MiWi^{TM} v6.0 Migration Guide

Contents

1 Introduction

This guide provides all the information need for a customer to migrate the MiWi^{TM} applications implemented on MiWi^{TM} Protocol available in Microchip Libraries for Applications (MLA) to SAM platforms (SAMR21 and SAMR30). The MiWi^{TM} Protocol is been upgraded from v5.30 to MiWi^{TM} Protocol v6.0 with the following feature additions detailed below…

1. MiWi^{TM} Protocol is been ported to Advanced Software Framework (ASF) to support easy integration of other components, services and drivers in application.
2. MiWi^{TM} P2P/Star is ported SAMR21 and SAMR30 platforms.
3. MiWi^{TM} Mesh is redesigned with new features to support large nodes.
4. MiApp API's are redesigned to support simple, easy to use and reliable data transfer.

2 Background

MiWi^{TM} is Microchip's proprietary Wireless Networking stack is designed to support low rate personal area networks (LRPAN's).
MiWi^{TM} supports three network topologies:

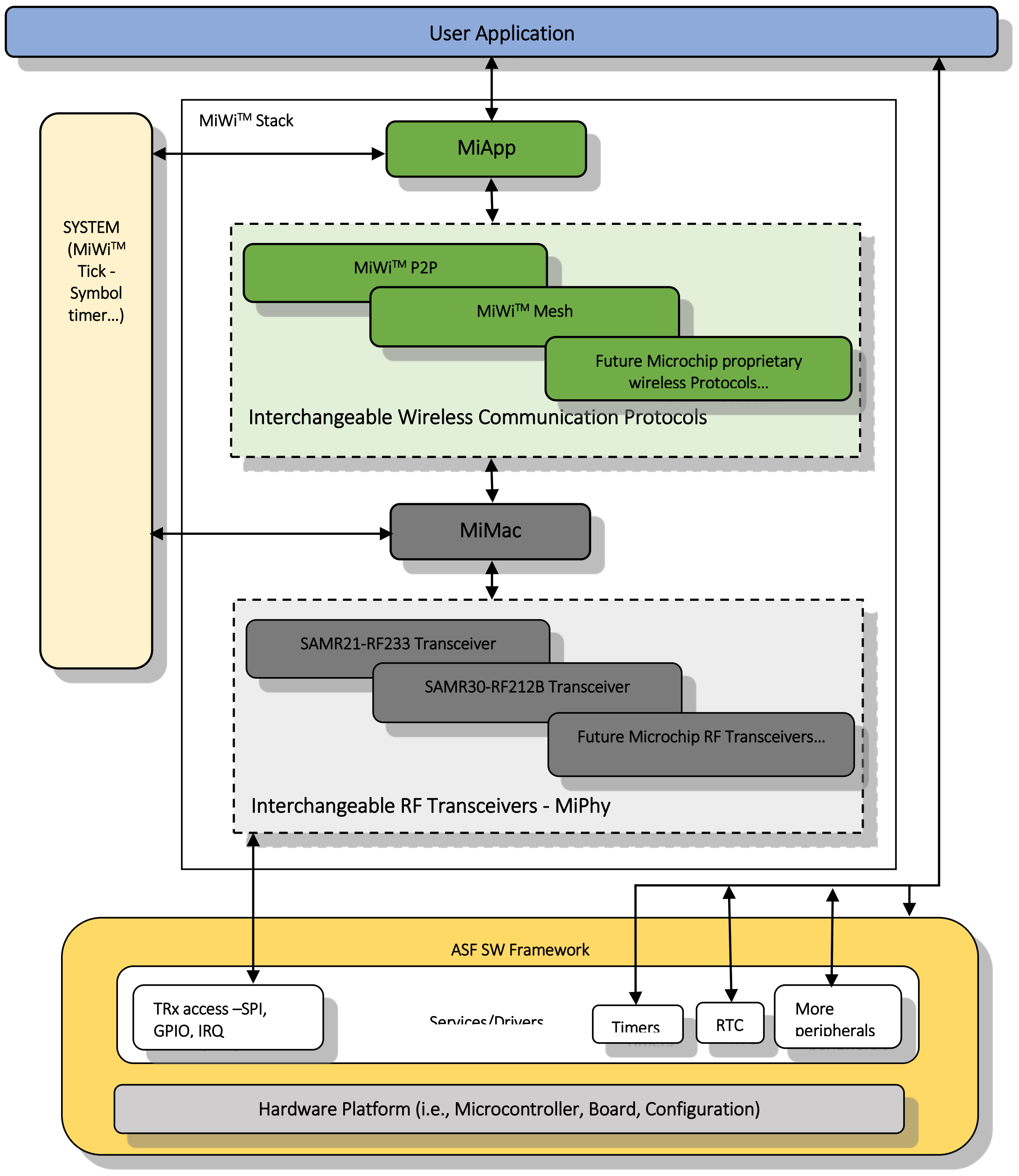
- Peer to Peer (P2P)
- STAR
- MESH

Older versions of MiWi^{TM} Mesh networking stack (i.e. up until version 2.10), released in MiWi^{TM} Protocol v5.30 of MLA version v2017-03-06, supports a library based Mesh networking stack. However this is been redesigned with following important changes as listed below:

1. Optimization of current APIs to improve simplicity.
2. Redesign of the MiWi^{TM} Mesh with additional features for next generation platforms.
3. A new Commissioning procedure added to improve the secured inclusion of devices to network.
4. Dynamic switching between device types in MiWi^{TM} Mesh.
5. Network Secure feature for all network messages.

3 Architecture

Architecture of the MiWi™ Protocol on ASF, which supports additional features for the application developer to pull in required components, services and drivers from ASF Wizard. More details on - http://www.microchip.com/webdoc/asf/asf.ModuleExplorerView.html


User Application
MiWi™ Stack
MiApp
SYSTEM (MiWi™ Tick - Symbol timer...)
MiWi™ P2P
MiWi™ Mesh
Future Microchip proprietary wireless Protocols...
Interchangeable Wireless Communication Protocols
MiMac
SAMR21-RF233 Transceiver
SAMR30-RF212B Transceiver
Future Microchip RF Transceivers...
Interchangeable RF Transceivers - MiPhy
ASF SW Framework
