (EmberMfgTokenId tokenId, EmberStatus status)

  Provides the result of a call to emberSetMfgToken.

- void emberSetPtaEnableReturn (EmberStatus status)

  Provides the result of a call to emberSetPtaEnable.

- void emberSetPtaOptionsReturn (EmberStatus status)

  Provides the result of a call to emberSetPtaOptions.

- void emberSetRadioHoldOffReturn (EmberStatus status)

  Provides the result of a call to emberSetRadioHoldOff.

- void emberSetRadioPowerReturn (EmberStatus status)

  Provides the result of a call to emberSetRadioPower() on the host.

- void emberSetSecurityParametersReturn (EmberStatus status)

  Provides the result of a call to emberSetSecurityParameters().
- void emberSetTxPowerModeReturn (EmberStatus status)

  Provides the result of a call to emberSetTxPowerMode() on the host.
- void emberSlaacServerChangeHandler (const uint8_t *prefix, uint8_t prefixLengthInBits, bool available)

  This is called when the stack knows about a new SLAAC prefix or if a SLAAC server has become unavailable.
- void emberStackPollForDataReturn (EmberStatus status)

  Provides the result of a call to emberStackPollForData().
- void emberStartHostJoinClientHandler (const uint8_t *parentAddress)

  Callback to tell the host to start security commissioning.
- void emberStateReturn (const EmberNetworkParameters *parameters, const EmberEui64 *localEui64, const EmberEui64 *macExtendedId, EmberNetworkStatus networkStatus)

  In a host/NCP setup, provides the result of a call to emberState() on the host.
- void emberSwitchToNextNetworkKeyHandler (EmberStatus status)

  This can be stubbed out on the SoC and host app. It is used by the NCP to update security on the driver when it is instructed to switch the network key by an over the air update.
- void emberSwitchToNextNetworkKeyReturn (EmberStatus status)

  Provides the result of a call to emberSwitchToNextNetworkKey().
- void emberSetDtlsDeviceCertificateReturn (uint32_t result)

  Provides the result of a call to emberSetDtlsDeviceCertificate().
- void emberSetDtlsPresharedKeyReturn (EmberStatus status)

  Provides the result of a call to emberSetDtlsPresharedKey().
- void emberOpenDtlsConnectionReturn (uint32_t result, const EmberIpv6Address *remoteAddress, uint16_t localPort, uint16_t remotePort)

  Provides the result of a call to emberOpenDtlsConnection().
- void emberDtlsSecureSessionEstablished (uint8_t flags, uint8_t sessionId, const EmberIpv6Address *local↩ Address, const EmberIpv6Address *remoteAddress, uint16_t localPort, uint16_t remotePort)

  Indicates to the application that a secure connection was successfully established.
- void emberGetSecureDtlsSessionIdReturn (uint8_t sessionId, const EmberIpv6Address *remoteAddress, uint16_t localPort, uint16_t remotePort)

  Provides the result of a call to emberGetSecureDtlsSessionId().
- void emberCloseDtlsConnectionReturn (uint8_t sessionId, EmberStatus status)

  Provides the result of a call to emberCloseDtlsConnection(), or indicates that the connection was closed on the other end.
- void emberProcessCoap (const uint8_t *message, uint16_t messageLength, EmberCoapRequestInfo *info)

  Process a CoAP message received over an alternate transport.
- void emberMicroBusyHandler (bool busy)

  Callback informing the application running on the micro of interruptions to normal processing. If ::busy is true, the micro will be busy processing and unavailable for an indefinite period of time. If ::busy is false, the micro has resumed normal operation. The main use case is jpake crypto on EM3xx processors. This gives the application a chance to prepare for the pause in regular processing.
- void halPowerMeterOverHeatStatusChangeCallback (uint8_t status)

  Over Heat Callback.
- void halPowerMeterOverCurrentStatusChangeCallback (uint8_t status)

  Over Current Callback.
- void halPowerMeterCalibrationFinishedCallback (uint16_t gainSetting)

  Calibration Finished Callback.
- void halSimEepromCallback (EmberStatus status)

  The Simulated EEPROM callback function, implemented by the application.
- void emberRadioNeedsCalibratingHandler (void)

  The radio calibration callback function.
- void emberAddAddressDataReturn (uint16_t shortId)

Callback for a debug command. Provides the result of ::emberAddAddressData.

- void emberAssertInfoReturn (const uint8_t *fileName, uint32_t lineNumber)

  Callback for a debug command. Provides the result of ::emberAssertInfo.
- void emberClearAddressCacheReturn (void)

  Callback for a debug command. Provides the result of ::emberClearAddressCache.
- void emberConfigUartReturn (void)

  Callback for a debug command. Provides the result of ::emberConfigUart.
- void emberEchoReturn (const uint8_t *data, uint8_t length)

  Callback for a debug command. Provides the result of emberEcho.
- void emberGetMulticastEntryReturn (uint8_t lastSequence, uint8_t windowBitmask, uint8_t dwellQs, const uint8_t *seed)

  Callback for a debug command. Provides the result of ::emberGetMulticastEntry.
- void emberGetNetworkKeyInfoReturn (EmberStatus status, uint32_t sequence, uint8_t state)

  Callback for a debug command. Provides the result of ::emberGetNetworkKeyInfo.
- void emberGetNodeStatusReturn (EmberStatus status, uint8_t ripId, EmberNodeId nodeId, uint8_t parentRipId, EmberNodeId parentId, const uint8_t *networkFragmentIdentifier, uint32_t networkFrameCounter)

  Callback for a debug command. Provides the result of ::emberGetNodeStatus.
- void emberLookupAddressDataReturn (uint16_t shortId)

  Callback for a debug command. Provides the result of ::emberLookupAddressData.
- void emberNcpUdpStormCompleteHandler (void)

  Callback for a debug command. Provides the result of ::emberNcpUdpStormComplete.
- void emberNcpUdpStormReturn (EmberStatus status)

  Callback for a debug command. Provides the result of ::emberNcpUdpStorm.
- void emberResetNcpAshReturn (void)

  Callback for a debug command. Provides the result of ::emberResetNcpAsh.
- void emberSendDoneReturn (void)

  Callback for a debug command. Provides the result of ::emberSendDone.
- void emberSetRandomizeMacExtendedIdReturn (void)

  Callback for a debug command. Provides the result of ::emberSetRandomizeMacExtendedId.
- void emberSetWakeupSequenceNumberReturn (void)

  Callback for a debug command. Provides the result of ::emberSetWakeupSequenceNumber.
- void emberStartUartStormReturn (void)

  Callback for a debug command. Provides the result of ::emberStartUartStorm.
- void emberStopUartStormReturn (void)

  Callback for a debug command. Provides the result of ::emberStopUartStorm.
- void emberUartSpeedTestReturn (uint32_t totalBytesSent, uint32_t payloadBytesSent, uint32_t timeout)

  Callback for a debug command. Provides the result of ::emberUartSpeedTest.
- void emberUdpHandler (const uint8_t *destination, const uint8_t *source, uint16_t localPort, uint16_t remotePort, const uint8_t *payload, uint16_t payloadLength)

  Application callback for an incoming UDP message.
- void emberUdpMulticastHandler (const uint8_t *destination, const uint8_t *source, uint16_t localPort, uint16_t remotePort, const uint8_t *payload, uint16_t payloadLength)

  Application callback for an incoming UDP multicast.

8.15 child.h File Reference

Functions

- EmberNodeId emberChildId (uint8_t childIndex)

This function converts a child index to a node ID.

- uint8_t emberChildIndex (EmberNodeId childId)

  This function converts a node ID to a child index.

- EmberStatus emberSetMessageFlag (EmberNodeId childId)

  This function sets a flag to indicate a message is pending for a child. Next time that the child polls, it will be informed that it has a pending message. The message is sent from emberPollHandler, which is called when the child requests the data.

- EmberStatus emberClearMessageFlag (EmberNodeId childId)

  This function clears a flag to indicate that no more messages are available for a child. Next time the child polls, it will be informed that it does not have any pending messages.

- void emberPollHandler (EmberNodeId childId, bool transmitExpected)

  A callback that allows the application to send a message in response to a poll from a child.

8.15.1 Function Documentation

8.15.1.1 EmberNodeId emberChildId (uint8_t *childIndex*)

child.h

Parameters

childIndex	The index.

Returns

The node ID of the child or EMBER_NULL_NODE_ID a child isn't at the childIndex specified.

8.15.1.2 uint8_t emberChildIndex (EmberNodeId *childId*)

Parameters

childId	The node ID of the child.

Returns

The child index or 0xFF if the node ID does not belong to a child.

8.15.1.3 EmberStatus emberClearMessageFlag (EmberNodeId *childId*)

Parameters

childId	The ID of the child that no longer has pending messages.

Returns

An EmberStatus value.

- EMBER_SUCCESS - The next time that the child polls, it will be informed that it does not have any pending messages.
- EMBER_NOT_JOINED - The child identified by childId is not our child (it is not in the PAN).

8.15.1.4 void emberPollHandler (EmberNodeId *childId,* bool *transmitExpected*)

This function is called when a child polls, provided that the pending message flag is set for that child (see emberSetMessageFlag(). The message should be sent to the child using ::emberSendUnicast() with the ::EMBER_APS_OPTION_POLL_RESPONSE option.

If the application includes ::emberPollHanlder(), it must define EMBER_APPLICATION_HAS_POLL_HANDLER in its CONFIGURATION_HEADER.

Parameters

childId	The ID of the child that is requesting data.
transmitExpected	true if the child is expecting an application- supplied data message. false otherwise.

8.15.1.5 EmberStatus emberSetMessageFlag (EmberNodeId *childId*)

Parameters

childId	The ID of the child that just polled for data.

Returns

An EmberStatus value.

- EMBER_SUCCESS - The next time that the child polls, it will be informed that it has pending data.
- EMBER_NOT_JOINED - The child identified by childId is not our child (it is not in the PAN).

8.16 coap-diagnostic.h File Reference

Diagnostic Functionality Over CoAP API.

Data Structures

- struct EmberDiagnosticData
- struct MacCountersData

Macros

- #define TYPE_LIST_TLV_LENGTH 16
- #define emberDiagnosticDataHasTlv(data, tlv) (((data)->tlvMask & BIT(tlv)) == BIT(tlv))

Enumerations

- enum EmberDiagnosticValue {
  DIAGNOSTIC_MAC_EXTENDED_ADDRESS = 0,
  DIAGNOSTIC_ADDRESS_16 = 1,
  DIAGNOSTIC_MODE = 2,
  DIAGNOSTIC_TIMEOUT = 3,
  DIAGNOSTIC_CONNECTIVITY = 4,
  DIAGNOSTIC_ROUTING_TABLE = 5,
  DIAGNOSTIC_LEADER_DATA = 6,
  DIAGNOSTIC_NETWORK_DATA = 7,
  DIAGNOSTIC_IPV6_ADDRESS_LIST = 8,
  DIAGNOSTIC_MAC_COUNTERS = 9,
  DIAGNOSTIC_BATTERY_LEVEL = 14,
  DIAGNOSTIC_VOLTAGE = 15,
  DIAGNOSTIC_CHILD_TABLE = 16,
  DIAGNOSTIC_CHANNEL_PAGES = 17,
  DIAGNOSTIC_TYPE_LIST = 18,
  LAST_DIAGNOSTIC_VALUE = DIAGNOSTIC_CHANNEL_PAGES }
- enum MacCountersValue {
  DIAGNOSTIC_PACKETS_SENT = 9,
  DIAGNOSTIC_PACKETS_RECEIVED = 10,
  DIAGNOSTIC_PACKETS_DROPPED_ON_TRANSMIT = 11,
  DIAGNOSTIC_PACKETS_DROPPED_ON_RECEIVE = 12,
  DIAGNOSTIC_SECURITY_ERRORS = 13,
  DIAGNOSTIC_NUMBER_OF_RETRIES = 14 }

Functions

- void emApiSendDiagnostic (const EmberIpv6Address *destination, const uint8_t *requestedTlvs, uint8_t length, const uint8_t *uri)
- void emberSendDiagnosticQuery (const EmberIpv6Address *destination, const uint8_t *requestedTlvs, uint8_t length)

  This function sends a CoAP diagnostic query. See emberDiagnosticAnswerHandler() for callback.
- void emberSendDiagnosticGet (const EmberIpv6Address *destination, const uint8_t *requestedTlvs, uint8_t length)

  This function sends a CoAP diagnostic get. See emberDiagnosticAnswerHandler() for callback.
- void emberSendDiagnosticReset (const EmberIpv6Address *destination, const uint8_t *requestedTlvs, uint8_t length)

  This function sends a CoAP diagnostic reset. See emberDiagnosticAnswerHandler() for callback.
- void emberDiagnosticAnswerHandler (EmberStatus status, const EmberIpv6Address *remoteAddress, const uint8_t *payload, uint8_t payloadLength)

  Application callback for emberSendDiagnosticQuery() and emberSendDiagnosticGet().

8.16.1 Macro Definition Documentation

8.16.1.1 #define emberDiagnosticDataHasTlv(*data, tlv*) (((data)->tlvMask & BIT(tlv)) == BIT(tlv))

8.16.1.2 #define TYPE_LIST_TLV_LENGTH 16

8.16.2 Enumeration Type Documentation

8.16.2.1 enum EmberDiagnosticValue

Enumerator

DIAGNOSTIC_MAC_EXTENDED_ADDRESS

DIAGNOSTIC_ADDRESS_16
DIAGNOSTIC_MODE
DIAGNOSTIC_TIMEOUT
DIAGNOSTIC_CONNECTIVITY
DIAGNOSTIC_ROUTING_TABLE
DIAGNOSTIC_LEADER_DATA
DIAGNOSTIC_NETWORK_DATA
DIAGNOSTIC_IPV6_ADDRESS_LIST
DIAGNOSTIC_MAC_COUNTERS
DIAGNOSTIC_BATTERY_LEVEL
DIAGNOSTIC_VOLTAGE
DIAGNOSTIC_CHILD_TABLE
DIAGNOSTIC_CHANNEL_PAGES
DIAGNOSTIC_TYPE_LIST
LAST_DIAGNOSTIC_VALUE

8.16.2.2 enum MacCountersValue

Enumerator

DIAGNOSTIC_PACKETS_SENT
DIAGNOSTIC_PACKETS_RECEIVED
DIAGNOSTIC_PACKETS_DROPPED_ON_TRANSMIT
DIAGNOSTIC_PACKETS_DROPPED_ON_RECEIVE
DIAGNOSTIC_SECURITY_ERRORS
DIAGNOSTIC_NUMBER_OF_RETRIES

8.16.3 Function Documentation

8.16.3.1 void emApiSendDiagnostic (const EmberIpv6Address * *destination,* const uint8_t * *requestedTlvs,* uint8_t *length,* const uint8_t * *uri*)

8.16.3.2 void emberSendDiagnosticGet (const EmberIpv6Address * *destination,* const uint8_t * *requestedTlvs,* uint8_t *length*)

Parameters

destination	The destination, which must be unicast.
requestedTlvs	An array of requested TLVs.
length	The length of requestedTlvs.

8.16.3.3 void emberSendDiagnosticQuery (const EmberIpv6Address * *destination,* const uint8_t * *requestedTlvs,* uint8_t *length*)

Parameters

destination	The destination, which may be unicast or multicast.
requestedTlvs	An array of requested TLVs.
length	The length of requestedTlvs.

8.16.3.4 void emberSendDiagnosticReset (const EmberIpv6Address * *destination*, const uint8_t * *requestedTlvs*, uint8_t *length*)

Parameters

destination	The destination, which may be unicast or multicast.
requestedTlvs	An array of TLVs marked for reset.
length	The length of requestedTlvs.

8.17 coap.h File Reference

Constrained Application Protocol (CoAP) API.

Data Structures

- struct EmberCoapOption

  Structure that includes options in outgoing requests and responses.
- struct EmberCoapResponseInfo

  Additional information about an incoming response.
- struct EmberCoapSendInfo

  Optional information when sending a message.
- struct EmberCoapRequestInfo

  Additional information about an incoming request.
- struct EmberCoapBlockOption

Macros

- #define MAKE_COAP_CODE(class, detail) ((class << 5) | detail)
- #define GET_COAP_CLASS(code) (((code) & 0xE0) >> 5)
- #define GET_COAP_DETAIL(code) ((code) & 0x1F)
- #define EMBER_COAP_PORT 5683
- #define EMBER_COAP_SECURE_PORT 5684
- #define EMBER_COAP_MAX_TOKEN_LENGTH 8
- #define EMBER_COAP_DEFAULT_TIMEOUT_MS 90000
- #define emberBlockOptionSize(option) (1 << (option)->logSize)

Typedefs

- typedef struct EmberCoapReadOptions_s EmberCoapReadOptions

  This function encapsulates incoming CoAP options.

- typedef bool(∗ EmberCoapTransmitHandler) (const uint8_t ∗payload, uint16_t payloadLength, const EmberIpv6Address ∗localAddress, uint16_t localPort, const EmberIpv6Address ∗remoteAddress, uint16_t remotePort, void ∗transmitHandlerData)

  Function type for alternative transports.

- typedef void(∗ EmberCoapResponseHandler) (EmberCoapStatus status, EmberCoapCode code, EmberCoapReadOptions ∗options, uint8_t ∗payload, uint16_t payloadLength, EmberCoapResponseInfo ∗info)

  Type definition for callback handlers for a response.

Enumerations

- enum EmberCoapClass {
  EMBER_COAP_CLASS_REQUEST = 0,
  EMBER_COAP_CLASS_SUCCESS_RESPONSE = 2,
  EMBER_COAP_CLASS_CLIENT_ERROR_RESPONSE = 4,
  EMBER_COAP_CLASS_SERVER_ERROR_RESPONSE = 5 }
- enum EmberCoapCode {
  EMBER_COAP_CODE_EMPTY = MAKE_COAP_CODE(0, 0),
  EMBER_COAP_CODE_GET = MAKE_COAP_CODE(0, 1),
  EMBER_COAP_CODE_POST = MAKE_COAP_CODE(0, 2),
  EMBER_COAP_CODE_PUT = MAKE_COAP_CODE(0, 3),
  EMBER_COAP_CODE_DELETE = MAKE_COAP_CODE(0, 4),
  EMBER_COAP_CODE_201_CREATED = MAKE_COAP_CODE(2, 1),
  EMBER_COAP_CODE_202_DELETED = MAKE_COAP_CODE(2, 2),
  EMBER_COAP_CODE_203_VALID = MAKE_COAP_CODE(2, 3),
  EMBER_COAP_CODE_204_CHANGED = MAKE_COAP_CODE(2, 4),
  EMBER_COAP_CODE_205_CONTENT = MAKE_COAP_CODE(2, 5),
  EMBER_COAP_CODE_400_BAD_REQUEST = MAKE_COAP_CODE(4, 0),
  EMBER_COAP_CODE_401_UNAUTHORIZED = MAKE_COAP_CODE(4, 1),
  EMBER_COAP_CODE_402_BAD_OPTION = MAKE_COAP_CODE(4, 2),
  EMBER_COAP_CODE_403_FORBIDDEN = MAKE_COAP_CODE(4, 3),
  EMBER_COAP_CODE_404_NOT_FOUND = MAKE_COAP_CODE(4, 4),
  EMBER_COAP_CODE_405_METHOD_NOT_ALLOWED = MAKE_COAP_CODE(4, 5),
  EMBER_COAP_CODE_406_NOT_ACCEPTABLE = MAKE_COAP_CODE(4, 6),
  EMBER_COAP_CODE_412_PRECONDITION_FAILED = MAKE_COAP_CODE(4, 12),
  EMBER_COAP_CODE_413_REQUEST_ENTITY_TOO_LARGE = MAKE_COAP_CODE(4, 13),
  EMBER_COAP_CODE_415_UNSUPPORTED_CONTENT_FORMAT = MAKE_COAP_CODE(4, 15),
  EMBER_COAP_CODE_500_INTERNAL_SERVER_ERROR = MAKE_COAP_CODE(5, 0),
  EMBER_COAP_CODE_501_NOT_IMPLEMENTED = MAKE_COAP_CODE(5, 1),
  EMBER_COAP_CODE_502_BAD_GATEWAY = MAKE_COAP_CODE(5, 2),
  EMBER_COAP_CODE_503_SERVICE_UNAVAILABLE = MAKE_COAP_CODE(5, 3),
  EMBER_COAP_CODE_504_GATEWAY_TIMEOUT = MAKE_COAP_CODE(5, 4),
  EMBER_COAP_CODE_505_PROXYING_NOT_SUPPORTED = MAKE_COAP_CODE(5, 5) }
- enum EmberCoapOptionType {

EMBER_COAP_NO_OPTION = 0,
EMBER_COAP_OPTION_IF_MATCH = 1,
EMBER_COAP_OPTION_URI_HOST = 3,
EMBER_COAP_OPTION_ETAG = 4,
EMBER_COAP_OPTION_IF_NONE_MATCH = 5,
EMBER_COAP_OPTION_OBSERVE = 6,
EMBER_COAP_OPTION_URI_PORT = 7,
EMBER_COAP_OPTION_LOCATION_PATH = 8,
EMBER_COAP_OPTION_URI_PATH = 11,
EMBER_COAP_OPTION_CONTENT_FORMAT = 12,
EMBER_COAP_OPTION_MAX_AGE = 14,
EMBER_COAP_OPTION_URI_QUERY = 15,
EMBER_COAP_OPTION_ACCEPT = 17,
EMBER_COAP_OPTION_LOCATION_QUERY = 20,
EMBER_COAP_OPTION_BLOCK2 = 23,
EMBER_COAP_OPTION_BLOCK1 = 27,
EMBER_COAP_OPTION_SIZE2 = 28,
EMBER_COAP_OPTION_PROXY_URI = 35,
EMBER_COAP_OPTION_PROXY_SCHEME = 39,
EMBER_COAP_OPTION_SIZE1 = 60 }

- enum EmberCoapContentFormatType {
  EMBER_COAP_CONTENT_FORMAT_TEXT_PLAIN = 0,
  EMBER_COAP_CONTENT_FORMAT_LINK_FORMAT = 40,
  EMBER_COAP_CONTENT_FORMAT_XML = 41,
  EMBER_COAP_CONTENT_FORMAT_OCTET_STREAM = 42,
  EMBER_COAP_CONTENT_FORMAT_EXI = 47,
  EMBER_COAP_CONTENT_FORMAT_JSON = 50,
  EMBER_COAP_CONTENT_FORMAT_CBOR = 60,
  EMBER_COAP_CONTENT_FORMAT_LINK_FORMAT_PLUS_CBOR = 65064,
  EMBER_COAP_CONTENT_FORMAT_NONE = -1 }
- enum EmberCoapStatus {
  EMBER_COAP_MESSAGE_TIMED_OUT,
  EMBER_COAP_MESSAGE_ACKED,
  EMBER_COAP_MESSAGE_RESET,
  EMBER_COAP_MESSAGE_RESPONSE }

  Status values passed to response handlers.

Functions

- bool emberCoapIsSuccessResponse (EmberCoapCode code)

  This function indicates whether the code represents a successful response.
- bool emberCoapIsClientErrorResponse (EmberCoapCode code)

  This function indicates whether the code represents a client error response.
- bool emberCoapIsServerErrorResponse (EmberCoapCode code)

  This function indicates whether the code represents a server error response.
- bool emberCoapIsRequest (EmberCoapCode code)

  This function indicates whether the code represents a request.
- bool emberCoapIsResponse (EmberCoapCode code)

  This function indicates whether the code represents a response.
- EmberCoapOptionType emberReadNextOption (EmberCoapReadOptions *options, const uint8_t **value↩ PointerLoc, uint16_t *valueLengthLoc)

  This function reads the next option from an incoming message.
- void emberResetReadOptionPointer (EmberCoapReadOptions *options)

  This function resets the internal pointer back to the first option.

- uint32_t emberReadOptionValue (const uint8_t *value, uint16_t valuelength)

  This function decodes the value of an integer option.
- bool emberReadIntegerOption (EmberCoapReadOptions *options, EmberCoapOptionType type, uint32_t *valueLoc)

  This function reads the value of an integer option.
- bool emberReadBytesOption (EmberCoapReadOptions *options, EmberCoapOptionType type, const uint8_t **valueLoc, uint16_t *valueLengthLoc)

  This function reads the value of an option.
- int16_t emberReadLocationPath (EmberCoapReadOptions *options, uint8_t *pathBuffer, uint16_t pathBufferLength)

  This function converts path options to a string.
- EmberStatus emberCoapSend (const EmberIpv6Address *destination, EmberCoapCode code, const uint8_t *path, const uint8_t *payload, uint16_t payloadLength, EmberCoapResponseHandler responseHandler, const EmberCoapSendInfo *info)

  This function sends a request.
- EmberStatus emberCoapGet (const EmberIpv6Address *destination, const uint8_t *path, EmberCoapResponseHandler responseHandler, const EmberCoapSendInfo *info)

  This function sends a GET request.
- EmberStatus emberCoapPut (const EmberIpv6Address *destination, const uint8_t *path, const uint8_t *payload, uint16_t payloadLength, EmberCoapResponseHandler responseHandler, const EmberCoapSendInfo *info)

  This function sends a PUT request.
- EmberStatus emberCoapPost (const EmberIpv6Address *destination, const uint8_t *path, const uint8_t *payload, uint16_t payloadLength, EmberCoapResponseHandler responseHandler, const EmberCoapSendInfo *info)

  This function sends a POST request.
- EmberStatus emberCoapDelete (const EmberIpv6Address *destination, const uint8_t *path, EmberCoapResponseHandler responseHandler, const EmberCoapSendInfo *info)

  This function sends a DELETE request.
- void emberCoapRequestHandler (EmberCoapCode code, uint8_t *uri, EmberCoapReadOptions *options, const uint8_t *payload, uint16_t payloadLength, const EmberCoapRequestInfo *info)

  Callback for incoming requests.
- EmberStatus emberCoapRespond (const EmberCoapRequestInfo *requestInfo, EmberCoapCode code, const EmberCoapOption *options, uint8_t numberOfOptions, const uint8_t *payload, uint16_t payloadLength)

  Sending a response.
- EmberStatus emberCoapRespondWithPath (const EmberCoapRequestInfo *requestInfo, EmberCoapCode code, const uint8_t *path, const EmberCoapOption *options, uint8_t numberOfOptions, const uint8_t *payload, uint16_t payloadLength)

  Sending a response that includes a location path.
- EmberStatus emberCoapRespondWithCode (const EmberCoapRequestInfo *requestInfo, EmberCoapCode code)

  Sending a response that consists of just a code.
- EmberStatus emberCoapRespondWithPayload (const EmberCoapRequestInfo *requestInfo, EmberCoapCode code, const uint8_t *payload, uint16_t payloadLength)

  Sending a response that consists of a code and a payload.
- void emberSaveRequestInfo (const EmberCoapRequestInfo *from, EmberCoapRequestInfo *to)

  This function saves a `EmberCoapRequestInfo` for later use.
- void emberProcessCoap (const uint8_t *message, uint16_t messageLength, EmberCoapRequestInfo *info)

  This function processes a CoAP message received over an alternate transport.
- uint32_t emberBlockOptionOffset (EmberCoapBlockOption *option)
- bool emberReadBlockOption (EmberCoapReadOptions *options, EmberCoapOptionType type, EmberCoapBlockOption *option)

- void emberParseBlockOptionValue (uint32_t value, EmberCoapBlockOption *option)
- uint32_t emberBlockOptionValue (bool more, uint8_t logSize, uint32_t number)
- void emberInitCoapOption (EmberCoapOption *option, EmberCoapOptionType type, uint32_t value)
- bool emberVerifyBlockOption (const EmberCoapBlockOption *blockOption, uint16_t payloadLength, uint8_t expectedLogSize)
- EmberStatus emberCoapRequestNextBlock (EmberCoapCode code, const uint8_t *path, EmberCoapBlockOption *block2Option, EmberCoapResponseHandler responseHandler, EmberCoapResponseInfo *responseInfo)

8.18 command-interpreter2.h File Reference

Data Structures

- struct EmberCommandEntry

  Command entry for a command table.
- struct EmberCommandState

  For use when declaring a separate command streams. The fields are not accessed directly by the application.

Macros

- #define MAX_TOKEN_COUNT (EMBER_MAX_COMMAND_ARGUMENTS + 1)
- #define MAX_COMMAND_TABLE_NESTING 16
- #define emberBinaryCommand emberBinaryCommandEntryAction
- #define emberBinaryNestedCommand emberBinaryCommandEntrySubMenu
- #define emberBinaryCommandEntryAction(identifier, command, arguments, description) { { (PGM_P)identifier }, command, arguments, description }
- #define emberBinaryCommandEntrySubMenu(identifier, nestedCommands, description) { { (PGM_P)identifier }, NULL, description, nestedCommands }
- #define emberCommandEntryAction(name, command, arguments, description) { { name }, command, arguments, description }
- #define emberCommandEntrySubMenu(name, nestedCommands, description) { { name }, NULL, (PGM_P)nestedCommands, description }
- #define emberCommandEntryTerminator() { { NULL }, NULL, NULL, NULL }
- #define emberCommand emberCommandEntryAction
- #define emberNestedCommand emberCommandEntrySubMenu
- #define emberProcessCommandInput(port) emberProcessCommandString(NULL, port)

  This function processes input coming in on the given serial port.

Command Table Settings

- #define EMBER_MAX_COMMAND_ARGUMENTS 14
- #define EMBER_COMMAND_BUFFER_LENGTH 100
- #define EMBER_CUSTOM_COMMAND_BUFFER_LENGTH (EMBER_COMMAND_BUFFER_LENGTH - 3)
- #define EMBER_COMMAND_INTERPRETER_HAS_DESCRIPTION_FIELD
- #define EMBER_COMMAND_INTERPRETER_NO_ERROR_MESSAGE

Typedefs

- typedef void(* CommandAction) (void)
- typedef void EmberCommandErrorHandler(EmberCommandStatus status, EmberCommandEntry *command)

  Type of error handlers; the command argument is currently always NULL.

Enumerations

- enum EmberCommandStatus {
  EMBER_CMD_SUCCESS,
  EMBER_CMD_ERR_PORT_PROBLEM,
  EMBER_CMD_ERR_NO_SUCH_COMMAND,
  EMBER_CMD_ERR_WRONG_NUMBER_OF_ARGUMENTS,
  EMBER_CMD_ERR_ARGUMENT_OUT_OF_RANGE,
  EMBER_CMD_ERR_ARGUMENT_SYNTAX_ERROR,
  EMBER_CMD_ERR_STRING_TOO_LONG,
  EMBER_CMD_ERR_INVALID_ARGUMENT_TYPE }

  Command error states.

Functions

- void emberCommandErrorHandler (EmberCommandStatus status, EmberCommandEntry *command)
- void emberPrintCommandUsage (EmberCommandEntry *entry)
- void emberPrintCommandUsageNotes (void)
- void emberPrintCommandTable (void)
- void emberCommandClearBuffer (void)
- void emberCommandReaderInit (void)

  This function nitializes the command interpreter.
- bool emberProcessCommandString (const uint8_t *input, uint16_t sizeOrPort)

  This function processes the given string as a command.
- void emberInitializeCommandState (EmberCommandState *state)

  This function must be called to initialize a command state before passing it to emberRunBinaryCommandInterpreter() or emberRunAsciiCommandInterpreter().
- bool emberRunBinaryCommandInterpreter (EmberCommandState *state, EmberCommandEntry *commands, EmberCommandErrorHandler *errorHandler, const uint8_t *input, uint16_t sizeOrPort)

  For use to process binary commands when additional different command streams are being used.
- bool emberRunAsciiCommandInterpreter (EmberCommandState *state, EmberCommandEntry *commands, EmberCommandErrorHandler *errorHandler, const uint8_t *input, uint16_t sizeOrPort)

  For use to process ASCII commands when additional different command streams are being used.
- void emberCommandReaderSetDefaultBase (uint8_t base)

Variables

- EmberCommandEntry emberCommandTable [ ]

Functions to Retrieve Arguments

Use the following functions in your functions that process commands to retrieve arguments from the command interpreter. These functions pull out unsigned integers, signed integers, and strings, and hex strings. Index 0 is the first command argument.

- #define emberGetKeyArgument(index, keyDataPointer)
- #define emberGetEui64Argument(index, eui64) emberGetExtendedPanIdArgument(index, (eui64)->bytes)
- uint8_t emberCommandArgumentCount (void)
- uint32_t emberUnsignedCommandArgument (uint8_t argNum)
- int32_t emberSignedCommandArgument (uint8_t argNum)
- uint8_t * emberStringCommandArgument (int8_t argNum, uint8_t *length)

- uint8_t * emberLongStringCommandArgument (int8_t argNum, uint16_t *length)
- const char * emberCommandName (void)
- uint8_t emberGetStringArgument (int8_t argNum, uint8_t *destination, uint8_t maxLength, bool leftPad)
- bool emberGetIpArgument (uint8_t index, uint8_t *target)
- bool emberGetIpv6AddressArgument (uint8_t index, EmberIpv6Address *dst)

  This function parses an IPv6 address in a CLI command.
- bool emberGetIpv6PrefixArgument (uint8_t index, EmberIpv6Address *dst, uint8_t *dstPrefixBits)

  This function parses an IPv6 prefix in a CLI command.
- void emberGetExtendedPanIdArgument (int8_t index, uint8_t *extendedPanId)

8.19 counters.h File Reference

Variables

- uint16_t emberCounters [EMBER_COUNTER_TYPE_COUNT]

8.19.1 Detailed Description

A library to tally up Ember stack counter events.

The Ember stack tracks a number of events defined by ::EmberCountersType and reports them to the application via the emberCounterHandler() callback. This library keeps a tally of the number of times each type of a counter event occurs. The application must define ::EMBER_APPLICATION_HAS_COUNTER_HANDLER in its CONFI↩ GURATION_HEADER to use this library.

8.19.2 Variable Documentation

8.19.2.1 uint16_t emberCounters[EMBER_COUNTER_TYPE_COUNT]

The ith entry in this array is the count of events of EmberCounterType i.

8.20 crc.h File Reference

Macros

- #define INITIAL_CRC 0xFFFFFFFFL
- #define CRC32_START INITIAL_CRC
- #define CRC32_END 0xDEBB20E3L

Functions

- uint16_t halCommonCrc16 (uint8_t newByte, uint16_t prevResult)

  Calculates 16-bit cyclic redundancy code (CITT CRC 16).
- uint32_t halCommonCrc32 (uint8_t newByte, uint32_t prevResult)

  Calculates 32-bit cyclic redundancy code.

8.20.1 Detailed Description

See Cyclic Redundancy Code (CRC) for detailed documentation.

8.21 dev0680.h File Reference

Macros

- #define PWRUP_CFG_SC1_TXD _GPIO_P_CFGL_Px0_OUT_ALT
  Give GPIO SC1 TXD and nRTS configurations friendly names.
- #define PWRDN_OUT_SC1_nRTS 1

Custom Baud Rate Definitions

Application Framework NCP Configuration Board Header

This board header (dev0680) is not supported in framework NCP applications. NCP applications must use either the dev0680spi or dev0680uart board headers when creating custom NCP applications through the framework.

The following define is used with defining a custom baud rate for the UART. This define provides a simple hook into the definition of the baud rates used with the UART. The baudSettings[] array in uart.c links the BAUD_∗ defines with the actual register values needed for operating the UART. The array baudSettings[] can be edited directly for a custom baud rate or another entry (the register settings) can be provided here with this define.

- #define EMBER_SERIAL_BAUD_CUSTOM 13
  This define is the register setting for generating a baud of.

Button Definitions

The following are used to aid in the abstraction with the Button connections. The microcontroller-specific sources use these definitions so they are able to work across a variety of boards which could have different connections. The names and ports/pins used below are intended to match with a schematic of the system to provide the abstraction.

The BUTTONn macros should always be used with manipulating the buttons as they directly refer to the GPIOs to which the buttons are connected.

Note

The GPIO number must match the IRQ letter

- #define BUTTON0 PORTB_PIN(6)
  The actual GPIO BUTTON0 is connected to. This define should be used whenever referencing BUTTON0.
- #define BUTTON0_IN (GPIO->P[1].IN)
  The GPIO input register for BUTTON0.
- #define BUTTON0_SEL() do {} while (0)
  Point the proper IRQ at the desired pin for BUTTON0.
- #define BUTTON0_ISR halIrqBIsr
  The interrupt service routine for BUTTON0.
- #define BUTTON0_INTCFG (EVENT_GPIO->CFGB)
  The interrupt configuration register for BUTTON0.
- #define BUTTON0_INT_EN_IRQN IRQB_IRQn
  The interrupt enable bit for BUTTON0.
- #define BUTTON0_INT_EN_BIT BIT32(BUTTON0_INT_EN_IRQN)
  The actual GPIO BUTTON0 is connected to. This define should be used whenever referencing BUTTON0.
- #define BUTTON0_FLAG_BIT EVENT_GPIO_FLAG_IRQB
  The interrupt flag bit for BUTTON0.
- #define BUTTON0_MISS_BIT EVENT_MISS_MISS_IRQB

The missed interrupt bit for BUTTON0.

- #define BUTTON1 PORTC_PIN(6)

  The actual GPIO BUTTON1 is connected to. This define should be used whenever referencing BUTTON1, such as controlling if pieces are compiled in. Remember there may be other things that might want to use IRQC.
- #define BUTTON1_IN (GPIO->P[2].IN)

  The GPIO input register for BUTTON1.
- #define BUTTON1_SEL() do { GPIO->IRQCSEL = PORTC_PIN(6); } while (0)

  Point the proper IRQ at the desired pin for BUTTON1. Remember there may be other things that might want to use IRQC.
- #define BUTTON1_ISR halIrqCIsr

  The interrupt service routine for BUTTON1. Remember there may be other things that might want to use IRQC.
- #define BUTTON1_INTCFG (EVENT_GPIO->CFGC)

  The interrupt configuration register for BUTTON1.
- #define BUTTON1_INT_EN_IRQN IRQC_IRQn

  The interrupt enable bit for BUTTON1.
- #define BUTTON1_INT_EN_BIT BIT32(BUTTON1_INT_EN_IRQN)

  The actual GPIO BUTTON0 is connected to. This define should be used whenever referencing BUTTON0.
- #define BUTTON1_FLAG_BIT EVENT_GPIO_FLAG_IRQC

  The interrupt flag bit for BUTTON1.
- #define BUTTON1_MISS_BIT EVENT_MISS_MISS_IRQC

  The missed interrupt bit for BUTTON1.

USB Power State

Define if the USB is self powered or bus powered since the configuration descriptor needs to report to the host the powered state.

