MiWi™ Software Design Guide

Introduction

The MiWi is Microchip's proprietary wireless networking stack designed to support Low Rate Personal Area Networks (LRPANs). This guide describes the MiWi applications implemented on the MiWi protocol available in the SAM platforms (SAMR21 and SAMR30).

The MiWi supports the following three network topologies:

- Peer-to-Peer (P2P)
- Star
- Mesh

Features

Earlier versions of the MiWi Mesh networking stack (until version 2.10), released in the MiWi protocol v5.30 of Microchip Libraries for Applications (MLA) v2017-03-06, supports a library-based Mesh networking stack. However, this is redesigned with the following changes:

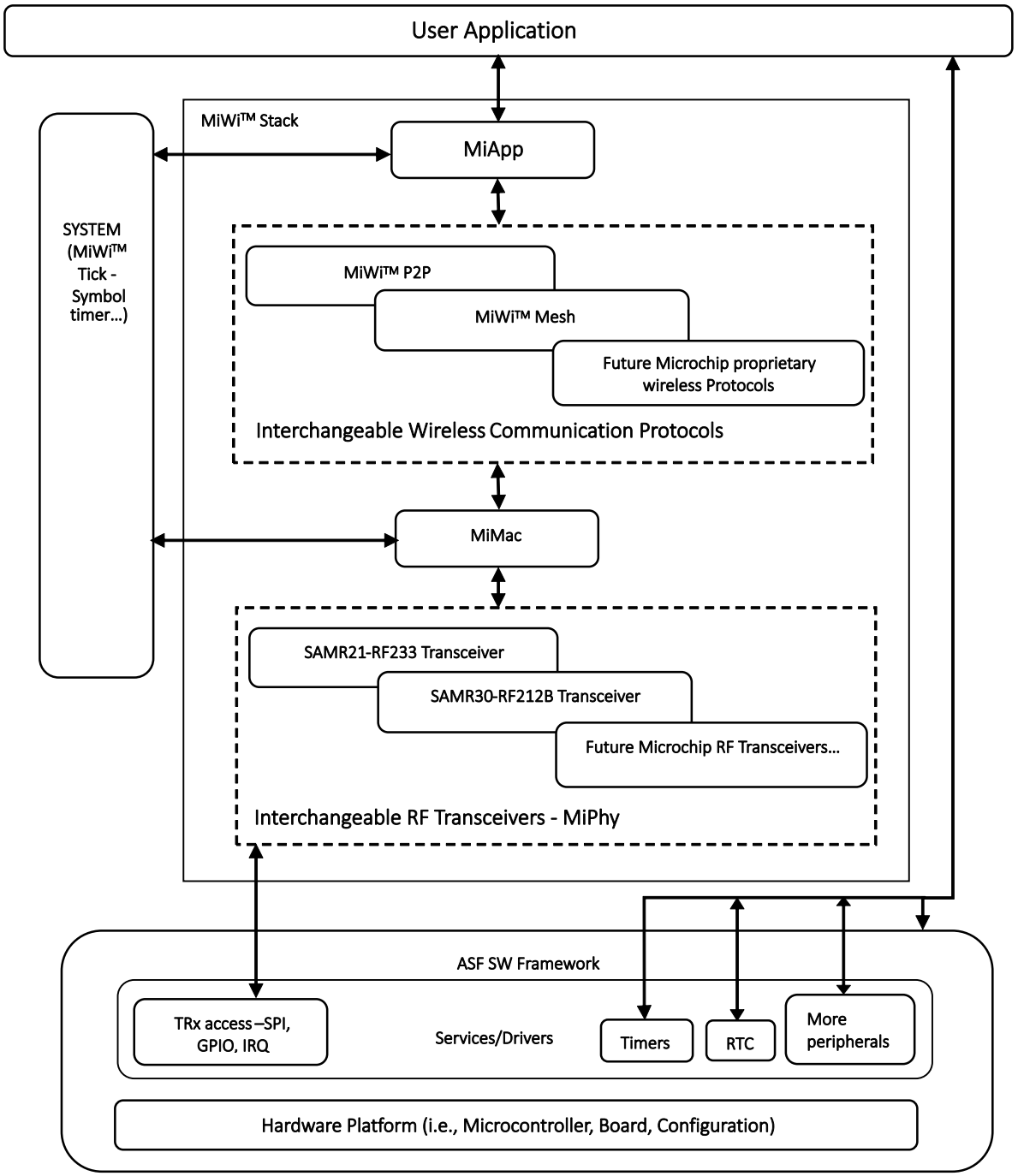
1. Optimization of current APIs to improve simplicity.
2. Redesign of the MiWi Mesh with additional features for next generation platforms.
3. A new commissioning procedure to improve the secured inclusion of devices to the network.
4. Dynamic switching between device types in the MiWi Mesh.
5. Network secure feature for all network messages.
6. Over-The-Air Upgrade to upgrade all the nodes in the network.

Table of Contents

1. MiWi Architecture

The following is the MiWi protocol architecture on Advanced Software Framework (ASF) which allows the user to obtain required components, services and drivers from ASF Wizard. For more details, refer to the ASF Wizard section at the Atmel Software Framework web page.

Figure 1-1. MiWi™ Architecture



2. MiWi Mesh Device Types

The MiWi Mesh protocol supports the following device types:

1. PAN Coordinator
   - 1.1. Starts the network
   - 1.2. Assigns and maintains the coordinators and its end-devices addresses
   - 1.3. Behaves as coordinator for routing frames
   - 1.4. Controls the devices which can be included into the network through commissioning
2. Coordinator
   - 2.1. Joins a network as an end-device
   - 2.2. Requests PAN coordinator for role upgrade to become a coordinator
   - 2.3. Supports routing of frames within the network
   - 2.4. Stores the commissioning information from PAN coordinator and allows only the commissioned devices to participate in the network
   - 2.5. Maintains its end-devices and their addresses
   - 2.6. Maintains data for sleeping end-devices
3. End-Device
   - 3.1. Joins to network though available coordinators
   - 3.2. Supports Rx-On end-device and Sleeping end-device for battery operated devices
   - 3.3. Supports dynamic switching between Rx-On to Sleeping end-device and vice versa

3. MiWi Mesh Frame Format

The network header and application payload of the MiWi Mesh are encapsulated inside the standard IEEE® 805.15.4 data frame payload, but the stack does not adhere to the standard. Therefore, the MiWi Mesh does not receive and process IEEE 805.15.4 command frames. The following figure illustrates a general frame format composed of an IEEE 805.15.4 MAC header, network header, application payload, optional message integrity code (MIC), and a check sum (CRC).

Figure 3-1. General MiWi™ Frame Format

2	1	2	2/8	2	0/2/8	1	1	1	0/2	0/2	0/2/8	0/5	Variable	0/4	2
Frame Control	Sequence number	Dest. PANID	Dest. Address	Source PANID	Source Address	Hops	Frame Control	Sequence number	Dest. PANID	Dest. Address	Source Address	Auxiliary Security Header	Payload	MIC	CRC
MAC Header						Network Header							Payload	Network Footer	

3.1 MAC Header – Frame Control Field

The following figure illustrates the Frame Control field of the MAC header.