TRx access –SPI, GPIO, IRQ
Services/Drivers
Timers
RTC
More peripherals
Hardware Platform (i.e., Microcontroller, Board, Configuration)

4 MiWi^{TM} Mesh – Device Types

There are 3 device types supported in MiWi^{TM} mesh protocol, they are listed below as follows:

1. PAN Coordinator
    a. Start the network
    b. Assigns and maintains the Coordinators and its end-devices addresses.
    c. Behaves as Coordinator for routing frames.
    d. Controls which devices can be included into the network through commissioning.
2. Coordinator
    a. Joins a network as End-device
    b. Requests PAN Coordinator for role upgrade to become Coordinator.
    c. Supports routing of frames within the network.
    d. Stores the commissioning information from PAN Coordinator and allows only the Commissioned devices to participate in the network.
    e. Maintains its end-devices and their addresses.
    f. Maintains data for sleeping end-devices.
3. End-device
    a. Joins to network though available Coordinators.
    b. Supports Rx-On end-device and Sleeping End-device for battery operated devices.
    c. Supports dynamic switching between Rx-On to Sleeping end-deice and vice versa.

5 MiWi^{TM} Mesh Frame Format

The MiWi^{TM} Mesh network header and application payload are encapsulated inside the standard IEEE 805.15.4 data frame payload, but the stack itself does not adhere to the standard, so it will not receive and process IEEE 805.15.4 command frames. Figure 1 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).

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

Figure 1: General Frame Format

5.1 MAC Header – Frame Control

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. Addr. Mode	Frame Version	Src. Addr. Mode

Figure 2: MAC Header- Frame Control Field

The following settings are used for a MAC Frame Control Field as described in Figure 2:

- Frame Type: Data

- Security Enabled: False
- Frame Pending: True if pending data available for sleeping end device or False otherwise.
- 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

5.2 NWK Header – Hops

The number of hops that the packet is allowed to be retransmitted (00h means don't retransmit this packet – 1 byte)

5.3 NWK Header – Frame Control

The Frame Control field is a bitmap as defined below, which defines the behavior of a packet.