Note

VBUS Monitoring is required for USB to function when the EM358 device is configured as self-powered.

- #define USB_SELFPWRD_STATE (1)

  The USB power state.

USB Remote Wakeup Enable

If the USB device needs to awake the host from suspend, then it needs to have remote wakeup enable.

Note

The host can deny remote wakeup, keeping the device in suspend.

If the device has remote wakeup enabled the configuration descriptor needs to report this fact to the host. Additionally, the USB core in the chip needs to be directly told. Set the define USB_REMOTEWKUPEN_STATE to 0 if remote wake is disabled or 1 if enabled.

- #define USB_REMOTEWKUPEN_STATE (1)

  USB Remote Wakeup Enable.

USB Maximum Power Consumption

The USB device must report the maximum power it will draw from the bus. This is done via the bMaxPower parameter in the Configuration Descriptor reported to the host. The value used is in units of 2mA.

Self-powered devices are low power devices and must draw less than 100mA.

Systems that have components such as a FEM are likely to consume more than 100mA and are considered high power and therefore must be bus-powered.

- #define USB_MAX_POWER (50)

  USB Max Power parameter (bMaxPower) the driver will report to the host in the Configuration Descriptor.

USB Enumeration Control

The following are used to aid in the abstraction of which GPIO is used for controlloing the pull-up resistor for enumeation.

The hardware setup connects the D+ signal to a GPIO via a 1.5kOhm pull-up resistor. Any GPIO can be used since it just needs to be a simple push-pull output configuration.

- #define ENUMCTRL PORTA_PIN(2)
  The actual GPIO ENUMCTRL is connected to. The GPIO only needs to be a simple push-pull output or input.
- #define ENUMCTRL_SETCFG(cfg)
  Set the GPIO's configuration to the provided state. The two states used are GPIOCFG_OUT when the device is enumerated and GPIOCFG_IN when the device is not enumerated.
- #define ENUMCTRL_SET() do { GPIO->P[0].SET = GPIO_P_SET_Px2; } while (0)
  When the GPIO used for enumeration is configured as push-pull, this macro makes it easy to set the output state high.
- #define ENUMCTRL_CLR() do { GPIO->P[0].CLR = GPIO_P_SET_Px2; } while (0)
  When the GPIO used for enumeration is configured as push-pull, this macro makes it easy to clear the output state low.

USB VBUS Monitoring Support

Note

VBUS Monitoring is required for USB to function when the EM358 device is configured as self-powered.

The following are used to aid in the abstraction of which GPIO and IRQ is used for VBUS Monitoring.

Remember that IRQA and IRQB are fixed to GPIO PB0 and PB6 respectively while IRQC and IRQD can be assigned to any GPIO. Since USB's D- and D+ data pins are fixed to PA0 and PA1 respectively, SC2 can't be used so it makes sense to allocate PA2 for enumeration control and PA3 for VBUS monitoring. Therefore, using PA3 for VBUS monitoring requires IRQC or IRQD.

The driver will only try to use VBUSMON functionality if USB_SELFPWRD_STATE is set to 1.

- #define VBUSMON GPIO_P_IN_Px3
  The actual GPIO VBUSMON is connected to. Remember that other pieces might want to use PA3.
- #define VBUSMON_IN (GPIO->P[0].IN)
  The GPIO input register for VBUSMON.
- #define VBUSMON_SETCFG()
  The GPIO configuration needed for VBUSMON. The configuration needs to be a simple input that will monitor for edge tansitions.
- #define VBUSMON_SEL() do { GPIO->IRQDSEL = PORTA_PIN(3); } while (0)
  Point the proper IRQ at the desired pin for VBUSMON. Remember that other pieces that might want to use IRQC.
- #define VBUSMON_ISR halIrqDIsr
  The interrupt service routine for VBUSMON. Remember that other pieces that might want to use IRQC.
- #define VBUSMON_INTCFG (EVENT_GPIO->CFGD)
  The interrupt configuration register for VBUSMON.
- #define VBUSMON_INT_EN_IRQN IRQD_IRQn
  The interrupt enable bit for VBUSMON.
- #define VBUSMON_INT_EN_BIT BIT32(VBUSMON_INT_EN_IRQN)
  The actual GPIO VBUSMON is connected to. Remember that other pieces might want to use PA3.
- #define VBUSMON_FLAG_BIT EVENT_GPIO_FLAG_IRQD
  The interrupt flag bit for VBUSMON.
- #define VBUSMON_MISS_BIT EVENT_MISS_MISS_IRQD
  The missed interrupt bit for VBUSMON.

Radio HoldOff Configuration Definitions

This define does not equate to anything. It is used as a trigger to enable Radio HoldOff support.

The following are used to aid in the abstraction with Radio HoldOff (RHO). The microcontroller-specific sources use these definitions so they are able to work across a variety of boards which could have different connections. The names and ports/pins used below are intended to match with a schematic of the system to provide the abstraction.

The Radio HoldOff input GPIO is abstracted like BUTTON0/1.

- #define RHO_ASSERTED 1
  The actual GPIO used to control Radio HoldOff.
- #define RHO_CFG (GPIO->P[0].CFGH)
  The GPIO configuration register for Radio HoldOff.
- #define RHO_IN (GPIO->P[0].IN)
  The GPIO input register for Radio HoldOff.
- #define RHO_OUT (GPIO->P[0].OUT)
  The GPIO output register for Radio HoldOff.
- #define RHO_SEL() do { GPIO->IRQDSEL = RHO_GPIO; } while (0)
  Point the proper IRQ at the desired pin for Radio HoldOff. Remember there may be other things that might want to use this IRQ.
- #define RHO_ISR halIrqDIsr
  The interrupt service routine for Radio HoldOff. Remember there may be other things that might want to use this IRQ.
- #define RHO_INTCFG (EVENT_GPIO->CFGD)
  The interrupt configuration register for Radio HoldOff.
- #define RHO_INT_EN_IRQN IRQD_IRQn
  The interrupt enable bit for Radio HoldOff.
- #define RHO_INT_EN_BIT BIT32(RHO_INT_EN_IRQN)
  The actual GPIO used to control Radio HoldOff.
- #define RHO_FLAG_BIT EVENT_GPIO_FLAG_IRQD
  The interrupt flag bit for Radio HoldOff.
- #define RHO_MISS_BIT EVENT_MISS_MISS_IRQD
  The missed interrupt bit for Radio HoldOff.
- #define PWRUP_CFG_DFL_RHO_FOR_RHO _GPIO_P_CFGL_Px0_IN_PUD
  Configuration of GPIO for Radio HoldOff operation.
- #define PWRUP_OUT_DFL_RHO_FOR_RHO 0 /* Deassert */
  The actual GPIO used to control Radio HoldOff.
- #define PWRDN_CFG_DFL_RHO_FOR_RHO _GPIO_P_CFGL_Px0_IN_PUD
  The actual GPIO used to control Radio HoldOff.
- #define PWRDN_OUT_DFL_RHO_FOR_RHO 0 /* Deassert */
  The actual GPIO used to control Radio HoldOff.
- #define PWRUP_CFG_DFL_RHO_FOR_DFL _GPIO_P_CFGL_Px0_OUT
  Configuration of GPIO for default behavior.
- #define PWRUP_OUT_DFL_RHO_FOR_DFL 1 /* LED default off */
  The actual GPIO used to control Radio HoldOff.
- #define PWRDN_CFG_DFL_RHO_FOR_DFL _GPIO_P_CFGL_Px0_OUT
  The actual GPIO used to control Radio HoldOff.
- #define PWRDN_OUT_DFL_RHO_FOR_DFL 1 /* LED off */
  The actual GPIO used to control Radio HoldOff.
- #define PWRUP_CFG_DFL_RHO PWRUP_CFG_DFL_RHO_FOR_DFL
  The following definitions are helpers for managing Radio HoldOff and should not be modified.
- #define PWRUP_OUT_DFL_RHO PWRUP_OUT_DFL_RHO_FOR_DFL
  The actual GPIO used to control Radio HoldOff.
- #define PWRDN_CFG_DFL_RHO PWRDN_CFG_DFL_RHO_FOR_DFL
  The actual GPIO used to control Radio HoldOff.
- #define PWRDN_OUT_DFL_RHO PWRDN_OUT_DFL_RHO_FOR_DFL
  The actual GPIO used to control Radio HoldOff.
- #define halInternalInitRadioHoldOff() /* no-op */
  The actual GPIO used to control Radio HoldOff.

Temperature sensor ADC channel

Define the analog input channel connected to the LM-20 temperature sensor. The scale factor compensates for different platform input ranges. PB5/ADC0 must be an analog input. PC7 must be an output and set to a high level to power the sensor.

- #define TEMP_SENSOR_ADC_CHANNEL ADC_SOURCE_ADC0_VREF2
  The analog input channel to use for the temperature sensor.

- #define TEMP_SENSOR_SCALE_FACTOR 1
  The scale factor to compensate for different input ranges.

Packet Trace

When PACKET_TRACE is defined, ::GPIO_PACFGH will automatically be setup by halInit() to enable Packet Trace support on PA4 and PA5, in addition to the configuration specified below.

Note

This define will override any settings for PA4 and PA5.

- #define PACKET_TRACE
  This define does not equate to anything. It is used as a trigger to enable Packet Trace support on the breakout board (dev0680).

ENABLE_OSC32K

When ENABLE_OSC32K is defined, halInit() will configure system timekeeping to utilize the external 32.768 kHz crystal oscillator rather than the internal 1 kHz RC oscillator.

Note

ENABLE_OSC32K is mutually exclusive with ENABLE_ALT_FUNCTION_NTX_ACTIVE since they define conflicting usage of GPIO PC6.

On initial powerup the 32.768 kHz crystal oscillator will take a little while to start stable oscillation. This only happens on initial powerup, not on wake-from-sleep, since the crystal usually stays running in deep sleep mode.

When ENABLE_OSC32K is defined the crystal oscillator is started as part of halInit(). After the crystal is started we delay for OSC32K_STARTUP_DELAY_MS (time in milliseconds). This delay allows the crystal oscillator to stabilize before we start using it for system timing.

If you set OSC32K_STARTUP_DELAY_MS to less than the crystal's startup time:

- *The system timer won't produce a reliable one millisecond tick before the crystal is stable.*
- *You may see some number of ticks of unknown period occur before the crystal is stable.*
- *halInit() will complete and application code will begin running, but any events based on the system timer will not be accurate until the crystal is stable.*
- *An unstable system timer will only affect the APIs in system-timer.h.*

Typical 32.768 kHz crystals measured by Ember take about 400 milliseconds to stabilize. Be sure to characterize your particular crystal's stabilization time since crystal behavior can vary.

- #define OSC32K_STARTUP_DELAY_MS (0)

Packet Trace Configuration Defines

Provide the proper set of pin configuration for when the Packet Trace is enabled (look above for the define which enables it). When Packet Trace is not enabled, leave the two PTI pins in their default configuration. If Packet Trace is not being used, feel free to set the pin configurations as desired. The config shown here is simply the Power On Reset defaults.

- #define PWRUP_CFG_PTI_EN _GPIO_P_CFGL_Px0_OUT_ALT
  Give the packet trace configuration a friendly name.
- #define PWRUP_OUT_PTI_EN 0
  Give the packet trace configuration a friendly name.
- #define PWRDN_CFG_PTI_EN _GPIO_P_CFGL_Px0_IN_PUD
  Give the packet trace configuration a friendly name.
- #define PWRDN_OUT_PTI_EN 0
  Give the packet trace configuration a friendly name.
- #define PWRUP_CFG_PTI_DATA _GPIO_P_CFGL_Px0_OUT_ALT

Give the packet trace configuration a friendly name.

- #define PWRUP_OUT_PTI_DATA 1

  Give the packet trace configuration a friendly name.
- #define PWRDN_CFG_PTI_DATA _GPIO_P_CFGL_Px0_IN_PUD

  Give the packet trace configuration a friendly name.
- #define PWRDN_OUT_PTI_DATA 1

  Give the packet trace configuration a friendly name.

32kHz Oscillator and nTX_ACTIVE Configuration Defines

Since the 32kHz Oscillator and nTX_ACTIVE both share PC6, their configuration defines are linked and instantiated together. Look above for the defines that enable the 32kHz Oscillator and nTX_ACTIVE.

Note

ENABLE_OSC32K is mutually exclusive with ENABLE_ALT_FUNCTION_NTX_ACTIVE since they define conflicting usage of GPIO PC6.

When using the 32kHz, configure PC6 and PC7 for analog for the XTAL.

When using nTX_ACTIVE, configure PC6 for alternate output while awake and a low output when deepsleeping. Also, configure PC7 for TEMP_EN.

When not using the 32kHz or nTX_ACTIVE, configure PC6 and PC7 for Button1 and TEMP_EN.

- #define PWRUP_CFG_BUTTON1 _GPIO_P_CFGL_Px0_IN_PUD

  Give GPIO PC6 configuration a friendly name.
- #define PWRUP_OUT_BUTTON1 1 /∗ Button needs a pullup ∗/

  Give GPIO PC6 configuration a friendly name.
- #define PWRDN_CFG_BUTTON1 _GPIO_P_CFGL_Px0_IN_PUD

  Give GPIO PC6 configuration a friendly name.
- #define PWRDN_OUT_BUTTON1 1 /∗ Button needs a pullup ∗/

  Give GPIO PC6 configuration a friendly name.
- #define CFG_TEMPEN _GPIO_P_CFGL_Px0_OUT

  Give GPIO PC7 configuration a friendly name.

TX_ACTIVE Configuration Defines

Provide the proper set of pin (PC5) configurations for when TX_ACTIVE is enabled (look above for the define which enables it). When TX_ACTIVE is not enabled, configure the pin for LED2.

- #define PWRUP_CFG_LED2 _GPIO_P_CFGL_Px0_OUT

  Give the TX_ACTIVE configuration a friendly name.
- #define PWRUP_OUT_LED2 1 /∗ LED default off ∗/

  Give the TX_ACTIVE configuration a friendly name.
- #define PWRDN_CFG_LED2 _GPIO_P_CFGL_Px0_OUT

  Give the TX_ACTIVE configuration a friendly name.
- #define PWRDN_OUT_LED2 1 /∗ LED default off ∗/

  Give the TX_ACTIVE configuration a friendly name.

USB Configuration Defines

Provide the proper set of pin configuration for when USB is not enumerated. Not enumerated primarily refers to the driver not being configured or deep sleep. The configuration used here is only for keeping the USB off the bus. The GPIO configuration used when active is controlled by the USB driver since the driver needs to control the enumeration process (which affects GPIO state.)

Note

: Using USB requires Serial port 3 to be defined and is only possible on EM3582/EM3586/EM3588/EM359 chips.

- #define PWRUP_CFG_USBDM _GPIO_P_CFGL_Px0_OUT_ALT
  Give the USB configuration a friendly name.
- #define PWRUP_OUT_USBDM 0
  Give the USB configuration a friendly name.
- #define PWRUP_CFG_USBDP _GPIO_P_CFGL_Px0_IN
  Give the USB configuration a friendly name.
- #define PWRUP_OUT_USBDP 0
  Give the USB configuration a friendly name.
- #define PWRUP_CFG_ENUMCTRL _GPIO_P_CFGL_Px0_OUT_ALT
  Give the USB configuration a friendly name.
- #define PWRUP_OUT_ENUMCTRL 0
  Give the USB configuration a friendly name.
- #define PWRUP_CFG_VBUSMON _GPIO_P_CFGL_Px0_OUT
  Give the USB configuration a friendly name.
- #define PWRUP_OUT_VBUSMON 1
  Give the USB configuration a friendly name.
- #define PWRDN_CFG_USBDM _GPIO_P_CFGL_Px0_IN_PUD
  Give the USB configuration a friendly name.
- #define PWRDN_OUT_USBDM 1
  Give the USB configuration a friendly name.
- #define PWRDN_CFG_USBDP _GPIO_P_CFGL_Px0_IN_PUD
  Give the USB configuration a friendly name.
- #define PWRDN_OUT_USBDP 1
  Give the USB configuration a friendly name.
- #define PWRDN_CFG_ENUMCTRL _GPIO_P_CFGL_Px0_IN_PUD
  Give the USB configuration a friendly name.
- #define PWRDN_OUT_ENUMCTRL 1
  Give the USB configuration a friendly name.
- #define PWRDN_CFG_VBUSMON _GPIO_P_CFGL_Px0_OUT
  Give the USB configuration a friendly name.
- #define PWRDN_OUT_VBUSMON 1
  Give the USB configuration a friendly name.

GPIO Wake Source Definitions

A convenient define that chooses if this external signal can be used as source to wake from deep sleep. Any change in the state of the signal will wake up the CPU.

- #define WAKE_ON_PA0 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PA1 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PA2 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PA3 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PA4 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PA5 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PA6 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PA7 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PB0 false

true if this GPIO can wake the chip from deep sleep, false if not.

- #define WAKE_ON_PB1 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PB2 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PB3 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PB4 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PB5 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PB6 true
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PB7 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PC0 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PC1 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PC2 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PC3 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PC4 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PC5 false
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PC6 true
  true if this GPIO can wake the chip from deep sleep, false if not.
- #define WAKE_ON_PC7 false
  true if this GPIO can wake the chip from deep sleep, false if not.

Board Specific Functions

The following macros exist to aid in the initialization, power up from sleep, and power down to sleep operations. These macros are responsible for either initializing directly, or calling initialization functions for any peripherals that are specific to this board implementation. These macros are called from halInit, halPowerDown, and halPowerUp respectively.

- #define halInternalInitBoard()
  Initialize the board. This function is called from halInit().
- #define halInternalPowerDownBoard()
  Power down the board. This function is called from halPowerDown().
- #define halInternalSuspendBoard()
  Suspend the board. This function is called from halSuspend().
- #define halInternalPowerUpBoard()
  Power up the board. This function is called from halPowerUp().
- #define halInternalResumeBoard()
  Resume the board. This function is called from halResume().

Enumerations

LED Definitions

The following are used to aid in the abstraction with the LED connections. The microcontroller-specific sources use these definitions so they are able to work across a variety of boards which could have different connections. The names and ports/pins used below are intended to match with a schematic of the system to provide the abstraction.

The HalBoardLedPins enum values should always be used when manipulating the state of LEDs, as they directly refer to the GPIOs to which the LEDs are connected.

Note: *LEDs 0 and 1 are on the RCM.*

Note: *LED 2 is on the breakout board (dev0680).*

Note: *LED 3 simply redirects to LED 2.*

- enum HalBoardLedPins {
  BOARDLED0 = PORTA_PIN(6),
  BOARDLED1 = PORTA_PIN(7),
  BOARDLED2 = PORTC_PIN(5),
  BOARDLED3 = BOARDLED2,
  BOARD_ACTIVITY_LED = BOARDLED0,
  BOARD_HEARTBEAT_LED = BOARDLED1 }

  Assign each GPIO with an LED connected to a convenient name. BOARD_ACTIVITY_LED and BOARD_HEARTBEAT_LED provide a further layer of abstraction on top of the 3 LEDs for verbose coding.

GPIO Configuration Macros

These macros define the GPIO configuration and initial state of the output registers for all the GPIO in the powerup and powerdown modes.

- #define DEFINE_GPIO_RADIO_POWER_BOARD_MASK_VARIABLE() GpioMaskType gpioRadioPowerBoardMask = 0

  Define the mask for GPIO relevant to the radio in the context of power state. Each bit in the mask indicates the corresponding GPIO which should be affected when invoking halStackRadioPowerUpBoard() or halStackRadioPowerDownBoard().
- #define DEFINE_POWERUP_GPIO_CFG_VARIABLES()

  Initialize GPIO powerup configuration variables.
- #define DEFINE_POWERUP_GPIO_OUTPUT_DATA_VARIABLES()

  Initialize GPIO powerup output variables.
- #define DEFINE_POWERDOWN_GPIO_CFG_VARIABLES()

  Initialize powerdown GPIO configuration variables.
- #define DEFINE_POWERDOWN_GPIO_OUTPUT_DATA_VARIABLES()

  Initialize powerdown GPIO output variables.
- #define SET_POWERUP_GPIO_CFG_REGISTERS()

  Set powerup GPIO configuration registers.
- #define SET_POWERUP_GPIO_OUTPUT_DATA_REGISTERS()

  Set powerup GPIO output registers.
- #define SET_POWERDOWN_GPIO_CFG_REGISTERS()

  Set powerdown GPIO configuration registers.
- #define SET_POWERDOWN_GPIO_OUTPUT_DATA_REGISTERS()

  Set powerdown GPIO output registers.
- #define SET_RESUME_GPIO_CFG_REGISTERS()

  Set resume GPIO configuration registers. Identical to SET_POWERUP.
- #define SET_RESUME_GPIO_OUTPUT_DATA_REGISTERS()

  Set resume GPIO output registers. Identical to SET_POWERUP.
- #define SET_SUSPEND_GPIO_CFG_REGISTERS()

  Set suspend GPIO configuration registers. SET_POWERDOWN minus USB regs.
- #define SET_SUSPEND_GPIO_OUTPUT_DATA_REGISTERS()

  Set suspend GPIO output registers. SET_POWERDOWN minus USB regs.

- #define CONFIGURE_EXTERNAL_REGULATOR_ENABLE() GPIO->DBGCFG &= ~GPIO_DBGCFG_E↩ XTREGEN;

  External regulator enable/disable macro.
- uint16_t gpioCfgPowerUp [6]

  Define the mask for GPIO relevant to the radio in the context of power state. Each bit in the mask indicates the corresponding GPIO which should be affected when invoking halStackRadioPowerUpBoard() or halStackRadio↩ PowerDownBoard().
- uint16_t gpioCfgPowerDown [6]

  Define the mask for GPIO relevant to the radio in the context of power state. Each bit in the mask indicates the corresponding GPIO which should be affected when invoking halStackRadioPowerUpBoard() or halStackRadio↩ PowerDownBoard().
- uint8_t gpioOutPowerUp [3]

  Define the mask for GPIO relevant to the radio in the context of power state. Each bit in the mask indicates the corresponding GPIO which should be affected when invoking halStackRadioPowerUpBoard() or halStackRadio↩ PowerDownBoard().
- uint8_t gpioOutPowerDown [3]

  Define the mask for GPIO relevant to the radio in the context of power state. Each bit in the mask indicates the corresponding GPIO which should be affected when invoking halStackRadioPowerUpBoard() or halStackRadio↩ PowerDownBoard().
- GpioMaskType gpioRadioPowerBoardMask

  Define the mask for GPIO relevant to the radio in the context of power state. Each bit in the mask indicates the corresponding GPIO which should be affected when invoking halStackRadioPowerUpBoard() or halStackRadio↩ PowerDownBoard().

8.21.1 Detailed Description

See Sample Breakout Board Configuration for detailed documentation.

8.21.2 Macro Definition Documentation

8.21.2.1 #define halInternalInitBoard( )

Value:

```
do {                              \
    halInternalPowerUpBoard();     \
    halInternalInitRadioHoldOff(); \
    halInternalInitButton();       \
  } while (0)
```

8.21.2.2 #define halInternalPowerDownBoard( )

Value:

```
do {                                      \
    /* Board peripheral deactivation */    \
    /* halInternalSleepAdc(); */           \
    SET_POWERDOWN_GPIO_OUTPUT_DATA_REGISTERS()
      \
    SET_POWERDOWN_GPIO_CFG_REGISTERS()     \
  } while (0)
```

8.21.2.3 #define halInternalPowerUpBoard( )

Value:

```
do {                                                    \
    SET_POWERUP_GPIO_OUTPUT_DATA_REGISTERS()
      \
    SET_POWERUP_GPIO_CFG_REGISTERS()                    \
    /*The radio GPIO should remain in the powerdown state */ \
    /*until the stack specifically powers them up. */   \
    halStackRadioPowerDownBoard();
      \
    CONFIGURE_EXTERNAL_REGULATOR_ENABLE()               \
    /* Board peripheral reactivation */                 \
    halInternalInitAdc();                               \
  } while (0)
```

8.21.2.4 #define halInternalResumeBoard( )

Value:

```
do {                                                    \
    SET_RESUME_GPIO_OUTPUT_DATA_REGISTERS()
      \
    SET_RESUME_GPIO_CFG_REGISTERS()                     \
    /*The radio GPIO should remain in the powerdown state */ \
    /*until the stack specifically powers them up. */   \
    halStackRadioPowerDownBoard();
      \
    CONFIGURE_EXTERNAL_REGULATOR_ENABLE()               \
    /* Board peripheral reactivation */                 \
    halInternalInitAdc();                               \
  } while (0)
```

8.21.2.5 #define halInternalSuspendBoard( )

Value:

```
do {                                    \
    /* Board peripheral deactivation */     \
    /* halInternalSleepAdc(); */            \
    SET_SUSPEND_GPIO_OUTPUT_DATA_REGISTERS()
      \
    SET_SUSPEND_GPIO_CFG_REGISTERS()        \
  } while (0)
```

8.22 diagnostic.h File Reference

Macros

- #define halResetWasCrash() (((1 << halGetResetInfo()) & RESET_CRASH_REASON_MASK) != 0U)

  Macro evaluating to true if the last reset was a crash, false otherwise.

Functions

- uint32_t halGetMainStackBytesUsed (void)

  Returns the number of bytes used in the main stack.
- void halPrintCrashSummary (uint8_t port)

  Print a summary of crash details.
- void halPrintCrashDetails (uint8_t port)

  Print the complete, decoded crash details.
- void halPrintCrashData (uint8_t port)

  Print the complete crash data.
- const HalAssertInfoType * halGetAssertInfo (void)

  If last reset was from an assert, return saved assert information.

8.22.1 Detailed Description

See Crash and Watchdog Diagnostics for detailed documentation.

8.23 dtls.h File Reference

DTLS API for dotdot.

Macros

- #define EMBER_DTLS_MODE_CERT 0x01

  Define the various modes of a DTLS connection.
- #define EMBER_DTLS_MODE_PSK 0x02
- #define EMBER_DTLS_MODE_PKEY 0x04

Typedefs

- typedef uint8_t EmberDtlsMode

Functions

- void emberSetDtlsDeviceCertificate (const CertificateAuthority **certAuthority, const DeviceCertificate *deviceCert)

  Set a device certificate to be used to create a certificate based secure session on the application. The expected arguments are DER encoded X.509 certificates. If this succeeds, emberSetDtlsDeviceCertificateReturn should return 0.
- void emberSetDtlsDeviceCertificateReturn (uint32_t result)

  Provides the result of a call to emberSetDtlsDeviceCertificate().
- void emberSetDtlsPresharedKey (const uint8_t *key, uint8_t keyLength, const EmberIpv6Address *remote↩ Address)

  Set a key to be used to create a PSK based secure session on the application. The maximum length of the key is 32 bytes.
- void emberSetDtlsPresharedKeyReturn (EmberStatus status)

  Provides the result of a call to emberSetDtlsPresharedKey().

- void emberOpenDtlsConnection (EmberDtlsMode dtlsMode, const EmberIpv6Address *remoteAddress, uint16_t localPort, uint16_t remotePort)

  Establish a DTLS connection with a peer on the Thread network. When established, this session can be used to send secure CoAP data. The device requesting the connection acts as a DTLS client.
- void emberOpenDtlsConnectionReturn (uint32_t result, const EmberIpv6Address *remoteAddress, uint16_t localPort, uint16_t remotePort)

  Provides the result of a call to emberOpenDtlsConnection().
- void emberDtlsSecureSessionEstablished (uint8_t flags, uint8_t sessionId, const EmberIpv6Address *localAddress, const EmberIpv6Address *remoteAddress, uint16_t localPort, uint16_t remotePort)

  Indicates to the application that a secure connection was successfully established.
- void emberGetSecureDtlsSessionId (const EmberIpv6Address *remoteAddress, uint16_t localPort, uint16_t remotePort)

  Request the session ID given connection parameters.
- void emberGetSecureDtlsSessionIdReturn (uint8_t sessionId, const EmberIpv6Address *remoteAddress, uint16_t localPort, uint16_t remotePort)

  Provides the result of a call to emberGetSecureDtlsSessionId().
- void emberCloseDtlsConnection (uint8_t sessionId)

  Close a currently active secure session on the application. When successful, emberCloseDtlsConnectionReturn should be called on both ends of the connection with EMBER_SUCCESS.
- void emberCloseDtlsConnectionReturn (uint8_t sessionId, EmberStatus status)

  Provides the result of a call to emberCloseDtlsConnection(), or indicates that the connection was closed on the other end.
- bool emberDtlsTransmitHandler (const uint8_t *payload, uint16_t payloadLength, const EmberIpv6Address *localAddress, uint16_t localPort, const EmberIpv6Address *remoteAddress, uint16_t remotePort, void *transmitHandlerData)

  Public DTLS transmit handler to be set in emberCoapSend. The secure payload is delivered via emberProcessCoap on the other end, with a matching session ID in the transmitHandlerData of its CoapRequestInfo. See emberProcessCoap (stack/include/coap.h)

8.23.1 Detailed Description

Note: If using an mbed TLS library, your stack must define EMBER_MBEDTLS_STACK.

8.24 em_usb.h File Reference

USB protocol stack library API for EFM32.

```
#include <string.h>
#include <stddef.h>
#include <uchar.h>
```

Data Structures

- struct USB_Setup_TypeDef

  USB Setup request package.
- struct USB_DeviceDescriptor_TypeDef

  USB Device Descriptor.
- struct USB_ConfigurationDescriptor_TypeDef

  USB Configuration Descriptor.

- struct USB_InterfaceDescriptor_TypeDef

  USB Interface Descriptor.
- struct USB_EndpointDescriptor_TypeDef

  USB Endpoint Descriptor.
- struct USB_StringDescriptor_TypeDef

  USB String Descriptor.
- struct USBD_Init_TypeDef

  USB Device stack initialization structure.
- struct USBD_Callbacks_TypeDef

  USB Device stack callback structure.

Macros

- #define USB_SETUP_DIR_OUT 0
- #define USB_SETUP_DIR_IN 1
- #define USB_SETUP_DIR_MASK 0x80
- #define USB_SETUP_DIR_D2H 0x80
- #define USB_SETUP_DIR_H2D 0x00
- #define USB_SETUP_TYPE_STANDARD 0
- #define USB_SETUP_TYPE_CLASS 1
- #define USB_SETUP_TYPE_VENDOR 2
- #define USB_SETUP_TYPE_STANDARD_MASK 0x00
- #define USB_SETUP_TYPE_CLASS_MASK 0x20
- #define USB_SETUP_TYPE_VENDOR_MASK 0x40
- #define USB_SETUP_RECIPIENT_DEVICE 0
- #define USB_SETUP_RECIPIENT_INTERFACE 1
- #define USB_SETUP_RECIPIENT_ENDPOINT 2
- #define USB_SETUP_RECIPIENT_OTHER 3
- #define GET_STATUS 0
- #define CLEAR_FEATURE 1
- #define SET_FEATURE 3
- #define SET_ADDRESS 5
- #define GET_DESCRIPTOR 6
- #define SET_DESCRIPTOR 7
- #define GET_CONFIGURATION 8
- #define SET_CONFIGURATION 9
- #define GET_INTERFACE 10
- #define SET_INTERFACE 11
- #define SYNCH_FRAME 12
- #define USB_HID_GET_REPORT 0x01
- #define USB_HID_GET_IDLE 0x02
- #define USB_HID_SET_REPORT 0x09
- #define USB_HID_SET_IDLE 0x0A
- #define USB_HID_SET_PROTOCOL 0x0B
- #define USB_CDC_SETLINECODING 0x20
- #define USB_CDC_GETLINECODING 0x21
- #define USB_CDC_SETCTRLLINESTATE 0x22
- #define USB_MSD_BOTRESET 0xFF
- #define USB_MSD_GETMAXLUN 0xFE
- #define USB_DEVICE_DESCRIPTOR 1
- #define USB_CONFIG_DESCRIPTOR 2
- #define USB_STRING_DESCRIPTOR 3
- #define USB_INTERFACE_DESCRIPTOR 4

- #define USB_ENDPOINT_DESCRIPTOR 5
- #define USB_DEVICE_QUALIFIER_DESCRIPTOR 6
- #define USB_OTHER_SPEED_CONFIG_DESCRIPTOR 7
- #define USB_INTERFACE_POWER_DESCRIPTOR 8
- #define USB_HUB_DESCRIPTOR 0x29
- #define USB_HID_DESCRIPTOR 0x21
- #define USB_HID_REPORT_DESCRIPTOR 0x22
- #define USB_CS_INTERFACE_DESCRIPTOR 0x24
- #define USB_DEVICE_DESCSIZE 18
- #define USB_CONFIG_DESCSIZE 9
- #define USB_INTERFACE_DESCSIZE 9
- #define USB_ENDPOINT_DESCSIZE 7
- #define USB_DEVICE_QUALIFIER_DESCSIZE 10
- #define USB_OTHER_SPEED_CONFIG_DESCSIZE 9
- #define USB_HID_DESCSIZE 9
- #define USB_CDC_HEADER_FND_DESCSIZE 5
- #define USB_CDC_CALLMNG_FND_DESCSIZE 5
- #define USB_CDC_ACM_FND_DESCSIZE 4
- #define USB_EP0_SIZE 8
- #define USB_EP1_SIZE 8
- #define USB_EP2_SIZE 8
- #define USB_EP3_SIZE 64
- #define USB_EP4_SIZE 32
- #define USB_EP5_SIZE 64
- #define USB_EP6_SIZE 512
- #define USB_MAX_EP_SIZE 64
- #define USB_EPTYPE_CTRL 0
- #define USB_EPTYPE_ISOC 1
- #define USB_EPTYPE_BULK 2
- #define USB_EPTYPE_INTR 3
- #define USB_EP_DIR_IN 0x80
- #define USB_SETUP_PKT_SIZE 8
- #define USB_EPNUM_MASK 0x0F
- #define USB_LANGID_ENUS 0x0409
- #define USB_MAX_DEVICE_ADDRESS 127
- #define CONFIG_DESC_BM_REMOTEWAKEUP 0x20
- #define CONFIG_DESC_BM_SELFPOWERED 0x40
- #define CONFIG_DESC_BM_RESERVED_D7 0x80
- #define CONFIG_DESC_BM_TRANSFERTYPE 0x03
- #define CONFIG_DESC_MAXPOWER_mA(x) (((x) + 1) / 2)
- #define DEVICE_IS_SELFPOWERED 0x0001
- #define REMOTE_WAKEUP_ENABLED 0x0002
- #define USB_FEATURE_ENDPOINT_HALT 0
- #define USB_FEATURE_DEVICE_REMOTE_WAKEUP 1
- #define HUB_FEATURE_PORT_RESET 4
- #define HUB_FEATURE_PORT_POWER 8
- #define HUB_FEATURE_C_PORT_CONNECTION 16
- #define HUB_FEATURE_C_PORT_RESET 20
- #define HUB_FEATURE_PORT_INDICATOR 22
- #define USB_CLASS_CDC 2
- #define USB_CLASS_CDC_DATA 0x0A
- #define USB_CLASS_CDC_ACM 2
- #define USB_CLASS_CDC_HFN 0
- #define USB_CLASS_CDC_CMNGFN 1
- #define USB_CLASS_CDC_ACMFN 2

- #define USB_CLASS_CDC_UNIONFN 6
- #define USB_CLASS_HID 3
- #define USB_CLASS_HID_KEYBOARD 1
- #define USB_CLASS_HID_MOUSE 2
- #define USB_CLASS_HUB 9
- #define USB_CLASS_MSD 8
- #define USB_CLASS_MSD_BOT_TRANSPORT 0x50
- #define USB_CLASS_MSD_SCSI_CMDSET 6
- #define USB_CLASS_MSD_CSW_CMDPASSED 0
- #define USB_CLASS_MSD_CSW_CMDFAILED 1
- #define USB_CLASS_MSD_CSW_PHASEERROR 2
- #define PORT_FULL_SPEED 1
- #define PORT_LOW_SPEED 2
- #define nibble2Ascii(n) ((n) + (((n) < 10) ? '0' : 'A' - 10));
- #define STATIC_CONST_STRING_DESC(_name, ...)
- #define STATIC_CONST_STRING_DESC_LANGID(_name, x, y)
- #define UBUF(x, y) EFM32_ALIGN(4) uint8_t x[((y) + 3) & ~3]
- #define STATIC_UBUF(x, y) EFM32_ALIGN(4) static uint8_t x[((y) + 3) & ~3]
- #define NUM_QTIMERS 0

Typedefs

- typedef int(* USB_XferCompleteCb_TypeDef) (USB_Status_TypeDef status, uint32_t xferred, uint32_t remaining)

  USB transfer callback function.
- typedef void(* USBTIMER_Callback_TypeDef) (void)

  USBTIMER callback function.
- typedef void(* USBD_UsbResetCb_TypeDef) (void)

  USB Reset callback function.
- typedef void(* USBD_SofIntCb_TypeDef) (uint16_t sofNr)

  USB Start Of Frame (SOF) interrupt callback function.
- typedef void(* USBD_DeviceStateChangeCb_TypeDef) (USBD_State_TypeDef oldState, USBD_State_TypeDef newState)

  USB State change callback function.
- typedef bool(* USBD_IsSelfPoweredCb_TypeDef) (void)

  USB power mode callback function.
- typedef int(* USBD_SetupCmdCb_TypeDef) (const USB_Setup_TypeDef *setup)

  USB setup request callback function.
- typedef struct USBD_Callbacks_TypeDef USBD_Callbacks_TypeDef

  USB Device stack callback structure.

Enumerations

- enum USB_Status_TypeDef {
  USB_STATUS_OK = 0,
  USB_STATUS_REQ_ERR = -1,
  USB_STATUS_EP_BUSY = -2,
  USB_STATUS_REQ_UNHANDLED = -3,
  USB_STATUS_ILLEGAL = -4,
  USB_STATUS_EP_STALLED = -5,
  USB_STATUS_EP_ABORTED = -6,
  USB_STATUS_EP_ERROR = -7,
  USB_STATUS_EP_NAK = -8,
  USB_STATUS_DEVICE_UNCONFIGURED = -9,
  USB_STATUS_DEVICE_SUSPENDED = -10,
  USB_STATUS_DEVICE_RESET = -11,
  USB_STATUS_TIMEOUT = -12,
  USB_STATUS_DEVICE_REMOVED = -13,
  USB_STATUS_HC_BUSY = -14,
  USB_STATUS_DEVICE_MALFUNCTION = -15,
  USB_STATUS_PORT_OVERCURRENT = -16 }

  USB transfer status enumerator.
- enum USBD_State_TypeDef {
  USBD_STATE_NONE = 0,
  USBD_STATE_ATTACHED = 1,
  USBD_STATE_POWERED = 2,
  USBD_STATE_DEFAULT = 3,
  USBD_STATE_ADDRESSED = 4,
  USBD_STATE_CONFIGURED = 4,
  USBD_STATE_SUSPENDED = 6,
  USBD_STATE_LASTMARKER = 7 }

  USB device state enumerator.