Figure 3-2. MAC Header – Frame Control Field

Bits:0-2	3	4	5	6	7:9	10:11	12:13	14:15
Frame Type	Security Enabled	Frame Pending	Ack. Request	PAN ID Compression	Reserved	Dest. Address Mode	Frame Version	Source Address Mode

This is the fixed MAC Frame control field settings used in MiWi Mesh. The following table lists the settings used for a Frame Control field of the MAC header.

Table 3-1. MAC Frame Control Field Settings

Field Name	Settings
Frame Type	Data
Security Enabled	False
Frame Pending	True if pending data available for sleeping end device, otherwise false
Acknowledgment Request	True for unicast frames and false for broadcast frames
PAN ID Compression	True
Destination Addressing Mode	0 for no address fields, 2 for 16-bit short address and 3 for 64-bit extended address
Frame Version	0
Source Addressing Mode	0 for no address fields, 2 for 16-bit short address and 3 for 64-bit extended address

3.1.1 Frame Type

The Frame Type subfield is 3 bits in length and shall be set to one of the non-reserved values (see Table 3-2).

3.1.2 Security Enabled

The Security Enabled subfield is 1 bit in length, and it shall be set to one if the frame is protected by the MAC sublayer and shall be set to zero otherwise. The Auxiliary Security Header field of the MHR shall be present only if the Security Enabled subfield is set to one.

Table 3-2. Values of the Frame Type Subfield

Frame Type Value $b_2\ b_1\ b_0$	Description
000	Beacon
001	Data
010	Acknowledgment
011	MAC command
100–111	Reserved

3.1.3 Frame Pending

The Frame Pending subfield is 1 bit in length and shall be set to one if the device sending the frame has more data for the recipient. This subfield shall be set to zero otherwise.

The Frame Pending subfield shall be used only in beacon frames or frames transmitted either during the CAP by devices operating on a beacon-enabled PAN or at any time by devices operating on a non-beacon enabled PAN. At all other times, it shall be set to zero on transmission and ignored on reception.

3.1.4 Acknowledgment Request

The Acknowledgment Request subfield is 1 bit in length and specifies whether an acknowledgment is required from the recipient device on receipt of a data or MAC command frame. If this subfield is set to one, the recipient device shall send an acknowledgment frame only if, upon reception, the frame passes the third level of filtering. If this subfield is set to zero, the recipient device shall not send an acknowledgment frame.

3.1.5 PAN ID Compression

The PAN ID Compression subfield is 1 bit in length and specifies whether the MAC frame is to be sent containing only one of the PAN identifier fields when both source and destination addresses are present. If this subfield is set to one and both the source and destination addresses are present, the frame shall contain only the Destination PAN Identifier field and the Source PAN Identifier field shall be assumed equal to that of the destination. If this subfield is set to zero and both the source and destination addresses are present, the frame shall contain both the Source PAN Identifier and Destination PAN Identifier fields. If only one of the addresses is present, this subfield shall be set to zero, and the frame shall contain the PAN Identifier field corresponding to the address. If neither address is present, this subfield shall be set to zero, and the frame shall not contain either PAN Identifier field.

3.1.6 Destination Addressing Mode

The Destination Addressing Mode subfield is 2 bits in length and shall be set to one of the nonreserved values as listed in the following table. If this subfield is equal to zero and the Frame Type subfield does

not specify that this frame is an acknowledgment or beacon frame, the Source Addressing Mode subfield shall be nonzero, implying that the frame is directed to the PAN coordinator with the PAN identifier as specified in the Source PAN Identifier field.

Table 3-3. Possible values of the Destination Addressing Mode and Source Addressing Mode subfields

Addressing Mode Value b_1b_0	Description
00	PAN Identifier and address fields are not present
01	Reserved
10	Address field contains a 16-bit short address
11	Address field contains a 64-bit extended address

3.1.7 Frame Version

The Frame Version subfield is 2 bits in length and specifies the version number corresponding to the frame. This subfield shall be set to 0x00 to indicate a frame compatible with IEEE Std 802.15.4-2003 and 0x01 to indicate an IEEE 802.15.4 frame. All other subfield values shall be reserved for future use.

3.1.8 Source Addressing Mode

The Source Addressing Mode subfield is 2 bits in length and shall be set to one of the nonreserved values listed in Table 3-3. If this subfield is equal to zero and the Frame Type subfield does not specify that this frame is an acknowledgment frame, the Destination Addressing Mode subfield shall be nonzero, implying that the frame has originated from the PAN coordinator with the PAN identifier as specified in the Destination PAN Identifier field.

3.2 Network Header

3.2.1 Hops Field

The Hops field provides the number of hops the packet is allowed to be retransmitted. For example, 00h indicates that the packet is not retransmitted. Maximum possible hop is 0xFF.

3.2.2 Frame Control Field

The Frame Control field is a bitmap which defines the behavior of a packet as shown in the following figure.

Figure 3-3. Network Header – Frame Control Field

Bits:0-1	2	3	4	5	6-7
Frame Type	Security Enabled	Infra Cluster	Ack. Request	Address same as MAC	Reserved

The following table details the Frame Control field of the Network Header.

Table 3-4. Network Header Frame Control Field Description

Bit Number	Field Name	Description
6-7	Reserved	Set the bit as ‘0’ for this implementation.
5	Address same as MAC	This bit is set when the MAC Address fields and Network Address fields are same. This is useful when the sleeping end-device polls the parent for data, with relatively less bytes over-the-air for single hop from the network layer.
4	Acknowledgment Request	This bit is set when the source device requests an Network layer acknowledgment of receipt from the destination device.
3	Intra Cluster	Reserved in this implementation. Set the bit as ‘1’.
2	Security Enabled	This bit is set when data packet is encrypted at the application level.
0-1	Frame Type	These bits indicate as following: • 00 – Data • 01 – Command • 10 – Manufacturer specific • 11 – Reserved

3.2.3 Sequence Number Field

The Sequence Number field is 1 byte in length and specifies the sequence identifier for the frame. The Sequence Number field shall be increased by 1 for every outgoing frame, originating on the node and it must not be changed for routed frames.

3.2.4 Destination PANID Field

The Destination PANID field is 2 bytes in length, specifies the PAN identifier of the intended recipient of the frame. This field will be present only if Address is same as MAC bit which is set to 0.

3.2.5 Source Address Field

The Source Address field is 2 bytes in length and specifies the network address of the node originating the frame.

3.2.6 Destination Address Field

The Destination Address field is 2 bytes in length and specifies the network address of the destination node. The Destination Address field can be set as per the following table for other frames except unicast to a node. Data transmission using long address is not supported.

Table 3-5. Network Header Destination Address Field Description

Destination Address Value	Description
0xFFFF	Broadcast to every device
0xFFFE	Multicast to all FFD’s

.....continued	
Destination Address Value	**Description**
0xFFFD	Multicast to all Coordinators

3.2.7 Auxiliary Security Header Field

The Auxiliary Security Header field specifies information required for security processing, including how the frame is protected (security level) and frame counter. This field shall be present only if the Security Enabled sub-field in Frame control field is set to one.