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

Figure 3: NWK Header - Frame Control Field

Bit 7-6 **Reserved:** Maintain as '0' in this implementation

Bit 5 **Address same as MAC:**

This bit will be set when the MAC address fields and NWK address fields are the same. This would be useful when the sleeping end device polls the parent for data, with relatively less bytes over-the-air for single hop from network layer.

Bit 4 **Ack Request:** Acknowledge Request bit

When set, the source device requests an upper layer Acknowledgement of receipt from the destination device.

Bit 3 **Intra Cluster:** Intra Cluster bit

Reserved in this implementation, maintain as '1'.

Bit 2 **Security Enabled:** Security enabled bit

When set, data packet is encrypted at the application level.

Bit 0-1 **Frame Type:** Frame type bits

Frame Type value b_1 b_0	Description
00	Data
01	Command
10 – 11	Reserved

5.4 NWK Header – 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 should not be change for routed frames.

5.5 NWK Header – Source Address field

The Source Address field is 2 bytes or 8 bytes in length and specifies the network address of the node originating the frame.8 bytes are used when security is enabled where source IEEE address is needed for security processing.

5.6 NWK Header – 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 below table for other frames except unicast to a node.

Destination Address value	Description
0xFFFF	Broadcast to every device
0xFFFE	Multicast to all FFD's
0xFFFD	Multicast to all Coordinators

5.7 NWK Header – Auxiliary security header

Bytes: 1	4
Security Level	Frame Counter

Figure 4: NWK Header - Auxiliary Security Header Field

The Auxiliary Security Header field specifies information required for security processing, including how the frame is actually protected (security level) and frame counter. This field shall be present only if the Security Enabled subfield is set to one. The supported security level are 0(No Security), 1(Authentication-4bytes MIC), 4(Encryption only), 5 (Encryption with Authentication-4 bytes MIC).

6 MiWi^{TM} Mesh – Device Addressing Mechanism

MiWi^{TM} Mesh uses 2 byte short address to specify nodes in the network when performing routing across the network. 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 (0x00 to identify Coordinator)

7 MiWi^{TM} Mesh – Networking

MiWi^{TM} Mesh network is enriched with features categorized as follows…

1. Network Commissioning
2. Start and Join Network
3. Routing in Network

7.1 Network Commissioning

Commissioning the network controls the devices which can participate in the network.

1. Application on PAN Coordinator reads the IEEE address (can be improved to read from Barcode) from one or more devices.
2. PAN Coordinator calculates the 64-byte bloom filter value with the read information.
3. The calculated bloom filter value is sent to all the coordinators in the network.
4. Coordinators provide beacon only to the devices which has its IEEE address in the bloom filter.
5. More information on bloom filter is available in - https://en.wikipedia.org/wiki/Bloom_filter

7.2 Start and Join Network

1. Only PAN Coordinator can start a network.
2. Joining device sends beacon request in order 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 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 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 Join Wish Byte has two bits C and ED, remaining bits are reserved.

   If both bits are set in JoinWish, then the particular device wishes to join as an End device if coordinator capacity is currently unavailable in the network.

   If only C bit is set, then the device wishes to join as a Coordinator only.

   If only ED bit is set, then the device wishes to join as an End device only.
5. If the Parent is PAN Coordinator and JoinWish filed is set with C and ED or C alone, then 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 ED alone set, then it will allocate end device address and sends the Mesh connection response.
6. If the Parent is 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 use 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, then based on Role Upgrade Timeout (Configurable) will send a Role Upgrade request packet to the PAN Coordinator in order to upgrade its role from an end device to Coordinator.
9. When the PAN Coordinator receives a role upgrade request, it checks whether coordinator address is available. If yes, 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 Role Upgrade response with failure status.

7.3 Routing in Network

1. During the joining procedure and role upgrade, the route table will be updated in all the coordinators.
2. The route table in Coordinators will be used to route the packet to destination device.
3. When the device doesn’t 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 will be generated from any node which has the next hop information in its routing table.
5. The source device (which initiated route request) will select the route reply for the destination based on the fewer hops and best LQI.
6. However in order to establish and synchronize the network periodically, the Route Table Update will be broadcast to a single hop based on pre-configured intervals.
7. This would ensure that coordinators in the network share the neighbor’s information with its neighbors.