Functions

- void USBTIMER_DelayMs (uint32_t msec)
- void USBTIMER_DelayUs (uint32_t usec)
- void USBTIMER_Init (void)
- void USBTIMER_Start (uint32_t id, uint32_t timeout, USBTIMER_Callback_TypeDef callback)
- void USBTIMER_Stop (uint32_t id)
- void USBD_AbortAllTransfers (void)

  Abort all pending transfers.
- int USBD_AbortTransfer (int epAddr)

  Abort a pending transfer on a specific endpoint.
- void USBD_Connect (void)

  Start USB device operation.
- void USBD_Disconnect (void)

  Stop USB device operation.
- bool USBD_EpIsBusy (int epAddr)

  Check if an endpoint is busy doing a transfer.
- USBD_State_TypeDef USBD_GetUsbState (void)

  Get current USB device state.
- const char * USBD_GetUsbStateName (USBD_State_TypeDef state)

  Get a string naming a device USB state.
- int USBD_Init (const USBD_Init_TypeDef *p)

Initializes USB device hardware and internal protocol stack data structures, then connects the data-line (D+ or D-) pullup resistor to signal host that enumeration can begin.

- int USBD_Read (int epAddr, void *data, int byteCount, USB_XferCompleteCb_TypeDef callback)

  Start a read (OUT) transfer on an endpoint.
- int USBD_RemoteWakeup (void)

  Perform a remote wakeup signalling sequence.
- bool USBD_SafeToEnterEM2 (void)
- int USBD_StallEp (int epAddr)

  Set an endpoint in the stalled (halted) state.
- void USBD_Stop (void)

  Stop USB device stack operation.
- int USBD_UnStallEp (int epAddr)

  Reset stall state on a stalled (halted) endpoint.
- int USBD_Write (int epAddr, void *data, int byteCount, USB_XferCompleteCb_TypeDef callback)

  Start a write (IN) transfer on an endpoint.
- void usbSuspendDsr (void)

  USB suspend delayed service routine.
- void usbTxData (void)
- void usbForceTxData (uint8_t *data, uint8_t length)
- void halInternalUart3RxIsr (uint8_t *rxData, uint8_t length, bool *pauseRx)

8.24.1 Detailed Description

Author

Nathaniel Ting

Version

3.20.3

8.24.2 Macro Definition Documentation

8.24.2.1 #define NUM_QTIMERS 0

8.24.3 Function Documentation

8.24.3.1 void halInternalUart3RxIsr (uint8_t * *rxData,* uint8_t *length,* bool * *pauseRx*)

8.24.3.2 void usbForceTxData (uint8_t * *data,* uint8_t *length*)

Referenced by USBD_RemoteWakeup().

8.24.3.3 void usbTxData (void)

Referenced by USBD_RemoteWakeup().

8.25 em_usbd.c File Reference

USB protocol stack library, device API.

```
#include <PLATFORM_HEADER>
#include "stack/include/ember.h"
#include "hal/hal.h"
#include "em_usb.h"
#include "em_usbhal.h"
#include "em_usbtypes.h"
#include "em_usbd.h"
#include "serial/serial.h"
```

Functions

- void USBD_AbortAllTransfers (void)

  Abort all pending transfers.
- int USBD_AbortTransfer (int epAddr)

  Abort a pending transfer on a specific endpoint.
- void USBD_Connect (void)

  Start USB device operation.
- void USBD_Disconnect (void)

  Stop USB device operation.
- USBD_State_TypeDef USBD_GetUsbState (void)

  Get current USB device state.
- const char * USBD_GetUsbStateName (USBD_State_TypeDef state)

  Get a string naming a device USB state.
- bool USBD_EpIsBusy (int epAddr)

  Check if an endpoint is busy doing a transfer.
- int USBD_StallEp (int epAddr)

  Set an endpoint in the stalled (halted) state.
- int USBD_UnStallEp (int epAddr)

  Reset stall state on a stalled (halted) endpoint.
- void USBD_Stop (void)

  Stop USB device stack operation.
- int USBD_Init (const USBD_Init_TypeDef *p)

  Initializes USB device hardware and internal protocol stack data structures, then connects the data-line (D+ or D-) pullup resistor to signal host that enumeration can begin.
- int USBD_Write (int epAddr, void *data, int byteCount, USB_XferCompleteCb_TypeDef callback)

  Start a write (IN) transfer on an endpoint.
- int USBD_Read (int epAddr, void *data, int byteCount, USB_XferCompleteCb_TypeDef callback)

  Start a read (OUT) transfer on an endpoint.
- void usbSuspendDsr (void)

  USB suspend delayed service routine.
- int USBD_RemoteWakeup (void)

  Perform a remote wakeup signalling sequence.

8.25.1 Detailed Description

Author

Nathaniel Ting

Version

3.20.3

8.26 em_usbd.h File Reference

USB protocol stack library API for EFM32.

```
#include "em_usb.h"
#include <PLATFORM_HEADER>
#include "stack/include/ember.h"
#include "hal/hal.h"
#include "em_usbhal.h"
```

8.26.1 Detailed Description

Author

Nathaniel Ting

Version

3.20.3

8.27 em_usbdch9.c File Reference

USB protocol stack library, USB chapter 9 command handler.

```
#include <PLATFORM_HEADER>
#include "stack/include/ember.h"
#include "hal/hal.h"
#include "em_usb.h"
#include "em_usbhal.h"
#include "em_usbtypes.h"
#include "em_usbd.h"
```

8.27.1 Detailed Description

Author

Nathaniel Ting

Version

3.20.3

8.28 em_usbdep.c File Reference

USB protocol stack library, USB device endpoint handlers.

```
#include <PLATFORM_HEADER>
#include "stack/include/ember.h"
#include "hal/hal.h"
#include "em_usb.h"
#include "em_usbhal.h"
#include "em_usbtypes.h"
#include "em_usbd.h"
```

8.28.1 Detailed Description

Author

Nathaniel Ting

Version

3.20.3

8.29 em_usbhal.c File Reference

USB protocol stack library, low level USB peripheral access.

```
#include <PLATFORM_HEADER>
#include "stack/include/ember.h"
#include "hal/hal.h"
#include "em_usb.h"
#include "em_usbtypes.h"
#include "em_usbhal.h"
```

8.29.1 Detailed Description

Author

Nathaniel Ting

Version

3.20.3

8.30 em_usbhal.h File Reference

USB protocol stack library, low level USB peripheral access.

```
#include "em_usbtypes.h"
```

8.30.1 Detailed Description

Author

Nathaniel Ting

Version

3.20.3

8.31 em_usbint.c File Reference

USB protocol stack library, USB device peripheral interrupt handlers.

```
#include <PLATFORM_HEADER>
#include "stack/include/ember.h"
#include "hal/hal.h"
#include "em_usb.h"
#include "em_usbhal.h"
#include "em_usbtypes.h"
#include "em_usbd.h"
#include "serial/serial.h"
```

8.31.1 Detailed Description

Author

Nathaniel Ting

Version

3.20.3

8.32 em_usbtypes.h File Reference

USB protocol stack library, internal type definitions.

8.32.1 Detailed Description

Author

Nathaniel Ting

Version

3.20.3

8.33 ember-configuration-defaults.h File Reference

User-configurable stack memory allocation defaults.

Macros

- #define EMBER_VERSION_NAME "Thread"

  If the application defined a configuration file, include it.
- #define EMBER_HEAP_SIZE 6000

  The minimum heap size allocated for an application.
- #define EMBER_MALLOC_HEAP_SIZE_BYTES 32768

  The default amount of heap allocated for the mbedtls malloc library, if in use.
- #define EMBER_ASSERT_SERIAL_PORT 1

  Settings to control if and where assert information will be printed.
- #define EMBER_INDIRECT_TRANSMISSION_TIMEOUT 30

  The maximum amount of time (in quarter seconds) that the MAC will hold a message for indirect transmission to a child.
- #define EMBER_CHILD_TABLE_SIZE 16

  The size of the child table. This include sleepy and powered end device children, as well as router eligible end devices.
- #define EMBER_RETRY_QUEUE_SIZE 8
- #define EMBER_SECURITY_LEVEL 5

  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).
- #define EMBER_SECURITY_TO_HOST false
- #define EMBER_TASK_COUNT (3)

  The number of event tasks that can be tracked for the purpose of processor idling. The Thread stack requires 1, an application and associated libraries may use additional tasks, though typically no more than 3 are needed for most applications.
- #define EMBER_SLEEPY_CHILD_POLL_TIMEOUT 240

  The number of seconds after which the parent will time an EMBER_SLEEPY_END_DEVICE out of its table if it has not heard a data poll from it.
- #define EMBER_END_DEVICE_POLL_TIMEOUT 240

  The maximum amount of time that an EMBER_END_DEVICE can wait between polls.
- #define EMBER_MFG_RX_NCP_TO_HOST_INTERVAL 50

  The number of packets received by an NCP before it decides to send aggregated packet information to the host when running an mfg send test.
- #define EMBER_USE_DIRECT_IP_CALLBACK false
- #define RIP_MAX_LURKERS 0

8.33.1 Detailed Description

Note

Application developers should **not** modify any portion of this file. Doing so may cause mysterious bugs. Allocations should be adjusted only by defining the appropriate macros in the application's CONFIGURATI↩ ON_HEADER.

See configuration for documentation.

8.34 ember-debug.h File Reference

Macros

- #define NO_DEBUG 0
- #define BASIC_DEBUG 1
- #define FULL_DEBUG 2
- #define emberDebugInit(port) do {} while (false)

  This function is obsolete and no longer required to initialize the debug system.

Functions

- void emberDebugAssert (PGM_P filename, int linenumber)

  Prints the filename and line number to the debug serial port.
- void emberDebugMemoryDump (uint8_t *start, uint8_t *end)

  Prints the contents of RAM to the debug serial port.
- void emberDebugBinaryPrintf (PGM_P formatString,...)

  Prints binary data to the debug channel.
- void emDebugSendVuartMessage (const uint8_t *buff, uint8_t len)

  An internal debug command used by the HAL to send vuart data out the the debug channel.
- void emberDebugError (EmberStatus code)

  Prints an EmberStatus return code to the serial port.
- bool emberDebugReportOff (void)

  Turns off all debug output.
- void emberDebugReportRestore (bool state)

  Restores the state of the debug output.
- void emberDebugPrintf (PGM_P formatString,...)

  Prints text debug messages.

8.34.1 Detailed Description

See Debugging Utilities for documentation.

8.35 ember-types.h File Reference

Ember data type definitions.

```
#include "stack/config/ember-configuration-defaults.h"
#include "stack/include/error.h"
```

Data Structures

- struct EmberEui64

  EUI 64-bit ID (an IEEE address).
- struct EmberIpv6Prefix

  An IPv6 Prefix structure.
- struct EmberIpv6Address

  An IPv6 Address structure.
- struct EmberKeyData

  This data structure contains the key data that is passed into various other functions.
- struct EmberVersion

  For use when declaring data that holds the Ember software version type.
- struct Ipv6Header

  A structure that holds an IPv6 header. All values are in their local byte order (as opposed to network byte order, which might be different).
- struct TlsSessionState

  Defines a TLS session state.
- struct Bytes8

  Defines a data type of size 8 bytes.
- struct Bytes16

  Defines a data type of size 16 bytes.
- struct CertificateAuthority

  Defines a certificate authority structure.
- struct DeviceCertificate

  Defines a device certificate structure.
- struct EventActions_s

  The static part of an event. Each event can be used with only one event queue.
- struct Event_s
- struct EventQueue_s

  An event queue is currently a list of events ordered by execution time.
- struct EmberEventControl

  Control structure for events.
- struct EmberTaskControl

  Control structure for tasks.

Macros

- #define INT16U_MAX ((uint16_t)(~(uint16_t)0))

  Defines the maximum value of an unsigned short data type.
- #define DEFAULT_SCAN_DURATION 5

  Default scan duration for an energy or active scan.
- #define EMBER_COUNTER_STRINGS

  Defines the CLI enumerations for the EmberCounterType enum.

Broadcast Addresses

Broadcasts are normally sent only to routers. Broadcasts can also be forwarded to end devices, either all of them or only those that do not sleep. Broadcasting to end devices is both significantly more resource-intensive and significantly less reliable than broadcasting to routers.

- #define EMBER_BROADCAST_ADDRESS 0xFFFC
- #define EMBER_RX_ON_WHEN_IDLE_BROADCAST_ADDRESS 0xFFFD

- #define EMBER_SLEEPY_BROADCAST_ADDRESS 0xFFFF

txPowerModes for emberSetTxPowerMode and mfglibSetPower

- #define EMBER_TX_POWER_MODE_DEFAULT 0x0000

  The application should call emberSetTxPowerMode() with the txPowerMode parameter set to this value to disable all power mode options resulting in normal power mode and bi-directional RF transmitter output.
- #define EMBER_TX_POWER_MODE_BOOST 0x0001

  The application should call emberSetTxPowerMode() with the txPowerMode parameter set to this value to enable boost power mode.
- #define EMBER_TX_POWER_MODE_ALTERNATE 0x0002

  The application should call emberSetTxPowerMode() with the txPowerMode parameter set to this value to enable the alternate transmitter output.
- #define EMBER_TX_POWER_MODE_BOOST_AND_ALTERNATE

  The application should call emberSetTxPowerMode() with the txPowerMode parameter set to this value to enable both boost mode and the alternate transmitter output.

Typedefs

- typedef uint8_t EmberTaskId
- typedef const struct EventActions_s EventActions

  The static part of an event. Each event can be used with only one event queue.
- typedef struct Event_s Event
- typedef struct EventQueue_s EventQueue

  An event queue is currently a list of events ordered by execution time.
- struct {
  EmberEventControl * control
  void(* handler)(void)
  } EmberEventData

  Complete events with a control and a handler procedure.

Enumerations

- enum EmberNodeType {
  EMBER_UNKNOWN_DEVICE = 0,
  EMBER_ROUTER = 2,
  EMBER_END_DEVICE = 3,
  EMBER_SLEEPY_END_DEVICE = 4,
  EMBER_MINIMAL_END_DEVICE = 5,
  EMBER_COMMISSIONER = 7 }
- enum EmberNetworkStatus {
  EMBER_NO_NETWORK,
  EMBER_SAVED_NETWORK,
  EMBER_JOINING_NETWORK,
  EMBER_JOINED_NETWORK_ATTACHED,
  EMBER_JOINED_NETWORK_NO_PARENT,
  EMBER_JOINED_NETWORK_ATTACHING }

  Defines the possible join states for a node.
- enum EmberJoinFailureReason {
  EMBER_JOIN_FAILURE_REASON_NONE,
  EMBER_JOIN_FAILURE_REASON_FORM_SCAN,
  EMBER_JOIN_FAILURE_REASON_ACTIVE_SCAN,
  EMBER_JOIN_FAILURE_REASON_COMMISSIONING,
  EMBER_JOIN_FAILURE_REASON_SECURITY }

Defines the reason why a network status change occurred.

- enum EmberNetworkScanType {
  EMBER_ENERGY_SCAN,
  EMBER_ACTIVE_SCAN }

Type for a network scan.

- enum EmberEventUnits {
  EMBER_EVENT_INACTIVE = 0,
  EMBER_EVENT_MS_TIME,
  EMBER_EVENT_QS_TIME,
  EMBER_EVENT_MINUTE_TIME,
  EMBER_EVENT_ZERO_DELAY }

Either marks an event as inactive or specifies the units for the event execution time.

- enum EmberCounterType {

EMBER_COUNTER_PHY_IN_PACKETS,
EMBER_COUNTER_PHY_OUT_PACKETS,
EMBER_COUNTER_PHY_IN_OCTETS,
EMBER_COUNTER_PHY_OUT_OCTETS,
EMBER_COUNTER_MAC_IN_UNICAST,
EMBER_COUNTER_MAC_IN_BROADCAST,
EMBER_COUNTER_MAC_OUT_UNICAST_SUCCESS,
EMBER_COUNTER_MAC_OUT_UNICAST_ACK_FAIL,
EMBER_COUNTER_MAC_OUT_UNICAST_CCA_FAIL,
EMBER_COUNTER_MAC_OUT_UNICAST_EXT_FAIL,
EMBER_COUNTER_MAC_OUT_UNICAST_RETRY,
EMBER_COUNTER_MAC_OUT_BROADCAST,
EMBER_COUNTER_MAC_OUT_BROADCAST_CCA_FAIL,
EMBER_COUNTER_MAC_DROP_IN_MEMORY,
EMBER_COUNTER_MAC_DROP_IN_NO_EUI,
EMBER_COUNTER_MAC_DROP_IN_FRAME_COUNTER,
EMBER_COUNTER_MAC_DROP_IN_DECRYPT,
EMBER_COUNTER_MAC_DROP_IN_DUPLICATE,
EMBER_COUNTER_IP_IN_UNICAST,
EMBER_COUNTER_IP_OUT_UNICAST,
EMBER_COUNTER_IP_IN_MULTICAST,
EMBER_COUNTER_IP_OUT_MULTICAST,
EMBER_COUNTER_UDP_IN,
EMBER_COUNTER_UDP_OUT,
EMBER_COUNTER_UART_IN_DATA,
EMBER_COUNTER_UART_IN_MANAGEMENT,
EMBER_COUNTER_UART_IN_FAIL,
EMBER_COUNTER_UART_OUT_DATA,
EMBER_COUNTER_UART_OUT_MANAGEMENT,
EMBER_COUNTER_UART_OUT_FAIL,
EMBER_COUNTER_ROUTE_2_HOP_LOOP,
EMBER_COUNTER_BUFFER_ALLOCATION_FAIL,
EMBER_ASH_V3_ACK_SENT,
EMBER_ASH_V3_ACK_RECEIVED,
EMBER_ASH_V3_NACK_SENT,
EMBER_ASH_V3_NACK_RECEIVED,
EMBER_ASH_V3_RESEND,
EMBER_ASH_V3_BYTES_SENT,
EMBER_ASH_V3_TOTAL_BYTES_RECEIVED,
EMBER_ASH_V3_VALID_BYTES_RECEIVED,
EMBER_ASH_V3_PAYLOAD_BYTES_SENT,
EMBER_COUNTER_PTA_LO_PRI_REQUESTED,
EMBER_COUNTER_PTA_HI_PRI_REQUESTED,
EMBER_COUNTER_PTA_LO_PRI_DENIED,
EMBER_COUNTER_PTA_HI_PRI_DENIED,
EMBER_COUNTER_PTA_LO_PRI_TX_ABORTED,
EMBER_COUNTER_PTA_HI_PRI_TX_ABORTED,
EMBER_COUNTER_TYPE_COUNT,
EMBER_COUNTER_ALL = 0xFF }

Defines the events reported to the application by the emberCounterHandler().

Miscellaneous Ember Types

- #define EUI64_SIZE 8

  Size of EUI64 (an IEEE address) in bytes (8).
- #define EMBER_ENCRYPTION_KEY_SIZE 16

Size of an encryption key in bytes (16).

- #define EXTENDED_PAN_ID_SIZE 8

  Size of an extended PAN identifier in bytes (8).

- #define LEADER_SIZE EUI64_SIZE

  Size of a leader EUI64 in bytes (8).

- #define EMBER_NETWORK_ID_SIZE 16

  Size of a network ID in bytes (16).

- #define EMBER_JOIN_KEY_MAX_SIZE 32

  Maximum size of a device join key (PSKd) in bytes (32).

- #define __EMBERSTATUS_TYPE__

  Return type for Ember functions.

- #define EMBER_MAX_802_15_4_CHANNEL_NUMBER 26

  The maximum 802.15.4 channel number is 26.

- #define EMBER_MIN_802_15_4_CHANNEL_NUMBER 11

  The minimum 802.15.4 channel number is 11.

- #define EMBER_NUM_802_15_4_CHANNELS (EMBER_MAX_802_15_4_CHANNEL_NUMBER - EMBE↩ R_MIN_802_15_4_CHANNEL_NUMBER + 1)

  There are sixteen 802.15.4 channels.

- #define EMBER_ALL_802_15_4_CHANNELS_MASK 0x07FFF800UL

  Bitmask to scan all 802.15.4 channels.

- #define EMBER_ZIGBEE_COORDINATOR_ADDRESS 0x0000

  The network ID of the coordinator in a ZigBee network is 0x0000.

- #define EMBER_NULL_NODE_ID 0xFFFF

  A distinguished network ID that will never be assigned to any node. Used to indicate the absence of a node ID.

- #define EMBER_VERSION_TYPE_MAX EMBER_VERSION_TYPE_LEGACY

  Size of EUI64 (an IEEE address) in bytes (8).

- #define EMBER_VERSION_TYPE_NAMES

  Size of EUI64 (an IEEE address) in bytes (8).

- #define NULL_BUFFER 0x0000

  Denotes a null buffer.

- #define TLS_SESSION_ID_SIZE 32

  Size of EUI64 (an IEEE address) in bytes (8).

- #define TLS_MASTER_SECRET_SIZE 48

  Size of EUI64 (an IEEE address) in bytes (8).

- enum EmberVersionType {
  EMBER_VERSION_TYPE_INTERNAL = 0,
  EMBER_VERSION_TYPE_ALPHA = 1,
  EMBER_VERSION_TYPE_BETA = 2,
  EMBER_VERSION_TYPE_GA = 3,
  EMBER_VERSION_TYPE_SPECIAL = 4,
  EMBER_VERSION_TYPE_LEGACY = 5 }

  Type of Ember software version.

- enum EmberIcmpType {

ICMP_DESTINATION_UNREACHABLE = 1,
ICMP_PACKET_TOO_BIG = 2,
ICMP_TIME_EXCEEDED = 3,
ICMP_PARAMETER_PROBLEM = 4,
ICMP_PRIVATE_EXPERIMENTATION_0 = 100,
ICMP_ECHO_REQUEST = 128,
ICMP_ECHO_REPLY = 129,
ICMP_ROUTER_SOLICITATION = 133,
ICMP_ROUTER_ADVERTISEMENT = 134,
ICMP_NEIGHBOR_SOLICITATION = 135,
ICMP_NEIGHBOR_ADVERTISEMENT = 136,
ICMP_RPL = 155,
ICMP_DUPLICATE_ADDRESS_REQUEST = 157,
ICMP_DUPLICATE_ADDRESS_CONFIRM = 158 }

Definitions for ICMP message types.

- enum EmberIcmpCode {
ICMP_CODE_NO_ROUTE_TO_DESTINATION = 0,
ICMP_CODE_ERROR_IN_SOURCE_ROUTING_HEADER = 7 }

  Definitions for ICMP message codes.
- enum EmberIpv6NextHeader {
IPV6_NEXT_HEADER_ICMP = 1,
IPV6_NEXT_HEADER_TCP = 6,
IPV6_NEXT_HEADER_UDP = 17,
IPV6_NEXT_HEADER_IPV6 = 41,
IPV6_NEXT_HEADER_ICMPV6 = 58,
IPV6_NEXT_HEADER_NO_NEXT = 59,
IPV6_NEXT_HEADER_MOBILITY = 137,
IPV6_NEXT_HEADER_HOP_BY_HOP = 0,
IPV6_NEXT_HEADER_DESTINATION = 60,
IPV6_NEXT_HEADER_ROUTING = 43,
IPV6_NEXT_HEADER_FRAGMENT = 44,
IPV6_NEXT_HEADER_UNKNOWN = 0xFF }

  Structure to hold an IPv6 "Next Header" See http://www.iana.org/assignments/protocol-numbers.
- typedef uint8_t EmberStatus

  Size of EUI64 (an IEEE address) in bytes (8).
- typedef uint8_t EmberEUI64[EUI64_SIZE]

  Obsolete version of EUI64 structure used by some platform-dependent applications. Use EmberEui64.
- typedef uint16_t EmberNodeId

  16-bit 802.15.4 network address.
- typedef uint16_t EmberPanId

  802.15.4 PAN ID.
- typedef uint16_t Buffer

  For use when declaring a Buffer.
- typedef uint16_t EmberMessageBuffer

  For use when declaring a buffer to hold a message.
- typedef Buffer PacketHeader

  For use when declaring a buffer to hold a packet header.
- typedef uint16_t ChildStatusFlags

  For use when declaring data that holds child status flags.

8.35.1 Detailed Description

See Utilities for details.

8.36 endian.h File Reference

Macros

- #define HTONL NTOHL
- #define HTONS NTOHS

Functions

- uint16_t NTOHS (uint16_t val)

  Converts a short (16-bit) value from network to host byte order.
- uint32_t NTOHL (uint32_t val)

  Converts a long (32-bit) value from network to host byte order.
- uint32_t SwapEndiannessInt32u (uint32_t val)

8.36.1 Detailed Description

See Network to Host Byte Order Conversion for detailed documentation.

8.37 error-def.h File Reference

Return-code definitions for EmberZNet stack API functions.

Macros

Generic Messages

These messages are system wide.

- #define EMBER_SUCCESS(x00)

  The generic "no error" message.
- #define EMBER_ERR_FATAL(x01)

  The generic "fatal error" message.
- #define EMBER_BAD_ARGUMENT(x02)

  An invalid value was passed as an argument to a function.
- #define EMBER_EEPROM_MFG_STACK_VERSION_MISMATCH(x04)

  The manufacturing and stack token format in non-volatile memory is different than what the stack expects (returned at initialization).
- #define EMBER_INCOMPATIBLE_STATIC_MEMORY_DEFINITIONS(x05)

  The static memory definitions in ember-static-memory.h are incompatible with this stack version.
- #define EMBER_EEPROM_MFG_VERSION_MISMATCH(x06)

  The manufacturing token format in non-volatile memory is different than what the stack expects (returned at initialization).
- #define EMBER_EEPROM_STACK_VERSION_MISMATCH(x07)

  The stack token format in non-volatile memory is different than what the stack expects (returned at initialization).

Packet Buffer Module Errors

- #define EMBER_NO_BUFFERS(x18)

  There are no more buffers.

Serial Manager Errors

- #define EMBER_SERIAL_INVALID_BAUD_RATE(x20)
  Specified an invalid baud rate.
- #define EMBER_SERIAL_INVALID_PORT(x21)
  Specified an invalid serial port.
- #define EMBER_SERIAL_TX_OVERFLOW(x22)
  Tried to send too much data.
- #define EMBER_SERIAL_RX_OVERFLOW(x23)
  There was not enough space to store a received character and the character was dropped.
- #define EMBER_SERIAL_RX_FRAME_ERROR(x24)
  Detected a UART framing error.
- #define EMBER_SERIAL_RX_PARITY_ERROR(x25)
  Detected a UART parity error.
- #define EMBER_SERIAL_RX_EMPTY(x26)
  There is no received data to process.
- #define EMBER_SERIAL_RX_OVERRUN_ERROR(x27)
  The receive interrupt was not handled in time, and a character was dropped.

MAC Errors

- #define EMBER_MAC_TRANSMIT_QUEUE_FULL(x39)
  The MAC transmit queue is full.
- #define EMBER_MAC_UNKNOWN_HEADER_TYPE(x3A)
  MAC header FCF error on receive.
- #define EMBER_MAC_ACK_HEADER_TYPE(x3B)
  MAC ACK header received.
- #define EMBER_MAC_SCANNING(x3D)
  The MAC can't complete this task because it is scanning.
- #define EMBER_MAC_NO_DATA(x31)
  No pending data exists for device doing a data poll.
- #define EMBER_MAC_JOINED_NETWORK(x32)
  Attempt to scan when we are joined to a network.
- #define EMBER_MAC_BAD_SCAN_DURATION(x33)
  Scan duration must be 0 to 14 inclusive. Attempt was made to scan with an incorrect duration value.
- #define EMBER_MAC_INCORRECT_SCAN_TYPE(x34)
  emberStartScan was called with an incorrect scan type.
- #define EMBER_MAC_INVALID_CHANNEL_MASK(x35)
  emberStartScan was called with an invalid channel mask.
- #define EMBER_MAC_COMMAND_TRANSMIT_FAILURE(x36)
  Failed to scan current channel because we were unable to transmit the relevent MAC command.
- #define EMBER_MAC_NO_ACK_RECEIVED(x40)
  We expected to receive an ACK following the transmission, but the MAC level ACK was never received.
- #define EMBER_MAC_INDIRECT_TIMEOUT(x42)
  Indirect data message timed out before polled.

Simulated EEPROM Errors

- #define EMBER_SIM_EEPROM_ERASE_PAGE_GREEN(x43)
  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.
- #define EMBER_SIM_EEPROM_ERASE_PAGE_RED(x44)
  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.
- #define EMBER_SIM_EEPROM_FULL(x45)
  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.
- #define EMBER_SIM_EEPROM_INIT_1_FAILED(x48)

Attempt 1 to initialize the Simulated EEPROM has failed.

- #define EMBER_SIM_EEPROM_INIT_2_FAILED(x49)

  Attempt 2 to initialize the Simulated EEPROM has failed.
- #define EMBER_SIM_EEPROM_INIT_3_FAILED(x4A)

  Attempt 3 to initialize the Simulated EEPROM has failed.
- #define EMBER_SIM_EEPROM_REPAIRING(x4D)

  The Simulated EEPROM is repairing itself.

Flash Errors

- #define EMBER_ERR_FLASH_WRITE_INHIBITED(x46)

  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.
- #define EMBER_ERR_FLASH_VERIFY_FAILED(x47)

  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 cycles of the flash.
- #define EMBER_ERR_FLASH_PROG_FAIL(x4B)
- #define EMBER_ERR_FLASH_ERASE_FAIL(x4C)

Bootloader Errors

- #define EMBER_ERR_BOOTLOADER_TRAP_TABLE_BAD(x58)

  The bootloader received an invalid message (failed attempt to go into bootloader).
- #define EMBER_ERR_BOOTLOADER_TRAP_UNKNOWN(x59)

  Bootloader received an invalid message (failed attempt to go into bootloader).
- #define EMBER_ERR_BOOTLOADER_NO_IMAGE(x05A)

  The bootloader cannot complete the bootload operation because either an image was not found or the image exceeded memory bounds.

Transport Errors

- #define EMBER_DELIVERY_FAILED(x66)

  The APS layer attempted to send or deliver a message, but it failed.
- #define EMBER_BINDING_INDEX_OUT_OF_RANGE(x69)

  This binding index is out of range for the current binding table.
- #define EMBER_ADDRESS_TABLE_INDEX_OUT_OF_RANGE(x6A)

  This address table index is out of range for the current address table.
- #define EMBER_INVALID_BINDING_INDEX(x6C)

  An invalid binding table index was given to a function.
- #define EMBER_INVALID_CALL(x70)

  The API call is not allowed given the current state of the stack.
- #define EMBER_COST_NOT_KNOWN(x71)

  The link cost to a node is not known.
- #define EMBER_MAX_MESSAGE_LIMIT_REACHED(x72)

  The maximum number of in-flight messages (i.e. ::EMBER_APS_UNICAST_MESSAGE_COUNT) has been reached.
- #define EMBER_MESSAGE_TOO_LONG(x74)

  The message to be transmitted is too big to fit into a single over-the-air packet.
- #define EMBER_BINDING_IS_ACTIVE(x75)

  The application is trying to delete or overwrite a binding that is in use.
- #define EMBER_ADDRESS_TABLE_ENTRY_IS_ACTIVE(x76)

  The application is trying to overwrite an address table entry that is in use.

HAL Module Errors

- #define EMBER_ADC_CONVERSION_DONE(x80)
  Conversion is complete.
- #define EMBER_ADC_CONVERSION_BUSY(x81)
  Conversion cannot be done because a request is being processed.
- #define EMBER_ADC_CONVERSION_DEFERRED(x82)
  Conversion is deferred until the current request has been processed.
- #define EMBER_ADC_NO_CONVERSION_PENDING(x84)
  No results are pending.
- #define EMBER_SLEEP_INTERRUPTED(x85)
  Sleeping (for a duration) has been abnormally interrupted and exited prematurely.

PHY Errors

- #define EMBER_PHY_TX_UNDERFLOW(x88)
  The transmit hardware buffer underflowed.
- #define EMBER_PHY_TX_INCOMPLETE(x89)
  The transmit hardware did not finish transmitting a packet.
- #define EMBER_PHY_INVALID_CHANNEL(x8A)
  An unsupported channel setting was specified.
- #define EMBER_PHY_INVALID_POWER(x8B)
  An unsupported power setting was specified.
- #define EMBER_PHY_TX_BUSY(x8C)
  The requested operation cannot be completed because the radio is currently busy, either transmitting a packet or performing calibration.
- #define EMBER_PHY_TX_CCA_FAIL(x8D)
  The transmit attempt failed because all CCA attempts indicated that the channel was busy.
- #define EMBER_PHY_OSCILLATOR_CHECK_FAILED(x8E)
  The software installed on the hardware doesn't recognize the hardware radio type.
- #define EMBER_PHY_ACK_RECEIVED(x8F)
  The expected ACK was received after the last transmission.

Return Codes Passed to emberStackStatusHandler()

See also ::emberStackStatusHandler().

- #define EMBER_NETWORK_UP(x90)
  The stack software has completed initialization and is ready to send and receive packets over the air.
- #define EMBER_NETWORK_DOWN(x91)
  The network is not operating.
- #define EMBER_JOIN_FAILED(x94)
  An attempt to join a network failed.
- #define EMBER_MOVE_FAILED(x96)
  After moving, a mobile node's attempt to re-establish contact with the network failed.
- #define EMBER_CANNOT_JOIN_AS_ROUTER(x98)
  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.
- #define EMBER_NODE_ID_CHANGED(x99)
  The local node ID has changed. The application can obtain the new node ID by calling ::emberGetNodeId().
- #define EMBER_PAN_ID_CHANGED(x9A)
  The local PAN ID has changed. The application can obtain the new PAN ID by calling emberGetPanId().
- #define EMBER_CHANNEL_CHANGED(x9B)
  The channel has changed.
- #define EMBER_NO_BEACONS(xAB)
  An attempt to join or rejoin the network failed because no router beacons could be heard by the joining node.
- #define EMBER_RECEIVED_KEY_IN_THE_CLEAR(xAC)
  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_REQUIRE_ENCRYPTED_↩ KEY).
- #define EMBER_NO_NETWORK_KEY_RECEIVED(xAD)

An attempt was made to join a Secured Network, but the device did not receive a Network Key.

- #define EMBER_NO_LINK_KEY_RECEIVED(xAE)

  After a device joined a Secured Network, a Link Key was requested (::EMBER_GET_LINK_KEY_WHEN_JOIN↩ ING) but no response was ever received.
- #define EMBER_PRECONFIGURED_KEY_REQUIRED(xAF)

  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.

Security Errors

- #define EMBER_KEY_INVALID(xB2)

  The passed key data is not valid. A key of all zeros or all F's are reserved values and cannot be used.
- #define EMBER_INVALID_SECURITY_LEVEL(x95)

  The chosen security level (the value of EMBER_SECURITY_LEVEL) is not supported by the stack.
- #define EMBER_APS_ENCRYPTION_ERROR(xA6)

  There was an error in trying to encrypt at the APS Level.
- #define EMBER_TRUST_CENTER_MASTER_KEY_NOT_SET(xA7)

  There was an attempt to form a network using High security without setting the Trust Center master key first.
- #define EMBER_SECURITY_STATE_NOT_SET(xA8)

  There was an attempt to form or join a network with security without calling ::emberSetInitialSecurityState() first.
- #define EMBER_KEY_TABLE_INVALID_ADDRESS(xB3)

  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 not valid addresses in 802.15.4.
- #define EMBER_SECURITY_CONFIGURATION_INVALID(xB7)

  There was an attempt to set a security configuration that is not valid given the other security settings.
- #define EMBER_TOO_SOON_FOR_SWITCH_KEY(xB8)

  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.
- #define EMBER_SIGNATURE_VERIFY_FAILURE(xB9)

  The received signature corresponding to the message that was passed to the CBKE Library failed verification, it is not valid.
- #define EMBER_KEY_NOT_AUTHORIZED(xBB)

  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).
- #define EMBER_MAC_COUNTER_ERROR(xDB)

  MAC encryption failed.
- #define EMBER_SECURITY_DATA_INVALID(xBD)

  The security data provided was not valid, or an integrity check failed.

Miscellaneous Network Errors

- #define EMBER_NOT_JOINED(x93)

  The node has not joined a network.
- #define EMBER_NETWORK_BUSY(xA1)

  A message cannot be sent because the network is currently overloaded.
- #define EMBER_INVALID_ENDPOINT(xA3)

  The application tried to send a message using an endpoint that it has not defined.
- #define EMBER_BINDING_HAS_CHANGED(xA4)

  The application tried to use a binding that has been remotely modified and the change has not yet been reported to the application.
- #define EMBER_INSUFFICIENT_RANDOM_DATA(xA5)

  An attempt to generate random bytes failed because of insufficient random data from the radio.
- #define EMBER_ROUTE_FAILURE(xA9)

  A route could not be found.
- #define EMBER_MANY_TO_ONE_ROUTE_FAILURE(xAA)

Miscellaneous Utility Errors

- #define EMBER_STACK_AND_HARDWARE_MISMATCH(xB0)

  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.
- #define EMBER_INDEX_OUT_OF_RANGE(xB1)

  An index was passed into the function that was larger than the valid range.
- #define EMBER_TABLE_FULL(xB4)

  There are no empty entries left in the table.
- #define EMBER_TABLE_ENTRY_ERASED(xB6)

  The requested table entry has been erased and contains no valid data.
- #define EMBER_LIBRARY_NOT_PRESENT(xB5)

  The requested function cannot be executed because the library that contains the necessary functionality is not present.
- #define EMBER_OPERATION_IN_PROGRESS(xBA)

  The stack accepted the command and is currently processing the request. The results will be returned via an appropriate handler.
- #define EMBER_TRUST_CENTER_EUI_HAS_CHANGED(xBC)

  The EUI of the Trust center has changed due to a successful rejoin. The device may need to perform other authentication to verify the new TC is authorized to take over.

Application Errors

These error codes are available for application use.

- #define EMBER_APPLICATION_ERROR_0(xF0)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_1(xF1)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_2(xF2)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_3(xF3)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_4(xF4)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_5(xF5)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_6(xF6)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_7(xF7)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_8(xF8)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_9(xF9)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_10(xFA)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_11(xFB)

  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.

- #define EMBER_APPLICATION_ERROR_12(xFC)
  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_13(xFD)
  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_14(xFE)
  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.
- #define EMBER_APPLICATION_ERROR_15(xFF)
  This error is reserved for customer application use. This will never be returned from any portion of the network stack or HAL.