Table 3-6. Auxiliary Security Header Field

Bytes: 1	4	8
Security Level	Frame Counter	Source long address

3.2.7.1 Security Level

The supported security level are:

- 0 (No Security)
- 1 (Authentication - 4 bytes MIC)
- 4 (Encryption only)
- 5 (Encryption with Authentication - 4 bytes MIC)

4. MiWi Mesh – Device Addressing Mechanism

The MiWi Mesh uses a 2 bytes short address to specify nodes in the network when performing routing across the network. The address is allocated during the joining process. The lower byte is used to identify the end-devices. The higher byte is used to identify the coordinators.

Bit 15:8	Bit 7	Bit 6:0
Coordinator identifier	RxOnWhenIdle	End-device identifier

5. Network Freezer

The Network Freezer feature saves critical network information into the Nonvolatile Memory (NVM) and restores them after power cycle. In this way, the application supports the power cycle scenario and the network can be restored to the previous state of the power cycle without many message exchanges after the power cycle.

Additionally, wear-leveling implementation reduces the number of “backup-erase-re-write” cycles and thereby improves the Flash lifetime. Refer to the `miwi_mesh_pds.c` file which specifies information about the parameters stored in the NVM.

5.1 Interface

The Network Freezer feature is enabled by defining `ENABLE_NETWORK_FREEZER` API in the configuration file of the application project. This feature is invoked by calling the MiApp function `MiApp_ProtocolInit`. When Network Freezer is enabled in the application, the network information is restored from NVM; otherwise, the wireless node starts from initial stage. If Network Freezer is disabled, the node always starts as a factory new device.

5.2 Additional Notes

The Network Freezer feature requires NVM to store the critical network information. The NVM used for this implementation is the internal Flash.

6. Sleep Mode

For most of the applications, it is critical to provide long battery life for the sleeping devices. A device can be in either the Active mode or Sleep mode. After being powered-up, a node always starts in the Active mode, with its MCU fully turned on. An application can check whether the stack is allowing it to sleep or not using the `ENABLE_SLEEP_FEATURE` API. If it allows, the application can go to sleep at a maximum of allowable time by stack for proper operation.

In the Sleep mode, the RF chip and the MCU are in Low-Power state and only the functionality required for MCU wake up remains active. Thus, in Sleep mode, the application cannot perform any radio Tx/Rx operations or communicate with the external periphery.

Major power is consumed during the Active mode, requesting for and sending data in the duty cycle. Therefore, for a device to be active is based on its polling period. This can be controlled using a configuration option. Among all nodes, only end-devices can sleep.

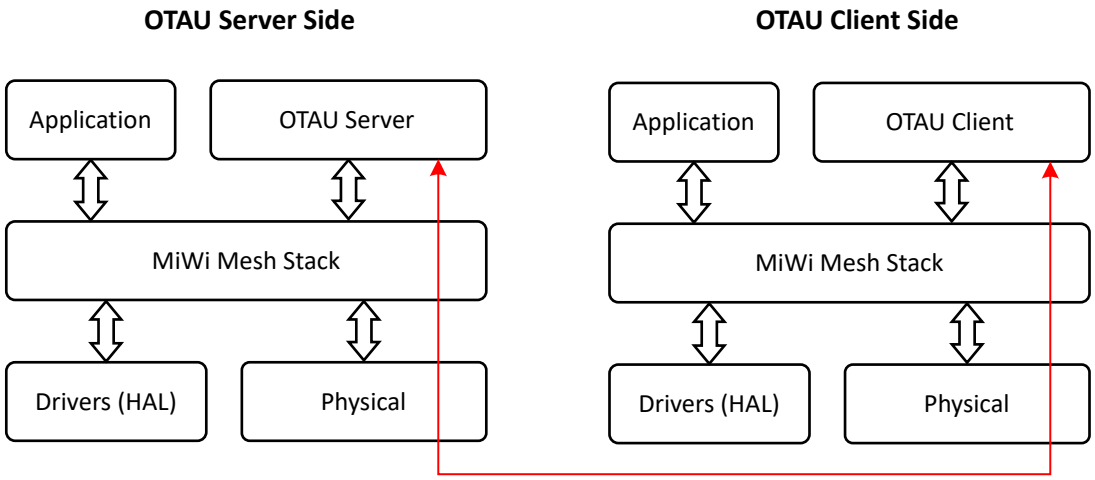
6.1 Interface

The Sleep mode can be enabled by defining `ENABLE_SLEEP_FEATURE` API in the configuration file of the application project. For more details, see MiApp API Description.

7. Over-The-Air Upgrade

The following figure shows the firmware architecture of the Over-The-Air Upgrade (OTAU).

Figure 7-1. OTAU Firmware Architecture



7.1 OTAU Server

The OTAU server receives or transmits the command from or to the PC through UART or USB. To upgrade, transmit required frames through MiWi Mesh stack layer to reach the clients. The server acts as a bridge between the clients and an OTAU tool running in the PC; that is, there is no additional intelligence in OTAU module on the server end.

7.2 OTAU Client

The OTAU client receives or transmits the proprietary commands over-the-air to communicate with the OTAU server.

7.3 Domains of OTAU

The following sections describe the Notify and Upgrade domains of the OTAU server and client.

7.3.1 Notify

1. Provides basic information about the client such as, IEEE address, short address, and next hop address to reach the OTAU server for plotting network topology.
2. Commands to power LED on clients to identify visually on large network.
3. Provision for user to fetch additional information related to the application such as, firmware name, firmware version, board name, and board version.

7.3.2 Upgrade

- Supports OTAU of each client through proprietary protocol exchange.
- Provides support to switch to an new image individually when all the nodes are upgraded.

7.4 Compiler Switches for OTAU

OTAU_ENABLED
OTAU_ENABLED switch must be included in project symbols to enable the upgrade support.

OTAU_SERVER
When OTAU_SERVER switch is enabled on the project symbols, the node acts as the OTAU server. If this symbol is not enabled, then the node acts as a Client for OTAU.

8. MiWi Mesh – Networking

The MiWi Mesh network features are categorized as follows:

1. Network commissioning
2. Start and join network
3. Routing in network

8.1 Network Commissioning

The network commissioning controls the devices which can participate in the network.

1. Application on PAN coordinator reads the IEEE address (for example, it can be improved to read from bar code) from one or more devices.
2. PAN coordinator calculates the 64 byte bloom filter value with the read information.
3. Calculated bloom filter value is sent to all the coordinators in the network.
4. Coordinators provide beacon only to the devices which have its IEEE address in the bloom filter.

8.2 Start and Join Network

1. Only the PAN coordinator can start a network.
2. Joining device sends a beacon request to obtain information about the available networks in its personal operating space.
3. The PAN coordinator or coordinator evaluates the beacon request by parsing the given IEEE address with the bloom filter value. If found, it sends a beacon frame with a beacon payload which includes PAN coordinator hop count and bloom filter value (64 bytes). If not found, it discards the packet.
4. Upon receiving the beacon frames, the joining device parses it and checks its own address in the bloom filter value and then decides its parent based on associate permit, children capacity and Link Quality Indicator (LQI) of the received beacons. After choosing the parent, it unicasts Mesh Connection Request packet (includes its capability and JoinWish field) to the selected parent. The JoinWish field has 2-bits C and ED, and remaining bits are reserved.
   - If both bits are set in JoinWish, then the particular device joins as an end-device if the coordinator capacity is currently unavailable in the network.
   - If only the C bit is set, then the device joins as a Coordinator only.
   - If only the ED bit is set, then the device joins as an end-device only.
5. If the parent is the PAN coordinator and the JoinWish field is set with C and ED or C only, then the PAN coordinator checks whether it has a new coordinator address. If available, it sends the Mesh Connection Response with device address as allocated new coordinator address. If address is unavailable or JoinWish field has only ED set, then it allocates the end-device address and sends the Mesh connection Response.
6. If the parent is the coordinator, then it allocates end-device address and sends Mesh Connection Response with device address as allocated end-device address.
7. The joining device parses the Mesh Connection Response and uses the received network address along with the received network key for further communications in the network.