8 MACROs for MiWi^{TM} Mesh

Sl. no.	MACRO	Default Value	Range	Description
1.	CHANNEL_MAP	(1<<25)	--	Channel map is a bit map used to select appropriate channels for starting or establishing connection in the network. Bitmap based on the physical layer. Set/Clear of any bits in the below range is valid… For 2.4GHz – 0x07FFF800 For SubGHz – 0x000007FF
2.	KEEP_ALIVE_COORDINATOR_SEND_INTERVAL	120	0-255	Time interval in which a coordinator device would send keep alive message to PAN Coordinator
3.	KEEP_ALIVE_COORDINATOR_TIMEOUT_IN_SEC	KEEP_ALIVE_COORDINATOR_SEND_INTERVAL * 10	0 to (2^32 – 1)	Timeout in seconds of a coordinator at which the PAN Coordinator would remove the coordinator from the coordinator table.
4.	KEEP_ALIVE_RXONENDDEVICE_SEND_INTERVAL	120	0-255	Time interval in which a Rx enabled end device would send keep alive message to its parent Coordinator
5.	KEEP_ALIVE_RXONENDDEVICE_TIMEOUT_IN_SEC	KEEP_ALIVE_RXONENDDEVICE_SEND_INTERVAL * 10	0 to (2^32 – 1)	Timeout in seconds of Rx On Enddevice at which the Coordinators would remove the Rx On Enddevice from the Enddevice table.
6.	MAX_NUMBER_OF_DEVICES_IN_NETWORK*	32	0 - 200	Table size of the maximum number of devices in the network to generate Bloom filter. This is applicable only for PAN Coordinator.
7.	DEVICE_TIMEOUT1	15000		Timeout of end device reported to coordinator.
8.	ROLE_UPGRADE_INTERVAL_IN_SEC	20	0 - 255	Time interval in which a coordinator capable device would request for role upgrade to PAN Coordinator
9.	CONNECTION_RESPONSE_WAIT_IN_SEC	5	0 - 255	Time interval to wait for connection response from the Coordinator.
10.	NUM_OF_COORDINATORS*	10	0 - 200	Maximum number of Coordinators anticipated for which the route table need to be maintained
11.	NUM_OF_NONSLEEPING_ENDDEVICES*	5	0 - 127	Maximum number of Rx-On end-device anticipated for coordinator.
12.	NUM_OF_SLEEPING_ENDDEVICES*	5	0 - 127	Maximum number of Sleeping end-device anticipated for coordinator.
13.	ROUTE_UPDATE_INTERVAL	20	0 - 255	Time Interval at which the coordinator sends out Route update

14.	ROUTE_REQ_WAIT_INTERVAL	5	0 - 255	Time to wait for route replies from other coordinators
15.	INDIRECT_DATA_WAIT_INTERVAL	10	0 - 255	Time interval to maintain the Indirect data for sleeping devices in coordinator.
16.	DATA_REQUEST_INTERVAL	5	0 - 255	Time interval to send data request from sleeping end device to coordinator.
17.	FRAME_ACK_WAIT_INTERVAL	5	0 - 255	Time interval to wait for network level acknowledgement.
18.	FRAME_RETRY	3	0 - 255	Number of retries to be done for application data frames
19.	REBROADCAST_TABLE_SIZE*	10	0 - 255	Size for rebroadcast table to maintain rebroadcasted frames in coordinator.
20.	REBROADCAST_TIMEOUT	5	0 - 255	Time out to remove entry from rebroadcast table.
21.	MAX_BEACON_RESULTS*	5	0 - 255	Maximum number of beacon results to be stored during search connection procedure
22.	MESH_SECURITY_LEVEL	5	0 - 7	Security levels for CCM* as defined in IEEE 802.15.4
23.	PUBLIC_KEY_DEFAULT	{0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F}	--	Public key of the network used to retrieve network key.
24.	NETWORK_KEY_DEFAULT	{0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD,	--	Network key used to transact after successful join to the network.