8.37.1 Detailed Description

See Utilities for documentation.

8.38 flash.h File Reference

```
#include "memmap.h"
```

Functions

- bool halFlashEraseIsActive (void)
  Tells the calling code if a Flash Erase operation is active.

8.38.1 Detailed Description

See Flash Memory Control for documentation.

8.39 hal.h File Reference

Generic set of HAL includes for all platforms.

```
#include "micro/micro.h"
#include "micro/pta.h"
#include "plugin/antenna/antenna.h"
#include "plugin/adc/adc.h"
#include "micro/button.h"
#include "plugin/buzzer/buzzer.h"
#include "micro/crc.h"
#include "micro/endian.h"
#include "micro/led.h"
#include "micro/random.h"
#include "micro/serial.h"
#include "micro/spi.h"
#include "micro/system-timer.h"
#include "micro/bootloader-eeprom.h"
#include "micro/token.h"
#include "micro/bootloader-interface.h"
#include "micro/diagnostic.h"
```

Macros

- #define HAL_PTA_OPTIONS DEFAULT_PTA_OPTIONS
- #define emAmHost() false

8.39.1 Detailed Description

See also Hardware Abstraction Layer (HAL) API Reference for more documentation.

Some HAL includes are not used or present in builds intended for the Host processor connected to the Ember Network Coprocessor.

8.39.2 Macro Definition Documentation

8.39.2.1 #define emAmHost( ) false

8.39.2.2 #define HAL_PTA_OPTIONS DEFAULT_PTA_OPTIONS

8.40 iar.h File Reference

```
#include "hal/micro/generic/compiler/platform-common.h"
```

Macros

- #define HAL_HAS_INT64

  Denotes that this platform supports 64-bit data-types.
- #define _HAL_USE_COMMON_PGM_

  Use the Master Program Memory Declarations from platform-common.h.
- #define _HAL_USE_COMMON_MEMUTILS_

  If the line below is uncommented we will use Ember memory APIs, otherwise, we will use the C Standard library (memset,memcpy,memmove) APIs.
- #define PLATCOMMONOKTOINCLUDE

  Include platform-common.h last to pick up defaults and common definitions.
- #define MAIN_FUNCTION_PARAMETERS void

  The kind of arguments the main function takes.
- #define MAIN_FUNCTION_ARGUMENTS

Portable segment names

- #define __NO_INIT__ ".noinit"

  Portable segment names.
- #define __DEBUG_CHANNEL__ "DEBUG_CHANNEL"

  Portable segment names.
- #define __INTVEC__ ".intvec"

  Portable segment names.
- #define __CSTACK__ "CSTACK"

  Portable segment names.
- #define __RESETINFO__ "RESETINFO"

Portable segment names.

- #define __DATA_INIT__ ".data_init"

  Portable segment names.
- #define __DATA__ ".data"

  Portable segment names.
- #define __BSS__ ".bss"

  Portable segment names.
- #define __APP_RAM__ "APP_RAM"

  Portable segment names.
- #define __CONST__ ".rodata"

  Portable segment names.
- #define __TEXT__ ".text"

  Portable segment names.
- #define __TEXTRW_INIT__ ".textrw_init"

  Portable segment names.
- #define __TEXTRW__ ".textrw"

  Portable segment names.
- #define __AAT__ "AAT"

  Portable segment names.
- #define __BAT__ "BAT"

  Portable segment names.
- #define __BAT_INIT__ "BAT"

  Portable segment names.
- #define __FAT__ "FAT"

  Portable segment names.
- #define __RAT__ "RAT"

  Portable segment names.
- #define __NVM__ "NVM"

  Portable segment names.
- #define __SIMEE__ "SIMEE"

  Portable segment names.
- #define __PSSTORE__ "PSSTORE"

  Portable segment names.
- #define __EMHEAP__ "EMHEAP"

  Portable segment names.
- #define __EMHEAP_OVERLAY__ "EMHEAP_overlay"

  Portable segment names.
- #define __GUARD_REGION__ "GUARD_REGION"

  Portable segment names.
- #define __DLIB_PERTHREAD_INIT__ "__DLIB_PERTHREAD_init"

  Portable segment names.
- #define __DLIB_PERTHREAD_INITIALIZED_DATA__ "DLIB_PERTHREAD_INITIALIZED_DATA"

  Portable segment names.
- #define __DLIB_PERTHREAD_ZERO_DATA__ "DLIB_PERTHREAD_ZERO_DATA"

  Portable segment names.
- #define __INTERNAL_STORAGE__ "INTERNAL_STORAGE"

  Portable segment names.
- #define __UNRETAINED_RAM__ "UNRETAINED_RAM"

  Portable segment names.
- #define _NO_INIT_SEGMENT_BEGIN __segment_begin(__NO_INIT__)

  Portable segment names.
- #define _DEBUG_CHANNEL_SEGMENT_BEGIN __segment_begin(__DEBUG_CHANNEL__)

  Portable segment names.
- #define _INTVEC_SEGMENT_BEGIN __segment_begin(__INTVEC__)

  Portable segment names.
- #define _CSTACK_SEGMENT_BEGIN __segment_begin(__CSTACK__)

  Portable segment names.
- #define _RESETINFO_SEGMENT_BEGIN __segment_begin(__RESETINFO__)

  Portable segment names.

- #define _DATA_INIT_SEGMENT_BEGIN __segment_begin(__DATA_INIT__)
  Portable segment names.
- #define _DATA_SEGMENT_BEGIN __segment_begin(__DATA__)
  Portable segment names.
- #define _BSS_SEGMENT_BEGIN __segment_begin(__BSS__)
  Portable segment names.
- #define _APP_RAM_SEGMENT_BEGIN __segment_begin(__APP_RAM__)
  Portable segment names.
- #define _CONST_SEGMENT_BEGIN __segment_begin(__CONST__)
  Portable segment names.
- #define _TEXT_SEGMENT_BEGIN __segment_begin(__TEXT__)
  Portable segment names.
- #define _TEXTRW_INIT_SEGMENT_BEGIN __segment_begin(__TEXTRW_INIT__)
  Portable segment names.
- #define _TEXTRW_SEGMENT_BEGIN __segment_begin(__TEXTRW__)
  Portable segment names.
- #define _AAT_SEGMENT_BEGIN __segment_begin(__AAT__)
  Portable segment names.
- #define _BAT_SEGMENT_BEGIN __segment_begin(__BAT__)
  Portable segment names.
- #define _BAT_INIT_SEGMENT_BEGIN __segment_begin(__BAT_INIT__)
  Portable segment names.
- #define _FAT_SEGMENT_BEGIN __segment_begin(__FAT__)
  Portable segment names.
- #define _RAT_SEGMENT_BEGIN __segment_begin(__RAT__)
  Portable segment names.
- #define _NVM_SEGMENT_BEGIN __segment_begin(__NVM__)
  Portable segment names.
- #define _SIMEE_SEGMENT_BEGIN __segment_begin(__SIMEE__)
  Portable segment names.
- #define _PSSTORE_SEGMENT_BEGIN __segment_begin(__PSSTORE__)
  Portable segment names.
- #define _EMHEAP_SEGMENT_BEGIN __segment_begin(__EMHEAP__)
  Portable segment names.
- #define _EMHEAP_OVERLAY_SEGMENT_BEGIN __segment_begin(__EMHEAP_OVERLAY__)
  Portable segment names.
- #define _GUARD_REGION_SEGMENT_BEGIN __segment_begin(__GUARD_REGION__)
  Portable segment names.
- #define _DLIB_PERTHREAD_INIT_SEGMENT_BEGIN __segment_begin(__DLIB_PERTHREAD_INI↩T__)
  Portable segment names.
- #define _DLIB_PERTHREAD_INITIALIZED_DATA_SEGMENT_BEGIN __segment_begin(__DLIB_PE↩RTHREAD_INITIALIZED_DATA__)
  Portable segment names.
- #define _DLIB_PERTHREAD_ZERO_DATA_SEGMENT_BEGIN __segment_begin(__DLIB_PERTHR↩EAD_ZERO_DATA__)
  Portable segment names.
- #define _INTERNAL_STORAGE_SEGMENT_BEGIN __segment_begin(__INTERNAL_STORAGE__)
  Portable segment names.
- #define _UNRETAINED_RAM_SEGMENT_BEGIN __segment_begin(__UNRETAINED_RAM__)
  Portable segment names.
- #define _NO_INIT_SEGMENT_END __segment_end(__NO_INIT__)
  Portable segment names.
- #define _DEBUG_CHANNEL_SEGMENT_END __segment_end(__DEBUG_CHANNEL__)
  Portable segment names.
- #define _INTVEC_SEGMENT_END __segment_end(__INTVEC__)
  Portable segment names.
- #define _CSTACK_SEGMENT_END __segment_end(__CSTACK__)
  Portable segment names.

- #define _RESETINFO_SEGMENT_END __segment_end(__RESETINFO__)
  Portable segment names.
- #define _DATA_INIT_SEGMENT_END __segment_end(__DATA_INIT__)
  Portable segment names.
- #define _DATA_SEGMENT_END __segment_end(__DATA__)
  Portable segment names.
- #define _BSS_SEGMENT_END __segment_end(__BSS__)
  Portable segment names.
- #define _APP_RAM_SEGMENT_END __segment_end(__APP_RAM__)
  Portable segment names.
- #define _CONST_SEGMENT_END __segment_end(__CONST__)
  Portable segment names.
- #define _TEXT_SEGMENT_END __segment_end(__TEXT__)
  Portable segment names.
- #define _TEXTRW_INIT_SEGMENT_END __segment_end(__TEXTRW_INIT__)
  Portable segment names.
- #define _TEXTRW_SEGMENT_END __segment_end(__TEXTRW__)
  Portable segment names.
- #define _AAT_SEGMENT_END __segment_end(__AAT__)
  Portable segment names.
- #define _BAT_SEGMENT_END __segment_end(__BAT__)
  Portable segment names.
- #define _BAT_INIT_SEGMENT_END __segment_end(__BAT_INIT__)
  Portable segment names.
- #define _FAT_SEGMENT_END __segment_end(__FAT__)
  Portable segment names.
- #define _RAT_SEGMENT_END __segment_end(__RAT__)
  Portable segment names.
- #define _NVM_SEGMENT_END __segment_end(__NVM__)
  Portable segment names.
- #define _SIMEE_SEGMENT_END __segment_end(__SIMEE__)
  Portable segment names.
- #define _PSSTORE_SEGMENT_END __segment_end(__PSSTORE__)
  Portable segment names.
- #define _EMHEAP_SEGMENT_END __segment_end(__EMHEAP__)
  Portable segment names.
- #define _EMHEAP_OVERLAY_SEGMENT_END __segment_end(__EMHEAP_OVERLAY__)
  Portable segment names.
- #define _GUARD_REGION_SEGMENT_END __segment_end(__GUARD_REGION__)
  Portable segment names.
- #define _DLIB_PERTHREAD_INIT_SEGMENT_END __segment_end(__DLIB_PERTHREAD_INIT__)
  Portable segment names.
- #define _DLIB_PERTHREAD_INITIALIZED_DATA_SEGMENT_END __segment_end(__DLIB_PERT↩HREAD_INITIALIZED_DATA__)
  Portable segment names.
- #define _DLIB_PERTHREAD_ZERO_DATA_SEGMENT_END __segment_end(__DLIB_PERTHREAD↩_ZERO_DATA__)
  Portable segment names.
- #define _INTERNAL_STORAGE_SEGMENT_END __segment_end(__INTERNAL_STORAGE__)
  Portable segment names.
- #define _UNRETAINED_RAM_SEGMENT_END __segment_end(__UNRETAINED_RAM__)
  Portable segment names.
- #define _NO_INIT_SEGMENT_SIZE __segment_size(__NO_INIT__)
  Portable segment names.
- #define _DEBUG_CHANNEL_SEGMENT_SIZE __segment_size(__DEBUG_CHANNEL__)
  Portable segment names.
- #define _INTVEC_SEGMENT_SIZE __segment_size(__INTVEC__)
  Portable segment names.
- #define _CSTACK_SEGMENT_SIZE __segment_size(__CSTACK__)

Portable segment names.

- #define _RESETINFO_SEGMENT_SIZE __segment_size(__RESETINFO__)

  Portable segment names.
- #define _DATA_INIT_SEGMENT_SIZE __segment_size(__DATA_INIT__)

  Portable segment names.
- #define _DATA_SEGMENT_SIZE __segment_size(__DATA__)

  Portable segment names.
- #define _BSS_SEGMENT_SIZE __segment_size(__BSS__)

  Portable segment names.
- #define _APP_RAM_SEGMENT_SIZE __segment_size(__APP_RAM__)

  Portable segment names.
- #define _CONST_SEGMENT_SIZE __segment_size(__CONST__)

  Portable segment names.
- #define _TEXT_SEGMENT_SIZE __segment_size(__TEXT__)

  Portable segment names.
- #define _TEXTRW_INIT_SEGMENT_SIZE __segment_size(__TEXTRW_INIT__)

  Portable segment names.
- #define _TEXTRW_SEGMENT_SIZE __segment_size(__TEXTRW__)

  Portable segment names.
- #define _AAT_SEGMENT_SIZE __segment_size(__AAT__)

  Portable segment names.
- #define _BAT_SEGMENT_SIZE __segment_size(__BAT__)

  Portable segment names.
- #define _BAT_INIT_SEGMENT_SIZE __segment_size(__BAT_INIT__)

  Portable segment names.
- #define _FAT_SEGMENT_SIZE __segment_size(__FAT__)

  Portable segment names.
- #define _RAT_SEGMENT_SIZE __segment_size(__RAT__)

  Portable segment names.
- #define _NVM_SEGMENT_SIZE __segment_size(__NVM__)

  Portable segment names.
- #define _SIMEE_SEGMENT_SIZE __segment_size(__SIMEE__)

  Portable segment names.
- #define _PSSTORE_SEGMENT_SIZE __segment_size(__PSSTORE__)

  Portable segment names.
- #define _EMHEAP_SEGMENT_SIZE __segment_size(__EMHEAP__)

  Portable segment names.
- #define _EMHEAP_OVERLAY_SEGMENT_SIZE __segment_size(__EMHEAP_OVERLAY__)

  Portable segment names.
- #define _GUARD_REGION_SEGMENT_SIZE __segment_size(__GUARD_REGION__)

  Portable segment names.
- #define _DLIB_PERTHREAD_INIT_SEGMENT_SIZE __segment_size(__DLIB_PERTHREAD_INIT__)

  Portable segment names.
- #define _DLIB_PERTHREAD_INITIALIZED_DATA_SEGMENT_SIZE __segment_size(__DLIB_PERT↩HREAD_INITIALIZED_DATA__)

  Portable segment names.
- #define _DLIB_PERTHREAD_ZERO_DATA_SEGMENT_SIZE __segment_size(__DLIB_PERTHREA↩D_ZERO_DATA__)

  Portable segment names.
- #define _INTERNAL_STORAGE_SEGMENT_SIZE __segment_size(__INTERNAL_STORAGE__)

  Portable segment names.
- #define _UNRETAINED_RAM_SEGMENT_SIZE __segment_size(__UNRETAINED_RAM__)

  Portable segment names.

Typedefs

Master Variable Types

These are a set of typedefs to make the size of all variable declarations explicitly known.

- typedef bool boolean
  A typedef to make the size of the variable explicitly known.
- typedef unsigned char int8u
  A typedef to make the size of the variable explicitly known.
- typedef signed char int8s
  A typedef to make the size of the variable explicitly known.
- typedef unsigned short int16u
  A typedef to make the size of the variable explicitly known.
- typedef signed short int16s
  A typedef to make the size of the variable explicitly known.
- typedef unsigned int int32u
  A typedef to make the size of the variable explicitly known.
- typedef signed int int32s
  A typedef to make the size of the variable explicitly known.
- typedef unsigned long long int64u
  A typedef to make the size of the variable explicitly known.
- typedef signed long long int64s
  A typedef to make the size of the variable explicitly known.
- typedef unsigned int PointerType
  A typedef to make the size of the variable explicitly known.

Functions

- void _executeBarrierInstructions (void)

External Declarations

These are routines that are defined in certain header files that we don't want to include, e.g. stdlib.h

- int abs (int I)
  Returns the absolute value of I (also called the magnitude of I). That is, if I is negative, the result is the opposite of I, but if I is nonnegative the result is I.

Miscellaneous Macros

- #define BIGENDIAN_CPU false
  A convenient method for code to know what endiannes processor it is running on. For the Cortex-M3, we are little endian.
- #define NTOHS(val) (__REV16(val))
  Define intrinsics for NTOHL and NTOHS to save code space by making endian.c compile to nothing.
- #define NTOHL(val) (__REV(val))
  A convenient method for code to know what endiannes processor it is running on. For the Cortex-M3, we are little endian.
- #define NO_STRIPPING __root
  A friendlier name for the compiler's intrinsic for not stripping.
- #define EEPROM errorerror
  A friendlier name for the compiler's intrinsic for eeprom reference.
- #define __SOURCEFILE__ __FILE__
  *The **SOURCEFILE** macro is used by asserts to list the filename if it isn't otherwise defined, set it to the compiler intrinsic which specifies the whole filename and path of the sourcefile.*
- #define assert(condition)
  A custom implementation of the C language assert macro. This macro implements the conditional evaluation and calls the function halInternalAssertFailed(). (see hal/micro/micro.h)
- #define halResetWatchdog() halInternalResetWatchDog()

A convenient method for code to know what endiannes processor it is running on. For the Cortex-M3, we are little endian.

- #define __attribute__(...)

  *Define **attribute** to nothing since it isn't handled by IAR.*
- #define UNUSED

  Declare a variable as unused to avoid a warning. Has no effect in IAR builds.
- #define SIGNED_ENUM

  Some platforms need to cast enum values that have the high bit set.
- #define STACK_FILL_VALUE 0xCDCDCDCDU

  Define the magic value that is interpreted by IAR C-SPY's Stack View.
- #define RAMFUNC __ramfunc

  Define a generic RAM function identifier to a compiler specific one.
- #define NO_OPERATION() __no_operation()

  Define a generic no operation identifier to a compiler specific one.
- #define SET_REG_FIELD(reg, field, value)

  A convenience macro that makes it easy to change the field of a register to any value.
- #define SET_CMSIS_REG_FIELD(reg, field, value)

  A convenience macro that makes it easy to change the field of a register, as defined in CMSIS Device headers, to any value. Example using EM35xx: SET_CMSIS_REG_FIELD(GPIO->P[0].CFGL, GPIO_P_CFGL_Px0, _GPIO↩ _P_CFGL_Px0_OUT);.
- #define simulatedTimePasses()

  Stub for code not running in simulation.
- #define simulatedTimePassesMs(x)

  Stub for code not running in simulation.
- #define simulatedSerialTimePasses()

  Stub for code not running in simulation.
- #define _HAL_USE_COMMON_DIVMOD_

  Use the Divide and Modulus Operations from platform-common.h.
- #define VAR_AT_SEGMENT(__variableDeclaration, __segmentName) __variableDeclaration @ __↩ segmentName

  Provide a portable way to specify the segment where a variable lives.
- #define STRINGIZE(X) #X

  Convinience macro for turning a token into a string.
- #define ALIGNMENT(X) _Pragma(STRINGIZE(data_alignment = X))

  Provide a portable way to align data.
- #define WEAK(__symbol) __weak __symbol

  Provide a portable way to specify a symbol as weak.
- #define NO_INIT(__symbol) __no_init __symbol

  Provide a portable way to specify a non initialized symbol.
- #define STATIC_ASSERT(__condition, __errorstr) static_assert(__condition, __errorstr)

  Provide a portable way to specify a compile time assert.
- void halInternalAssertFailed (const char *filename, int linenumber)

  A prototype definition for use by the assert macro. (see hal/micro/micro.h)
- void halInternalResetWatchDog (void)

  Macro to reset the watchdog timer. Note: be very very careful when using this as you can easily get into an infinite loop if you are not careful.

8.40.1 Detailed Description

See IAR PLATFORM_HEADER Configuration for detailed documentation.

8.41 icmp.h File Reference

Simple ICMP API.

Functions

- EmberStatus emberIcmpListen (const uint8_t *address)

  This function sets up a listener for ICMP messages for the given address.
- bool emberIpPing (uint8_t *destination, uint16_t id, uint16_t sequence, uint16_t length, uint8_t hopLimit)

  This function sends an ICMP ECHO REQUEST message.
- void emberIncomingIcmpHandler (Ipv6Header *ipHeader)

  An application callback for an incoming ICMP message.

8.42 ip-modem-library.h File Reference

Data Structures

- struct ipModemThreadParamStruct

Typedefs

- typedef struct ipModemThreadParamStruct ipModemThreadParam

Functions

- void ipModemThreadMain (void *pvParameters)

8.42.1 Typedef Documentation

8.42.1.1 typedef struct ipModemThreadParamStruct ipModemThreadParam

8.42.2 Function Documentation

8.42.2.1 void ipModemThreadMain (void * *pvParameters*)

8.43 ip-modem-link.h File Reference

```
#include "stack/include/ember-types.h"
```

Enumerations

- enum ManagementType {
  MANAGEMENT_IDLE,
  MANAGEMENT_COMMAND,
  MANAGEMENT_NOTIFY,
  MANAGEMENT_RESPONSE_DONE,
  MANAGEMENT_RESPONSE_ERROR }

Functions

- void ipModemLinkInit (void)
- void ipModemLinkReset (void)
- bool ipModemLinkWaitForIncoming (uint32_t timeoutMs)
- void ipModemLinkAbortWaitForIncoming (void)
- void ipModemLinkProcessIncoming (void)
- void ipModemLinkIpPacketReceived (SerialLinkMessageType type, Buffer b)
- bool ipModemLinkSendManagementCommand (ManagementType managementType, const uint8_t *data, uint8_t len)
- bool ipModemLinkSendManagementCommandHost (ManagementType managementType, const uint8_t *data, uint8_t len)
- void ipModemLinkMarkBuffers (void)
- void * ipModemLinkMemoryAllocate (uint32_t size, void **freePtr)
- void ipModemLinkMemoryFree (void *ptr)
- bool ipModemLinkPreparingForPowerDown (void)
- void ipModemLinkManagementErrorHandler (const uint8_t *data, uint8_t len, bool hostToNcp)
- bool ipModemLinkIpPacketHandler (SerialLinkMessageType type, Buffer b)
- bool ipModemLinkManagementPacketHandler (ManagementType managementType, uint8_t *data, uint8_t len)
- void ipModemLinkUartTransmit (uint8_t type, PacketHeader header)

8.43.1 Enumeration Type Documentation

8.43.1.1 enum ManagementType

Enumerator

MANAGEMENT_IDLE

MANAGEMENT_COMMAND

MANAGEMENT_NOTIFY

MANAGEMENT_RESPONSE_DONE

MANAGEMENT_RESPONSE_ERROR

8.43.2 Function Documentation

8.43.2.1 void ipModemLinkAbortWaitForIncoming (void)

This function attempts to exit the wait state entered by the ipModemLinkWaitForIncoming() command. It can be used to stop the IP modem main loop from blocking after an asynchronous event, such as an interrupt. NOTE: Code in this function must be safe to call from within the interrupt context.

8.43.2.2 void ipModemLinkInit (void)

This function initializes the communication link used by the IP modem. It should be implemented for the application-specific communication mechanism. If it is ommitted, a default initialization that does nothing is used.

8.43.2.3 bool ipModemLinkIpPacketHandler (SerialLinkMessageType *type*, Buffer *b*)

8.43.2.4 void ipModemLinkIpPacketReceived (SerialLinkMessageType *type*, Buffer *b*)

This function sends an IP packet received by the node to the host application or thread for further processing.

Parameters

secured	If true, this packet is to be secured on output.
len	The length in bytes of the data packet.

8.43.2.5 void ipModemLinkManagementErrorHandler (const uint8_t * *data*, uint8_t *len*, bool *hostToNcp*)

This function is called when an error occurs sending management commands over the IP modem link. In RTOS builds it can be called by either the host or VNCP threads as indicated by the hostToNcp parameter.

Parameters

data	A pointer to the data that could not be sent.
len	The length of the data that we could not send in bytes.
hostToNcp	True if this message originated from the host to VNCP in RTOS builds and false otherwise.

8.43.2.6 bool ipModemLinkManagementPacketHandler (ManagementType *managementType*, uint8_t * *data*, uint8_t *len*)

8.43.2.7 void ipModemLinkMarkBuffers (void)

If this link is using any Ember buffers, it should mark them here to prevent garbage collection. If not using Ember buffers, this function should be stubbed out.

8.43.2.8 void* ipModemLinkMemoryAllocate (uint32_t *size*, void ** *freePtr*)

This optional function allows the IP modem to use memory allocated from the IP modem link instead of Ember buffers. You must return a pointer to the memory allocated as well as set the pointer to be passed to the ipModemLinkMemoryFree() function when garbage collecting. This function must be thread-safe.

Parameters

size	The number of bytes to allocate.
freePtr	The pointer that should be passed to ipModemLinkMemoryFree() when releasing this memory. If NULL, no callback will happen.

Returns

The address of the data pointer to use or NULL if this routine is not implemented and only Ember buffers should be used.

8.43.2.9 void ipModemLinkMemoryFree (void * *ptr*)

This function is called when freeing memory that was allocated with the ipModemLinkMemoryAllocate() function. It is passed the freePtr that was specified by the allocate routine. This function must be thread-safe.

Parameters

ptr	The value of the pointer that was set in freePtr when the ipModemLinkMemoryAllocate() function was called.

8.43.2.10 bool ipModemLinkPreparingForPowerDown (void)

8.43.2.11 void ipModemLinkProcessIncoming (void)

This function should check for incoming messages for the IP modem, read them out, and call the appropriate processing functions. It must be implemented for the specific link and should not block waiting for new messages.

8.43.2.12 void ipModemLinkReset (void)

This function resets the IP modem communications link. It should clear any state held in the link and ensure that it's ready to work again.

8.43.2.13 bool ipModemLinkSendManagementCommand (ManagementType *managementType*, const uint8_t * *data*, uint8_t *len*)

This function takes a management command and sends it out over the current link. The type is given and should be included so that it can be decoded on the other side of the link.

Parameters

managementType	The type of management command being sent.
data	A pointer to the management command data.
dataLength	The length of the management command being sent.

Returns

Returns True if the management command can be sent or false if unable to send.

8.43.2.14 bool ipModemLinkSendManagementCommandHost (ManagementType *managementType*, const uint8_t * *data*, uint8_t *len*)

This function is only used in environments where both the host and stack code run on the same chip (e.g., RTOS). In this situation, the function is called to send a management command from the host to the virtual NCP thread. On

platforms that don't support this return false.

Parameters

managementType	The type of management command being sent.
data	A pointer to the management command data.
dataLength	The length of the management command being sent.

Returns

Returns True if the management command could be sent or false if we were unable to send it for any reason.

8.43.2.15 void ipModemLinkUartTransmit (uint8_t *type*, PacketHeader *header*)

8.43.2.16 bool ipModemLinkWaitForIncoming (uint32_t *timeoutMs*)

This function should block until an incoming message is available for the IP modem link or the timeout expires.

Parameters

timeoutMs	The timeout in miliseconds.

Returns

Returns true if a message is available and false if the timeout expired.

8.44 led.h File Reference

Typedefs

- typedef enum HalBoardLedPins HalBoardLed

  Ensures that the definitions from the BOARD_HEADER are always used as parameters to the LED functions.

Functions

- void halInternalInitLed (void)

  Configures GPIOs pertaining to the control of LEDs.
- void halToggleLed (HalBoardLed led)

  Atomically wraps an XOR or similar operation for a single GPIO pin attached to an LED.
- void halSetLed (HalBoardLed led)

  Turns on (sets) a GPIO pin connected to an LED so that the LED turns on.
- void halClearLed (HalBoardLed led)

  Turns off (clears) a GPIO pin connected to an LED, which turns off the LED.
- void halStackIndicateActivity (bool turnOn)

  Called by the stack to indicate activity over the radio (for both transmission and reception). It is called once with `turnOn` *true and shortly thereafter with* `turnOn` *false.*

8.44.1 Detailed Description

See LED Control for documentation.

8.45 mfglib.h File Reference

Functions

- MfgStatus mfglibStart (void(∗mfglibRxCallback)(uint8_t ∗packet, uint8_t linkQuality, int8_t rssi))

  Activates use of MFGLIB test routines and enables the radio receiver to report packets it receives to the caller-specified mfglibRxCallback() routine.
- MfgStatus mfglibEnd (void)

  Deactivates use of MFGLIB test routines.
- MfgStatus mfglibStartTone (void)

  Starts transmitting the tone feature of the radio.
- MfgStatus mfglibStopTone (void)

  Stops transmitting a tone started by mfglibStartTone().
- MfgStatus mfglibStartStream (void)

  Starts transmitting a random stream of characters. This is so that the radio modulation can be measured.
- MfgStatus mfglibStopStream (void)

  Stops transmitting a random stream of characters started by mfglibStartStream().
- MfgStatus mfglibSendPacket (uint8_t ∗packet, uint16_t repeat)

  Sends a single packet, (repeat + 1) times.
- MfgStatus mfglibSetChannel (uint8_t channel)

  Selects the radio channel. The channel range is from 11 to 26.
- MfgStatus_U mfglibGetChannel (void)

  Get the current radio channel, as previously set via mfglibSetChannel().
- MfgStatus mfglibSetPower (uint16_t txPowerMode, int8_t power)

  Set the transmit power mode and the radio transmit power.
- MfgStatus_S mfglibGetPower (void)

  Get the current radio power setting as previously set via mfglibSetPower().
- MfgStatus_UU mfglibGetPowerMode (void)

  Get the radio transmit power mode setting as previously set via mfglibSetPower().
- void mfglibSetSynOffset (int8_t synOffset)

  Set the synth offset in 11.7kHz steps.
- MfgStatus_S mfglibGetSynOffset (void)

  Get the current synth offset in 11.7kHz steps.
- void mfglibTestContModCal (uint8_t channel, uint32_t duration)

  Run mod DAC calibration on the given channel for the given amount of time.
- MfgStatus mfglibSetOptions (uint8_t options)

  Set manufacturing library options.
- MfgStatus_U mfglibGetOptions (void)

  Get the current manufacturing library options, as previously set via mfglibSetOptions().
- void mfglibStartReturn (EmberStatus status)

  This function provides the result of a call to mfglibStart().
- void mfglibEndReturn (EmberStatus status, uint32_t receiveCount)

  This function provides the result of a call to mfglibEnd().
- void mfglibStartToneReturn (EmberStatus status)

  This function provides the result of a call to mfglibStartTone().

- void mfglibStopToneReturn (EmberStatus status)

  This function provides the result of a call to mfglibStopTone().
- void mfglibStartStreamReturn (EmberStatus status)

  This function provides the result of a call to mfglibStartStream().
- void mfglibStopStreamReturn (EmberStatus status)

  This function provides the result of a call to mfglibStopStream().
- void mfglibSendPacketReturn (EmberStatus status)

  This function provides the result of a call to mfglibSendPacket().
- void mfglibSetChannelReturn (EmberStatus status)

  This function provides the result of a call to mfglibSetChannel().
- void mfglibGetChannelReturn (uint8_t channel)

  This function provides the result of a call to mfglibGetChannel().
- void mfglibSetPowerReturn (EmberStatus status)

  This function provides the result of a call to mfglibSetPower().
- void mfglibGetPowerReturn (int8_t power)

  This function provides the result of a call to mfglibGetPower().
- void mfglibGetPowerModeReturn (uint16_t txPowerMode)

  This function provides the result of a call to mfglibGetPowerMode().
- void mfglibGetSynOffsetReturn (int8_t synthOffset)

  This function provides the result of a call to mfglibGetSynOffset().
- void mfglibSetOptionsReturn (EmberStatus status)

  This function provides the result of a call to mfglibSetOptions().
- void mfglibGetOptionsReturn (uint8_t options)

  This function provides the result of a call to mfglibGetOptions().
- void mfglibRxHandler (uint8_t *packet, uint8_t linkQuality, int8_t rssi)

  RX Handler for the mfglib test library.

8.45.1 Detailed Description

See MFGLIB for documentation.

8.46 micro-common.h File Reference

Minimal Hal functions common across all microcontroller-specific files. See Common Microcontroller Functions for documentation.

Data Structures

- struct RTCCRamData

Macros

- #define MICRO_DISABLE_WATCH_DOG_KEY 0xA5U

  The value that must be passed as the single parameter to halInternalDisableWatchDog() in order to successfully disable the watchdog timer.
- #define GPIO_MASK_SIZE 24
- #define GPIO_MASK 0xFFFFFF
- #define WAKE_GPIO_MASK GPIO_MASK
- #define WAKE_GPIO_SIZE GPIO_MASK_SIZE
- #define WAKE_MASK_INVALID (-1)
- #define WAKE_EVENT_SIZE WakeMask
- #define DEBUG_TOGGLE(n)

Typedefs

- typedef uint32_t WakeEvents
- typedef uint32_t WakeMask

Enumerations

- enum SleepModes {
  SLEEPMODE_RUNNING = 0U,
  SLEEPMODE_IDLE = 1U,
  SLEEPMODE_WAKETIMER = 2U,
  SLEEPMODE_MAINTAINTIMER = 3U,
  SLEEPMODE_NOTIMER = 4U,
  SLEEPMODE_HIBERNATE = 5U,
  SLEEPMODE_RESERVED = 6U,
  SLEEPMODE_POWERDOWN = 7U,
  SLEEPMODE_POWERSAVE = 8U }

  Enumerations for the possible microcontroller sleep modes.

Functions

- void halInit (void)

  Initializes microcontroller-specific peripherals.
- void halReboot (void)

  Restarts the microcontroller and therefore everything else.
- void halPowerUp (void)

  Powers up microcontroller peripherals and board peripherals.
- void halPowerDown (void)

  Powers down microcontroller peripherals and board peripherals.
- void halResume (void)

  Resumes microcontroller peripherals and board peripherals.
- void halSuspend (void)

  Suspends microcontroller peripherals and board peripherals.
- void halInternalEnableWatchDog (void)

  Enables the watchdog timer.
- void halInternalDisableWatchDog (uint8_t magicKey)

  Disables the watchdog timer.
- bool halInternalWatchDogEnabled (void)

  Determines whether the watchdog has been enabled or disabled.
- void halSleep (SleepModes sleepMode)

  Puts the microcontroller to sleep in a specified mode.
- void halCommonDelayMicroseconds (uint16_t us)

  Blocks the current thread of execution for the specified amount of time, in microseconds.
- void halCommonDisableVreg1v8 (void)

  Disable the 1.8V regulator. This function is to be used when the 1.8V supply is provided externally. Disabling the regulator saves current consumption. Disabling the regulator will cause ADC readings of external signals to be wrong. These exteranl signals include analog sources ADC0 thru ADC5 and VDD_PADS/4.
- void halCommonEnableVreg1v8 (void)

  Enable the 1.8V regulator. Normally the 1.8V regulator is enabled out of reset. This function is only needed if the 1.8V regulator has been disabled and ADC conversions on external signals are needed. These exteranl signals include analog sources ADC0 thru ADC5 and VDD_PADS/4. The state of 1v8 survives deep sleep.
- void halBeforeEM4 (uint32_t duration, RTCCRamData input)
- RTCCRamData halAfterEM4 (void)

Variables

- volatile int8_t halCommonVreg1v8EnableCount

  Helper variable to track the state of 1.8V regulator.

8.47 micro-types.h File Reference

This file handles defines and enums related to all the micros.

8.47.1 Detailed Description

THIS IS A GENERATED FILE. DO NOT EDIT.

8.48 micro.h File Reference

Utility and convenience functions for EM35x microcontroller. See Common Microcontroller Functions for documentation.

```
#include "micro-common.h"
```

Macros

- #define PTA_SUPPORT
- #define PTA_GPIOCFG_INPUT GPIOCFG_IN_PUD
- #define PTA_GPIOCFG_OUTPUT GPIOCFG_OUT
- #define PTA_GPIOCFG_WIRED_OR GPIOCFG_OUT_OD
- #define PTA_GPIOCFG_WIRED_AND GPIOCFG_OUT_OD

Vector Table Index Definitions

These are numerical definitions for vector table. Indices 0 through 15 are Cortex-M3 standard exception vectors and indices 16 through 35 are EM3XX specific interrupt vectors.

- #define STACK_VECTOR_INDEX 0U

  A numerical definition for a vector.
- #define RESET_VECTOR_INDEX 1U

  A numerical definition for a vector.
- #define NMI_VECTOR_INDEX 2U

  A numerical definition for a vector.
- #define HARD_FAULT_VECTOR_INDEX 3U

  A numerical definition for a vector.
- #define MEMORY_FAULT_VECTOR_INDEX 4U

  A numerical definition for a vector.
- #define BUS_FAULT_VECTOR_INDEX 5U

  A numerical definition for a vector.
- #define USAGE_FAULT_VECTOR_INDEX 6U

  A numerical definition for a vector.
- #define RESERVED07_VECTOR_INDEX 7U

  A numerical definition for a vector.

- #define RESERVED08_VECTOR_INDEX 8U
  A numerical definition for a vector.
- #define RESERVED09_VECTOR_INDEX 9U
  A numerical definition for a vector.
- #define RESERVED10_VECTOR_INDEX 10U
  A numerical definition for a vector.
- #define SVCALL_VECTOR_INDEX 11U
  A numerical definition for a vector.
- #define DEBUG_MONITOR_VECTOR_INDEX 12U
  A numerical definition for a vector.
- #define RESERVED13_VECTOR_INDEX 13U
  A numerical definition for a vector.
- #define PENDSV_VECTOR_INDEX 14U
  A numerical definition for a vector.
- #define SYSTICK_VECTOR_INDEX 15U
  A numerical definition for a vector.
- #define TIMER1_VECTOR_INDEX 16U
  A numerical definition for a vector.
- #define TIMER2_VECTOR_INDEX 17U
  A numerical definition for a vector.
- #define MANAGEMENT_VECTOR_INDEX 18U
  A numerical definition for a vector.
- #define BASEBAND_VECTOR_INDEX 19U
  A numerical definition for a vector.
- #define SLEEP_TIMER_VECTOR_INDEX 20U
  A numerical definition for a vector.
- #define SC1_VECTOR_INDEX 21U
  A numerical definition for a vector.
- #define SC2_VECTOR_INDEX 22U
  A numerical definition for a vector.
- #define SECURITY_VECTOR_INDEX 23U
  A numerical definition for a vector.
- #define MAC_TIMER_VECTOR_INDEX 24U
  A numerical definition for a vector.
- #define MAC_TX_VECTOR_INDEX 25U
  A numerical definition for a vector.
- #define MAC_RX_VECTOR_INDEX 26U
  A numerical definition for a vector.
- #define ADC_VECTOR_INDEX 27U
  A numerical definition for a vector.
- #define IRQA_VECTOR_INDEX 28U
  A numerical definition for a vector.
- #define IRQB_VECTOR_INDEX 29U
  A numerical definition for a vector.
- #define IRQC_VECTOR_INDEX 30U
  A numerical definition for a vector.
- #define IRQD_VECTOR_INDEX 31U
  A numerical definition for a vector.
- #define DEBUG_VECTOR_INDEX 32U
  A numerical definition for a vector.
- #define SC3_VECTOR_INDEX 33U
  A numerical definition for a vector.
- #define SC4_VECTOR_INDEX 34U
  A numerical definition for a vector.
- #define USB_VECTOR_INDEX 35U
  A numerical definition for a vector.
- #define VECTOR_TABLE_LENGTH 36U
  Number of vectors.