8. The joining device which is coordinator capable, receives an end-device address, and based on Role Upgrade Timeout (configurable), the device sends a role upgrade request packet to the PAN coordinator in order to upgrade its role from an end-device to the coordinator.
9. When the PAN coordinator receives a role upgrade request, it checks whether coordinator address is available. If the address is available, it allocates a new coordinator address and sends the role upgrade response with the allocated address and status as success. If an address is unavailable, then it sends a role upgrade response with failure status.

8.3 Routing in Network

1. During the joining procedure and role upgrade, the route table is updated in all the coordinators.
2. The route table in coordinators is used to route the packet to the destination device.
3. When the device does not have the next hop address for the destination, it will trigger a broadcast for a route request to the destination.
4. Unlike the legacy route request in AODV routing protocols, the reply is generated from any node which has the next hop information in its routing table.
5. The source device (which initiated the route request) selects the route reply for the destination based on the fewer hops and best LQI.
6. However, to establish and synchronize the network periodically, the route table update is broadcasted to a single hop based on pre-configured intervals.
7. This ensures that the coordinators in the network share the neighbor’s information with its neighbors.

9. Macros for MiWi Mesh

This section describes the macros for the MiWi Mesh.

9.1 CHANNEL_MAP

Description	Channel map is a bit map used to select appropriate channels for starting or establishing connection in the network.
Default Value	• For SAMR21 - (1<<25) • For SAMR30 - (1<<2)
Range	Bit map based on the physical layer. Set or clear of any bits in the below range is valid. • For 2.4GHz (SAMR21) – 0x07FFF800 • For SubGHz (SAMR30) – 0x000007FF
Memory Usage	None
Configurable in	`miwi_config_mesh.h`
Remarks	For SAMR30, only channels 1-10 can be changed using this Macro. To use Channel 0, `PHY_Init()` must be modified to include TX Power and PHY Mode setting as per the recommendation from the data sheet for European band.

9.2 KEEP_ALIVE_COORDINATOR_SEND_INTERVAL

Description	Time interval in seconds on which a coordinator capable device sends keep alive frame to PAN coordinator. Upon reception of this frame, PAN coordinator refreshes the timeout for that particular coordinator.
Default Value	120
Range	1 – 65535
Memory Usage	None
Configurable in	`miwi_config_mesh.h`
Remarks	`KEEP_ALIVE_COORDINATOR_TIMEOUT_IN_SEC` is based on this value.

9.3 KEEP_ALIVE_COORDINATOR_TIMEOUT_IN_SEC

Description	Timeout in seconds for which the PAN coordinator maintains the entry of coordinator, for holding its address. Each coordinator is expected to send at least one keep alive frame to PANC within this timeout.
Default Value	KEEP_ALIVE_COORDINATOR_SEND_INTERVAL *10 The default value is 1200 when KEEP_ALIVE_COORDINATOR_SEND_INTERVAL is set as 120.

Range	1 – 65535
Memory Usage	None
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.4 KEEP_ALIVE_RXONENDDEVICE_SEND_INTERVAL

Description	Time interval in seconds on which an end-device sends keep alive frame to its coordinator. Upon reception of this frame, coordinator refreshes the timeout for that particular end-device.
Default Value	120
Range	1 – 65535
Memory Usage	None
Configurable in	`miwi_config_mesh.h`
Remarks	`KEEP_ALIVE_RXONENDDEVICE_TIMEOUT_IN_SEC` is based on this value.

9.5 KEEP_ALIVE_RXONENDDEVICE_TIMEOUT_IN_SEC

Description	Timeout in seconds for which the coordinator maintains the entry of end-device, for holding its address. Each end-device is expected to send at least one keep alive frame to coordinator within this timeout.
Default Value	KEEP_ALIVE_RXONENDDEVICE_SEND_INTERVAL *10 (that is, 1200) The default value is 1200 when KEEP_ALIVE_COORDINATOR_SEND_INTERVAL is set as 120.
Range	1 – 65535
Memory Usage	None
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.6 DATA_REQUEST_SEND_INTERVAL

Description	Time interval in seconds on which a sleeping end-device sends Data Request frame to its coordinator. Upon reception of this frame, coordinator refreshes the timeout for that particular sleeping end-device and sends any data cached in indirect queue.
Default Value	3
Range	1 – 254
Memory Usage	None

Configurable in	`miwi_config_mesh.h`
Remarks	`RXOFF_DEVICE_TIMEOUT_IN_SEC` and `MAXIMUM_DATA_REQUEST_SEND_INTERVAL` is based on this value

9.7 RXOFF_DEVICE_TIMEOUT_IN_SEC

Description	Timeout in seconds for which the coordinator maintains the entry of sleeping end-device, to hold its address. Each sleeping end-device is expected to send at least one Data Request to coordinator within this timeout.
Default Value	DATA_REQUEST_SEND_INTERVAL * 20 (that is, 60) The default value is 60 when DATA_REQUEST_SEND_INTERVAL is set as 3.
Range	1 – 65535
Memory Usage	None
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.8 MAXIMUM_DATA_REQUEST_SEND_INTERVAL

Description	Maximum time interval in seconds for Data Request of end-device in the network.
Default Value	DATA_REQUEST_SEND_INTERVAL * 2 (that is, 6)
Range	1 – 254
Memory Usage	None
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.9 MAX_NUMBER_OF_DEVICES_IN_NETWORK

Description	This macro is used to configure the number of device's IEEE addresses to be stored for commissioning.
Default Value	32
Range	1 – 255
Memory Usage	256 bytes of RAM for 32 entries, that is, 8 bytes per entry
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.10 JOIN_WISH

Description	Configuration to join the network based on defined roles. For more information, see 8.2 Start and Join Network.
Default Value	• For Coordinator – JOINWISH_ANY • For End-device – JOINWISH_ENDEVICE
Range	• JOINWISH_ENDEVICE - 0x01 • JOINWISH_COORD_ALONE - 0x02 • JOINWISH_ANY - 0x03
Memory Usage	None
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.11 ROLE_UPGRADE_INTERVAL_IN_SEC

Description	Time interval in seconds on which a coordinator capable end-device requests the PANC to upgrade its role to coordinator. For more information on Role Upgrade, see 8.2 Start and Join Network.
Default Value	25
Range	1 – 254
Memory Usage	None
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.12 CONNECTION_RESPONSE_WAIT_IN_SEC

Description	Time interval in seconds to wait for Connection Response after sending Connection Request to any coordinator in the network.
Default Value	5
Range	1 – 254
Memory Usage	None
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.13 NUM_OF_COORDINATORS