		0xEE, 0xFF}		

* - Increasing the value will impact the memory in the system

9 MiApp API changes

The following table lists the API changes with respect to old APIs …

Sl. No.	Old APIs	New APIs	Stack Supported
1.	bool MiApp_ProtocolInit(bool bNetworkFreezer);	bool MiApp_ProtocolInit (defaultParametersRomOrRam_t *defaultRomOrRamParams, defaultParametersRamOnly_t *defaultRamOnlyParams)	P2P/Star/Mesh
2.	bool MiApp_SetChannel(uint8_t Channel)	bool MiApp_Set(enum id, uint8_t value)	P2P/Star/Mesh
3.	bool MiApp_StartConnection(uint8_t Mode, uint8_t ScanDuration, uint32_t ChannelMap)	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)	uint8_t MiApp_SearchConnection(uint8_t ScanDuration, uint32_t ChannelMap, FUNC ConfCallback)	P2P/Star/Mesh
5.	uint8_t MiApp_EstablishConnection(uint8_t ActiveScanIndex, uint8_t Mode)	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)	void MiApp_RemoveConnection(uint8_t ConnectionIndex)	P2P/Star/Mesh
7.	void MiApp_ConnectionMode(uint8_t Mode)	void MiApp_ConnectionMode(uint8_t Mode)	P2P/Star/Mesh
8.	#define MiApp_FlushTx() {TxData = PAYLOAD_START;}	MiApp_SendData(uint8_t addr_len, uint8_t addr, uint8_t len, uint8_t pointer, FUNC ConfCallback)	P2P/Star/Mesh
9.	#define MiApp_WriteData(a) TxBuffer[TxData++] = a		P2P/Star/Mesh
10.	bool MiApp_BroadcastPacket(bool SecEn)		P2P/Star/Mesh
11.	bool MiApp_UnicastConnection(uint8_t ConnectionIndex, bool SecEn)		P2P/Star/Mesh
12.	bool MiApp_UnicastAddress(uint8_t DestinationAddress, bool PermanentAddr, bool SecEn)		P2P/Star/Mesh
13.	bool MiApp_MessageAvailable(void)	MiApp_SubscribeDataIndicationCallback (FUNC callback)	P2P/Star/Mesh
14.	void MiApp_DiscardMessage(void)	Not Required	P2P/Star/Mesh
15.	uint8_t MiApp_NoiseDetection(uint32_t ChannelMap, uint8_t ScanDuration,	uint8_t MiApp_NoiseDetection(uint32_t ChannelMap, uint8_t ScanDuration,	P2P/Star/Mesh

	uint8_t DetectionMode, OUTPUT uint8_t NoiseLevel)	uint8_t DetectionMode, OUTPUT uint8_t NoiseLevel)	
16.	uint8_t MiApp_TransceiverPowerState(uint8_t Mode)	uint8_t MiApp_TransceiverPowerState(uint8_t Mode)	P2P/Star/Mesh
17.	bool MiApp_InitChannelHopping(uint32_t ChannelMap)	bool MiApp_InitChannelHopping(uint32_t ChannelMap)	P2P/Star/Mesh
18.	bool MiApp_ResyncConnection(uint8_t ConnectionIndex, uint32_t ChannelMap)	bool MiApp_ResyncConnection(uint8_t ConnectionIndex, uint32_t ChannelMap)	P2P/Star/Mesh
19.	uint8_t Total_Connections(void)	uint8_t Total_Connections(void)	P2P/Star/Mesh
20.	void MiApp_BroadcastConnectionTable()	void MiApp_BroadcastConnectionTable()	Star
21.	bool SW_Ack_SrED(uint8_t)	bool SW_Ack_SrED(uint8_t)	Star
22.	void send_link_status(void)	void send_link_status(void)	Star
23.	void Find_InActiveDevices(void)	void Find_InActiveDevices(void)	Star
24.	void MiApp_leave_network(void)	void MiApp_leave_network(void)	Star
25.	bool MiApp_UnicastStar (bool SecEn)	bool MiApp_UnicastStar (bool SecEn)	Star
26.	void MiApp_SetAddressPan(uint8_t address, uint16_t panid)	bool MiApp_Set(enum id, uint8_t value)	Mesh
27.	bool MiApp_IsMemberOfNetwork(void)	bool MiApp_IsMemberOfNetwork(void)	Mesh
28.	addr_t MiApp_GetParentAddress(void)	bool MiApp_Get(enum id, uint8_t value)	Mesh
29.	void MiApp_RequestData(void)	Not required	Mesh
30.	bool MiApp_SendDataRequest(void)	Not required	Mesh
31.	uint16_t MiApp_InitSleepRFDBuffers(uint8_t buffer, uint16_t bufferSize, uint16_t rfdMaxDataSize)	Not required	Mesh
32.	void MiApp_SetNetworkSecure(bool isSecure)	bool MiApp_Set(enum id, uint8_t value)	Mesh
33.	Not Available	bool MiApp_SubscribeDataIndicationCallback (PacketIndCallback_t callback)	Mesh
34.	Not Available	bool MiApp_RoleUpgradeNotification_Subscribe(roleUpgrade_callback_t callback)	Mesh
35.	Not Available	bool MiApp_Commissioning_AddNewDevice(uint64_t joinerAddress, bool triggerBloomUpdate)	Mesh
36.	Not Available	bool MiApp_SubscribeLinkFailureCallback(LinkFailureCallback_t callback)	Mesh