Functions

- void halInternalSysReset (uint16_t extendedCause)

  Records the specified reset cause then forces a reboot.
- uint16_t halGetExtendedResetInfo (void)

  Returns the Extended Reset Cause information.
- PGM_P halGetExtendedResetString (void)

  Calls halGetExtendedResetInfo() and supplies a string describing the extended cause of the reset. halGetReset↩String() should also be called to get the string for the base reset cause.
- EmberStatus halSetRadioHoldOff (bool enable)

  Enables or disables Radio HoldOff support.
- bool halGetRadioHoldOff (void)

  Returns whether Radio HoldOff has been enabled or not.
- void halStackRadioPowerDownBoard (void)

  To assist with saving power when the radio automatically powers down, this function allows the stack to tell the HAL to put pins specific to radio functionality in their powerdown state. The pin state used is the state used by halInternal↩PowerDownBoard, but applied only to the pins identified in the global variable gpioRadioPowerBoardMask. The stack will automatically call this function as needed, but it will only change GPIO state based on gpioRadioPowerBoard↩Mask. Most commonly, the bits set in gpioRadioPowerBoardMask petain to using a Front End Module. This function is often called from interrupt context.
- void halStackRadio2PowerDownBoard (void)

  To assist with saving power when radio2 automatically powers down, this function allows the stack to tell the HAL to put pins specific to radio functionality in their powerdown state. The pin state used is the state used by halInternal↩PowerDownBoard, but applied only to the pins identified in the global variable gpioRadioPowerBoardMask. The stack will automatically call this function as needed, but it will only change GPIO state based on gpioRadioPowerBoard↩Mask. Most commonly, the bits set in gpioRadioPowerBoardMask petain to using a Front End Module. This function is often called from interrupt context.
- void halStackRadioPowerUpBoard (void)

  To assist with saving power when the radio automatically powers up, this function allows the stack to tell the HAL to put pins specific to radio functionality in their powerup state. The pin state used is the state used by halInternal↩PowerUpBoard, but applied only to the pins identified in the global variable gpioRadioPowerBoardMask. The stack will automatically call this function as needed, but it will only change GPIO state based on gpioRadioPowerBoardMask. Most commonly, the bits set in gpioRadioPowerBoardMask petain to using a Front End Module. This function can be called from interrupt context.
- void halStackRadio2PowerUpBoard (void)

  To assist with saving power when radio2 automatically powers up, this function allows the stack to tell the HAL to put pins specific to radio functionality in their powerup state. The pin state used is the state used by halInternalPower↩UpBoard, but applied only to the pins identified in the global variable gpioRadioPowerBoardMask. The stack will automatically call this function as needed, but it will only change GPIO state based on gpioRadioPowerBoardMask. Most commonly, the bits set in gpioRadioPowerBoardMask petain to using a Front End Module. This function can be called from interrupt context.
- void halStackRadioPowerMainControl (bool powerUp)

  This function is called automatically by the stack prior to Radio power-up and after Radio power-down. It can be used to prepare for the radio being powered on and to clean up after it's been powered off. Unlike halStackRadioPower↩UpBoard() and halStackRadioPowerDownBoard(), which can be called from interrupt context, this function is only called from main-line context.
- void halRadioPowerUpHandler (void)

  Handler called in main context prior to radio being powered on.
- void halRadioPowerDownHandler (void)

  Handler called in main context after radio has been powered off.

8.49 micro.h File Reference

Full HAL functions common across all microcontroller-specific files. See Common Microcontroller Functions for documentation.

```
#include "hal/micro/generic/em2xx-reset-defs.h"
#include "hal/micro/micro-types.h"
```

Macros

- #define halGetEm2xxResetInfo() halGetResetInfo()

  Calls halGetExtendedResetInfo() and translates the EM35x or COBRA reset code to the corresponding value used by the EM2XX HAL. Any reset codes not present in the EM2XX are returned after being OR'ed with 0x80.

Functions

- void halStackProcessBootCount (void)

  Called from emberInit and provides a means for the HAL to increment a boot counter, most commonly in non-volatile memory.
- uint8_t halGetResetInfo (void)

  Gets information about what caused the microcontroller to reset.
- PGM_P halGetResetString (void)

  Calls halGetResetInfo() and supplies a string describing it.

8.49.1 Detailed Description

Some functions in this file return an EmberStatus value. See error-def.h for definitions of all EmberStatus return values.

8.50 network-management.h File Reference

Network Management API.

```
#include <stddef.h>
#include "stack/include/ember-types.h"
```

Data Structures

- struct EmberNetworkParameters

  An application structure to hold useful network parameters.
- struct EmberRipEntry

  Structure that holds information about a routing table entry for use on the application. See emberGetRipEntry.
- struct EmberMacBeaconData

  Structure to hold information about an 802.15.4 beacon for use on the application.
- struct EmberSecurityParameters

  Values of security parameters for use in forming or joining a network.
- struct EmberDnsResponse

  Structure for returning information from a DNS lookup. A structure is used to make it easier to add additional values.

Macros

- #define EMBER_HIGH_PRIORITY_TASKS (EMBER_OUTGOING_MESSAGES | EMBER_INCOMING_↩ MESSAGES | EMBER_RADIO_IS_ON)

  A mask of the high priority tasks that prevent a device from sleeping. Devices should not sleep if any high priority tasks are active.
- #define ISLAND_ID_SIZE 5

  Size of the island (aka network fragment) ID.
- #define EMBER_NETWORK_KEY_OPTION BIT(0)

  Define the various options for setting network parameters. Note: Only the EMBER_NETWORK_KEY_OPTION works at this time.
- #define EMBER_PSK_JOINING_OPTION BIT(1)
- #define EMBER_ECC_JOINING_OPTION BIT(2)
- #define EMBER_MAX_IPV6_ADDRESS_COUNT 10

  The maximum number of IPv6 addresses configured for the device. See emberGetLocalIpAddress.
- #define EMBER_MAX_IPV6_GLOBAL_ADDRESS_COUNT (EMBER_MAX_IPV6_ADDRESS_COUNT - 2)
- #define EMBER_MAX_IPV6_EXTERNAL_ROUTE_COUNT (EMBER_MAX_IPV6_ADDRESS_COUNT - 2)
- #define EMBER_MAX_LIFETIME_DELAY_SEC ((HALF_MAX_INT32U_VALUE - 1) / 1000)

  We enforce this limit to avoid overflow when converting lifetimes from seconds to milliseconds.
- #define EMBER_MIN_PREFERRED_LIFETIME_SEC 1800

  There should be at least half an hour of preferred lifetime remaining to advertise a DHCP server.
- #define EMBER_MIN_VALID_LIFETIME_SEC 60

  Renew when we are down to one minute of valid lifetime.
- #define EMBER_MAX_DNS_NAME_LENGTH 128

  The maximum length of a domain name that may be passed to emberDnsLookup().
- #define EMBER_MAX_DNS_QUERY_APP_DATA_LENGTH 64

  The maximum number of bytes of application data that may be passed to emberDnsLookup().
- #define EMBER_USE_DEFAULTS 0

  The following denotes which network parameters to use when forming or joining a network. Construct an uint16_t "options" flag for use in various network formation calls.
- #define EMBER_NETWORK_ID_OPTION BIT(0)
- #define EMBER_ULA_PREFIX_OPTION BIT(1)
- #define EMBER_EXTENDED_PAN_ID_OPTION BIT(2)
- #define EMBER_PAN_ID_OPTION BIT(3)
- #define EMBER_NODE_TYPE_OPTION BIT(4)
- #define EMBER_TX_POWER_OPTION BIT(5)
- #define EMBER_MASTER_KEY_OPTION BIT(6)
- #define EMBER_LEGACY_ULA_OPTION BIT(7)
- #define EMBER_JOIN_KEY_OPTION BIT(8)
- #define EMBER_NETWORK_DATA_LEADER_SIZE 8

  Network data values.
- #define EMBER_IPV6_ADDRESS_STRING_SIZE 40

  Maximum size of a string written by emberIpv6AddressToString, including a NUL terminator. It is sufficient to store the string "ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff" and a NUL terminator.
- #define EMBER_IPV6_PREFIX_STRING_SIZE 44

  Maximum size of a string written by emberIpv6PrefixToString, including a NUL terminator. It is sufficient to store the string "ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff/128" and a NUL terminator.
- #define EMBER_IPV6_BITS 128

  Number of bits in an IPv6 address.
- #define EMBER_IPV6_BYTES (EMBER_IPV6_BITS / 8)

  Number of bytes in an IPv6 address.
- #define EMBER_IPV6_FIELDS (EMBER_IPV6_BYTES / 2)

  Number of fields in an IPv6 address.
- #define EMBER_IPV6_MTU 1280

  Size of the largest supported IPv6 packet.

Typedefs

- typedef uint8_t EmberTokenId

  Read token values stored on the Ember chip.
- typedef uint8_t EmberMfgTokenId

  Token identifier used when reading and writing manufacturing tokens.
- typedef uint16_t EmberBorderRouterTlvFlag
- typedef uint8_t EmberDefaultRouteTlvFlag
- typedef uint8_t LocalServerFlag
- typedef void(∗ EmberDnsResponseHandler) (EmberDnsLookupStatus status, const uint8_t ∗domain↩ Name, uint8_t domainNameLength, const EmberDnsResponse ∗response, void ∗applicationData, uint16_t applicationDataLength)

  Type definition for callback handlers for DNS responses.

Enumerations

- enum EmberIdleRadioState {
  IDLE_WITH_RADIO_ON,
  IDLE_WITH_POLLING,
  IDLE_WITH_RADIO_OFF }

  Required radio state while stack is idle.
- enum {
  EMBER_OUTGOING_MESSAGES = 0x01,
  EMBER_INCOMING_MESSAGES = 0x02,
  EMBER_RADIO_IS_ON = 0x04 }

  This function defines tasks that prevent the stack from sleeping.
- enum EmberResetCause {
  EMBER_RESET_UNKNOWN,
  EMBER_RESET_FIB,
  EMBER_RESET_BOOTLOADER,
  EMBER_RESET_EXTERNAL,
  EMBER_RESET_POWERON,
  EMBER_RESET_WATCHDOG,
  EMBER_RESET_SOFTWARE,
  EMBER_RESET_CRASH,
  EMBER_RESET_FLASH,
  EMBER_RESET_FATAL,
  EMBER_RESET_FAULT,
  EMBER_RESET_BROWNOUT }

  Enumerate the various chip reset causes.
- enum { EMBER_CHANNEL_CAL_DATA_TOKEN }

  Enumerate the various token values that can be retrieved by the application.
- enum {
  EMBER_CUSTOM_EUI_64_MFG_TOKEN,
  EMBER_EZSP_STORAGE_MFG_TOKEN,
  EMBER_CTUNE_MFG_TOKEN }

  Enumerate the various manufacturing token values that can be read or written by the application.
- enum EmberLocalAddressScope {
  REALM_SCOPE = 0,
  LINK_SCOPE = 1,
  GLOBAL_SCOPE = 2 }

- enum EmberBorderRouterTlvFlag_e {
  EMBER_BORDER_ROUTER_ND_DNS_FLAG = 0x0080,
  EMBER_BORDER_ROUTER_ON_MESH_FLAG = 0x0100,
  EMBER_BORDER_ROUTER_DEFAULT_ROUTE_FLAG = 0x0200,
  EMBER_BORDER_ROUTER_CONFIGURE_FLAG = 0x0400,
  EMBER_BORDER_ROUTER_DHCP_FLAG = 0x0800,
  EMBER_BORDER_ROUTER_SLAAC_FLAG = 0x1000,
  EMBER_BORDER_ROUTER_PREFERRED_FLAG = 0x2000,
  EMBER_BORDER_ROUTER_PREFERENCE_MASK = 0xC000,
  EMBER_BORDER_ROUTER_HIGH_PREFERENCE = 0x4000,
  EMBER_BORDER_ROUTER_MEDIUM_PREFERENCE = 0x0000,
  EMBER_BORDER_ROUTER_LOW_PREFERENCE = 0xC000 }

  Border router flags (see Thread spec chapter 5 for more information)
- enum EmberExternalRouteTlvFlag_e {
  EMBER_EXTERNAL_ROUTE_PREFERENCE_MASK = 0xC0,
  EMBER_EXTERNAL_ROUTE_HIGH_PREFERENCE = 0x40,
  EMBER_EXTERNAL_ROUTE_MEDIUM_PREFERENCE = 0x00,
  EMBER_EXTERNAL_ROUTE_LOW_PREFERENCE = 0xC0 }

  External route router flags (see Thread spec chapter 5 for more information)
- enum LocalServerFlag_e {
  EMBER_GLOBAL_ADDRESS_AM_GATEWAY = 0x01,
  EMBER_GLOBAL_ADDRESS_AM_DHCP_SERVER = 0x02,
  EMBER_GLOBAL_ADDRESS_AM_SLAAC_SERVER = 0x04,
  EMBER_GLOBAL_ADDRESS_DHCP = 0x08,
  EMBER_GLOBAL_ADDRESS_SLAAC = 0x10,
  EMBER_GLOBAL_ADDRESS_CONFIGURED = 0x20,
  EMBER_GLOBAL_ADDRESS_REQUEST_SENT = 0x40,
  EMBER_GLOBAL_ADDRESS_REQUEST_FAILED = 0x80,
  EMBER_LOCAL_ADDRESS = 0xFF }

  Address configuration flags. These flags denote the properties of a Thread IPv6 address.
- enum EmberDnsLookupStatus {
  EMBER_DNS_LOOKUP_SUCCESS,
  EMBER_DNS_LOOKUP_NO_BORDER_ROUTER,
  EMBER_DNS_LOOKUP_NO_BORDER_ROUTER_RESPONSE,
  EMBER_DNS_LOOKUP_BORDER_ROUTER_RESPONSE_ERROR,
  EMBER_DNS_LOOKUP_NO_DNS_SERVER,
  EMBER_DNS_LOOKUP_NO_DNS_RESPONSE,
  EMBER_DNS_LOOKUP_NO_DNS_RESPONSE_ERROR,
  EMBER_DNS_LOOKUP_NO_ENTRY_FOR_NAME,
  EMBER_DNS_LOOKUP_NO_BUFFERS }

  Status values passed to DNS response handlers.
- enum {
  EMBER_NO_COMMISSIONER = 0,
  EMBER_HAVE_COMMISSIONER = BIT(0),
  EMBER_AM_COMMISSIONER = BIT(1),
  EMBER_JOINING_ENABLED = BIT(2),
  EMBER_JOINING_WITH_EUI_STEERING = BIT(3) }

  Flag values for emberCommissionerStatusHandler().
- enum EmberJoiningMode {
  EMBER_NO_JOINING,
  EMBER_JOINING_ALLOW_ALL_STEERING,
  EMBER_JOINING_ALLOW_EUI_STEERING,
  EMBER_JOINING_ALLOW_SMALL_EUI_STEERING }

  Joining modes, passed to emberSetJoiningMode() on the commissioner. No change takes place until emberSend↩ SteeringData() is called. If steering is used, the EUI-64s of the joining devices should be passed to emberAdd↩ SteeringEui64() before calling emberSendSteeringData().

Functions

- void emberInit (void)

  This function initializes the Ember stack.
- void emberInitReturn (EmberStatus status)

  This function provides the result of a call to emberInit().
- void emberTick (void)

  A periodic tick routine that must be called in the application's main event loop.
- void emberResetNetworkState (void)

  This function erases the network state stored in nonvolatile memory after which the network status will be EMBER_NO_NETWORK. This function should not be called to rejoin a former network; use emberResumeNetwork() instead. There may be difficulties joining a former network after resetting the network state, due to security considerations.
- void emberResetNetworkStateReturn (EmberStatus status)

  This function provides the result of a call to emberResetNetworkState().
- bool emberDeepSleepTick (void)

  An application handler for deep sleep on sleepy end devices. This call is ignored for non-sleepy devices. The device may or may not sleep depending on the internal state.
- void emberDeepSleep (bool sleep)

  This function turns chip deep sleep on or off for sleepy end devices. This call is ignored on non-sleepy devices. The device may or may not sleep depending on the internal state.
- void emberDeepSleepReturn (EmberStatus status)

  This function provides the result of a call to emberDeepSleep().
- void emberDeepSleepCompleteHandler (uint16_t sleepDuration)

  For a sleepy end device, report how long the chip went to deep sleep. In a NCP + host setup, the stack reports this to the host app.
- uint32_t emberStackIdleTimeMs (EmberIdleRadioState *radioStateResult)

  This function returns the time the stack will be idle, in milliseconds. Also sets radioStateResult to the required radio state while the stack is idle.
- bool emberOkToNap (void)

  This function indicates whether the stack is currently in a state where there are no high priority tasks and may sleep.
- void emberOkToNapReturn (uint8_t stateMask)

  If implementing event-driven sleep on an NCP host, this method will return the bitmask indicating the stack's current tasks. (see enum above)
- void emberEventDelayUpdatedFromIsrHandler (Event *event)

  This method is called any time an event is scheduled from within an ISR context. It can be used to determine when to stop a long running sleep to see what application or stack events now need to be processed.
- void emberStackPrepareForPowerDown (void)

  This function gets the stack ready for power down, or deep sleep. Purges the MAC indirect queue, and empties the phy-to-mac and mac-to-network queues.
- bool emberStackPreparingForPowerDown (void)

  This function returns true if the stack is currently emptying any message queues or false if the MAC queue is currently not empty.
- void emberStackPowerDown (void)

  Immediately turns the radio power completely off.
- void emberStackPowerUp (void)

  This function initializes the radio. Typically called coming out of deep sleep.
- void emberStackPollForData (uint32_t pollMs)

  For sleepy hosts, use this call to have the stack manage polling for sleepy end devices. In a host/NCP setup, this means that the NCP app will take care of periodic data polling.
- void emberStackPollForDataReturn (EmberStatus status)

  This function provides the result of a call to emberStackPollForData().
- void emberPollForData (void)

  Use this call if setting up polling for sleepy end devices on the application.

- void emberPollForDataReturn (EmberStatus status)

  This function provides the result of a call to emberPollForData().
- const EmberEui64 * emberEui64 (void)

  This function returns the EUI64 of the Ember chip.
- EmberNetworkStatus emberNetworkStatus (void)

  This function returns the current status of the network. Prior to calling emberInitNetwork(), the status is EMBER_NETWORK_UNINITIALIZED.
- void emberNetworkStatusHandler (EmberNetworkStatus newNetworkStatus, EmberNetworkStatus oldNetworkStatus, EmberJoinFailureReason reason)

  This function reports a change to the network status. For example, the network status changes while going through the joining process, or while reattaching to the network, which can happen for a variety of reasons. In particular, after issuing a form, join, resume, or attach command, the application knows that the device is on the network and ready to communicate when this handler is called with a newNetworkStatus of EMBER_JOINED_NETWORK_ATTACHED.
- void emberGetNetworkParameters (EmberNetworkParameters *parameters)

  This function fetches the current network parameters into the supplied pointer.
- EmberPanId emberGetPanId (void)

  This function returns the pan id of the network.
- void emberGetRipEntry (uint8_t index)

  This function gets the EmberRipEntry at the specified index of the RIP table. The result is returned to the application via the emberGetRipEntryReturn() callback.
- void emberGetRipEntryReturn (uint8_t index, const EmberRipEntry *entry)

  This function provides the result of a call to emberGetRipEntry().
- void emberGetCounter (EmberCounterType type)

  This function gets the value for the specified counter. The result is returned to the application via emberGetCounterReturn().
- void emberGetCounterReturn (EmberCounterType type, uint16_t value)

  This function provides the result of a call to emberGetCounter().
- void emberClearCounters (void)

  This function resets all counter values to 0.
- void emberCounterHandler (EmberCounterType type, uint16_t increment)

  A callback invoked to inform the application of the occurrence of an event defined by EmberCounterType, for example, transmissions and receptions at different layers of the stack.
- uint16_t emberCounterValueHandler (EmberCounterType type)

  A callback invoked to query the application for the countervalue of an event defined by EmberCounterType.
- bool emberForwardIpv6Packet (const uint8_t *packet, const uint16_t packetLength)

  This API provides a means to forward a raw IPv6 packet on the mesh.
- void emberStartScan (EmberNetworkScanType scanType, uint32_t channelMask, uint8_t duration)

  This function starts a scan. Note that while a scan can be initiated while the node is currently joined to a network, the node will generally be unable to communicate with its PAN during the scan period, so care should be taken when performing scans of any significant duration while presently joined to an existing PAN.
- void emberEnergyScanHandler (uint8_t channel, int8_t maxRssiValue)

  This function reports the maximum RSSI value measured on the channel.
- void emberActiveScanHandler (const EmberMacBeaconData *beaconData)

  This function reports an incoming beacon during an active scan.
- void emberScanReturn (EmberStatus status)

  This function provides the status upon completion of a scan.
- void emberStopScan (void)

  This function terminates a scan in progress.
- void emberResetMicro (void)

  This function resets the Ember chip.
- void emberResetMicroHandler (EmberResetCause cause)

  This function notifies the application of a reset on the Ember chip due to the indicated cause.
- void emberGetStandaloneBootloaderInfo (void)

This function detects if the standalone bootloder is installed, and if so returns the installed version and info about the platform, micro and phy. If not version will be set to 0xffff. A returned version of 0x1234 would indicate version 1.2 build 34.

- void emberGetStandaloneBootloaderInfoReturn (uint16_t version, uint8_t platformId, uint8_t microId, uint8_t phyId)

  This function provides the result of a call to emberGetStandaloneBootloaderInfo.
- void emberLaunchStandaloneBootloader (uint8_t mode)

  This function launches the standalone bootloader (if installed). The function returns an error if the standalone bootloader is not present.
- void emberLaunchStandaloneBootloaderReturn (EmberStatus status)

  This function provides the result of a call to emberLaunchStandaloneBootloader.
- void emberInitHost (void)

  In a host/NCP setup, inform the NCP to send the network state and version information.
- void emberState (void)

  In a host/NCP setup, get the network parameters, the network status and eui64 all at once.
- void emberStateReturn (const EmberNetworkParameters *parameters, const EmberEui64 *localEui64, const EmberEui64 *macExtendedId, EmberNetworkStatus networkStatus)

  In a host/NCP setup, provides the result of a call to emberState() on the host.
- void emberHostStateHandler (const EmberNetworkParameters *parameters, const EmberEui64 *localEui64, const EmberEui64 *macExtendedId, EmberNetworkStatus networkStatus)

  In a host/NCP setup, notifies the host to changes in the network parameters.
- void emberSetRadioPower (int8_t power)

  This function sets the radio output power at which a node is to operate. 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. Note: If the requested power level is not available on a given radio, this function will use the next higher available power level.
- void emberSetRadioPowerReturn (EmberStatus status)

  This function provides the result of a call to emberSetRadioPower() on the host.
- void emberGetRadioPower (void)

  This function gets 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.
- void emberGetRadioPowerReturn (int8_t power)

  This function provides the result of a call to emberGetRadioPower() on the host.
- EmberStatus emberSetTxPowerMode (uint16_t txPowerMode)

  This function enables boost power mode and/or the alternate transmit path.
- void emberSetTxPowerModeReturn (EmberStatus status)

  This function provides the result of a call to emberSetTxPowerMode() on the host.
- void emberGetTxPowerMode (void)

  This function requests the current configuration of boost power mode and alternate transmitter output.
- void emberGetTxPowerModeReturn (uint16_t txPowerMode)

  This function provides the result of a call to emberGetTxPowerMode() on the host.
- void emberSetSecurityParameters (const EmberSecurityParameters *parameters, uint16_t options)

  This function is called before forming or joining. Fails if already formed or joined or if the arguments are inconsistent with the stack (i.e. if ECC is wanted and we have no ECC).
- void emberSetSecurityParametersReturn (EmberStatus status)

  This function provides the result of a call to emberSetSecurityParameters().
- void emberSwitchToNextNetworkKey (void)

  This function changes MAC encryption over to the next key. Fails if there is no next network key.
- void emberSwitchToNextNetworkKeyReturn (EmberStatus status)

  This function provides the result of a call to emberSwitchToNextNetworkKey().

- void emberSwitchToNextNetworkKeyHandler (EmberStatus status)

  This function can be stubbed out on the SoC and host app. It is used by the NCP to update security on the driver when it is instructed to switch the network key by an over the air update.
- void emberGetVersions (void)

  This function gets various versions: The stack version name (versionName) The management version number (managementVersionNumber, if applicable, otherwise set to 0xFFFF) The stack version number (stackVersion↩ Number) The stack build number (stackBuildNumber) The version type (versionType) The date / time of the build (buildTimestamp)
- void emberGetVersionsReturn (const uint8_t *versionName, uint16_t managementVersionNumber, uint16↩ _t stackVersionNumber, uint16_t stackBuildNumber, EmberVersionType versionType, const uint8_t *build↩ Timestamp)

  Provides the result of a call to emberGetVersions().
- void emberSetCcaThreshold (int8_t threshold)

  This function sets the CCA threshold level - the noise floor above which the channel is normally considered busy. The threshold parameter is expected to be a signed 2's complement value, in dBm.
- void emberSetCcaThresholdReturn (EmberStatus status)

  This function provides the result of a call to emberSetCcaThreshold().
- void emberGetCcaThreshold (void)

  This function gets the current CCA threshold level.
- void emberGetCcaThresholdReturn (int8_t threshold)

  This function provides the result of a call to emberGetCcaThreshold().
- void emberMacPassthroughMessageHandler (PacketHeader header)

  Application handler to intercept "passthrough" packets and handle them at the application.
- bool emberMacPassthroughFilterHandler (uint8_t *macHeader)

  Application handler to define "passthrough" packets.
- bool emberMacRssiFilterHandler (uint8_t *macHeader)

  Application handler to filter 802.15.4 packets to be observed for signal strength.
- void emberMacRssiHandler (int8_t currentRssi)

  Gets the received signal strength indication (RSSI) for the last 802.15.4 packet received by the stack.
- void emberGetIndexedToken (EmberTokenId tokenId, uint8_t index)

  This function gets the indexed token stored in non-volatile memory on the Ember chip. The result is returned depending on the tokenId provided (see enum above) to the appropriate Return() API.
- void emberGetChannelCalDataTokenReturn (uint8_t lna, int8_t tempAtLna, uint8_t modDac, int8_t tempAt↩ ModDac)

  This function gets the token information for tokenId = EMBER_CHANNEL_CAL_DATA_TOKEN.
- void emberGetMfgToken (EmberMfgTokenId tokenId)

  This function gets the manufacturer token stored in non-volatile memory on the Ember chip.
- void emberGetMfgTokenReturn (EmberMfgTokenId tokenId, EmberStatus status, const uint8_t *tokenData, uint8_t tokenDataLength)

  This function provides the result of a call to emberGetMfgToken.
- void emberSetMfgToken (EmberMfgTokenId tokenId, const uint8_t *tokenData, uint8_t tokenDataLength)

  This function sets the manufacturer token stored in non-volatile memory on the Ember chip.
- void emberSetMfgTokenReturn (EmberMfgTokenId tokenId, EmberStatus status)

  This function provides the result of a call to emberSetMfgToken.
- void emberGetCtune (void)

  This function gets the CTUNE value. (Only valid on EFR32)
- void emberGetCtuneReturn (uint16_t tune, EmberStatus status)

  This function provides the result of a call to emberGetCtune.
- void emberSetCtune (uint16_t tune)

  This function changes the CTUNE value. Involves switching to HFRCO and turning off the HFXO temporarily. (Only valid on EFR32)
- void emberSetCtuneReturn (EmberStatus status)

This function provides the result of a call to emberSetCtune.

- void emberRegisterDropIncomingMessageCallback (bool(∗drop)(PacketHeader header, Ipv6Header ∗ipHeader))

  This function registers a callback function so that the application can define rules to drop incoming packets. The callback function MUST be of the form: bool func_name(PacketHeader header, Ipv6Header ∗ipHeader) { ... }.
- void emberRegisterSerialTransmitCallback (void(∗serialTransmit)(uint8_t type, PacketHeader header))

  This function registers a callback function so that the application can define serial transmit logic. This should only be used for NCPs, and will have no effect for SoCs. The callback function MUST be of the form: void uartTransmit(uint8_t type, Buffer b) { ... }.
- bool emberGetLocalIpAddress (uint8_t index, EmberIpv6Address ∗address)

  This function fetches one of the device IPv6 addresses into the supplied pointer. Since there may be multiple addresses, an index argument between 0 and EMBER_MAX_IPV6_ADDRESS_COUNT must be supplied.
- void emberGetRoutingLocator (void)

  This function fetches the Thread Routing Locator (RLOC).
- void emberGetRoutingLocatorReturn (const EmberIpv6Address ∗rloc)

  This function provides the result of a call to emberGetRoutingLocator.
- void emberSetLocalNetworkData (const uint8_t ∗networkData, uint16_t length)

  Sets the Network Data that describes the local node's Border Router and server capabilities. This is passed a set of Network Data TLVs that may include Prefix, Has Route, Border Router, Service and Server TLVS.
- void emberSetLocalNetworkDataReturn (EmberStatus status, uint16_t length)

  Provides the result of a call to ::emberSetServerNetworkData.
- void emberConfigureGateway (EmberBorderRouterTlvFlag borderRouterFlags, bool isStable, const uint8_t ∗prefix, const uint8_t prefixLengthInBits, uint8_t domainId, uint32_t preferredLifetime, uint32_t validLifetime)

  This function configures the border router behavior, such as whether this device has a default route to the Internet, and whether it have a prefix that can be used by network devices to configure routable addresses.
- void emberConfigureGatewayReturn (EmberStatus status)

  This function provides the result of a call to emberConfigureGateway.
- void emberSetNdData (const uint8_t ∗data, uint16_t length)
- void emberSetNdDataReturn (EmberStatus status, uint16_t length)

  This function provides the result of a call to emberSetNdData.
- void emberConfigureExternalRoute (EmberDefaultRouteTlvFlag extRouteFlags, bool isStable, const uint8_t ∗extRoutePrefix, uint8_t extRoutePrefixLengthInBits, uint8_t extRouteDomainId)

  This function defines an external route set, a route for a Thread network IPv6 packet that must traverse a border router and be forwarded to an exterior network.
- void emberConfigureExternalRouteReturn (EmberStatus status)

  This function provides the result of a call to emberConfigureExternalRoute.
- void emberAddressConfigurationChangeHandler (const EmberIpv6Address ∗address, uint32_t preferredLifetime, uint32_t validLifetime, uint8_t addressFlags)

  This function is called when a new address is configured on the application.
- void emberGetGlobalPrefixes (void)

  This function returns the list of global prefixes that we know about.
- void emberGetGlobalPrefixReturn (uint8_t flags, bool isStable, const uint8_t ∗prefix, uint8_t prefixLengthInBits, uint8_t domainId, uint32_t preferredLifetime, uint32_t validLifetime)

  This function provides the result of a call to ::emberGetGlobalPrefix.
- void emberDhcpServerChangeHandler (const uint8_t ∗prefix, uint8_t prefixLengthInBits, bool available)

  This function is called when the stack knows about a new dhcp server or if a dhcp server has become unavailable.
- void emberRequestDhcpAddress (const uint8_t ∗prefix, uint8_t prefixLengthInBits)

  The application can choose to request a new DHCP address when it is informed via emberDhcpServerChangeHandler of an available DHCP server.
- void emberRequestDhcpAddressReturn (EmberStatus status, const uint8_t ∗prefix, uint8_t prefixLengthInBits)

  This function provides the result of a call to emberRequestDhcpAddress.
- void emberSlaacServerChangeHandler (const uint8_t ∗prefix, uint8_t prefixLengthInBits, bool available)

This function is called when the stack knows about a new SLAAC prefix or if a SLAAC server has become unavailable.

- void emberRequestSlaacAddress (const uint8_t *prefix, uint8_t prefixLengthInBits)

  The application can choose to request a new SLAAC address when it is informed via emberSlaacServerChangeHandler of an available SLAAC prefix.
- void emberRequestSlaacAddressReturn (EmberStatus status, const uint8_t *prefix, uint8_t prefixLengthInBits)

  This function provides the result of a call to emberRequestSlaacAddress.
- void emberGetGlobalAddresses (const uint8_t *prefix, uint8_t prefixLengthInBits)

  This function returns the list of global addresses configured on this device.
- void emberGetGlobalAddressReturn (const EmberIpv6Address *address, uint32_t preferredLifetime, uint32_t validLifetime, uint8_t addressFlags)

  This function provides the result of a call to emberGetGlobalAddresses.
- void emberResignGlobalAddress (const EmberIpv6Address *address)

  This function resigns this IPv6 global address from this node. If this is a DHCP address, then the server is informed about it. If it is a SLAAC address, we remove it locally.
- void emberResignGlobalAddressReturn (EmberStatus status)

  This function provides the result of a call to emberResignGlobalAddress().
- void emberExternalRouteChangeHandler (const uint8_t *prefix, uint8_t prefixLengthInBits, bool available)

  This function is called when the stack knows about a border router that has an external route to a prefix.
- void emberNetworkDataChangeHandler (const uint8_t *networkData, uint16_t length)

  This function is called when the stack receives new Thread Network Data. The networkData argument may be NULL, in which case emberGetNetworkData can be used to obtain the new Thread Network Data.
- void emberGetNetworkData (uint8_t *networkDataBuffer, uint16_t bufferLength)

  This function is called to obtain the current Thread Network Data.
- void emberGetNetworkDataReturn (EmberStatus status, uint8_t *networkData, uint16_t bufferLength)

  This function provides the result of a call to emberGetNetworkData.
- EmberStatus emberDnsLookup (const uint8_t *domainName, uint8_t domainNameLength, const uint8_t *prefix64, EmberDnsResponseHandler responseHandler, uint8_t *appData, uint16_t appDataLength)

  This function initiates a DNS name lookup.
- void emberConfigureNetwork (const EmberNetworkParameters *parameters, uint16_t options)

  This function configures network parameters.
- void emberFormNetwork (const EmberNetworkParameters *parameters, uint16_t options, uint32_t channelMask)

  This function forms a new network.
- void emberFormNetworkReturn (EmberStatus status)

  A callback that indicates whether a prior call to emberFormNetwork() successfully initiated the form process. The status argument is either EMBER_SUCCSS, or EMBER_INVALID_CALL if resume was called when the network status was not EMBER_NO_NETWORK, or a scan was underway.
- void emberJoinNetwork (const EmberNetworkParameters *parameters, uint16_t options, uint32_t channelMask)

  This function joins an existing network.
- void emberJoinCommissioned (int8_t radioTxPower, EmberNodeType nodeType, bool requireConnectivity)

  This function joins an already-commissioned network.
- void emberJoinNetworkReturn (EmberStatus status)

  A callback that indicates whether the join process was successfully initiated via a prior call to emberJoinNetwork() or emberJoinCommissioned(). The possible EmberStatus values are: EMBER_SUCCESS, EMBER_BAD_ARGUMENT, or EMBER_INVALID_CALL (if join was called when the network status was something other than EMBER_NO_NETWORK).
- void emberResumeNetwork (void)

  This function resumes network operation after a reboot of the Ember micro.
- void emberResumeNetworkReturn (EmberStatus status)

  A callback that indicates whether a prior call to emberResumeNetwork() successfully initiated the resume process. The status argument is either EMBER_SUCCESS, or EMBER_INVALID_CALL if resume was called when the network status was not EMBER_SAVED_NETWORK, or while a scan was underway.

- void emberAttachToNetwork (void)

  On an end device, this initiates an attach with any available router-eligible devices in the network. This call must only be made if the network materials have been pre-commissioned on this device, or if previously completed obtaining the commissioning materials from another device.
- void emberAttachToNetworkReturn (EmberStatus status)

  A callback that indicates whether the attach process was successfully initiated via a prior call to emberAttachToNetwork(). The status argument is either EMBER_SUCCESS, or EMBER_INVALID_CALL if attach was called when the network status was not EMBER_JOINED_NETWORK_NO_PARENT, or while an attach was underway.
- void emberSetAddressHandler (const uint8_t *address)

  A callback that is generated when the host's address changes.
- void emberSetDriverAddressHandler (const uint8_t *address)

  A callback to the IP driver to tell it to change its address.
- void emberStartHostJoinClientHandler (const uint8_t *parentAddress)

  A callback to tell the host to start security commissioning.
- void emberSetNetworkKeysHandler (uint32_t sequence, const uint8_t *masterKey, uint32_t sequence2, const uint8_t *masterKey2)

  A callback to the IP driver to tell it the network keys.
- void emberSetCommProxyAppParametersHandler (const uint8_t *extendedPanId, const uint8_t *networkId, const uint8_t *ulaPrefix, uint16_t panId, uint8_t channel, const EmberEui64 *eui64, const EmberEui64 *macExtendedId, EmberNetworkStatus networkStatus)

  A callback to provide the commission-proxy-app on the host with the requisite network parameters.
- void emberSetCommProxyAppSecurityHandler (const uint8_t *masterKey, uint32_t sequenceNumber)

  A callback to provide the commission-proxy-app on the host with the requisite security material.
- void emberSetCommProxyAppAddressHandler (const uint8_t *address)

  A callback to provide the commission-proxy-app on the host with our mesh local address.
- void emberSetCommProxyAppPskcHandler (const uint8_t *pskc)

  A callback to provide the commission-proxy-app on the host with the pskc.
- void emberChangeNodeType (EmberNodeType newType)

  This function changes the node type of a joined device.
- void emberChangeNodeTypeReturn (EmberStatus status)

  This function provides the result of a call to emberChangeNodeType(): either EMBER_SUCCESS, or EMBER_INVALID_CALL.
- void emberBecomeCommissioner (const uint8_t *deviceName, uint8_t deviceNameLength)

  This function petitions to make this device the commissioner for the network. This will succeed if there is no active commissioner and fail if there is one.
- void emberBecomeCommissionerReturn (EmberStatus status)

  Return call for emberBecomeCommissioner(). The status is EMBER_SUCCESS if a petition was sent or EMBER_ERR_FATAL if some temporary resource shortage prevented doing so.
- void emberStopCommissioning (void)

  This function causes this device to cease being the active commissioner. This call always succeeds and has no return.
- void emberGetCommissioner (void)

  This function causes the stack to call emberCommissionerStatusHandler() to report the current commissioner status. This always succeeds and has no return.
- void emberAllowNativeCommissioner (bool on)

  This function causes the stack to allow a connection to a native commissioner.
- void emberAllowNativeCommissionerReturn (EmberStatus status)

  This function provides the result of a call to emberAllowNativeCommissioner(): either EMBER_SUCCESS or EMBER_INVALID_CALL.
- void emberSetCommissionerKey (const uint8_t *commissionerKey, uint8_t commissionerKeyLength)

  This function sets the key that a native commissioner must use to establish a connection to a Thread router. The commissionerKey argument is known as the "commissioning credential" in the Thread spec and must be between 6 and 255 bytes in length. Internally, it is hashed to derive the 16-byte Pre-Shared Key for the commissioner, known as the PSKc.