Description	This macro is used to configure the number of coordinators in the network. Also used to allocate the coordinator table on PANC to maintain the IEEE addresses and timeout for each coordinator.
Default Value	64
Range	1 – 200
Memory Usage	768 bytes of RAM for 64 entries, that is, 12 bytes per entry
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.14 NUM_OF_NONSLEEPING_ENDDEVICES

Description	This macro is used to configure the number of non-sleeping end-devices in the network. Also used to allocate the device table on each coordinator to maintain the IEEE addresses and timeout for each non-sleeping end-device.
Default Value	5
Range	1 – 127
Memory Usage	80 bytes of RAM for 5 entries, that is, 16 bytes per entry
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.15 NUM_OF_SLEEPING_ENDDEVICES

Description	This macro is used to configure the number of sleeping end-devices in the network. Also used to allocate the sleeping device table on each coordinator to maintain the IEEE addresses and timeout for each sleeping end-device.
Default Value	5
Range	1 – 128
Memory Usage	100 bytes of RAM for 5 entries, that is, 20 bytes per entry
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.16 ROUTE_UPDATE_INTERVAL

Description	Periodic time interval in seconds to send route update for neighboring devices after joining the network.

Default Value	60
Range	1 – 254
Memory Usage	None
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.17 ROUTE_REQ_WAIT_INTERVAL

Description	Timeout in seconds to wait for route replies after sending route request to discover route for a specific coordinator.
Default Value	5
Range	1 – 254
Memory Usage	None
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.18 INDIRECT_DATA_WAIT_INTERVAL

Description	Timeout in seconds to hold indirect data to its sleeping end-devices. This must be maintained at least more than twice the Data Request interval to ensure reliable data transfer.
Default Value	25
Range	1 – 254
Memory Usage	None
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.19 ED_LINK_FAILURE_ATTEMPTS

Description	Number of consecutive attempts on end-device made with the parent before confirming link failure.
Default Value	15
Range	1 – 254
Memory Usage	None
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.20 FRAME_RETRY

Description	Defines the number of retries to be performed during failure to reach the destination.
Default Value	3
Range	0 – 254
Memory Usage	None
Configurable in	`miwi_config_mesh.h`
Remarks	This configuration to retry is apart from the basic MAC level retries. **Note:** Any frame going out of the device is retried three times at MAC layer.

9.21 REBROADCAST_TABLE_SIZE

Description	This macro is used to configure the number of entries to be stored to avoid duplicate rebroadcast for every broadcast in the network.
Default Value	10
Range	1 – 255
Memory Usage	40 bytes of RAM for 10 entries, that is, 4 bytes per entry.
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.22 REBROADCAST_TIMEOUT

Description	Timeout in seconds to hold the broadcasted data in rebroadcast table to avoid rebroadcasting again.
Default Value	5
Range	1 – 254
Memory Usage	None
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.23 DUPLICATE_REJECTION_TABLE_SIZE

Description	Size of duplicate rejection table used to avoid multiple data indication to the application.
Default Value	10
Range	1 – 255

Memory Usage	40 bytes of RAM for 10 entries, that is, 4 bytes per entry.
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.24 MAX_BEACON_RESULTS

Description	Number of entries allocated to receive beacon response during active scan.
Default Value	5
Range	1 – 255
Memory Usage	90 bytes of RAM for 5 entries, that is, 18 bytes per entry.
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.25 MESH_SECURITY_LEVEL

Description	Security levels for CCM* as defined in IEEE 802.15.4.
Default Value	5
Range	0 – 7
Memory Usage	None
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.26 PUBLIC_KEY_DEFAULT

Description	Public key is the initial key stored in all devices, the initial communications use this key until it gets the network key.
Default Value	{0x00,0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F}
Range	Any 16 bytes value
Memory Usage	None
Configurable in	`miwi_config_mesh.h`
Remarks	None

9.27 NETWORK_KEY_DEFAULT

Description	Network key used to transact after successful join to the network.

Default Value	{0x00,0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF}
Range	Any 16 bytes value
Memory Usage	None
Configurable in	`miwi_config_mesh.h`
Remarks	None

10. Recommendation for Macros

For example, consider the default tested network (with WSN Demo application) which has following considerations.

1. Network size as 1 PAN coordinator, 50 coordinators, and 30 end-devices.
2. All the devices in network must report to PAN Coordinator at a periodic interval.
3. Data flow is mostly unidirectional (that is, uplink requires more bandwidth).
4. Report from all the devices are monitored using WSNMonitor.

This network is programmed with default values (see 9. Macros for MiWi Mesh), and tested to be working for more than 48 hours.

By default, in the WSN Demo application, each end-device joins to a coordinator based on its available end-device capacity. Since the number of coordinators are greater than end-devices, most end-devices join to coordinators with 100% end-device capacity.

The following macros must be proportionally modified (that is, when the number of coordinators increases, configuration values must be increased and vice versa) based on the change in number of coordinators in the network.

1. NUM_OF_COORDINATORS
2. KEEP_ALIVE_COORDINATOR_SEND_INTERVAL
3. KEEP_ALIVE_COORDINATOR_TIMEOUT_IN_SEC
4. ROUTE_UPDATE_INTERVAL
5. ROLE_UPGRADE_INTERVAL_IN_SEC

The following macros must be proportionally modified (that is, when the number of coordinators increases, configuration values must be increased and vice versa) based on the change in number of end-devices in the network.

- For sleeping end-device:
  1. NUM_OF_SLEEPING_ENDDEVICES
  2. DATA_REQUEST_SEND_INTERVAL
  3. RXOFF_DEVICE_TIMEOUT_IN_SEC
  4. MAXIMUM_DATA_REQUEST_SEND_INTERVAL
  5. INDIRECT_DATA_WAIT_INTERVAL
- For non-sleeping (that is, RXON end-device) end-device:
  1. NUM_OF_NONSLEEPING_ENDDEVICES
  2. KEEP_ALIVE_RXONENDDEVICE_SEND_INTERVAL
  3. KEEP_ALIVE_RXONENDDEVICE_TIMEOUT_IN_SEC

Note: This recommendation of macros is tested on SAMR21 (that is, 2.4 GHz); therefore, for the same network size in SAMR30 (that is, Sub-GHz) the values must be increased to compensate on the reduction in data rate.

10.1 Extending Battery Life for Sleeping End-device

Apart from the Sleep mode supported by the controller, DATA_REQUEST_SEND_INTERVAL configuration directly impacts the frequency of wake up from sleep and the consumption of battery.

11. MiApp API

The following table lists the supported APIs.