37.	Not Available	uint16_t MiApp_MeshGetNextHopAddr(uint16_t destAddress)	Mesh

10 MiApp API Description

10.1 MiApp_ProtocolInit

API	`bool 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 has been done.
Parameters	defaultParametersRomOrRam_t defaultRomOrRamParams - Default parameters for MiWi^{TM} Mesh. defaultParametersRamOnly_t defaultRamOnlyParams - Default parameters for MiWi^{TM} Mesh. Ignored in case of P2P / Star
Returns	True if successfully

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

Remarks	None

10.2 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^{TM} stack.
Pre-Condition	Protocol initialization must be done.
Parameters	set_params id - The identifier of the value to be set value - value to be set
Returns	A boolean to indicate if set operation has been performed successfully

Example	<code>
	if(true == MiApp_Set(CHANNEL, 15))
	{

	// channel changes successfully
	}
	</code>

Remarks	None

10.3 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 who 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… START_CONN_DIRECT Start PAN directly without noise scan START_CONN_ENERGY_SCN Perform an energy scan first, then start the PAN on the channel with least noise. START_CONN_CS_SCN Perform a carrier-sense scan first, then start the PAN on the channel with least noise. uint8_t ScanDuration - The 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 (2^ScanDuration + 1) 10^(-6) second ScanDuration is discarded if the connection mode is START_CONN_DIRECT. uint32_t ChannelMap - The bit map of channels to perform noise scan. The 32-bit double word parameter use 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 - The callback routine which will be called upon the initiated connection procedure is performed.
Returns	A boolean to indicate if PAN has been 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

10.4 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 will be 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 has been done.
Parameters	uint8_t ScanDuration - The 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 (2^ScanDuration + 1) 10^(-6) second. uint32_t ChannelMap - The bit map of channels to perform noise scan. The 32-bit double word parameter use one bit to represent corresponding channels from 0 to 31. For instance, 0x00000003 represent to scan channel 0 and channel 1. SearchConnectionConf_callback_t ConfCallback - The callback routine which will be called upon 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

10.5 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. There are multiple ways to establish connection(s), all depends on the input parameters.
Pre-Condition	Protocol initialization has been done. If only to establish connection with a predefined device, an active scan must be performed before and valid active scan result has been saved.
Parameters	uint8_t channel - The 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 - The 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 have been established through this function call, the return value points to the index of one of the peer devices.

10.6 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 simply remove 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 will get 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 will get out of network.
Pre-Condition	Transceiver has been initialized. Node has establish one or more connections
Parameters	uint8_t ConnectionIndex - The index of the connection in the connection table to be removed
Returns	None
Example	<code> MiApp_RemoveConnection(0x00); </code>
Remarks	None

10.7 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 accept connection request.
Pre-Condition	Protocol initialization has been done.
Parameters	uint8_t Mode - The mode to accept 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… ENABLE_ALL_CONN Enable response to all connection request ENABLE_PREV_CONN Enable response to connection request from device already in the connection table. ENABLE_ACTIVE_SCAN_RSP Enable response to active scan only DISABLE_ALL_CONN Disable response to connection request, including an active scan request.

Returns	None

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

Remarks	None

10.8 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 has been done.
Parameters	uint8_t addr_len - destionation address length uint8_t *addr - destionation 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 ack (Note- Discarded for broadcast data) DataConf_callback_t ConfCallback - The callback routine which will be called upon 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

10.9 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 for message indication callback to the application. The function will call the protocol stack state machine to keep the stack running.