- void emberSetCommissionerKeyReturn (EmberStatus status)

  This function provides the result of a call to emberSetCommissionerKey(): either EMBER_SUCCESS or EMBER_↩ INVALID_CALL.
- void emberSetPskcHandler (const uint8_t *pskc)

  Handler to let application know that a PSKc TLV was successfully set.
- void emberCommissionerStatusHandler (uint16_t flags, const uint8_t *commissionerName, uint8_↩ t commissionerNameLength)

  This function reports on the current commissioner state.
- void emberSetJoiningMode (EmberJoiningMode mode, uint8_t length)

  This function sets the joining mode, clearing the steering data if steering is to be used.
- void emberAddSteeringEui64 (const EmberEui64 *eui64)

  This function adds the given EUI64 to the steering data if this device is the active commissioner; has no effect otherwise.
- void emberSendSteeringData (void)

  This function sends the current steering data to the network, enabling joining in the process.
- void emberSendSteeringDataReturn (EmberStatus status)

  This function provides the result of a call to emberSendSteeringData().
- void emberSetJoinKey (const EmberEui64 *eui64, const uint8_t *key, uint8_t keyLength)

  This function supplies the commissioner with the key a joining node will be using.
- void emberSetJoinKeyReturn (EmberStatus status)

  This function provides the result of a call to emberSetJoinKey().
- void emberEnableHostDtlsClient (bool enable)

  This function allows DTLS implementations on the host.
- void emberCommissionNetwork (uint8_t preferredChannel, uint32_t fallbackChannelMask, const uint8_↩ t *networkId, uint8_t networkIdLength, uint16_t panId, const uint8_t *ulaPrefix, const uint8_t *extended↩ PanId, const EmberKeyData *key, uint32_t keySequence)

  This function commissions the network.
- void emberCommissionNetworkReturn (EmberStatus status)

  This function provides the result of a call to emberCommissionNetwork.
- void emberPermitJoining (uint16_t durationSeconds)

  Deprecated, not for use by Thread networks. Tells the stack to allow other nodes to join the network with this node as their parent. Joining is initially disabled by default. This function may only be called when the network status is EMBER_JOINED.
- void emberPermitJoiningReturn (EmberStatus status)

  This function provides the result of a call to emberPermitJoining().
- void emberPermitJoiningHandler (bool joiningAllowed)

  This function informs the application when the permit joining value changes.
- void emberCustomHostToNcpMessage (const uint8_t *message, uint8_t messageLength)

  This function sends a custom message from the Host to the NCP.
- void emberCustomHostToNcpMessageHandler (const uint8_t *message, uint8_t messageLength)

  NCP handler called to process a custom message from the Host.
- void emberCustomNcpToHostMessage (const uint8_t *message, uint8_t messageLength)

  This function sends a custom message from the NCP to the Host.
- void emberCustomNcpToHostMessageHandler (const uint8_t *message, uint8_t messageLength)

  Host handler called to process a custom message from the NCP.
- void emberSetEui64 (const EmberEui64 *eui64)

  This function sets the EUI.
- void emberHostToNcpNoOp (const uint8_t *bytes, uint8_t bytesLength)

  This function sends a no-op with data payload from the Host to the NCP.
- void emberNcpToHostNoOp (const uint8_t *bytes, uint8_t bytesLength)

  This function sends a no-op with data payload from the NCP to the Host.
- void emberLeaderDataHandler (const uint8_t *leaderData)

A callback invoked when the leader data changes.

- void emberGetNetworkDataTlv (uint8_t type, uint8_t index)

  This function gets a Network Data TLV.
- void emberGetNetworkDataTlvReturn (uint8_t type, uint8_t index, uint8_t versionNumber, const uint8_t *tlv, uint8_t tlvLength)

  This function provides the result of a call to emberGetNetworkDataTlv().
- void emberEcho (const uint8_t *data, uint8_t length)

  Test command. Echo data to the NCP.
- void emberEchoReturn (const uint8_t *data, uint8_t length)

  Callback for a debug command. Provides the result of emberEcho.
- void emberAssertInfoReturn (const uint8_t *fileName, uint32_t lineNumber)

  Sent from the NCP to the host when an assert occurs.
- void emberStartXonXoffTest (void)
- bool emberPing (const uint8_t *destination, uint16_t id, uint16_t sequence, uint16_t length, uint8_t hopLimit)
- void emberEnableNetworkFragmentation (void)
- void emberHostJoinClientComplete (uint32_t keySequence, const uint8_t *key, const uint8_t *ulaPrefix)
- bool emberIpv6AddressToString (const EmberIpv6Address *src, uint8_t *dst, size_t dstSize)

  This function converts an EmberIpv6Address to a NUL-terminated string.
- bool emberIpv6PrefixToString (const EmberIpv6Address *src, uint8_t srcPrefixBits, uint8_t *dst, size_t dstSize)

  This function converts an EmberIpv6Address and prefix length to a NUL-terminated string.
- bool emberIpv6StringToAddress (const uint8_t *src, EmberIpv6Address *dst)

  This function converts a NUL-terminated string to an EmberIpv6Address.
- bool emberIpv6StringToPrefix (const uint8_t *src, EmberIpv6Address *dst, uint8_t *dstPrefixBits)

  This function converts a NUL-terminated string to an EmberIpv6Address with a prefix length.
- bool emberIsIpv6UnspecifiedAddress (const EmberIpv6Address *address)

  This function checks an EmberIpv6Address to see if it is set to all zeroes which represents an unspecified address.
- bool emberIsIpv6LoopbackAddress (const EmberIpv6Address *address)

  This function checks an EmberIpv6Address to see if it is all zeroes, except the last byte, which is set to one, representing the loopback address.
- void emberSetRadioHoldOff (bool enable)

  This function enables or disables Radio HoldOff support.
- void emberSetRadioHoldOffReturn (EmberStatus status)

  This function provides the result of a call to emberSetRadioHoldOff.
- void emberGetPtaEnable (void)

  This function fetches whether packet traffic arbitration is enabled or disabled.
- void emberGetPtaEnableReturn (bool enabled)

  This function provides the result of a call to emberGetPtaEnable.
- void emberSetPtaEnable (bool enabled)

  This function enables or disables packet traffic arbitration.
- void emberSetPtaEnableReturn (EmberStatus status)

  This function provides the result of a call to emberSetPtaEnable.
- void emberGetPtaOptions (void)

  This function fetches packet traffic arbitration configuration options.
- void emberGetPtaOptionsReturn (uint32_t options)

  This function provides the result of a call to emberGetPtaOptions.
- void emberSetPtaOptions (uint32_t options)

  This function configures packet traffic arbitration options.
- void emberSetPtaOptionsReturn (EmberStatus status)

  This function provides the result of a call to emberSetPtaOptions.
- void emberGetAntennaMode (void)

This function fetches the current antenna mode.

- void emberGetAntennaModeReturn (EmberStatus status, uint8_t mode)

  This function provides the result of a call to emberGetAntennaMode.
- void emberSetAntennaMode (uint8_t mode)

  This function configures the antenna mode.
- void emberSetAntennaModeReturn (EmberStatus status)

  This function provides the result of a call to emberSetAntennaMode.
- void emberRadioGetRandomNumbers (uint8_t count)

  This function gets a true random number out of radios that support this. This will typically take a while, and so should be used to seed a PRNG and not as a source of random numbers for regular use.
- void emberRadioGetRandomNumbersReturn (EmberStatus status, const uint16_t *rn, uint8_t count)

  This function provides the result of a call to emberRadioGetRandomNumbers.
- void emberMicroBusyHandler (bool busy)

  Callback informing the application running on the micro of interruptions to normal processing. If ::busy is true, the micro will be busy processing and unavailable for an indefinite period of time. If ::busy is false, the micro has resumed normal operation. The main use case is jpake crypto on EM3xx processors. This gives the application a chance to prepare for the pause in regular processing.

8.50.1 Detailed Description

This file contains network management API functions that are available on both host and SOC platforms.

Callback naming conventions: ...Handler() // These are used for asynchronous status notifications // as a result of a unilateral event on the stack.

...Return() // These are explicitly the result of an API call, and // constitute a command/return event initiated by the // application.

The network APIs with a command/response model were designed to be the same, both for SoC and host contexts. On a host, every command follows the asynchronous model as we have to wait for the NCP stack to respond via a callback. Because of this design, on SoCs, ...Return() callbacks may be called within the command request, and before such a request completes.

8.50.2 Macro Definition Documentation

8.50.2.1 #define EMBER_IPV6_ADDRESS_STRING_SIZE 40

8.50.2.2 #define EMBER_IPV6_BITS 128

8.50.2.3 #define EMBER_IPV6_BYTES (EMBER_IPV6_BITS / 8)

8.50.2.4 #define EMBER_IPV6_FIELDS (EMBER_IPV6_BYTES / 2)

8.50.2.5 #define EMBER_IPV6_MTU 1280

8.50.2.6 #define EMBER_IPV6_PREFIX_STRING_SIZE 44

8.50.2.7 #define EMBER_NETWORK_DATA_LEADER_SIZE 8

8.50.3 Function Documentation

8.50.3.1 void emberCustomHostToNcpMessage (const uint8_t * *message*, uint8_t *messageLength*)

Parameters

message	message to send
messageLength	length of message

8.50.3.2 void emberCustomNcpToHostMessage (const uint8_t * *message*, uint8_t *messageLength*)

Parameters

message	message to send
messageLength	length of message

8.50.3.3 void emberEcho (const uint8_t * *data*, uint8_t *length*)

8.50.3.4 void emberEnableNetworkFragmentation (void)

8.50.3.5 void emberGetAntennaMode (void)

8.50.3.6 void emberGetNetworkDataTlv (uint8_t *type*, uint8_t *index*)

Parameters

type	the type for requested TLV
index	if there are multiple TLVs of the given type then this value indicates which one to return. A value of 0 will return the first TLV of the given type.

8.50.3.7 void emberGetPtaEnable (void)

8.50.3.8 void emberGetPtaOptions (void)

8.50.3.9 void emberHostJoinClientComplete (uint32_t *keySequence*, const uint8_t * *key*, const uint8_t * *ulaPrefix*)

8.50.3.10 void emberHostToNcpNoOp (const uint8_t * *bytes*, uint8_t *bytesLength*)

Parameters

bytes	bytes of payload
bytesLength	length of payload

8.50.3.11 bool emberIpv6AddressToString (const EmberIpv6Address * *src*, uint8_t * *dst*, size_t *dstSize*)

Parameters

src	the EmberIpv6Address to convert
dst	the buffer where the string will be written
dstSize	the size of buffer in bytes

Returns

true if the address was converted.

8.50.3.12 bool emberIpv6PrefixToString (const EmberIpv6Address * *src*, uint8_t *srcPrefixBits*, uint8_t * *dst*, size_t *dstSize*)

Parameters

src	the EmberIpv6Address to convert
srcPrefixBits	the size of the prefix in bits
dst	the buffer where the string will be written
dstSize	the size of buffer in bytes

Returns

true if the prefix was converted.

8.50.3.13 bool emberIpv6StringToAddress (const uint8_t * *src*, EmberIpv6Address * *dst*)

Parameters

src	the string to convert
dst	the EmberIpv6Address where the address will be written

Returns

true if the string was converted.

8.50.3.14 bool emberIpv6StringToPrefix (const uint8_t * *src*, EmberIpv6Address * *dst*, uint8_t * *dstPrefixBits*)

Parameters

src	the string to convert
dst	the EmberIpv6Address where the address will be written
dstPrefixBits	the number of prefix bits in the string

Returns

true if the string was converted.

8.50.3.15 bool emberIsIpv6LoopbackAddress (const EmberIpv6Address * *address*)

Parameters

address	the EmberIpv6Address to check

Returns

true if the address is zero for bytes 0-14 and one for byte 15

8.50.3.16 bool emberIsIpv6UnspecifiedAddress (const EmberIpv6Address * *address*)

Parameters

address	the EmberIpv6Address to check

Returns

true if the address is all zeroes (unspecified address).

8.50.3.17 void emberNcpToHostNoOp (const uint8_t * *bytes*, uint8_t *bytesLength*)

Parameters

bytes	bytes of payload
bytesLength	length of payload

8.50.3.18 void emberPermitJoining (uint16_t *durationSeconds*)

Parameters

durationSeconds	A value of 0 disables joining. Any other value enables joining for that number of seconds.

8.50.3.19 void emberPermitJoiningHandler (bool *joiningAllowed*)

Parameters

joiningAllowed	Set to true when permit joining is allowed.

8.50.3.20 void emberPermitJoiningReturn (EmberStatus *status*)

8.50.3.21 bool emberPing (const uint8_t * *destination*, uint16_t *id*, uint16_t *sequence*, uint16_t *length*, uint8_t *hopLimit*)

8.50.3.22 void emberRadioGetRandomNumbers (uint8_t *count*)

Parameters

count	- the count of uint16_t values to be returned.

8.50.3.23 void emberSetAntennaMode (uint8_t *mode*)

Parameters

mode	0-primary, 1-secondary, 2-toggle on tx ack fail

8.50.3.24 void emberSetEui64 (const EmberEui64 * *eui64*)

Parameters

eui64	Value of EUI to be set.

8.50.3.25 void emberSetPtaEnable (bool *enabled*)

Parameters

enabled	When true, enables packet traffic arbitration. When false, disables packet traffic arbitration.

8.50.3.26 void emberSetPtaOptions (uint32_t *options*)

Parameters

indicates	packet traffic arbitration options bit field. Field Bit Position Size(bits) RX retry timeout ms 0 8 Enable ack radio holdoff 8 1 Abort mid TX if grant is lost 9 1 TX request is high priority 10 1 RX request is high prioirity 11 1 RX retry request is high priority 12 1 RX retry request is enabled 13 1 Radio holdoff is enabled 14 1 Toggle request on mac retransmit 15 1 Reserved 16 15 Hold request across CCA failures 31 1

8.50.3.27 void emberSetRadioHoldOff (bool *enable*)

Parameters

enable	When true, configures ::RHO_GPIO in BOARD_HEADER as an input which, when asserted, will prevent the radio from transmitting. When false, configures ::RHO_GPIO for its original default purpose.

8.50.3.28 void emberStartXonXoffTest (void)

Need documentation comments for these.

8.51 nvic-config.h File Reference

8.52 ota-bootload-client.h File Reference

```
#include <PLATFORM_HEADER>
#include <CONFIGURATION_HEADER>
#include <EMBER_AF_API_ZCL_CORE>
```

Data Structures

- struct EmberZclOtaBootloadClientServerInfo_t

8.53 ota-bootload-core.h File Reference

```
#include <PLATFORM_HEADER>
#include <CONFIGURATION_HEADER>
#include <EMBER_AF_API_ZCL_CORE>
```

Data Structures

- struct EmberZclOtaBootloadHardwareVersionRange_t
- struct EmberZclOtaBootloadFileSpec_t
- struct EmberZclOtaBootloadFileHeaderInfo_t

Macros

- #define EMBER_ZCL_OTA_BOOTLOAD_FILE_MAGIC_NUMBER 0x0BEEF11E
- #define EMBER_ZCL_OTA_BOOTLOAD_FILE_MAGIC_NUMBER_SIZE 4
- #define EMBER_ZCL_OTA_BOOTLOAD_FILE_VERSION 0x2000
- #define EMBER_ZCL_OTA_BOOTLOAD_HEADER_STRING_SIZE 32
- #define EMBER_ZCL_OTA_BOOTLOAD_HEADER_MAX_SIZE 93
- #define EMBER_ZCL_OTA_BOOTLOAD_FILE_VERSION_NULL ((EmberZclOtaBootloadFileVersion_t)-1)
- #define EMBER_ZCL_OTA_BOOTLOAD_HARDWARE_VERSION_NULL ((EmberZclOtaBootload↩ HardwareVersion_t)-1)

Typedefs

- typedef uint32_t EmberZclOtaBootloadFileVersion_t
- typedef uint16_t EmberZclOtaBootloadHardwareVersion_t

Enumerations

- enum EmberZclOtaBootloadFileHeaderFieldControl_t {
  EMBER_ZCL_OTA_BOOTLOAD_FILE_HEADER_FIELD_CONTROL_DESTINATION = 0x0002,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_HEADER_FIELD_CONTROL_HARDWARE_VERSION = 0x0004,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_HEADER_FIELD_CONTROL_NULL = ((EmberZclOtaBootload↩ FileHeaderFieldControl_t)-1) }
- enum EmberZclOtaBootloadFileType_t {
  EMBER_ZCL_OTA_BOOTLOAD_FILE_TYPE_MANUFACTURER_SPECIFIC_MAXIMUM = 0xFFBF,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_TYPE_SECURITY_CREDENTIALS = 0xFFC0,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_TYPE_CONFIGURATION = 0xFFC1,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_TYPE_LOG = 0xFFC2,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_TYPE_PICTURE = 0xFFC3,
  EMBER_ZCL_OTA_BOOTLOAD_FILE_TYPE_WILDCARD = 0xFFFF }
- enum EmberZclOtaBootloadStackVersion_t {
  EMBER_ZCL_OTA_BOOTLOAD_STACK_VERSION_IP = 0x0004,
  EMBER_ZCL_OTA_BOOTLOAD_STACK_VERSION_NONE = ((EmberZclOtaBootloadStackVersion_t)-1) }

- enum EmberZclOtaBootloadSecurityCredentialVersion_t { EMBER_ZCL_OTA_BOOTLOAD_SECURITY_CREDENTIAL_VERSION_IP = 0x03, EMBER_ZCL_OTA_BOOTLOAD_SECURITY_CREDENTIAL_VERSION_NULL = ((EmberZclOta↩ BootloadSecurityCredentialVersion_t)-1) }
- enum EmberZclOtaBootloadFileStatus_t { EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_VALID = 0x00, EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_INVALID_MAGIC_NUMBER = 0x01, EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_INVALID_VERSION = 0x02, EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_INVALID_HEADER_SIZE = 0x03, EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_INVALID_STACK_VERSION = 0x04, EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_INVALID_SECURITY_CREDENTIAL_VERSION = 0x05, EMBER_ZCL_OTA_BOOTLOAD_FILE_STATUS_NULL = 0xFF }

Functions

- void emberZclOtaBootloadInitFileHeaderInfo (EmberZclOtaBootloadFileHeaderInfo_t *headerInfo)
- bool emberZclOtaBootloadFileSpecsAreEqual (const EmberZclOtaBootloadFileSpec_t *s1, const Ember↩ ZclOtaBootloadFileSpec_t *s2)
- size_t emberZclOtaBootloadFetchFileSpec (const uint8_t *data, EmberZclOtaBootloadFileSpec_t *fileSpec)
- size_t emberZclOtaBootloadStoreFileSpec (const EmberZclOtaBootloadFileSpec_t *fileSpec, uint8_t *data)
- EmberZclOtaBootloadFileStatus_t emberZclOtaBootloadFetchFileHeaderInfo (const uint8_t *data, Ember↩ ZclOtaBootloadFileHeaderInfo_t *fileHeaderInfo)
- EmberZclOtaBootloadFileStatus_t emberZclOtaBootloadStoreFileHeaderInfo (uint8_t *data, EmberZclOta↩ BootloadFileHeaderInfo_t *fileHeaderInfo, size_t imageDataSize)

Variables

- const EmberZclOtaBootloadFileSpec_t emberZclOtaBootloadFileSpecNull

8.54 ota-bootload-storage-core.h File Reference

```
#include <PLATFORM_HEADER>
#include <CONFIGURATION_HEADER>
#include <EMBER_AF_API_ZCL_CORE>
#include <EMBER_AF_API_ZCL_OTA_BOOTLOAD_CORE>
```

Data Structures

- struct EmberZclOtaBootloadStorageInfo_t
- struct EmberZclOtaBootloadStorageFileInfo_t

Typedefs

- typedef void(* EmberZclOtaBootloadStorageDeleteCallback) (EmberZclOtaBootloadStorageStatus_t)

Enumerations

- enum EmberZclOtaBootloadStorageStatus_t {
  EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_SUCCESS = 0x00,
  EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_FAILED = 0x01,
  EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_OUT_OF_RANGE = 0x02,
  EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_INVALID_FILE = 0x03,
  EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_OUT_OF_SPACE = 0x04,
  EMBER_ZCL_OTA_BOOTLOAD_STORAGE_STATUS_NULL = 0xFF }

Functions

- void emberZclOtaBootloadStorageGetInfo (EmberZclOtaBootloadStorageInfo_t *info, EmberZclOta↩BootloadFileSpec_t *returnedFiles, size_t returnedFilesMaxCount)
- EmberZclOtaBootloadStorageStatus_t emberZclOtaBootloadStorageFind (const EmberZclOtaBootload↩FileSpec_t *fileSpec, EmberZclOtaBootloadStorageFileInfo_t *fileInfo)
- EmberZclOtaBootloadStorageStatus_t emberZclOtaBootloadStorageCreate (const EmberZclOtaBootload↩FileSpec_t *fileSpec)
- EmberZclOtaBootloadStorageStatus_t emberZclOtaBootloadStorageRead (const EmberZclOtaBootload↩FileSpec_t *fileSpec, size_t offset, void *data, size_t dataLength)
- EmberZclOtaBootloadStorageStatus_t emberZclOtaBootloadStorageWrite (const EmberZclOtaBootload↩FileSpec_t *fileSpec, size_t offset, const void *data, size_t dataLength)
- EmberZclOtaBootloadStorageStatus_t emberZclOtaBootloadStorageDelete (const EmberZclOtaBootload↩FileSpec_t *fileSpec, EmberZclOtaBootloadStorageDeleteCallback callback)

8.55 platform-common.h File Reference

Macros

- #define MEMSET(d, v, l) memset(d, v, l)

  Friendly convenience macro pointing to the C Stdlib functions.
- #define MEMCOPY(d, s, l) memcpy(d, s, l)
- #define MEMMOVE(d, s, l) memmove(d, s, l)
- #define MEMPGMCOPY(d, s, l) memcpy(d, s, l)
- #define MEMCOMPARE(s0, s1, l) memcmp(s0, s1, l)
- #define MEMPGMCOMPARE(s0, s1, l) memcmp(s0, s1, l)

Generic Types

- #define TRUE 1

  An alias for one, used for clarity.
- #define FALSE 0

  An alias for zero, used for clarity.
- #define NULL ((void *)0)

  The null pointer.

Bit Manipulation Macros

- #define BIT(x) (1U << (x))

  Useful to reference a single bit of a byte.
- #define BIT32(x) (((uint32_t) 1) << (x))

  Useful to reference a single bit of an uint32_t type.

- #define SETBIT(reg, bit) (reg) |= BIT(bit)
  Sets `bit` in the `reg` register or byte.
- #define SETBITS(reg, bits) (reg) |= (bits)
  Sets the bits in the `reg` register or the byte as specified in the bitmask `bits`.
- #define CLEARBIT(reg, bit) (reg) &= ~(BIT(bit))
  Clears a bit in the `reg` register or byte.
- #define CLEARBITS(reg, bits) (reg) &= ~(bits)
  Clears the bits in the `reg` register or byte as specified in the bitmask `bits`.
- #define READBIT(reg, bit) ((reg) & (BIT(bit)))
  Returns the value of `bit` within the register or byte `reg`.
- #define READBITS(reg, bits) ((reg) & (bits))
  Returns the value of the bitmask `bits` within the register or byte `reg`.

Byte Manipulation Macros

- #define LOW_BYTE(n) ((uint8_t)((n) & 0xFF))
  Returns the low byte of the 16-bit value `n` as an `uint8_t`.
- #define HIGH_BYTE(n) ((uint8_t)(LOW_BYTE((n) >> 8)))
  Returns the high byte of the 16-bit value `n` as an `uint8_t`.
- #define HIGH_LOW_TO_INT(high, low)
  Returns the value built from the two `uint8_t` values `high` and `low`.
- #define INT8U_TO_INT32U(byte3, byte2, byte1, byte0)
  Returns the value built from the four `uint8_t` as an `uint32_t`.
- #define BYTE_0(n) ((uint8_t)((n) & 0xFF))
  Returns the low byte of the 32-bit value `n` as an `uint8_t`.
- #define BYTE_1(n) BYTE_0((n) >> 8)
  Returns the second byte of the 32-bit value `n` as an `uint8_t`.
- #define BYTE_2(n) BYTE_0((n) >> 16)
  Returns the third byte of the 32-bit value `n` as an `uint8_t`.
- #define BYTE_3(n) BYTE_0((n) >> 24)
  Returns the high byte of the 32-bit value `n` as an `uint8_t`.
- #define BYTE_4(n) BYTE_0((n) >> 32)
  Returns the fifth byte of the 64-bit value `n` as an `uint8_t`.
- #define BYTE_5(n) BYTE_0((n) >> 40)
  Returns the sixth byte of the 64-bit value `n` as an `uint8_t`.
- #define BYTE_6(n) BYTE_0((n) >> 48)
  Returns the seventh byte of the 64-bit value `n` as an `uint8_t`.
- #define BYTE_7(n) BYTE_0((n) >> 56)
  Returns the high byte of the 64-bit value `n` as an `uint8_t`.
- #define COUNTOF(a) (sizeof(a) / sizeof(a[0]))
  Returns the number of entries in an array.

Time Manipulation Macros

- #define elapsedTimeInt8u(oldTime, newTime) ((uint8_t) ((uint8_t)(newTime) - (uint8_t)(oldTime)))
  Returns the elapsed time between two 8 bit values. Result may not be valid if the time samples differ by more than 127.
- #define elapsedTimeInt16u(oldTime, newTime) ((uint16_t) ((uint16_t)(newTime) - (uint16_t)(oldTime)))
  Returns the elapsed time between two 16 bit values. Result may not be valid if the time samples differ by more than 32767.
- #define elapsedTimeInt32u(oldTime, newTime) ((uint32_t) ((uint32_t)(newTime) - (uint32_t)(oldTime)))
  Returns the elapsed time between two 32 bit values. Result may not be valid if the time samples differ by more than 2147483647.
- #define MAX_INT8U_VALUE (0xFF)
  Returns true if t1 is greater than t2. Can only account for 1 wrap around of the variable before it is wrong.
- #define HALF_MAX_INT8U_VALUE (0x80)
  Returns the elapsed time between two 8 bit values. Result may not be valid if the time samples differ by more than 127.

- #define timeGTorEqualInt8u(t1, t2) (elapsedTimeInt8u(t2, t1) <= (HALF_MAX_INT8U_VALUE))
  Returns the elapsed time between two 8 bit values. Result may not be valid if the time samples differ by more than 127.
- #define MAX_INT16U_VALUE (0xFFFF)
  Returns true if t1 is greater than t2. Can only account for 1 wrap around of the variable before it is wrong.
- #define HALF_MAX_INT16U_VALUE (0x8000)
  Returns the elapsed time between two 8 bit values. Result may not be valid if the time samples differ by more than 127.
- #define timeGTorEqualInt16u(t1, t2) (elapsedTimeInt16u(t2, t1) <= (HALF_MAX_INT16U_VALUE))
  Returns the elapsed time between two 8 bit values. Result may not be valid if the time samples differ by more than 127.
- #define MAX_INT32U_VALUE (0xFFFFFFFFUL)
  Returns true if t1 is greater than t2. Can only account for 1 wrap around of the variable before it is wrong.
- #define HALF_MAX_INT32U_VALUE (0x80000000UL)
  Returns the elapsed time between two 8 bit values. Result may not be valid if the time samples differ by more than 127.
- #define timeGTorEqualInt32u(t1, t2) (elapsedTimeInt32u(t2, t1) <= (HALF_MAX_INT32U_VALUE))
  Returns the elapsed time between two 8 bit values. Result may not be valid if the time samples differ by more than 127.

Miscellaneous Macros

- #define UNUSED_VAR(x) (void)(x)
- #define DEBUG_LEVEL BASIC_DEBUG
  Set debug level based on whether DEBUG or DEBUG_STRIPPED are defined.
- #define STATIC_ASSERT(__condition, __errorstr)
  Disable static assertions on compilers that don't support them.

8.55.1 Detailed Description

See Common PLATFORM_HEADER Configuration for detailed documentation.

8.56 random.h File Reference

Functions

- void halStackSeedRandom (uint32_t seed)
  Seeds the halCommonGetRandom() pseudorandom number generator.
- uint16_t halCommonGetRandom (void)
  Runs a standard LFSR to generate pseudorandom numbers.

8.56.1 Detailed Description

See Random Number Generation for detailed documentation.

8.57 reset-def.h File Reference

Definitions for all the reset cause types.

8.58 rtos-ipc-link.h File Reference

Macros

- #define IP_MODEM_LINK_DATA_INPUT_QUEUE_LEN 5
- #define IP_MODEM_LINK_MGMT_OUTPUT_QUEUE_LEN 2
- #define IP_MODEM_LINK_MGMT_INPUT_QUEUE_LEN 2

Functions

- void rtosIpcLinkInit (void)
- void hostAppProcessManagementCommands (void)
- bool hostAppSuspend (uint32_t timeoutMs)
- bool hostAppIdle (uint32_t timeoutMs)

8.58.1 Macro Definition Documentation

8.58.1.1 #define IP_MODEM_LINK_DATA_INPUT_QUEUE_LEN 5

8.58.1.2 #define IP_MODEM_LINK_MGMT_INPUT_QUEUE_LEN 2

8.58.1.3 #define IP_MODEM_LINK_MGMT_OUTPUT_QUEUE_LEN 2

8.58.2 Function Documentation

8.58.2.1 bool hostAppIdle (uint32_t *timeoutMs*)

8.58.2.2 void hostAppProcessManagementCommands (void)

8.58.2.3 bool hostAppSuspend (uint32_t *timeoutMs*)

8.58.2.4 void rtosIpcLinkInit (void)

8.59 serial.h File Reference

Serial hardware abstraction layer interfaces. See Serial UART Communication for documentation.

```
#include "stack/include/ember-types.h"
```

Macros

Serial Mode Definitions

These are numerical definitions for the possible serial modes so that code can test for the one being used. There may be additional modes defined in the micro-specific micro.h.

- #define EMBER_SERIAL_UNUSED 0

  A numerical definition for a possible serial mode the code can test for.
- #define EMBER_SERIAL_FIFO 1

  A numerical definition for a possible serial mode the code can test for.
- #define EMBER_SERIAL_BUFFER 2

  A numerical definition for a possible serial mode the code can test for.
- #define EMBER_SERIAL_LOWLEVEL 3

  A numerical definition for a possible serial mode the code can test for.

FIFO Utility Macros

These macros manipulate the FIFO queue data structures to add and remove data.

- #define FIFO_ENQUEUE(queue, data, size)

  Macro that enqueues a byte of data in a FIFO queue.
- #define FIFO_DEQUEUE(queue, size)

  Macro that de-queues a byte of data from a FIFO queue.

Enumerations

- enum SerialBaudRate {
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0,
  DEFINE_BAUD =(300) = 0 }

  Assign numerical values for variables that hold Baud Rate parameters.
- enum SerialParity {
  DEFINE_PARITY =(NONE) = 0U,
  DEFINE_PARITY =(NONE) = 0U,
  DEFINE_PARITY =(NONE) = 0U }

  CORTEXM3_EFM32_MICRO.

Functions

- void halHostFlushBuffers (void)
- uint16_t halHostEnqueueTx (const uint8_t *data, uint16_t length)
- void halHostFlushTx (void)
- uint16_t serialCopyFromRx (const uint8_t *data, uint16_t length)
- void emLoadSerialTx (void)

Buffered Serial Utility APIs

The higher-level serial code implements these APIs, which the HAL uses to deal with buffered serial output.

- void emSerialBufferNextMessageIsr (EmSerialBufferQueue *q)

  When new serial transmission is started and `bufferQueue->nextByte` is equal to NULL, this can be called to set up `nextByte` and `lastByte` for the next message.
- void emSerialBufferNextBlockIsr (EmSerialBufferQueue *q, uint8_t port)

  When a serial transmission is in progress and `bufferQueue->nextByte` has been sent and incremented leaving it equal to lastByte, this should be called to set up `nextByte` and `lastByte` for the next block.

Virtual UART API

API used by the stack in debug builds to receive data arriving over the virtual UART.

- void halStackReceiveVuartMessage (uint8_t *data, uint8_t length)

  When using a debug build with virtual UART support, this API is called by the stack when virtual UART data has been received over the debug channel.

Serial HAL APIs

These functions must be implemented by the HAL in order for the serial code to operate. Only the higher-level serial code uses these functions, so they should not be called directly. The HAL should also implement the appropriate interrupt handlers to drain the TX queues and fill the RX FIFO queue.

- #define halInternalUartFlowControl(port) do {} while (false)

  This function is used in FIFO mode when flow control is enabled. It is called from emberSerialReadByte(), and based on the number of bytes used in the uart receive queue, decides when to tell the host it may resume transmission.
- #define halInternalUartRxPump(port) do {} while (false)

  This function exists only in software UART (SOFTUART) mode on the EM3xx. This function is called by ::ember↩ SerialReadByte(). It is responsible for maintaining synchronization between the emSerialRxQueue and the UART DMA.
- #define halInternalUart1FlowControlRxIsEnabled() halInternalUartFlowControlRxIsEnabled(1)

  This function is used in FIFO mode when flow control is enabled. It is called from emberSerialReadByte(), and based on the number of bytes used in the uart receive queue, decides when to tell the host it may resume transmission.
- #define halInternalUart1XonRefreshDone() halInternalUartXonRefreshDone(1)

  This function is used in FIFO mode when flow control is enabled. It is called from emberSerialReadByte(), and based on the number of bytes used in the uart receive queue, decides when to tell the host it may resume transmission.
- #define halInternalUart1TxIsIdle() halInternalUartTxIsIdle(1)

  This function is used in FIFO mode when flow control is enabled. It is called from emberSerialReadByte(), and based on the number of bytes used in the uart receive queue, decides when to tell the host it may resume transmission.
- EmberStatus halInternalUartInit (uint8_t port, SerialBaudRate rate, SerialParity parity, uint8_t stopBits)

  Initializes the UART to the given settings (same parameters as ::emberSerialInit()).
- void halInternalPowerDownUart (void)

  This function is typically called by halPowerDown() and it is responsible for performing all the work internal to the UART needed to stop the UART before a sleep cycle.
- void halInternalPowerUpUart (void)

This function is typically called by halPowerUp() and it is responsible for performing all the work internal to the UART needed to restart the UART after a sleep cycle.

- void halInternalStartUartTx (uint8_t port)

  Called by serial code whenever anything is queued for transmission to start any interrupt-driven transmission. May be called when transmission is already in progess.
- void halInternalStopUartTx (uint8_t port)

  Called by serial code to stop any interrupt-driven serial transmission currently in progress.
- EmberStatus halInternalForceWriteUartData (uint8_t port, uint8_t *data, uint8_t length)

  Directly writes a byte to the UART for transmission, regardless of anything currently queued for transmission. Should wait for anything currently in the UART hardware registers to finish transmission first, and block until `data` is finished being sent.
- EmberStatus halInternalForceReadUartByte (uint8_t port, uint8_t *dataByte)

  Directly reads a byte from the UART for reception, regardless of anything currently queued for reception. Does not block if a data byte has not been received.
- void halInternalWaitUartTxComplete (uint8_t port)

  Blocks until the UART has finished transmitting any data in its hardware registers.
- void halInternalRestartUart (void)

  This function is typically called by halInternalPowerUpBoard() and it is responsible for performing all the work internal to the UART needed to restart the UART after a sleep cycle. (For example, resyncing the DMA hardware and the serial FIFO.)
- bool halInternalUartFlowControlRxIsEnabled (uint8_t port)

  Checks to see if the host is allowed to send serial data to the ncp - i.e., it is not being held off by nCTS or an XOFF. Returns true is the host is able to send.
- bool halInternalUartXonRefreshDone (uint8_t port)

  When Xon/Xoff flow control is used, returns true if the host is not being held off and XON refreshing is complete.
- bool halInternalUartTxIsIdle (uint8_t port)

  Returns true if the uart transmitter is idle, including the transmit shift register.
- bool serialDropPacket (void)

  Testing function implemented by the upper layer. Determines whether the next packet should be dropped. Returns true if the next packet should be dropped, false otherwise.

8.60 sim-eeprom.h File Reference

Simulated EEPROM system for wear leveling token storage across flash. See Simulated EEPROM for documentation.

Functions

- void halSimEepromCallback (EmberStatus status)

  The Simulated EEPROM callback function, implemented by the application.
- uint8_t halSimEepromErasePage (void)

  Erases a hardware flash page, if needed.
- uint8_t halSimEepromPagesRemainingToBeErased (void)

  Get count of pages to be erased.
- void halSimEepromStatus (uint16_t *freeWordsUntilFull, uint16_t *totalPageUseCount)

  Provides two basic statistics.

8.61 stack-info.h File Reference

SOC-only radio APIs.

Functions

- EmberStatus emberSetRadioChannel (uint8_t channel)

  This function sets the channel 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.
- uint8_t emberGetRadioChannel (void)

  This function gets the radio channel to which a node is set. The possible return values depend on the radio in use. For a list of available radio channels, see the technical specification for the RF communication module in your Developer Kit.