Table 11-1. MiApp API

S. No.	Supported APIs	Topology Supported
1	`miwi_status_t MiApp_ProtocolInit (defaultParametersRomOrRam_t *defaultRomOrRamParams, defaultParametersRamOnly_t *defaultRamOnlyParams)`	P2P/Star/ Mesh
2	`bool MiApp_Set(enum id, uint8_t value )`	P2P/Star/ Mesh
3	`bool MiApp_StartNetwork(uint8_t Mode, uint8_t ScanDuration, uint32_t ChannelMap, FUNC ConfCallback)`	P2P/Star/ Mesh
4	`uint8_t MiApp_SearchConnection(uint8_t ScanDuration, uint32_t ChannelMap, FUNC ConfCallback)`	P2P/Star/ Mesh
5	`uint8_t MiApp_EstablishConnection(uint8_t Channel, uint8_t addr_len, uint8_t addr, uint8_t Capability_info, FUNC ConfCallback)`	P2P/Star/ Mesh
6	`void MiApp_RemoveConnection(uint8_t ConnectionIndex)`	P2P/Star/ Mesh
7	`void MiApp_ConnectionMode(uint8_t Mode)`	P2P/Star/ Mesh
8	`MiApp_SendData(uint8_t addr_len, uint8_t addr, uint8_t len, uint8_t pointer, FUNC ConfCallback)`	P2P/Star/ Mesh
9	`MiApp_SubscribeDataIndicationCallback(FUNC callback)`	P2P/Star/ Mesh
10	`uint8_t MiApp_NoiseDetection(uint32_t ChannelMap, uint8_t ScanDuration, uint8_t DetectionMode, OUTPUT uint8_t NoiseLevel)`	P2P/Star/ Mesh
11	`uint8_t MiApp_TransceiverPowerState(uint8_t Mode)`	P2P/Star/ Mesh
12	`bool MiApp_InitChannelHopping(uint32_t ChannelMap)`	P2P/Star/ Mesh
13	`bool MiApp_ResyncConnection(uint8_t ConnectionIndex, uint32_t ChannelMap)`	P2P/Star/ Mesh
14	`uint8_t Total_Connections(void)`	P2P/Star/ Mesh
15	`void MiApp_BroadcastConnectionTable()`	Star
16	`bool SW_Ack_SrED(uint8_t )`	Star

.....continued

S. No.	Supported APIs	Topology Supported
17	void send_link_status(void)	Star
18	`void Find_InActiveDevices(void)`	Star
19	`void MiApp_leave_network(void)`	Star
20	`bool MiApp_UnicastStar (bool SecEn)`	Star
21	`bool MiApp_Set(enum id, uint8_t value )`	Mesh
22	`bool MiApp_IsMemberOfNetwork(void)`	Mesh
23	`bool MiApp_Get(enum id, uint8_t value )`	Mesh
24	`bool MiApp_Set(enum id, uint8_t value )`	Mesh
25	`bool MiApp_SubscribeReConnectionCallback(ReconnectionCallback_t callback)`	P2P/Star/ Mesh
26	`bool MiApp_ResetToFactoryNew(void)`	P2P/Star/ Mesh
27	`bool MiApp_ReadyToSleep(uint32_t* sleepTime)`	Mesh
28	`bool MiApp_ManuSpecSendData(uint8_t addr_len, uint8_t *addr, uint8_t msglen, uint8_t *msgpointer, uint8_t msghandle, bool ackReq, DataConf_callback_t ConfCallback)`	Mesh
29	`bool MiApp_SubscribeManuSpecDataIndicationCallback(PacketIndCallback_t callback)`	Mesh
30	`bool MiApp_IsConnected(void)`	Mesh
31	`uint16_t MiApp_MeshGetNextHopAddr(uint16_t destAddress)`	Mesh

12. MiApp API Description

This section describes the MiApp APIs.

12.1 MiApp_ProtocolInit

API	`miwi_status_t MiApp_ProtocolInit(defaultParametersRomOrRam_t *defaultRomOrRamParams, defaultParametersRamOnly_t *defaultRamOnlyParams)`
Description	This is the primary user interface function to initialize the Microchip proprietary wireless protocol, which is chosen by the application layer. Usually, this function must be called after the hardware initialization, before any other MiApp interface can be called.
Pre-Condition	Hardware initialization must be done.
Parameters	• `defaultParametersRomOrRam_t defaultRomOrRamParams` – Default parameters for MiWi™ Mesh. • `defaultParametersRamOnly_t defaultRamOnlyParams` – Default parameters for MiWi™ Mesh. • Ignored in case of P2P / Star
Returns	Status of Initialization

Example	`<code>`
	`HardwareInit();`
	`MiApp_ProtocolInit();`
	`</code>`

Remarks	If RECONNECTION_IN_PROGRESS status is received, then application needs to wait for reconnection callback before proceeding to call further MiApp API's.

12.2 MiApp_Set

API	`bool MiApp_Set(set_params id, uint8_t *value)`
Description	This is the primary user interface function to set the different values in the MiWi™ stack.
Pre-Condition	Protocol initialization must be done.
Parameters	• `set_params id` – The identifier of the value to be set • `value` – The value to be set
Returns	A boolean to indicate if set operation is performed successfully.

Example	`<code>`
	`if( true == MiApp_Set(CHANNEL, 15) )`
	`{`
	`// channel changes successfully`
	`}`
	`</code>`

Remarks	None

12.3 MiApp_StartConnection

API	`bool MiApp_StartConnection(uint8_t Mode, uint8_t ScanDuration, uint32_t ChannelMap,connectionConf_callback_t ConfCallback)`
Description	This is the primary user interface function for the application layer to start PAN. Usually, this function is called by the PAN coordinator which is the first in the PAN. The PAN coordinator may start the PAN after a noise scan if specified in the input mode.
Pre-Condition	Protocol initialization must be done.
Parameters	• `uint8_t Mode` – whether to start a PAN after a noise scan. Possible modes are as follows. – START_CONN_DIRECT – starts PAN directly without noise scan. – START_CONN_ENERGY_SCN – performs an energy scan first, then starts the PAN on the channel with least noise. – START_CONN_CS_SCN – performs a carrier-sense scan first, then starts the PAN on the channel with least noise. • `uint8_t ScanDuration` – maximum time to perform scan on single channel. The value is from 5 to 14. The real time to perform scan can be calculated in following formula from IEEE 802.15.4 specification: 960 x (2^ScanDuration + 1) x 10^(-6) second ScanDuration is discarded if the connection mode is START_CONN_DIRECT. • `uint32_t ChannelMap` – bit map of channels to perform noise scan. The 32-bit double word parameter uses one bit to represent corresponding channels from 0 to 31. For instance, 0x00000003 represent to scan channel 0 and channel 1. `ChannelMap` is discarded if the connection mode is START_CONN_DIRECT. • `connectionConf_callback_t ConfCallback` – callback routine which is called upon the initiated connection procedure is performed.
Returns	A boolean to indicate if PAN is started successfully.

Example	`<code>`
	`// start the PAN on the least noisy channel after scanning all possible channels.`
	`MiApp_StartConnection(START_CONN_ENERGY_SCN, 10, 0x07FFF800, callback);`
	`</code>`

Remarks	None

12.4 MiApp_SearchConnection

API	`uint8_t MiApp_SearchConnection(uint8_t ScanDuartion, uint32_t ChannelMap, SearchConnectionConf_callback_t ConfCallback)`

Description	This is the primary user interface function for the application layer to perform an active scan. After this function call, all active scan response is stored in the global variable `ActiveScanResults` in the format of structure ACTIVE_SCAN_RESULT. The return value indicates the total number of valid active scan response in the active scan result array.
Pre-Condition	Protocol initialization is done.
Parameters	`uint8_t ScanDuration` – maximum time to perform scan on single channel. The value is from 5 to 14. The real time to perform scan can be calculated with the following formula from the IEEE 802.15.4 specification: 960 x (2^ScanDuration + 1) x 10^(-6) second. `uint32_t ChannelMap` – bit map of channels to perform noise scan. The 32-bit double word parameter uses one bit to represent corresponding channels from 0 to 31. For instance, 0x00000003 represents to scan channel 0 and channel 1. `SearchConnectionConf_callback_t ConfCallback` – callback routine which is called when the initiated connection procedure is performed.
Returns	The number of valid active scan response stored in the global variable `ActiveScanResults`.