Pre-Condition	Protocol initialization has been done.
Parameters	
Returns	A boolean to indicates if the subscription operation is successful or not.

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

Remarks	None

10.10 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 functions for the application layer to perform noise detection on multiple channels.
Pre-Condition	Protocol initialization has been done.
Parameters	uint32_t ChannelMap - The bit map of channels to perform noise scan. The 32-bit double word parameter use one bit to represent corresponding channels from 0 to 31. For instance, 0x00000003 represent to scan channel 0 and channel 1. uint8_t ScanDuration - The 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 (2^ScanDuration + 1) 10^(-6) second uint8_t DetectionMode - The noise detection mode to perform the scan. The two possible scan modes are NOISE_DETECT_ENERGY Energy detection scan mode NOISE_DETECT_CS Carrier sense detection scan mode uint8_t NoiseLevel - The 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

10.11 MiApp_TransceiverPowerState

API	`uint8_t MiApp_TransceiverPowerState(uint8_t Mode)`
Description	This is the primary user interface functions for the application layer to put RF transceiver into sleep or wake it up. This function is only available to those wireless nodes that may have to disable the transceiver to save battery power.

Pre-Condition	Protocol initialization has been done.
Parameters	uint8_t Mode - The mode of power state for the RF transceiver to be set. The possible power states are following POWER_STATE_SLEEP The deep sleep mode for RF transceiver POWER_STATE_WAKEUP Wake up state, or operating state for RF transceiver POWER_STATE_WAKEUP_DR Put device into wakeup mode and then transmit a data request to the device's associated device
Returns	The status of the operation. The following are the possible status SUCCESS Operation successful ERR_TRX_FAIL Transceiver fails to go to sleep or wake up 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 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	

10.12 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^{TM} stack
Pre-Condition	Protocol initialization has been done.
Parameters	get_params id - The identifier of the value to be set
Returns	A boolean to indicate if get operation has been performed successfully
Example	<code> value = MiApp_get(CHANNEL) </code>
Remarks	None

10.13 MiApp_RoleUpgradeNotification_Subscribe

API	`bool MiApp_RoleUpgradeNotification_Subscribe (roleUpgrade_callback_t callback)`
Description	This is used to subscribe to notify the role upgrade. Upon successful role upgrade, callback will be called with new short address
Pre-Condition	Protocol initialization has been done.
Parameters	roleUpgrade_callback_t callback - The callback routine which will be called upon the role upgrade done
Returns	A boolean to indicates if the subscribtion is success or not

10.14 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 PAN Coordinator. This function is applicable only for PAN Coordinator.
Pre-Condition	Protocol initialization has been done.
Parameters	uint8_t joinerAddress - the ieee address to be added. bool triggerBloomUpdate - if set to true then bloom update is sent immediately
Returns	True if successfully added, false otherwise.

10.15 MiApp_SubscribeLinkFailureCallback

API	`bool MiApp_SubscribeLinkFailureCallback(LinkFailureCallback_t callback)`
Description	This is used to subscribe to notify the link failure. Upon link failure in coordinator or end device, this callback will be called
Pre-Condition	Protocol initialization has been done.
Parameters	LinkFailureCallback_t callback - The callback routine which will be called upon link failure
Returns	A boolean to indicates if the subscription is success or not

10.16 MiApp_MeshGetNextHopAddr

API	`uint16_t MiApp_MeshGetNextHopAddr(uint16_t destAddress)`
Description	This is used to get the next hop address for the given destination address from the routing table
Pre-Condition	Protocol initialization has been done.
Parameters	uint16_t destAddress - any valid network address
Returns	Next hop address if available, otherwise 0xFFFF

11 Limitations

The list of known limitations are described below:

1. Functional testing for MiWi^{TM} Mesh was performed on network consisting of 32 hops with 1 PAN Coordinator and 32 Coordinators.
2. The current MiWi^{TM} P2P/Star protocol frame format and examples are compatible with older versions of MiWi^{TM} P2P/Star.
3. The current version 6.0 and future version of the MiWi^{TM} Mesh Protocol is not interoperable or backward compatible to the older versions of MiWi^{TM} Mesh Protocol.