Radio-specific Functions

- void emberRadioNeedsCalibratingHandler (void)

  This function enables boost power mode and/or the alternate transmit path.
- void emberCalibrateCurrentChannel (void)

  This function calibrates the current channel. The stack will notify the application of the need for channel calibration via the emberRadioNeedsCalibratingHandler() callback function during emberTick(). This function should only be called from within the context of the emberRadioNeedsCalibratingHandler() callback function. Calibration can take up to 150 ms. Note, if this function is called when the radio is off, it will turn the radio on and leave it on.

8.62 standalone-bootloader.h File Reference

Functions

Required Custom Functions

- void bootloaderMenu (void)

  This function must be implemented, providing a bootloader menu.

Available Bootloader Library Functions

Functions implemented by the bootloader library that may be used by custom functions.

- BL_Status receiveImage (uint8_t commState)

  Puts the bootloader into a mode where it will receive an image. commState indicates whether the image is received via serial (COMM_SERIAL) or over the air (COMM_RADIO)
- bool checkDebugMenuOption (uint8_t ch)

  A hook to the bootloader library for it to check for extra menu options. Only used for ember internal debug builds, not normally needed.
- BL_Status initOtaState (void)

  Initialize OTA Bootloader state.
- BL_Status checkOtaStart (void)

  Check to see if the bootloader has detected an OTA upload start.
- BL_Status receiveOtaImage (void)

  Puts the bootloader into a mode where it will receive an image over the air. The function checkOtaStart() should have been called first and it should have returned with a status of BL_SUCCESS before calling this function.
- bool paIsPresent (void)

  Uses the information in the PHY_CONFIG token to determine if a power amplifier is present in the node design.
- bool halCheckIntegrity (void)

  Validate application integrity by running AES-MMO hash and comparing to AAT.

8.62.1 Detailed Description

See Standalone for detailed documentation.

8.63 symbol-timer.h File Reference

Functions that provide access to symbol time. One symbol period is 16 microseconds.

Functions

Symbol Timer Functions

- void halInternalStartSymbolTimer (void)

  Initializes the symbol timer. When a dedicated symbol timer peripheral exists (e.g. EM2xx, EM3xx) this initialization is generally performed directly by the PHY, so this routine may be a no-op.
- uint32_t halStackGetInt32uSymbolTick (void)

  Returns the current symbol time in symbol ticks (units are platform-dependent, but typically on the order of microseconds).
- bool halStackInt32uSymbolTickGTorEqual (uint32_t st1, uint32_t st2)

  Returns true if symbol tick time st1 is greater than symbol tick time st2, as determined by half the range of the symbol timer. Can only account for 1 wrap around between st1 and st2 before it is wrong.
- uint32_t halStackGetSymbolTicksPerSecond (void)

  Obtains the number of symtol timer ticks in one second of real time. Can be used for conversion between real time and symbol ticks.

MAC Timer Support Functions

These functions are used for MAC layer timing and symbol-based delays.

Applications should not directly call these functions. They are used internally by the operation of the stack.

- enum EmHalSymbolDelayChannel_t {
  EM_HAL_SYMBOL_DELAY_CHANNEL_A,
  EM_HAL_SYMBOL_DELAY_CHANNEL_B,
  EM_HAL_SYMBOL_DELAY_CHANNELS }

  Specifies two independent channels for symbol delay operations.
- typedef void(∗ EmHalSymbolDelayCallback_t) (EmHalSymbolDelayChannel_t delayChan)

  Specifies the callback API triggered when the symbol timer channel expires.
- uint32_t halStackOrderSymbolDelay (EmHalSymbolDelayChannel_t delayChan, EmHalSymbolDelay↩Callback_t callback, uint32_t microseconds)

  Sets up a delay timer to call the indicated interrupt-context callback when it expires.
- void halStackCancelSymbolDelay (EmHalSymbolDelayChannel_t delayChan, EmHalSymbolDelay↩Callback_t callback)

  Cancels the delay set up by an earlier halStackOrderSymbolDelay() call.
- void halStackOrderInt16uSymbolDelayA (uint16_t symbols)

  Sets up a timer and calls an interrupt-context callback when it expires.
- void halStackCancelSymbolDelayA (void)

  Cancels the timer set up by halStackOrderInt16uSymbolDelayA().
- void halStackSymbolDelayAIsr (void)

  This is the interrupt level callback into the stack that is called when the timers set by halStackOrderInt16uSymbol↩DelayA expire.

8.63.1 Detailed Description

See Symbol Timer Control for documentation.

8.64 system-timer.h File Reference

Macros

- #define halIdleForMilliseconds(duration) halCommonIdleForMilliseconds((duration))

Functions

- uint16_t halInternalStartSystemTimer (void)

  Initializes the system tick.
- uint16_t halCommonGetInt16uMillisecondTick (void)

  Returns the current system time in system ticks, as a 16-bit value.
- uint32_t halCommonGetInt32uMillisecondTick (void)

  Returns the current system time in system ticks, as a 32-bit value.
- uint16_t halCommonGetInt16uQuarterSecondTick (void)

  Returns the current system time in quarter second ticks, as a 16-bit value.
- EmberStatus halSleepForQuarterSeconds (uint32_t *duration)

  Uses the system timer to enter SLEEPMODE_WAKETIMER for approximately the specified amount of time (provided in quarter seconds).
- EmberStatus halSleepForMilliseconds (uint32_t *duration)

  Uses the system timer to enter SLEEPMODE_WAKETIMER for approximately the specified amount of time (provided in milliseconds). Note that since the system timer ticks at a rate of 1024Hz, a second is comprised of 1024 milliseconds in this function.
- EmberStatus halCommonIdleForMilliseconds (uint32_t *duration)

  Uses the system timer to enter SLEEPMODE_IDLE for approximately the specified amount of time (provided in milliseconds).

8.64.1 Detailed Description

See System Timer Control for documentation.

8.65 Thread_API.md File Reference

8.66 tmsp-enum.h File Reference

Macros

- #define TMSP_VERSION 0x0E00
- #define TMSP_MFGLIB_ACTIVITIES_MAX 0x01
- #define TMSP_MFGLIB_VALUES_MAX 0x04
- #define SET_LARGE_VALUE_ID_MAX 0x01

Typedefs

- typedef uint8_t TmspMfglibActivities
- typedef uint8_t TmspMfglibValues
- typedef uint8_t SetLargeValueId

Enumerations

- enum {
  TMSP_MFGLIB_TONE = 0x00,
  TMSP_MFGLIB_STREAM = 0x01 }

- enum {
  TMSP_MFGLIB_CHANNEL = 0x00,
  TMSP_MFGLIB_POWER = 0x01,
  TMSP_MFGLIB_POWER_MODE = 0x02,
  TMSP_MFGLIB_SYN_OFFSET = 0x03,
  TMSP_MFGLIB_OPTIONS = 0x04 }

- enum {
  SET_ND_DATA = 0x00,
  SET_LOCAL_NETWORK_DATA = 0x01 }

- enum {

EMBER_RESET_MICRO_COMMAND_IDENTIFIER = 0x6900,
EMBER_RESET_NETWORK_STATE_COMMAND_IDENTIFIER = 0x6901,
EMBER_INIT_HOST_COMMAND_IDENTIFIER = 0x6902,
EMBER_STATE_COMMAND_IDENTIFIER = 0x6903,
EMBER_GET_VERSIONS_COMMAND_IDENTIFIER = 0x6904,
EMBER_FORM_NETWORK_COMMAND_IDENTIFIER = 0x6905,
EMBER_JOIN_NETWORK_COMMAND_IDENTIFIER = 0x6906,
EMBER_RESUME_NETWORK_COMMAND_IDENTIFIER = 0x6907,
EMBER_ATTACH_TO_NETWORK_COMMAND_IDENTIFIER = 0x6908,
EMBER_HOST_JOIN_CLIENT_COMPLETE_COMMAND_IDENTIFIER = 0x690C,
EMBER_SET_SECURITY_PARAMETERS_COMMAND_IDENTIFIER = 0x690D,
EMBER_SWITCH_TO_NEXT_NETWORK_KEY_COMMAND_IDENTIFIER = 0x690F,
EMBER_START_SCAN_COMMAND_IDENTIFIER = 0x6913,
EMBER_STOP_SCAN_COMMAND_IDENTIFIER = 0x6915,
EMBER_GET_RIP_ENTRY_COMMAND_IDENTIFIER = 0x6917,
EMBER_GET_MULTICAST_TABLE_COMMAND_IDENTIFIER = 0x6918,
EMBER_GET_COUNTER_COMMAND_IDENTIFIER = 0x6919,
EMBER_CLEAR_COUNTERS_COMMAND_IDENTIFIER = 0x691A,
EMBER_SET_TX_POWER_MODE_COMMAND_IDENTIFIER = 0x691C,
EMBER_GET_TX_POWER_MODE_COMMAND_IDENTIFIER = 0x691D,
EMBER_FF_WAKEUP_COMMAND_IDENTIFIER = 0x6924,
EMBER_GET_CCA_THRESHOLD_COMMAND_IDENTIFIER = 0x6925,
EMBER_SET_CCA_THRESHOLD_COMMAND_IDENTIFIER = 0x6926,
EMBER_GET_RADIO_POWER_COMMAND_IDENTIFIER = 0x6927,
EMBER_SET_RADIO_POWER_COMMAND_IDENTIFIER = 0x6928,
EMBER_ECHO_COMMAND_IDENTIFIER = 0x692A,
EMBER_CONFIGURE_GATEWAY_COMMAND_IDENTIFIER = 0x6944,
EMBER_GET_INDEXED_TOKEN_COMMAND_IDENTIFIER = 0x6945,
EMBER_POLL_FOR_DATA_COMMAND_IDENTIFIER = 0x6946,
EMBER_DEEP_SLEEP_COMMAND_IDENTIFIER = 0x6947,
EMBER_STACK_POLL_FOR_DATA_COMMAND_IDENTIFIER = 0x6948,
EMBER_OK_TO_NAP_COMMAND_IDENTIFIER = 0x6949,
EMBER_PING_COMMAND_IDENTIFIER = 0x694F,
EMBER_JOIN_COMMISSIONED_COMMAND_IDENTIFIER = 0x6951,
EMBER_COMMISSION_NETWORK_COMMAND_IDENTIFIER = 0x6952,
EMBER_REQUEST_DHCP_ADDRESS_COMMAND_IDENTIFIER = 0x6956,
EMBER_HOST_TO_NCP_NO_OP_COMMAND_IDENTIFIER = 0x6958,
EMBER_GET_GLOBAL_ADDRESSES_COMMAND_IDENTIFIER = 0x6959,
EMBER_GET_DHCP_CLIENTS_COMMAND_IDENTIFIER = 0x6962,
EMBER_ADD_STEERING_EUI64_COMMAND_IDENTIFIER = 0x6965,
EMBER_BECOME_COMMISSIONER_COMMAND_IDENTIFIER = 0x6966,
EMBER_GET_COMMISSIONER_COMMAND_IDENTIFIER = 0x6967,
EMBER_SEND_STEERING_DATA_COMMAND_IDENTIFIER = 0x6968,
EMBER_STOP_COMMISSIONING_COMMAND_IDENTIFIER = 0x6969,
EMBER_SET_JOIN_KEY_COMMAND_IDENTIFIER = 0x696A,
EMBER_SET_JOINING_MODE_COMMAND_IDENTIFIER = 0x696B,
EMBER_CHANGE_NODE_TYPE_COMMAND_IDENTIFIER = 0x696E,
EMBER_GET_GLOBAL_PREFIXES_COMMAND_IDENTIFIER = 0x696F,
EMBER_RESIGN_GLOBAL_ADDRESS_COMMAND_IDENTIFIER = 0x6970,
EMBER_SET_EUI64_COMMAND_IDENTIFIER = 0x6971,
EMBER_REQUEST_SLAAC_ADDRESS_COMMAND_IDENTIFIER = 0x6972,
EMBER_CUSTOM_HOST_TO_NCP_MESSAGE_COMMAND_IDENTIFIER = 0x6973,
EMBER_GET_NETWORK_DATA_TLV_COMMAND_IDENTIFIER = 0x6977,
EMBER_GET_ROUTING_LOCATOR_COMMAND_IDENTIFIER = 0x6978,
EMBER_SET_RANDOMIZE_MAC_EXTENDED_ID_COMMAND_IDENTIFIER = 0x6979,
EMBER_CONFIGURE_EXTERNAL_ROUTE_COMMAND_IDENTIFIER = 0x697a,
EMBER_ALLOW_NATIVE_COMMISSIONER_COMMAND_IDENTIFIER = 0x697b,
EMBER_SET_COMMISSIONER_KEY_COMMAND_IDENTIFIER = 0x697c,
EMBER_GET_STANDALONE_BOOTLOADER_INFO_COMMAND_IDENTIFIER = 0x697d,
EMBER_LAUNCH_STANDALONE_BOOTLOADER_COMMAND_IDENTIFIER = 0x697e,
EMBER_GET_MFG_TOKEN_COMMAND_IDENTIFIER = 0x697f,
EMBER_SET_MFG_TOKEN_COMMAND_IDENTIFIER = 0x6980,
EMBER_ENABLE_HOST_DTLS_CLIENT_COMMAND_IDENTIFIER = 0x6981,
EMBER_GET_CTUNE_COMMAND_IDENTIFIER = 0x6982,

EMBER_START_XON_XOFF_TEST_COMMAND_IDENTIFIER = 0x694D }

8.66.1 Macro Definition Documentation

8.66.1.1 #define SET_LARGE_VALUE_ID_MAX 0x01

8.66.1.2 #define TMSP_MFGLIB_ACTIVITIES_MAX 0x01

8.66.1.3 #define TMSP_MFGLIB_VALUES_MAX 0x04

8.66.1.4 #define TMSP_VERSION 0x0E00

8.66.2 Typedef Documentation

8.66.2.1 typedef uint8_t SetLargeValueId

8.66.2.2 typedef uint8_t TmspMfglibActivities

8.66.2.3 typedef uint8_t TmspMfglibValues

8.66.3 Enumeration Type Documentation

8.66.3.1 anonymous enum

Enumerator

TMSP_MFGLIB_TONE

TMSP_MFGLIB_STREAM

8.66.3.2 anonymous enum

Enumerator

TMSP_MFGLIB_CHANNEL

TMSP_MFGLIB_POWER

TMSP_MFGLIB_POWER_MODE

TMSP_MFGLIB_SYN_OFFSET

TMSP_MFGLIB_OPTIONS

8.66.3.3 anonymous enum

Enumerator

SET_ND_DATA

SET_LOCAL_NETWORK_DATA

8.66.3.4 anonymous enum

Enumerator

EMBER_RESET_MICRO_COMMAND_IDENTIFIER
EMBER_RESET_NETWORK_STATE_COMMAND_IDENTIFIER
EMBER_INIT_HOST_COMMAND_IDENTIFIER
EMBER_STATE_COMMAND_IDENTIFIER
EMBER_GET_VERSIONS_COMMAND_IDENTIFIER
EMBER_FORM_NETWORK_COMMAND_IDENTIFIER
EMBER_JOIN_NETWORK_COMMAND_IDENTIFIER
EMBER_RESUME_NETWORK_COMMAND_IDENTIFIER
EMBER_ATTACH_TO_NETWORK_COMMAND_IDENTIFIER
EMBER_HOST_JOIN_CLIENT_COMPLETE_COMMAND_IDENTIFIER
EMBER_SET_SECURITY_PARAMETERS_COMMAND_IDENTIFIER
EMBER_SWITCH_TO_NEXT_NETWORK_KEY_COMMAND_IDENTIFIER
EMBER_START_SCAN_COMMAND_IDENTIFIER
EMBER_STOP_SCAN_COMMAND_IDENTIFIER
EMBER_GET_RIP_ENTRY_COMMAND_IDENTIFIER
EMBER_GET_MULTICAST_TABLE_COMMAND_IDENTIFIER
EMBER_GET_COUNTER_COMMAND_IDENTIFIER
EMBER_CLEAR_COUNTERS_COMMAND_IDENTIFIER
EMBER_SET_TX_POWER_MODE_COMMAND_IDENTIFIER
EMBER_GET_TX_POWER_MODE_COMMAND_IDENTIFIER
EMBER_FF_WAKEUP_COMMAND_IDENTIFIER
EMBER_GET_CCA_THRESHOLD_COMMAND_IDENTIFIER
EMBER_SET_CCA_THRESHOLD_COMMAND_IDENTIFIER
EMBER_GET_RADIO_POWER_COMMAND_IDENTIFIER
EMBER_SET_RADIO_POWER_COMMAND_IDENTIFIER
EMBER_ECHO_COMMAND_IDENTIFIER
EMBER_CONFIGURE_GATEWAY_COMMAND_IDENTIFIER
EMBER_GET_INDEXED_TOKEN_COMMAND_IDENTIFIER
EMBER_POLL_FOR_DATA_COMMAND_IDENTIFIER
EMBER_DEEP_SLEEP_COMMAND_IDENTIFIER
EMBER_STACK_POLL_FOR_DATA_COMMAND_IDENTIFIER
EMBER_OK_TO_NAP_COMMAND_IDENTIFIER
EMBER_PING_COMMAND_IDENTIFIER
EMBER_JOIN_COMMISSIONED_COMMAND_IDENTIFIER
EMBER_COMMISSION_NETWORK_COMMAND_IDENTIFIER
EMBER_REQUEST_DHCP_ADDRESS_COMMAND_IDENTIFIER
EMBER_HOST_TO_NCP_NO_OP_COMMAND_IDENTIFIER
EMBER_GET_GLOBAL_ADDRESSES_COMMAND_IDENTIFIER
EMBER_GET_DHCP_CLIENTS_COMMAND_IDENTIFIER
EMBER_ADD_STEERING_EUI64_COMMAND_IDENTIFIER
EMBER_BECOME_COMMISSIONER_COMMAND_IDENTIFIER
EMBER_GET_COMMISSIONER_COMMAND_IDENTIFIER

EMBER_SEND_STEERING_DATA_COMMAND_IDENTIFIER
EMBER_STOP_COMMISSIONING_COMMAND_IDENTIFIER
EMBER_SET_JOIN_KEY_COMMAND_IDENTIFIER
EMBER_SET_JOINING_MODE_COMMAND_IDENTIFIER
EMBER_CHANGE_NODE_TYPE_COMMAND_IDENTIFIER
EMBER_GET_GLOBAL_PREFIXES_COMMAND_IDENTIFIER
EMBER_RESIGN_GLOBAL_ADDRESS_COMMAND_IDENTIFIER
EMBER_SET_EUI64_COMMAND_IDENTIFIER
EMBER_REQUEST_SLAAC_ADDRESS_COMMAND_IDENTIFIER
EMBER_CUSTOM_HOST_TO_NCP_MESSAGE_COMMAND_IDENTIFIER
EMBER_GET_NETWORK_DATA_TLV_COMMAND_IDENTIFIER
EMBER_GET_ROUTING_LOCATOR_COMMAND_IDENTIFIER
EMBER_SET_RANDOMIZE_MAC_EXTENDED_ID_COMMAND_IDENTIFIER
EMBER_CONFIGURE_EXTERNAL_ROUTE_COMMAND_IDENTIFIER
EMBER_ALLOW_NATIVE_COMMISSIONER_COMMAND_IDENTIFIER
EMBER_SET_COMMISSIONER_KEY_COMMAND_IDENTIFIER
EMBER_GET_STANDALONE_BOOTLOADER_INFO_COMMAND_IDENTIFIER
EMBER_LAUNCH_STANDALONE_BOOTLOADER_COMMAND_IDENTIFIER
EMBER_GET_MFG_TOKEN_COMMAND_IDENTIFIER
EMBER_SET_MFG_TOKEN_COMMAND_IDENTIFIER
EMBER_ENABLE_HOST_DTLS_CLIENT_COMMAND_IDENTIFIER
EMBER_GET_CTUNE_COMMAND_IDENTIFIER
EMBER_SET_CTUNE_COMMAND_IDENTIFIER
EMBER_SET_RADIO_HOLD_OFF_COMMAND_IDENTIFIER
EMBER_GET_PTA_ENABLE_COMMAND_IDENTIFIER
EMBER_SET_PTA_ENABLE_COMMAND_IDENTIFIER
EMBER_GET_ANTENNA_MODE_COMMAND_IDENTIFIER
EMBER_SET_ANTENNA_MODE_COMMAND_IDENTIFIER
EMBER_RADIO_GET_RANDOM_NUMBERS_COMMAND_IDENTIFIER
EMBER_GET_PTA_OPTIONS_COMMAND_IDENTIFIER
EMBER_SET_PTA_OPTIONS_COMMAND_IDENTIFIER
EMBER_NCP_GET_NETWORK_DATA_COMMAND_IDENTIFIER
EMBER_NOTE_EXTERNAL_COMMISSIONER_COMMAND_IDENTIFIER
EMBER_NCP_SET_ND_DATA_COMMAND_IDENTIFIER
EMBER_NCP_SET_LARGE_DATA_COMMAND_IDENTIFIER
CB_RESET_MICRO_COMMAND_IDENTIFIER
CB_STATE_COMMAND_IDENTIFIER
CB_GET_VERSIONS_COMMAND_IDENTIFIER
CB_GET_RIP_ENTRY_COMMAND_IDENTIFIER
CB_INIT_COMMAND_IDENTIFIER
CB_GET_COUNTER_COMMAND_IDENTIFIER
CB_SET_SECURITY_PARAMETERS_COMMAND_IDENTIFIER
CB_SWITCH_TO_NEXT_NETWORK_KEY_COMMAND_IDENTIFIER
CB_FORM_NETWORK_COMMAND_IDENTIFIER
CB_JOIN_NETWORK_COMMAND_IDENTIFIER

CB_RESUME_NETWORK_COMMAND_IDENTIFIER
CB_ATTACH_TO_NETWORK_COMMAND_IDENTIFIER
CB_ENERGY_SCAN_COMMAND_IDENTIFIER
CB_ACTIVE_SCAN_COMMAND_IDENTIFIER
CB_SCAN_COMMAND_IDENTIFIER
CB_SET_ADDRESS_COMMAND_IDENTIFIER
CB_SET_DRIVER_ADDRESS_COMMAND_IDENTIFIER
CB_START_HOST_JOIN_CLIENT_COMMAND_IDENTIFIER
CB_SET_TX_POWER_MODE_COMMAND_IDENTIFIER
CB_GET_TX_POWER_MODE_COMMAND_IDENTIFIER
CB_SET_NETWORK_KEYS_COMMAND_IDENTIFIER
CB_GET_MULTICAST_ENTRY_COMMAND_IDENTIFIER
CB_GET_CCA_THRESHOLD_COMMAND_IDENTIFIER
CB_SET_CCA_THRESHOLD_COMMAND_IDENTIFIER
CB_GET_RADIO_POWER_COMMAND_IDENTIFIER
CB_SET_RADIO_POWER_COMMAND_IDENTIFIER
CB_ECHO_COMMAND_IDENTIFIER
CB_ASSERT_INFO_COMMAND_IDENTIFIER
CB_CONFIGURE_GATEWAY_COMMAND_IDENTIFIER
CB_GET_CHANNEL_CAL_DATA_TOKEN_COMMAND_IDENTIFIER
CB_POLL_FOR_DATA_COMMAND_IDENTIFIER
CB_DEEP_SLEEP_COMMAND_IDENTIFIER
CB_STACK_POLL_FOR_DATA_COMMAND_IDENTIFIER
CB_OK_TO_NAP_COMMAND_IDENTIFIER
CB_DEEP_SLEEP_COMPLETE_COMMAND_IDENTIFIER
CB_RESET_NETWORK_STATE_COMMAND_IDENTIFIER
CB_EXTERNAL_ROUTE_CHANGE_COMMAND_IDENTIFIER
CB_DHCP_SERVER_CHANGE_COMMAND_IDENTIFIER
CB_NCP_NETWORK_DATA_CHANGE_COMMAND_IDENTIFIER
CB_NCP_GET_NETWORK_DATA_COMMAND_IDENTIFIER
CB_ADDRESS_CONFIGURATION_CHANGE_COMMAND_IDENTIFIER
CB_REQUEST_DHCP_ADDRESS_COMMAND_IDENTIFIER
CB_COMMISSION_NETWORK_COMMAND_IDENTIFIER
CB_GET_GLOBAL_ADDRESS_COMMAND_IDENTIFIER
CB_GET_DHCP_CLIENT_COMMAND_IDENTIFIER
CB_BECOME_COMMISSIONER_COMMAND_IDENTIFIER
CB_SEND_STEERING_DATA_COMMAND_IDENTIFIER
CB_CHANGE_NODE_TYPE_COMMAND_IDENTIFIER
CB_GET_GLOBAL_PREFIX_COMMAND_IDENTIFIER
CB_RESIGN_GLOBAL_ADDRESS_COMMAND_IDENTIFIER
CB_SLAAC_SERVER_CHANGE_COMMAND_IDENTIFIER
CB_REQUEST_SLAAC_ADDRESS_COMMAND_IDENTIFIER
CB_CUSTOM_NCP_TO_HOST_MESSAGE_COMMAND_IDENTIFIER
CB_NCP_TO_HOST_NO_OP_COMMAND_IDENTIFIER
CB_COMMISSIONER_STATUS_COMMAND_IDENTIFIER

CB_LEADER_DATA_COMMAND_IDENTIFIER
CB_GET_NETWORK_DATA_TLV_COMMAND_IDENTIFIER
CB_GET_ROUTING_LOCATOR_COMMAND_IDENTIFIER
CB_SET_RANDOMIZE_MAC_EXTENDED_ID_COMMAND_IDENTIFIER
CB_CONFIGURE_EXTERNAL_ROUTE_COMMAND_IDENTIFIER
CB_ALLOW_NATIVE_COMMISSIONER_COMMAND_IDENTIFIER
CB_SET_COMMISSIONER_KEY_COMMAND_IDENTIFIER
CB_SET_COMM_PROXY_APP_PARAMETERS_COMMAND_IDENTIFIER
CB_SET_COMM_PROXY_APP_SECURITY_COMMAND_IDENTIFIER
CB_SET_COMM_PROXY_APP_ADDRESS_COMMAND_IDENTIFIER
CB_NETWORK_STATUS_COMMAND_IDENTIFIER
CB_GET_STANDALONE_BOOTLOADER_INFO_COMMAND_IDENTIFIER
CB_LAUNCH_STANDALONE_BOOTLOADER_COMMAND_IDENTIFIER
CB_GET_MFG_TOKEN_COMMAND_IDENTIFIER
CB_SET_MFG_TOKEN_COMMAND_IDENTIFIER
CB_HOST_STATE_COMMAND_IDENTIFIER
CB_GET_CTUNE_COMMAND_IDENTIFIER
CB_SET_CTUNE_COMMAND_IDENTIFIER
CB_SET_RADIO_HOLD_OFF_COMMAND_IDENTIFIER
CB_GET_PTA_ENABLE_COMMAND_IDENTIFIER
CB_SET_PTA_ENABLE_COMMAND_IDENTIFIER
CB_GET_ANTENNA_MODE_COMMAND_IDENTIFIER
CB_SET_ANTENNA_MODE_COMMAND_IDENTIFIER
CB_RADIO_GET_RANDOM_NUMBERS_COMMAND_IDENTIFIER
CB_GET_PTA_OPTIONS_COMMAND_IDENTIFIER
CB_SET_PTA_OPTIONS_COMMAND_IDENTIFIER
CB_SET_JOIN_KEY_COMMAND_IDENTIFIER
CB_SET_ND_DATA_COMMAND_IDENTIFIER
CB_SET_LOCAL_NETWORK_DATA_COMMAND_IDENTIFIER
CB_SET_COMM_PROXY_APP_PSKC_COMMAND_IDENTIFIER
CB_SET_PSKC_COMMAND_IDENTIFIER
EMBER_MFGLIB_START_COMMAND_IDENTIFIER
EMBER_MFGLIB_END_COMMAND_IDENTIFIER
EMBER_MFGLIB_START_ACTIVITY_COMMAND_IDENTIFIER
EMBER_MFGLIB_STOP_ACTIVITY_COMMAND_IDENTIFIER
EMBER_MFGLIB_SEND_PACKET_COMMAND_IDENTIFIER
EMBER_MFGLIB_SET_COMMAND_IDENTIFIER
EMBER_MFGLIB_GET_COMMAND_IDENTIFIER
EMBER_MFGLIB_TEST_CONT_MOD_CAL_COMMAND_IDENTIFIER
CB_MFGLIB_START_TEST_COMMAND_IDENTIFIER
CB_MFGLIB_RX_COMMAND_IDENTIFIER
CB_MFGLIB_END_TEST_COMMAND_IDENTIFIER
CB_MFGLIB_START_COMMAND_IDENTIFIER
CB_MFGLIB_STOP_COMMAND_IDENTIFIER
CB_MFGLIB_SEND_PACKET_EVENT_COMMAND_IDENTIFIER

CB_MFGLIB_SET_COMMAND_IDENTIFIER

CB_MFGLIB_GET_CHANNEL_COMMAND_IDENTIFIER

CB_MFGLIB_GET_POWER_COMMAND_IDENTIFIER

CB_MFGLIB_GET_POWER_MODE_COMMAND_IDENTIFIER

CB_MFGLIB_GET_SYN_OFFSET_COMMAND_IDENTIFIER

CB_MFGLIB_GET_OPTIONS_COMMAND_IDENTIFIER

EMBER_CONFIG_UART_COMMAND_IDENTIFIER

EMBER_RESET_NCP_ASH_COMMAND_IDENTIFIER

EMBER_RESET_IP_DRIVER_ASH_COMMAND_IDENTIFIER

EMBER_START_UART_STORM_COMMAND_IDENTIFIER

EMBER_STOP_UART_STORM_COMMAND_IDENTIFIER

EMBER_SEND_DONE_COMMAND_IDENTIFIER

CB_CONFIG_UART_COMMAND_IDENTIFIER

CB_RESET_NCP_ASH_COMMAND_IDENTIFIER

CB_START_UART_STORM_COMMAND_IDENTIFIER

CB_STOP_UART_STORM_COMMAND_IDENTIFIER

CB_SEND_DONE_COMMAND_IDENTIFIER

EMBER_GET_NETWORK_KEY_INFO_COMMAND_IDENTIFIER

EMBER_RESET_NCP_GPIO_COMMAND_IDENTIFIER

EMBER_ENABLE_RESET_NCP_GPIO_COMMAND_IDENTIFIER

EMBER_FORCE_ASSERT_COMMAND_IDENTIFIER

EMBER_GET_NODE_STATUS_COMMAND_IDENTIFIER

EMBER_ADD_ADDRESS_DATA_COMMAND_IDENTIFIER

EMBER_CLEAR_ADDRESS_CACHE_COMMAND_IDENTIFIER

EMBER_LOOKUP_ADDRESS_DATA_COMMAND_IDENTIFIER

EMBER_START_UART_SPEED_TEST_COMMAND_IDENTIFIER

EMBER_NCP_UDP_STORM_COMMAND_IDENTIFIER

CB_GET_NETWORK_KEY_INFO_COMMAND_IDENTIFIER

CB_GET_NODE_STATUS_COMMAND_IDENTIFIER

CB_ADD_ADDRESS_DATA_COMMAND_IDENTIFIER

CB_CLEAR_ADDRESS_CACHE_COMMAND_IDENTIFIER

CB_LOOKUP_ADDRESS_DATA_COMMAND_IDENTIFIER

CB_UART_SPEED_TEST_COMMAND_IDENTIFIER

CB_NCP_UDP_STORM_COMMAND_IDENTIFIER

CB_NCP_UDP_STORM_COMPLETE_COMMAND_IDENTIFIER

EMBER_START_XON_XOFF_TEST_COMMAND_IDENTIFIER

8.67 token-manufacturing.h File Reference

Definitions for manufacturing tokens.

Macros

- #define TOKEN_NEXT_ADDRESS(region, address)
- #define CREATOR_MFG_CHIP_DATA 0xC344
- #define CREATOR_MFG_PART_DATA 0xF064
- #define CREATOR_MFG_TESTER_DATA 0xF464
- #define CREATOR_MFG_EMBER_EUI_64 0xE545
- #define CREATOR_MFG_ANALOG_TRIM_NORMAL 0xF46E
- #define CREATOR_MFG_ANALOG_TRIM_BOOST 0xF442
- #define CREATOR_MFG_ANALOG_TRIM_BOTH 0xF462
- #define CREATOR_MFG_REG_TRIM 0xF274
- #define CREATOR_MFG_1V8_REG_VOLTAGE 0xF276
- #define CREATOR_MFG_VREF_VOLTAGE 0xF676
- #define CREATOR_MFG_TEMP_CAL 0xF463
- #define CREATOR_MFG_TEST_TEMP 0xF474
- #define CREATOR_MFG_FIB_VERSION 0xFF09
- #define CREATOR_MFG_FIB_CHECKSUM 0xE663
- #define CREATOR_MFG_FIB_OBS 0xE66F
- #define CREATOR_MFG_CIB_OBS 0xE36F
- #define CREATOR_MFG_CUSTOM_VERSION 0xC356
- #define CREATOR_MFG_CUSTOM_EUI_64 0xE345
- #define CREATOR_MFG_STRING 0xED73
- #define CREATOR_MFG_BOARD_NAME 0xC24E
- #define CREATOR_MFG_MANUF_ID 0xC944
- #define CREATOR_MFG_PHY_CONFIG 0xD043
- #define CREATOR_MFG_BOOTLOAD_AES_KEY 0xC24B
- #define CREATOR_MFG_EZSP_STORAGE 0xCD53
- #define CREATOR_MFG_ASH_CONFIG 0xC143
- #define CREATOR_MFG_CBKE_DATA 0xC342
- #define CREATOR_MFG_INSTALLATION_CODE 0xC943
- #define CREATOR_MFG_OSC24M_BIAS_TRIM 0xB254
- #define CREATOR_MFG_SYNTH_FREQ_OFFSET 0xD346
- #define CREATOR_MFG_OSC24M_SETTLE_DELAY 0xB253
- #define CREATOR_MFG_SECURITY_CONFIG 0xD343
- #define CREATOR_MFG_CCA_THRESHOLD 0xC343
- #define CREATOR_MFG_SECURE_BOOTLOADER_KEY 0xD342
- #define CREATOR_MFG_ETHERNET_ADDRESS 0xC554
- #define CREATOR_MFG_CBKE_283K1_DATA 0xC345
- #define CREATOR_MFG_XO_TUNE 0xD854
- #define CREATOR_MFG_THREAD_JOIN_KEY 0xCA4B
- #define CREATOR_MFG_EUI_64 0xB634
- #define CURRENT_MFG_TOKEN_VERSION 0x01FE
- #define VALID_MFG_TOKEN_VERSIONS { 0x01FE, 0x02FD }
- #define CURRENT_MFG_CUSTOM_VERSION 0x01FE

Convenience Macros

The following convenience macros are used to simplify the definition process for commonly specified parameters to the basic TOKEN_DEF macro. Please see hal/micro/token.h for a more complete explanation.

- #define DEFINE_MFG_TOKEN(name, type, address, ...)

8.67.1 Detailed Description

CAUTION: This file is generated by gen_all.py which invokes gen_em3xx_token_mfg_h.py.

This file should not be included directly. It is accessed by the other token files.

Please see stack/config/token-stack.h and hal/micro/token.h for a full explanation of the tokens.

The tokens listed below are the manufacturing tokens. This token definitions file is included from the master definitions file: stack/config/token-stack.h Please see that file for more details.

The user application can include its own manufacturing tokens in a header file similar to this one. The macro ::A↩ PPLICATION_MFG_TOKEN_HEADER should be defined to equal the name of the header file in which application manufacturing tokens are defined.

The macro DEFINE_MFG_TOKEN() should be used when instantiating a manufacturing token. Refer to the list of ∗_LOCATION defines to see what memory is allocated and what memory is unused/available.

REMEMBER: By definition, manufacturing tokens exist at fixed addresses. Tokens should not overlap.

Here is a basic example of a manufacturing token header file:

```
#define CREATOR_MFG_EXAMPLE 0x4242
#ifdef DEFINETYPES
typedef uint8_t tokTypeMfgExample[8];
#endif
#ifdef DEFINETOKENS
#define MFG_EXAMPLE_LOCATION 0x0980
DEFINE_MFG_TOKEN(MFG_EXAMPLE,
                 tokTypeMfgExample,
                 MFG_EXAMPLE_LOCATION,
                 {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF})
#endif
```

Since this file contains both the typedefs and the token defs, there are two #defines used to select which one is needed when this file is included. #define DEFINETYPES is used to select the type definitions and #define DEFINETOKENS is used to select the token definitions. Refer to token.h and token.c to see how these are used.