Example	`<code>`
	`// Perform an active scan on all possible channels`
	`NumOfActiveScanResponse = MiApp_SearchConnection(10, 0xFFFFFFFF, callback);`
	`</code>`

Remarks	None

12.5 MiApp_EstablishConnection

API	`uint8_t MiApp_EstablishConnection(uint8_t Channel, uint8_t addr_len, uint8_t *addr, uint8_t Capability_info, connectionConf_callback_t ConfCallback)`
Description	This is the primary user interface function for the application layer to start communication with an existing PAN. For P2P protocol, this function call can establish one or more connections. For network protocol, this function can be used to join the network, or establish a virtual socket connection with a node out of the radio range.
Pre-Condition	Protocol initialization is done. If only to establish connection with a predefined device, an active scan must be performed before and valid active scan result must be saved.
Parameters	• `uint8_t channel` – selected channel to invoke join procedure. • `uint8_t addr_len` – address length • `uint8_t *addr` – address of the parent • `uint8_t Capability_info` – capability information of the device • `connectionConf_callback_t ConfCallback` – callback routine which will be called upon the initiated connection procedure is performed
Returns	The index of the peer device on the connection table.

Example	<code>
	// Establish one or more connections with any device
	PeerIndex = MiApp_EstablishConnection(14, 8, 0x12345678901234567,0x80,
	callback);
	</code>

Remarks	If more than one connections is established through this function call, the return value points to the index of one of the peer devices.

12.6 MiApp_RemoveConnection

API	`void MiApp_RemoveConnection(uint8_t ConnectionIndex)`
Description	This is the primary user interface function to disconnect connection(s). For a P2P protocol, it removes the connection. For a network protocol, if the device referred by the input parameter is the parent of the device calling this function, the calling device gets out of network along with its children. If the device referred by the input parameter is children of the device calling this function, the target device gets out of network.
Pre-Condition	Transceiver is initialized. Node establishes one or more connections.
Parameters	`uint8_t ConnectionIndex` – index of the connection in the connection table to be removed.
Returns	None
Example	<code> MiApp_RemoveConnection(0x00); </code>
Remarks	None

12.7 MiApp_ConnectionMode

API	`void MiApp_ConnectionMode(uint8_t Mode)`
Description	This is the primary user interface function for the application layer to configure the way that the host device accepts the connection request.
Pre-Condition	Protocol initialization is done.

Parameters	`uint8_t Mode` - mode to accept the connection request. The privilege for those modes decreases gradually as defined. The higher privilege mode has all the rights of the lower privilege modes. The possible modes are as follows: • ENABLE_ALL_CONN – enables response to all connection request • ENABLE_PREV_CONN – enables response to connection request from device already in the connection table • ENABLE_ACTIVE_SCAN_RSP – enables response to active scan only • DISABLE_ALL_CONN – disables response to the connection request, including an active scan request
Returns	None

Example	`<code>`
	`// Enable all connection request`
	`MiApp_ConnectionMode(ENABLE_ALL_CONN);`
	`</code>`

Remarks	None

12.8 MiApp_SendData

API	`bool MiApp_SendData(uint8_t addr_len, uint8_t *addr,`
	`uint8_t msglen, uint8_t *msgpointer,uint8_t msghandle,`
	`bool ackReq, DataConf_callback_t ConfCallback)`

Description	This is one of the primary user interface functions for the application layer to unicast a message. The destination device is specified by the input parameter `DestinationAddress`. The application payload is filled using `msgpointer`.
Pre-Condition	Protocol initialization is done.
Parameters	• `uint8_t addr_len` – destination address length • `uint8_t *addr` – destination address • `uint8_t msglen` – length of the message • `uint8_t *msgpointer` – message/frame pointer • `uint8_t msghandle` – message handle • `bool ackReq` – set to receive network level acknowledgment **Note:** Discarded for broadcast data. • `DataConf_callback_t ConfCallback` – callback routine which is called when the initiated data procedure is performed.
Returns	A boolean to indicate if the unicast procedure is successful.

Example	`<code>`
	`// Secure and then broadcast the message stored in msgpointer to the permanent address`
	`// specified in the input parameter.`
	`MiApp_SendData(SHORT_ADDR_LEN, 0x0004, 5, “hello”,1, callback);`
	`</code>`

Remarks	None

12.9 MiApp_SubscribeDataIndicationCallback

API	`bool MiApp_SubscribeDataIndicationCallback(PacketIndCallback_t callback)`
Description	This is the primary user interface functions for the application layer to call the Microchip proprietary protocol stack to register the message indication callback to the application. The function calls the protocol stack state machine to keep the stack running.
Pre-Condition	Protocol initialization is done.
Parameters	None
Returns	A boolean to indicate if the subscription operation is successful or not.

Example	`<code>`
	`if( true == MiApp_SubscribeDataIndicationCallback(ind) )`
	`{`
	`}`
	`</code\>`

Remarks	None

12.10 MiApp_NoiseDetection

API	`uint8_t MiApp_NoiseDetection( uint32_t ChannelMap, uint8_t ScanDuration, uint8_t DetectionMode, uint8_t NoiseLevel)`
Description	This is the primary user interface function for the application layer to perform noise detection on multiple channels.
Pre-Condition	Protocol initialization is done.
Parameters	• `uint32_t ChannelMap` – bit map of channels to perform a noise scan. The 32-bit double word parameter uses one bit to represent corresponding channels from 0 to 31. For example, 0x00000003 represents to scan channel 0 and channel 1. • `uint8_t ScanDuration` – maximum time to perform a scan on a single channel. The valid value is from 5 to 14. The real time to perform a scan can be calculated in the following formula from IEEE 802.15.4 specification: 960 x (2^ScanDuration + 1) x 10^(-6) second • `uint8_t DetectionMode` – the noise detection mode to perform the scan. The two possible scan modes are as following. – `NOISE_DETECT_ENERGY` – Energy detection scan mode – `NOISE_DETECT_CS` – Carrier sense detection scan mode • `uint8_t NoiseLevel` - noise level at the channel with least noise level
Returns	The channel that has the lowest noise level.