8.67.2 Macro Definition Documentation

8.67.2.1 #define CREATOR_MFG_1V8_REG_VOLTAGE 0xF276

8.67.2.2 #define CREATOR_MFG_ANALOG_TRIM_BOOST 0xF442

8.67.2.3 #define CREATOR_MFG_ANALOG_TRIM_BOTH 0xF462

8.67.2.4 #define CREATOR_MFG_ANALOG_TRIM_NORMAL 0xF46E

8.67.2.5 #define CREATOR_MFG_ASH_CONFIG 0xC143

8.67.2.6 #define CREATOR_MFG_BOARD_NAME 0xC24E

8.67.2.7 #define CREATOR_MFG_BOOTLOAD_AES_KEY 0xC24B

8.67.2.8 #define CREATOR_MFG_CBKE_283K1_DATA 0xC345

8.67.2.9 #define CREATOR_MFG_CBKE_DATA 0xC342

8.67.2.10 #define CREATOR_MFG_CCA_THRESHOLD 0xC343

8.67.2.11 #define CREATOR_MFG_CHIP_DATA 0xC344

8.67.2.12 #define CREATOR_MFG_CIB_OBS 0xE36F

8.67.2.13 #define CREATOR_MFG_CUSTOM_EUI_64 0xE345

8.67.2.14 #define CREATOR_MFG_CUSTOM_VERSION 0xC356

8.67.2.15 #define CREATOR_MFG_EMBER_EUI_64 0xE545

8.67.2.16 #define CREATOR_MFG_ETHERNET_ADDRESS 0xC554

8.67.2.17 #define CREATOR_MFG_EUI_64 0xB634

8.67.2.18 #define CREATOR_MFG_EZSP_STORAGE 0xCD53

8.67.2.19 #define CREATOR_MFG_FIB_CHECKSUM 0xE663

8.67.2.20 #define CREATOR_MFG_FIB_OBS 0xE66F

8.67.2.21 #define CREATOR_MFG_FIB_VERSION 0xFF09

8.67.2.22 #define CREATOR_MFG_INSTALLATION_CODE 0xC943

8.67.2.23 #define CREATOR_MFG_MANUF_ID 0xC944

8.67.2.24 #define CREATOR_MFG_OSC24M_BIAS_TRIM 0xB254

8.67.2.25 #define CREATOR_MFG_OSC24M_SETTLE_DELAY 0xB253

8.67.2.26 #define CREATOR_MFG_PART_DATA 0xF064

8.67.2.27 #define CREATOR_MFG_PHY_CONFIG 0xD043

8.67.2.28 #define CREATOR_MFG_REG_TRIM 0xF274

8.67.2.29 #define CREATOR_MFG_SECURE_BOOTLOADER_KEY 0xD342

8.67.2.30 #define CREATOR_MFG_SECURITY_CONFIG 0xD343

8.67.2.31 #define CREATOR_MFG_STRING 0xED73

8.67.2.32 #define CREATOR_MFG_SYNTH_FREQ_OFFSET 0xD346

8.67.2.33 #define CREATOR_MFG_TEMP_CAL 0xF463

8.67.2.34 #define CREATOR_MFG_TEST_TEMP 0xF474

8.67.2.35 #define CREATOR_MFG_TESTER_DATA 0xF464

8.67.2.36 #define CREATOR_MFG_THREAD_JOIN_KEY 0xCA4B

8.67.2.37 #define CREATOR_MFG_VREF_VOLTAGE 0xF676

8.67.2.38 #define CREATOR_MFG_XO_TUNE 0xD854

8.67.2.39 #define CURRENT_MFG_CUSTOM_VERSION 0x01FE

8.67.2.40 #define CURRENT_MFG_TOKEN_VERSION 0x01FE

8.67.2.41 #define DEFINE_MFG_TOKEN(*name, type, address, ...*)

Value:

```
TOKEN_NEXT_ADDRESS(name, (address))                \
  TOKEN_MFG(name, CREATOR_##name, 0, 0, type, 1, __VA_ARGS__)
```

8.67.2.42 #define TOKEN_NEXT_ADDRESS(*region, address*)

8.67.2.43 #define VALID_MFG_TOKEN_VERSIONS { 0x01FE, 0x02FD }

8.68 token-stack.h File Reference

Definitions for stack tokens. See Hardware Abstraction Layer (HAL) API Reference for documentation.

```
#include "token-phy.h"
```

Macros

- #define TOKEN_NEXT_ADDRESS(region, address)

  By default, tokens are automatically located after the previous token.
- #define CURRENT_STACK_TOKEN_VERSION 0x03FC

  The current version number of the stack tokens. MSB is the version, LSB is a complement.

Convenience Macros

The following convenience macros are used to simplify the definition process for commonly specified parameters to the basic TOKEN_DEF macro. Please see hal/micro/token.h for a more complete explanation.

- #define DEFINE_BASIC_TOKEN(name, type, ...) TOKEN_DEF(name, CREATOR_##name, 0, 0, type, 1, __VA_ARGS__)
- #define DEFINE_COUNTER_TOKEN(name, type, ...) TOKEN_DEF(name, CREATOR_##name, 1, 0, type, 1, __VA_ARGS__)
- #define DEFINE_INDEXED_TOKEN(name, type, arraysize, ...) TOKEN_DEF(name, CREATOR_##name, 0, 1, type, (arraysize), __VA_ARGS__)
- #define DEFINE_FIXED_BASIC_TOKEN(name, type, address, ...)
- #define DEFINE_FIXED_COUNTER_TOKEN(name, type, address, ...)
- #define DEFINE_FIXED_INDEXED_TOKEN(name, type, arraysize, address, ...)
- #define DEFINE_MFG_TOKEN(name, type, address, ...)

Creator Codes

The CREATOR is used as a distinct identifier tag for the token.

The CREATOR is necessary because the token name is defined differently depending on the hardware platform, therefore the CREATOR makes sure that token definitions and data stay tagged and known. The only requirement is that each creator definition must be unique. Please see hal/micro/token.h for a more complete explanation.

- #define CREATOR_STACK_NVDATA_VERSION 0xFF01
- #define CREATOR_STACK_BOOT_COUNTER 0xE263
- #define CREATOR_STACK_NONCE_COUNTER 0xE563
- #define CREATOR_STACK_ANALYSIS_REBOOT 0xE162
- #define CREATOR_STACK_KEYS 0xEB79
- #define CREATOR_STACK_NODE_DATA 0xEE64
- #define CREATOR_STACK_CLASSIC_DATA 0xE364
- #define CREATOR_STACK_ALTERNATE_KEY 0xE475
- #define CREATOR_STACK_APS_FRAME_COUNTER 0xE123
- #define CREATOR_STACK_TRUST_CENTER 0xE124
- #define CREATOR_STACK_NETWORK_MANAGEMENT 0xE125
- #define CREATOR_STACK_BINDING_TABLE 0xE274
- #define CREATOR_STACK_CHILD_TABLE 0xFF0D
- #define CREATOR_STACK_KEY_TABLE 0xE456
- #define CREATOR_STACK_CERTIFICATE_TABLE 0xE500
- #define CREATOR_STACK_PSL_DATA 0xE501
- #define CREATOR_STACK_HOST_REGISTRY 0xE502

8.68.1 Detailed Description

The file token-stack.h should not be included directly. It is accessed by the other token files.

8.69 token.h File Reference

Cortex-M3 Token system for storing non-volatile information. See Tokens for documentation.

```
#include "mfg-token.h"
#include "stack/config/token-stack.h"
```

Macros

- #define DEFINETYPES
- #define DEFINETOKENS
- #define TOKEN_DEF(name, creator, iscnt, isidx, type, arraysize, ...) TOKEN_##name,
- #define TOKEN_DEF(name, creator, iscnt, isidx, type, arraysize, ...) TOKEN_##name##_SIZE = sizeof(type),
- #define COUNTER_TOKEN_PAD 50
- #define TOKEN_DEF(name, creator, iscnt, isidx, type, arraysize, ...) typedef type TOKEN_##name##_TYPE;
- #define halCommonGetToken(data, token) halInternalGetTokenData(data, token, 0x7F, token##_SIZE)
- #define halCommonGetIndexedToken(data, token, index) halInternalGetTokenData(data, token, index, token##_SIZE)
- #define halStackGetIndexedToken(data, token, index, size) halInternalGetTokenData(data, token, index, size)
- #define halStackGetIdxTokenPtrOrData(ptr, token, index) halInternalGetIdxTokenPtr(ptr, token, index, token##_SIZE)
- #define halCommonSetToken(token, data) halInternalSetTokenData(token, 0x7F, data, token##_SIZE)
- #define halCommonSetIndexedToken(token, index, data) halInternalSetTokenData(token, index, data, token##_SIZE)
- #define halStackSetIndexedToken(token, index, data, size) halInternalSetTokenData(token, index, data, size)
- #define halCommonIncrementCounterToken(token) halInternalIncrementCounterToken(token);

Enumerations

- enum { TOKEN_COUNT }
- enum

Functions

- void halInternalGetTokenData (void *data, uint16_t token, uint8_t index, uint8_t len)
- void halInternalSetTokenData (uint16_t token, uint8_t index, void *data, uint8_t len)
- void halInternalIncrementCounterToken (uint8_t token)
- void halInternalGetIdxTokenPtr (void *ptr, uint16_t ID, uint8_t index, uint8_t len)

Variables

- const uint16_t tokenCreators [ ]
- const bool tokenIsCnt [ ]
- const uint8_t tokenSize [ ]
- const uint8_t tokenArraySize [ ]
- const void *const tokenDefaults [ ]

8.69.1 Detailed Description

DOXYGEN NOTE: This file contains definitions, functions, and information that are internal only and should not be accessed by appilications. This information is still documented, but should not be published in the generated doxygen.

8.69.2 Macro Definition Documentation

8.69.2.1 #define COUNTER_TOKEN_PAD 50

A define for the token and Simulated EEPROM system that specifies, in bytes, the space allocated to a counter token for +1 marks. The number of +1 marks varies between chips based on the minimum write granularity for a chip's flash. EM35x chips can use 8bit per +1 while EFM32/EZM32/EZR32 chips use 16bit per +1.

8.69.2.2 #define DEFINETOKENS

8.69.2.3 #define DEFINETYPES

Simple declarations of all of the token types so that they can be referenced from anywhere in the code base.

8.69.2.4 #define halCommonGetIndexedToken(*data, token, index*) halInternalGetTokenData(data, token, index, token##_SIZE)

8.69.2.5 #define halCommonGetToken(*data, token*) halInternalGetTokenData(data, token, 0x7F, token##_SIZE)

8.69.2.6 #define halCommonIncrementCounterToken(*token*) halInternalIncrementCounterToken(token);

8.69.2.7 #define halCommonSetIndexedToken(*token, index, data*) halInternalSetTokenData(token, index, data, token##_SIZE)

8.69.2.8 #define halCommonSetToken(*token, data*) halInternalSetTokenData(token, 0x7F, data, token##_SIZE)

8.69.2.9 #define halStackGetIdxTokenPtrOrData(*ptr, token, index*) halInternalGetIdxTokenPtr(ptr, token, index, token##_SIZE)

8.69.2.10 #define halStackGetIndexedToken(*data, token, index, size*) halInternalGetTokenData(data, token, index, size)

8.69.2.11 #define halStackSetIndexedToken(*token, index, data, size*) halInternalSetTokenData(token, index, data, size)

8.69.2.12 #define TOKEN_DEF(*name, creator, iscnt, isidx, type, arraysize, ...*) TOKEN_##name,

Enum for translating token defs into a number. This number is used as an index into the cache of token information the token system and Simulated EEPROM hold.

The special entry TOKEN_COUNT is always at the top of the enum, allowing the token and sim-eeprom system to know how many tokens there are.

Parameters

name	The name of the token.

Macro for translating token definitions into size variables. This provides a convenience for abstracting the 'sizeof(type)' anywhere.

Parameters

name	The name of the token.
type	The token type. The types are found in token-stack.h.

Macro for typedef'ing the CamelCase token type found in token-stack.h to a capitalized TOKEN style name that ends in _TYPE. This macro allows other macros below to use 'token##_TYPE' to declare a local copy of that token.

Parameters

name	The name of the token.
type	The token type. The types are found in token-stack.h.

8.69.2.13 #define TOKEN_DEF(*name, creator, iscnt, isidx, type, arraysize, ...*) TOKEN_##name##_SIZE = sizeof(type),

Enum for translating token defs into a number. This number is used as an index into the cache of token information the token system and Simulated EEPROM hold.

The special entry TOKEN_COUNT is always at the top of the enum, allowing the token and sim-eeprom system to know how many tokens there are.

Parameters

name	The name of the token.

Macro for translating token definitions into size variables. This provides a convenience for abstracting the 'sizeof(type)' anywhere.

Parameters

name	The name of the token.
type	The token type. The types are found in token-stack.h.

Macro for typedef'ing the CamelCase token type found in token-stack.h to a capitalized TOKEN style name that ends in _TYPE. This macro allows other macros below to use 'token##_TYPE' to declare a local copy of that token.

Parameters

name	The name of the token.
type	The token type. The types are found in token-stack.h.

8.69.2.14 #define TOKEN_DEF(*name, creator, iscnt, isidx, type, arraysize, ...*) typedef type TOKEN_##name##_TYPE;

Enum for translating token defs into a number. This number is used as an index into the cache of token information the token system and Simulated EEPROM hold.

The special entry TOKEN_COUNT is always at the top of the enum, allowing the token and sim-eeprom system to know how many tokens there are.

Parameters

name	The name of the token.

Macro for translating token definitions into size variables. This provides a convenience for abstracting the 'sizeof(type)' anywhere.

Parameters

name	The name of the token.
type	The token type. The types are found in token-stack.h.

Macro for typedef'ing the CamelCase token type found in token-stack.h to a capitalized TOKEN style name that ends in _TYPE. This macro allows other macros below to use 'token##_TYPE' to declare a local copy of that token.

Parameters

name	The name of the token.
type	The token type. The types are found in token-stack.h.

8.69.3 Enumeration Type Documentation

8.69.3.1 anonymous enum

Enumerator

TOKEN_COUNT

8.69.3.2 anonymous enum

8.69.4 Function Documentation

8.69.4.1 void halInternalGetIdxTokenPtr (void * *ptr,* uint16_t *ID,* uint8_t *index,* uint8_t *len*)

8.69.4.2 void halInternalGetTokenData (void * *data,* uint16_t *token,* uint8_t *index,* uint8_t *len*)

Copies the token value from non-volatile storage into a RAM location. This is the internal function that the two exposed APIs (halCommonGetToken and halCommonGetIndexedToken) expand out to. The API simplifies the access into this function by hiding the size parameter and hiding the value 0 used for the index parameter in scalar tokens.

Note

Only the public function should be called since the public function provides the correct parameters.

Parameters

data	A pointer to where the data being read should be placed.
token	The name of the token to get data from. On this platform that name is defined as an address.
index	The index to access. If the token being accessed is not an indexed token, this parameter is set by the API to be 0.
len	The length of the token being worked on. This value is automatically set by the API to be the size of the token.

8.69.4.3 void halInternalIncrementCounterToken (uint8_t *token*)

Increments the value of a token that is a counter. This is the internal function that the exposed API (halCommon↩ IncrementCounterToken) expand out to. This internal function is used as a level of simple redirection providing clean separation from the lower token handler code.

Note

Only the public function should be called since the public function provides the correct parameters.

Parameters

token	The name of the token.

8.69.4.4 void halInternalSetTokenData (uint16_t *token,* uint8_t *index,* void ∗ *data,* uint8_t *len*)

Sets the value of a token in non-volatile storage. This is the internal function that the two exposed APIs (hal↩ CommonSetToken and halCommonSetIndexedToken) expand out to. The API simplifies the access into this function by hiding the size parameter and hiding the value 0 used for the index parameter in scalar tokens.

Note

Only the public function should be called since the public function provides the correct parameters.

Parameters

token	The name of the token to get data from. On this platform that name is defined as an address.
index	The index to access. If the token being accessed is not an indexed token, this parameter is set by the API to be 0.
data	A pointer to the data being written.
len	The length of the token being worked on. This value is automatically set by the API to be the size of the token.

8.69.5 Variable Documentation

8.69.5.1 const uint8_t tokenArraySize[ ]

External declaration of an array of array sizes. Since the token and sim-eeprom systems identify tokens through an enum (see below for the enum) and these two systems need to know the array size of each token, this array provides that information.

Parameters

arraysize	The array size.

8.69.5.2 const uint16_t tokenCreators[ ]

External declaration of an array of creator codes. Since the token and sim-eeprom systems identify tokens through an enum (see below for the enum) and these two systems need to link creator codes to their tokens, this array instantiates that link.

Parameters

creator	The creator code type. The codes are found in token-stack.h.

8.69.5.3 const void* const tokenDefaults[ ]

External declaration of an array of all token default values. This array is filled with pointers to the set of constant declarations of all of the token default values. Therefore, the index into this array chooses which token's defaults to access, and the address offset chooses the byte in the defaults to use.

For example, to get the n-th byte of the i-th token, use: uint8_t byte = *(((uint8_t *)tokenDefaults[i])+(n)

Parameters

TOKEN_	*#name##_DEFAULTS*	A constant declaration of the token default values, generated for all tokens.

8.69.5.4 const bool tokenIsCnt[ ]

External declaration of an array of IsCnt flags. Since the token and sim-eeprom systems identify tokens through an enum (see below for the enum) and these two systems need to know which tokens are counter tokens, this array provides that information.

Parameters

iscnt	The flag indicating if the token is a counter. The iscnt's are found in token-stack.h.

8.69.5.5 const uint8_t tokenSize[ ]

External declaration of an array of sizes. Since the token and sim-eeprom systems identify tokens through an enum (see below for the enum) and these two systems need to know the size of each token, this array provides that information.

Parameters

type	The token type. The types are found in token-stack.h.

8.70 token.h File Reference

Token system for storing non-volatile information. See Tokens for documentation.

Macros

- #define halCommonGetToken(data, token)

  Macro that copies the token value from non-volatile storage into a RAM location. This macro can only be used with tokens that are defined using DEFINE_BASIC_TOKEN.
- #define halCommonGetMfgToken(data, token)

  Macro that copies the token value from non-volatile storage into a RAM location. This macro can only be used with tokens that are defined using DEFINE_MFG_TOKEN.
- #define halCommonGetIndexedToken(data, token, index)

  Macro that copies the token value from non-volatile storage into a RAM location. This macro can only be used with tokens that are defined using DEFINE_INDEXED_TOKEN.
- #define halCommonSetToken(token, data)

  Macro that sets the value of a token in non-volatile storage. This macro can only be used with tokens that are defined using DEFINE_BASIC_TOKEN.
- #define halCommonSetIndexedToken(token, index, data)

  Macro that sets the value of a token in non-volatile storage. This macro can only be used with tokens that are defined using DEFINE_INDEXED_TOKEN.
- #define halCommonIncrementCounterToken(token)

  Macro that increments the value of a token that is a counter. This macro can only be used with tokens that are defined using either DEFINE_COUNTER_TOKEN.

Functions

- EmberStatus halStackInitTokens (void)

  Initializes and enables the token system. Checks if the manufacturing and stack non-volatile data versions are correct.

8.71 udp-peer.h File Reference

Connection-oriented UDP API.

Data Structures

- struct EmberUdpConnectionData

  Data stored for each connection.

Macros

- #define NULL_UDP_HANDLE 0
- #define UDP_CONNECTED 0x0001
- #define UDP_USING_DTLS 0x0002

Typedefs

- typedef uint8_t EmberUdpConnectionHandle
- typedef void(* EmberUdpConnectionStatusHandler) (EmberUdpConnectionData *connection, UdpStatus status)

  This function is called by the stack when the status of a connection changes.
- typedef void(* EmberUdpConnectionReadHandler) (EmberUdpConnectionData *connection, uint8_t *packet, uint16_t length)

  This function is called by the stack when a UDP packet arrives.

Enumerations

- enum UdpStatus {
  EMBER_UDP_CONNECTED = 0x01,
  EMBER_UDP_OPEN_FAILED = 0x02,
  EMBER_UDP_DISCONNECTED = 0x10 }

Functions

- EmberStatus emberGetUdpConnectionData (EmberUdpConnectionHandle connection, EmberUdpConnectionData *data)

  This function populates the EmberUdpConnectionData structure with the connections's info.
- EmberStatus emberUdpListenLocal (uint16_t port)

  This function listens for incoming UDP messages.
- bool emberUdpAmListening (uint16_t port)
- void emberUdpStopListening (uint16_t port)
- EmberStatus emberUdpMulticastListen (uint16_t port, const uint8_t *multicastAddress)
- EmberStatus emUdpListen (uint16_t port, const uint8_t *localAddress)

8.71.1 Macro Definition Documentation

8.71.1.1 #define NULL_UDP_HANDLE 0

8.71.1.2 #define UDP_CONNECTED 0x0001

8.71.1.3 #define UDP_USING_DTLS 0x0002

8.71.2 Typedef Documentation

8.71.2.1 typedef uint8_t EmberUdpConnectionHandle

8.71.2.2 typedef void(∗ EmberUdpConnectionReadHandler) (EmberUdpConnectionData ∗connection, uint8_t ∗packet, uint16_t length)

8.71.2.3 typedef void(∗ EmberUdpConnectionStatusHandler) (EmberUdpConnectionData ∗connection, UdpStatus status)

8.71.3 Enumeration Type Documentation

8.71.3.1 enum UdpStatus

Enumerator

EMBER_UDP_CONNECTED

EMBER_UDP_OPEN_FAILED

EMBER_UDP_DISCONNECTED

8.71.4 Function Documentation

8.71.4.1 EmberStatus emberGetUdpConnectionData (EmberUdpConnectionHandle *connection,* EmberUdpConnectionData ∗ *data*)

8.71.4.2 bool emberUdpAmListening (uint16_t *port*)

8.71.4.3 EmberStatus emberUdpListenLocal (uint16_t *port*)

8.71.4.4 EmberStatus emberUdpMulticastListen (uint16_t *port,* const uint8_t ∗ *multicastAddress*)

8.71.4.5 void emberUdpStopListening (uint16_t *port*)

8.71.4.6 EmberStatus emUdpListen (uint16_t *port,* const uint8_t ∗ *localAddress*)

8.72 udp.h File Reference

Simple UDP API.

Functions

- EmberStatus emberUdpListen (uint16_t port, const uint8_t *address)

  This function sets up a listener for UDP messages for a given address.
- EmberStatus emberSendUdp (const uint8_t *destination, uint16_t sourcePort, uint16_t destinationPort, uint8_t *payload, uint16_t payloadLength)

  This function sends a UDP message.
- void emberUdpHandler (const uint8_t *destination, const uint8_t *source, uint16_t localPort, uint16_t remotePort, const uint8_t *payload, uint16_t payloadLength)

  An application callback for an incoming UDP message.
- void emberUdpMulticastHandler (const uint8_t *destination, const uint8_t *source, uint16_t localPort, uint16_t remotePort, const uint8_t *payload, uint16_t payloadLength)

  An application callback for an incoming UDP multicast.

8.73 zcl-core-types.h File Reference

Data Structures

- struct EmberZclStringType_t
- struct EmberZclUid_t
- struct EmberZclClusterSpec_t
- struct EmberZclAttributeContext_t
- struct EmberZclBindingContext_t
- struct EmberZclCommandContext_t
- struct EmberZclNotificationContext_t
- struct EmberZclCoapEndpoint_t
- struct EmberZclApplicationDestination_t
- struct EmberZclDestination_t
- struct EmberZclAttributeWriteData_t
- struct EmberZclBindingEntry_t
- struct EmberZclGroupEntry_t
- struct EmberZclReportingConfiguration_t

Macros

- #define EMBER_ZCL_URI_MAX_LENGTH 120
- #define EMBER_ZCL_URI_PATH_MAX_LENGTH 29
- #define EMBER_ZCL_URI_PATH_CLUSTER_ID_MAX_LENGTH 11
- #define EMBER_ZCL_URI_PATH_MANUFACTURER_CODE_CLUSTER_ID_SEPARATOR '_'
- #define EMBER_ZCL_STRING_OVERHEAD 1
- #define EMBER_ZCL_STRING_LENGTH_MAX 0xFE
- #define EMBER_ZCL_STRING_LENGTH_INVALID 0xFF
- #define EMBER_ZCL_LONG_STRING_OVERHEAD 2
- #define EMBER_ZCL_LONG_STRING_LENGTH_MAX 0xFFFE
- #define EMBER_ZCL_LONG_STRING_LENGTH_INVALID 0xFFFF
- #define EMBER_ZCL_UID_BITS 256
- #define EMBER_ZCL_UID_SIZE EMBER_BITS_TO_BYTES(EMBER_ZCL_UID_BITS)
- #define EMBER_ZCL_UID_STRING_LENGTH (EMBER_ZCL_UID_BITS / 4)
- #define EMBER_ZCL_UID_STRING_SIZE (EMBER_ZCL_UID_STRING_LENGTH + 1)
- #define EMBER_ZCL_UID_BASE64URL_LENGTH (((EMBER_ZCL_UID_SIZE * 8) + 5) / 6)
- #define EMBER_ZCL_UID_BASE64URL_SIZE (EMBER_ZCL_UID_BASE64URL_LENGTH + 1)

- #define EMBER_ZCL_ENDPOINT_MIN 0x01
- #define EMBER_ZCL_ENDPOINT_MAX 0xF0
- #define EMBER_ZCL_ENDPOINT_NULL ((EmberZclEndpointId_t)-1)
- #define EMBER_ZCL_ENDPOINT_INDEX_NULL ((EmberZclEndpointIndex_t)-1)
- #define EMBER_ZCL_DEVICE_ID_NULL ((EmberZclDeviceId_t)-1)
- #define EMBER_ZCL_GROUP_ALL_ENDPOINTS 0xFFFF
- #define EMBER_ZCL_GROUP_MIN 0x0001
- #define EMBER_ZCL_GROUP_MAX 0xFFF7
- #define EMBER_ZCL_GROUP_NULL 0x0000
- #define EMBER_ZCL_MANUFACTURER_CODE_NULL 0x0000
- #define EMBER_ZCL_CLUSTER_NULL ((EmberZclClusterId_t)-1)
- #define EMBER_ZCL_ATTRIBUTE_CLUSTER_REVISION 0xFFFD
- #define EMBER_ZCL_ATTRIBUTE_REPORTING_STATUS 0xFFFE
- #define EMBER_ZCL_ATTRIBUTE_NULL ((EmberZclAttributeId_t)-1)
- #define EMBER_ZCL_CLUSTER_REVISION_PRE_ZCL6 0
- #define EMBER_ZCL_CLUSTER_REVISION_ZCL6 1
- #define EMBER_ZCL_CLUSTER_REVISION_NULL ((EmberZclClusterRevision_t)-1)
- #define EMBER_ZCL_BINDING_NULL ((EmberZclBindingId_t)-1)
- #define EMBER_ZCL_COMMAND_NULL ((EmberZclCommandId_t)-1)
- #define EMBER_ZCL_REPORTING_CONFIGURATION_DEFAULT 0
- #define EMBER_ZCL_REPORTING_CONFIGURATION_NULL ((EmberZclReportingConfigurationId_t)-1)
- #define EMBER_ZCL_MAX_GROUP_NAME_LENGTH 0

Typedefs

- typedef uint8_t data8_t
- typedef uint16_t data16_t
- typedef uint32_t data32_t
- typedef uint64_t data64_t
- typedef uint8_t bitmap8_t
- typedef uint16_t bitmap16_t
- typedef uint32_t bitmap32_t
- typedef uint64_t bitmap64_t
- typedef uint8_t enum8_t
- typedef uint16_t enum16_t
- typedef uint32_t utc_time_t
- typedef uint8_t EmberZclEndpointId_t
- typedef uint8_t EmberZclEndpointIndex_t
- typedef uint16_t EmberZclDeviceId_t
- typedef uint16_t EmberZclGroupId_t
- typedef uint16_t EmberZclManufacturerCode_t
- typedef uint16_t EmberZclClusterId_t
- typedef uint16_t EmberZclAttributeId_t
- typedef uint16_t EmberZclClusterRevision_t
- typedef uint8_t EmberZclBindingId_t
- typedef uint8_t EmberZclCommandId_t
- typedef uint8_t EmberZclReportingConfigurationId_t
- typedef void(* EmberZclReadAttributeResponseHandler) (EmberZclMessageStatus_t status, const EmberZclAttributeContext_t *context, const void *buffer, size_t bufferLength)
- typedef void(* EmberZclWriteAttributeResponseHandler) (EmberZclMessageStatus_t status, const EmberZclAttributeContext_t *context)
- typedef void(* EmberZclBindingResponseHandler) (EmberZclMessageStatus_t status, const EmberZclBindingContext_t *context, const EmberZclBindingEntry_t *entry)

Enumerations

- enum EmberZclStatus_t {
  EMBER_ZCL_STATUS_SUCCESS = 0x00,
  EMBER_ZCL_STATUS_FAILURE = 0x01,
  EMBER_ZCL_STATUS_NOT_AUTHORIZED = 0x7E,
  EMBER_ZCL_STATUS_RESERVED_FIELD_NOT_ZERO = 0x7F,
  EMBER_ZCL_STATUS_MALFORMED_COMMAND = 0x80,
  EMBER_ZCL_STATUS_UNSUP_CLUSTER_COMMAND = 0x81,
  EMBER_ZCL_STATUS_UNSUP_GENERAL_COMMAND = 0x82,
  EMBER_ZCL_STATUS_UNSUP_MANUF_CLUSTER_COMMAND = 0x83,
  EMBER_ZCL_STATUS_UNSUP_MANUF_GENERAL_COMMAND = 0x84,
  EMBER_ZCL_STATUS_INVALID_FIELD = 0x85,
  EMBER_ZCL_STATUS_UNSUPPORTED_ATTRIBUTE = 0x86,
  EMBER_ZCL_STATUS_INVALID_VALUE = 0x87,
  EMBER_ZCL_STATUS_READ_ONLY = 0x88,
  EMBER_ZCL_STATUS_INSUFFICIENT_SPACE = 0x89,
  EMBER_ZCL_STATUS_DUPLICATE_EXISTS = 0x8A,
  EMBER_ZCL_STATUS_NOT_FOUND = 0x8B,
  EMBER_ZCL_STATUS_UNREPORTABLE_ATTRIBUTE = 0x8C,
  EMBER_ZCL_STATUS_INVALID_DATA_TYPE = 0x8D,
  EMBER_ZCL_STATUS_INVALID_SELECTOR = 0x8E,
  EMBER_ZCL_STATUS_WRITE_ONLY = 0x8F,
  EMBER_ZCL_STATUS_INCONSISTENT_STARTUP_STATE = 0x90,
  EMBER_ZCL_STATUS_DEFINED_OUT_OF_BAND = 0x91,
  EMBER_ZCL_STATUS_INCONSISTENT = 0x92,
  EMBER_ZCL_STATUS_ACTION_DENIED = 0x93,
  EMBER_ZCL_STATUS_TIMEOUT = 0x94,
  EMBER_ZCL_STATUS_ABORT = 0x95,
  EMBER_ZCL_STATUS_INVALID_IMAGE = 0x96,
  EMBER_ZCL_STATUS_WAIT_FOR_DATA = 0x97,
  EMBER_ZCL_STATUS_NO_IMAGE_AVAILABLE = 0x98,
  EMBER_ZCL_STATUS_REQUIRE_MORE_IMAGE = 0x99,
  EMBER_ZCL_STATUS_NOTIFICATION_PENDING = 0x9A,
  EMBER_ZCL_STATUS_HARDWARE_FAILURE = 0xC0,
  EMBER_ZCL_STATUS_SOFTWARE_FAILURE = 0xC1,
  EMBER_ZCL_STATUS_CALIBRATION_ERROR = 0xC2,
  EMBER_ZCL_STATUS_NULL = 0xFF }
- enum EmberZclMessageStatus_t {
  EMBER_ZCL_MESSAGE_STATUS_COAP_TIMEOUT = EMBER_COAP_MESSAGE_TIMED_OUT,
  EMBER_ZCL_MESSAGE_STATUS_COAP_ACK = EMBER_COAP_MESSAGE_ACKED,
  EMBER_ZCL_MESSAGE_STATUS_COAP_RESET = EMBER_COAP_MESSAGE_RESET,
  EMBER_ZCL_MESSAGE_STATUS_COAP_RESPONSE = EMBER_COAP_MESSAGE_RESPONSE,
  EMBER_ZCL_MESSAGE_STATUS_DISCOVERY_TIMEOUT,
  EMBER_ZCL_MESSAGE_STATUS_NULL = 0xFF }
- enum EmberZclRole_t {
  EMBER_ZCL_ROLE_CLIENT = 0,
  EMBER_ZCL_ROLE_SERVER = 1 }
- enum {
  EMBER_ZCL_NO_FLAGS = 0x00,
  EMBER_ZCL_USE_COAPS_FLAG = 0x01,
  EMBER_ZCL_HAVE_IPV6_ADDRESS_FLAG = 0x02,
  EMBER_ZCL_HAVE_UID_FLAG = 0x04 }
- enum EmberZclApplicationDestinationType_t {
  EMBER_ZCL_APPLICATION_DESTINATION_TYPE_ENDPOINT = 0x00,
  EMBER_ZCL_APPLICATION_DESTINATION_TYPE_GROUP = 0x01 }
- enum EmberZclScheme_t {
  EMBER_ZCL_SCHEME_COAP = 0x00,
  EMBER_ZCL_SCHEME_COAPS = 0x01 }

- enum EmberZclNetworkDestinationType_t {
  EMBER_ZCL_NETWORK_DESTINATION_TYPE_ADDRESS = 0x00,
  EMBER_ZCL_NETWORK_DESTINATION_TYPE_UID = 0x01 }

8.74 zcl-core-well-known.h File Reference

```
#include <PLATFORM_HEADER>
#include <CONFIGURATION_HEADER>
#include "zcl-core.h"
```

Enumerations

- enum EmberZclDiscoveryRequestMode {
  EMBER_ZCL_DISCOVERY_REQUEST_SINGLE_QUERY = 0,
  EMBER_ZCL_DISCOVERY_REQUEST_MULTIPLE_QUERY = 1,
  EMBER_ZCL_DISCOVERY_REQUEST_MODE_MAX = 2 }

Functions

- void emberZclDiscInit (void)
- bool emberZclDiscSetMode (EmberZclDiscoveryRequestMode mode)
- bool emberZclDiscSend (EmberCoapResponseHandler responseHandler)
- bool emberZclDiscByClusterId (const EmberZclClusterSpec_t *clusterSpec, EmberCoapResponseHandler responseHandler)
- bool emberZclDiscByEndpoint (EmberZclEndpointId_t endpointId, EmberZclDeviceId_t deviceId, EmberCoapResponseHandler responseHandler)
- bool emberZclDiscByUid (const EmberZclUid_t *uid, uint16_t uidBits, EmberCoapResponseHandler responseHandler)
- bool emberZclDiscByClusterRev (EmberZclClusterRevision_t version, EmberCoapResponseHandler responseHandler)
- bool emberZclDiscByDeviceId (EmberZclDeviceId_t deviceId, EmberCoapResponseHandler responseHandler)
- bool emberZclDiscByResourceVersion (EmberZclClusterRevision_t version, EmberCoapResponseHandler responseHandler)

8.75 zcl-core.h File Reference

```
#include <PLATFORM_HEADER>
#include <EMBER_AF_API_STACK>
#include "zclip-struct.h"
#include "cbor.h"
#include "zcl-core-types.h"
#include "thread-zclip.h"
```

Functions

- EmberZclEndpointIndex_t emberZclEndpointIdToIndex (EmberZclEndpointId_t endpointId, const EmberZcl↩ ClusterSpec_t *clusterSpec)
- EmberZclEndpointId_t emberZclEndpointIndexToId (EmberZclEndpointIndex_t index, const EmberZcl↩ ClusterSpec_t *clusterSpec)
- bool emberZclIsEndpointInGroup (EmberZclEndpointId_t endpointId, EmberZclGroupId_t groupId)
- bool emberZclGetGroupName (EmberZclEndpointId_t endpointId, EmberZclGroupId_t groupId, uint8_↩ t *groupName, uint8_t *groupNameLength)
- EmberZclStatus_t emberZclAddEndpointToGroup (EmberZclEndpointId_t endpointId, EmberZclGroupId_↩ t groupId, const uint8_t *groupName, uint8_t groupNameLength)
- EmberZclStatus_t emberZclRemoveEndpointFromGroup (EmberZclEndpointId_t endpointId, EmberZcl↩ GroupId_t groupId)
- EmberZclStatus_t emberZclRemoveEndpointFromAllGroups (EmberZclEndpointId_t endpointId)
- EmberZclStatus_t emberZclRemoveGroup (EmberZclGroupId_t groupId)
- EmberZclStatus_t emberZclRemoveAllGroups (void)
- EmberStatus emberZclStartEzMode (void)
- void emberZclStopEzMode (void)
- bool emberZclEzModeIsActive (void)
- int32_t emberZclCompareClusterSpec (const EmberZclClusterSpec_t *s1, const EmberZclClusterSpec_↩ t *s2)
- bool emberZclAreClusterSpecsEqual (const EmberZclClusterSpec_t *s1, const EmberZclClusterSpec_t *s2)
- void emberZclReverseClusterSpec (const EmberZclClusterSpec_t *s1, EmberZclClusterSpec_t *s2)
- void emberZclResetAttributes (EmberZclEndpointId_t endpointId)
- EmberZclStatus_t emberZclReadAttribute (EmberZclEndpointId_t endpointId, const EmberZclClusterSpec↩ _t *clusterSpec, EmberZclAttributeId_t attributeId, void *buffer, size_t bufferLength)
- EmberZclStatus_t emberZclWriteAttribute (EmberZclEndpointId_t endpointId, const EmberZclClusterSpec↩ _t *clusterSpec, EmberZclAttributeId_t attributeId, const void *buffer, size_t bufferLength)
- EmberZclStatus_t emberZclExternalAttributeChanged (EmberZclEndpointId_t endpointId, const EmberZcl↩ ClusterSpec_t *clusterSpec, EmberZclAttributeId_t attributeId, const void *buffer, size_t bufferLength)
- EmberStatus emberZclSendAttributeRead (const EmberZclDestination_t *destination, const EmberZcl↩ ClusterSpec_t *clusterSpec, const EmberZclAttributeId_t *attributeIds, size_t attributeIdsCount, const EmberZclReadAttributeResponseHandler handler)
- EmberStatus emberZclSendAttributeWrite (const EmberZclDestination_t *destination, const EmberZcl↩ ClusterSpec_t *clusterSpec, const EmberZclAttributeWriteData_t *attributeWriteData, size_t attributeWrite↩ DataCount, const EmberZclWriteAttributeResponseHandler handler)
- bool emberZclHasBinding (EmberZclBindingId_t bindingId)
- bool emberZclGetBinding (EmberZclBindingId_t bindingId, EmberZclBindingEntry_t *entry)
- bool emberZclSetBinding (EmberZclBindingId_t bindingId, const EmberZclBindingEntry_t *entry)
- EmberZclBindingId_t emberZclAddBinding (const EmberZclBindingEntry_t *entry)
- bool emberZclRemoveBinding (EmberZclBindingId_t bindingId)
- bool emberZclRemoveAllBindings (void)
- EmberStatus emberZclSendAddBinding (const EmberZclDestination_t *destination, const EmberZcl↩ BindingEntry_t *entry, const EmberZclBindingResponseHandler handler)
- EmberStatus emberZclSendUpdateBinding (const EmberZclDestination_t *destination, const EmberZcl↩ BindingEntry_t *entry, EmberZclBindingId_t bindingId, const EmberZclBindingResponseHandler handler)
- EmberStatus emberZclSendRemoveBinding (const EmberZclDestination_t *destination, const EmberZcl↩ ClusterSpec_t *clusterSpec, EmberZclBindingId_t bindingId, const EmberZclBindingResponseHandler han↩ dler)
- bool emberZclGetDestinationFromBinding (const EmberZclClusterSpec_t *clusterSpec, EmberZclBinding↩ Id_t *bindingIdx, EmberZclDestination_t *destination)
- EmberStatus emberZclSendDefaultResponse (const EmberZclCommandContext_t *context, EmberZcl↩ Status_t status)
- void emberZclReportingConfigurationsFactoryReset (EmberZclEndpointId_t endpointId)

  This function performs a factory reset of the reporting configurations:-.
- uint8_t emberZclStringLength (const uint8_t *buffer)

- uint8_t emberZclStringSize (const uint8_t *buffer)
- uint16_t emberZclLongStringLength (const uint8_t *buffer)
- uint16_t emberZclLongStringSize (const uint8_t *buffer)

Index