Example	<code> uint8_t NoiseLevel; OptimalChannel = MiApp_NoiseDetection(0xFFFFFFFF, 10, NOISE_DETECT_ENERGY, &NoiseLevel); </code>
Remarks	None

12.11 MiApp_TransceiverPowerState

API	`uint8_t MiApp_TransceiverPowerState(uint8_t Mode)`
Description	This is the primary user interface function for the application layer to set the RF transceiver into sleep or wake up. This function is only available to those wireless nodes that have to disable the transceiver to save battery power.
Pre-Condition	Protocol initialization is done.
Parameters	`uint8_t Mode` – mode of the power state for the RF transceiver to be set. The possible power states are following. • `POWER_STATE_SLEEP` – deep sleep mode for the RF transceiver • `POWER_STATE_WAKEUP` – Wake-Up state, or operating state for the RF transceiver • `POWER_STATE_WAKEUP_DR` – Set the device into the Wake-Up mode and transmit the data request to the device's associated device
Returns	The status of the operation. The following are the possible status. • SUCCESS – operation is successful. • ERR_TRX_FAIL – Transceiver fails to go to the Sleep or Wake-Up mode. • ERR_TX_FAIL – transmission of Data Request command failed. Only available if the input mode is POWER_STATE_WAKEUP_DR. • ERR_RX_FAIL – failed to receive any response to Data Request command. Only available if the input mode is POWER_STATE_WAKEUP_DR. • ERR_INVLAID_INPUT – invalid input mode.

Example	<code>
	// put RF transceiver into sleep
	MiApp_TransceiverPowerState(POWER_STATE_SLEEP;
	// Put the MCU into sleep
	Sleep();
	// wakes up the MCU by WDT, external interrupt or any other means
	// make sure that RF transceiver to wake up and send out Data Request
	MiApp_TransceiverPowerState(POWER_STATE_WAKEUP_DR);
	</code>

Remarks	None

12.12 MiApp_Get

API	`bool MiApp_Get(set_params id, uint8_t *value )`

Description	This is the primary user interface function to get the different values in the MiWi™ stack
Pre-Condition	Protocol initialization is done
Parameters	`get_params id` – identifier of the value to be set
Returns	A boolean to indicate if the get operation is performed successfully

Example	`<code>`
	`value = MiApp_get(CHANNEL)`
	`</code>`

Remarks	None

12.13 MiApp_RoleUpgradeNotification_Subscribe

API	`bool MiApp_RoleUpgradeNotification_Subscribe (roleUpgrade_callback_t callback)` This is applicable only for coordinator.
Description	This API subscribes to notify the role upgrade. Upon successful role upgrade, callback is called with new short address.
Pre-Condition	Protocol initialization is done.
Parameters	`roleUpgrade_callback_t callback` – callback routine which is called upon the role upgrade completion
Returns	A boolean to indicate if the subscription is success or not

12.14 MiApp_Commissioning_AddNewDevice

API	`bool MiApp_Commissioning_AddNewDevice(uint64_t joinerAddress, bool triggerBloomUpdate)`
Description	This is used to add a device to bloom filter on the PAN coordinator. This function is applicable only for the PAN coordinator.
Pre-Condition	Protocol initialization is done.
Parameters	• `uint8_t joinerAddress` – the IEEE address to be added • `bool triggerBloomUpdate` – if set to true then bloom update is sent
Returns	True if successfully added, false otherwise.

12.15 MiApp_SubscribeReConnectionCallback

API	`bool MiApp_SubscribeReConnectionCallback(ReconnectionCallback_t callback)`
Description	This API subscribes to notify the reconnection after power recycle when the device was in network before power recycle. Upon reconnection on a device, this callback is called.

Pre-Condition	Protocol initialization is done.
Parameters	`ReconnectionCallback_t callback`- callback routine which is called upon reconnection
Returns	A boolean to indicate if the subscription is success or not

12.16 MiApp_ResetToFactoryNew

API	`bool MiApp_ResetToFactoryNew(void)`
Description	This API erases all the persistent items in the NVM and resets the system
Pre-Condition	None
Parameters	None
Returns	A boolean to indicate if the operation is success or not

12.17 MiApp_ReadyToSleep

API	`bool MiApp_ReadyToSleep(uint32_t* sleepTime)`
Description	This API helps to know if the stack is ready to sleep and how much time stack allows to sleep if it is ready
Pre-Condition	None
Parameters	`uint32_t* sleep Time` – pointer to sleep time which gets filled with the sleep time if the stack is ready to sleep
Returns	A boolean to indicate if the stack is ready to sleep or not

12.18 MiApp_ManuSpecSendData

API	`bool MiApp_ManuSpecSendData(uint8_t addr_len, uint8_t *addr, uint8_t msglen, uint8_t *msgpointer,uint8_t msghandle, bool ackReq, DataConf_callback_t ConfCallback)`
Description	This is an interface function for the manufacturer-specific data. The destination device is specified by the input parameter `DestinationAddress`. The OTAU module uses this API for upgrade support.
Pre-Condition	Protocol initialization is done.

Parameters	• `uint8_t addr_len` – destination address length • `uint8_t *addr` – destination address • `uint8_t msglen` – length of the message • `uint8_t *msgpointer` – message/frame pointer • `uint8_t msghandle` – message handle • `bool ackReq` – set to receive network level acknowledgment **Note:** Discarded for the broadcast data. • `DataConf_callback_t ConfCallback` - callback routine which is called when the initiated data procedure is performed.
Returns	A boolean indicates if the unicast procedure is successful.

Example	`<code>`
	`// Secure and then broadcast the message stored in msgpointer to the permanent address`
	`// specified in the input parameter.`
	`MiApp_ManuSpecSendData(SHORT_ADDR_LEN, 0x0004, 5, “hello”,1, callback);`
	`</code>`

Remarks	None

12.19 MiApp_SubscribeManuSpecDataIndicationCallback

API	`bool MiApp_SubscribeManuSpecDataIndicationCallback (PacketIndCallback_t callback)`
Description	This is the primary user interface functions for the OTAU module to register for manufacturer-specific indication callback.
Pre-Condition	Protocol initialization is done.
Parameters	None
Returns	A boolean indicates if the subscription operation is successful or not.

Example	`<code>`
	`if( true == MiApp_SubscribeManuSpecDataIndicationCallback (ind) )`
	`{`
	`}`
	`</code\>`

Remarks	None

12.20 MiApp_IsConnected

API	`bool MiApp_IsConnected(void)`
Description	This is used to check the connection of MiWi™ Mesh to a network.
Pre-Condition	None
Parameters	None

Returns	A boolean true indicates the node is connected to a network.

12.21 MiApp_MeshGetNextHopAddr

API	`uint16_t MiApp_MeshGetNextHopAddr(uint16_t destAddress)`
Description	This is used to get the address of next hop to reach the `destAddress`.
Pre-Condition	None
Parameters	`uint16_t destAddress` – destination address of the device to which the next hop is required.
Returns	Address of the next hop to reach the `destAddress`.

13. Limitations

The following are the known limitations:

1. Random behavior in some SAMR30 devices that the Back mode sleep fails to wake up when run continuously for 1 or 2 days.
2. It is possible to miss confirmation callback for data in P2P/Star if initiated too fast. It is recommended to have proper debounce in customer applications.
3. OTAU - Device unable (randomly) to wake up from sleep when CPU works at 48 MHz which is derived using DFLL with External 32 KHz crystal/clock source.

14. Document Revision History

Revision	Date	Section	Description
A	02/2019	Document	Initial Revision

The Microchip Web Site

Microchip provides online support via our web site at http://www.microchip.com/. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Customer Change Notification Service

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at http://www.microchip.com/. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.

- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip’s code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Trademarks

ISBN: 978-1-5224-4185-4

Quality Management System Certified by DNV

ISO/TS 16949

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4450-2828 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-67-3636 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820