# Microchip Wireless MiWi™ Application Programming Interface – MiApp

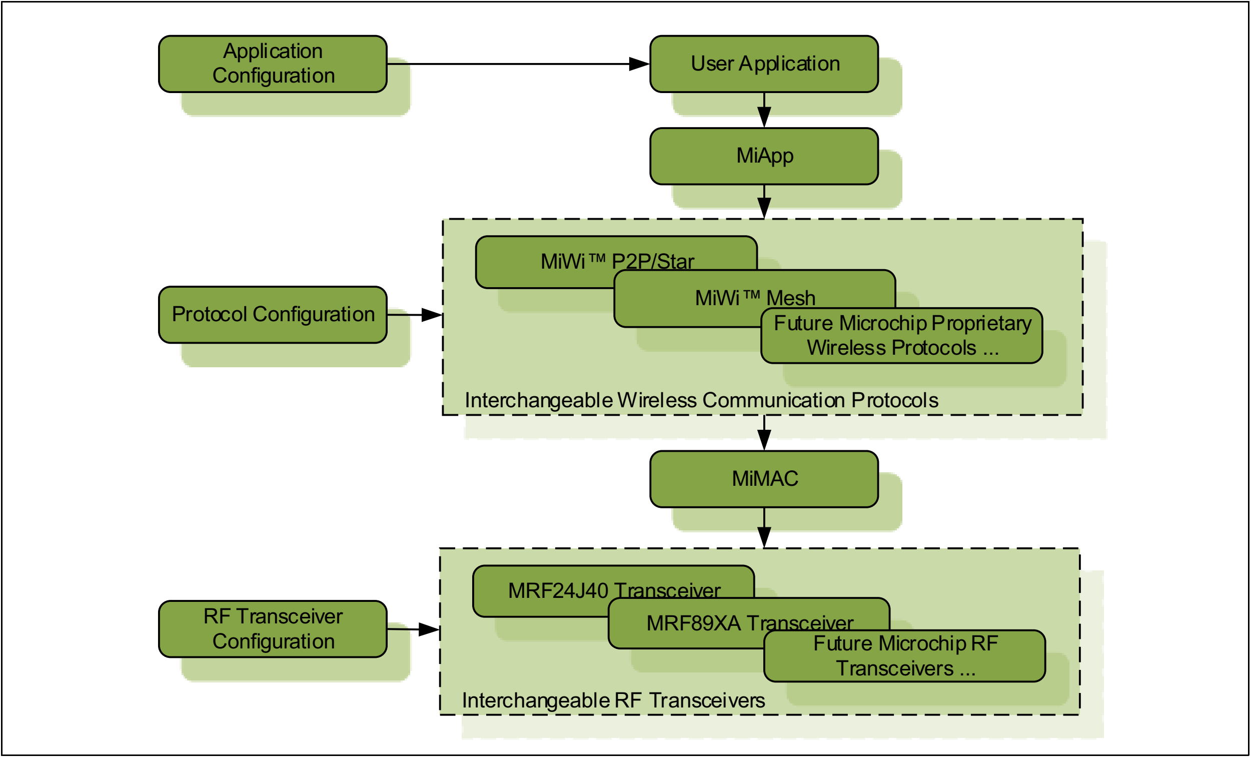
*Authors:* *Yifeng Yang*
*Pradeep Shamanna*
*Derrick Lattibeaudiere*
*Vivek Anchalia*
*Microchip Technology Inc.*

## INTRODUCTION

Developing wireless applications can be challenging. Apart from Radio Frequency (RF) circuit designs, the firmware development process may require the developers to understand the details of RF transceivers, as well as the different wireless communication protocols. Microchip has developed a way to handle the complex RF hardware or communication protocol stack software development, or both, which enables wireless application developers to focus on their own application development. This type of wireless communication is achieved through a concise, yet powerful communication programming interface in the application layer called *MiApp* which is defined in this application note.

The MiApp specification defines the programming interfaces between the application layer and Microchip proprietary wireless communication protocols. The MiApp programming interface is implemented in two ways: as configuration parameters defined in the configuration file, and as a set of function calls to the Microchip proprietary wireless communication protocols. Complying with the MiApp specification defined in this application note, applications can use any Microchip proprietary wireless protocols. With little or no modification in the application layer, software development can be easily changed between a proprietary P2P/Star topology connection protocol to a full Mesh proprietary networking protocol for small or big networks, depending on the application needs.

**FIGURE 1: BLOCK DIAGRAM OF MICROCHIP WIRELESS MiWi™ STACK**


Application Configuration
User Application
MiApp
MiWi™ P2P/Star
MiWi™ Mesh
Future Microchip Proprietary Wireless Protocols ...
Protocol Configuration
Interchangeable Wireless Communication Protocols
MiMAC
MRF24J40 Transceiver
MRF89XA Transceiver
Future Microchip RF Transceivers ...
RF Transceiver Configuration
Interchangeable RF Transceivers

The MiApp specification benefits wireless application developers in multiple ways:

- Wireless application development focuses on its application. Complex RF or protocol considerations are transparently handled by the MiApp programming interface.
- The MiApp specification allows maximum flexibility to choose a wireless protocol at any stage of application software development with little effort, thus greatly lowering the risk of software development. Application requirement changes in networking capabilities have little or no impact in application development.
- MiApp uses the same control interface for Microchip wireless proprietary protocols. Once you are familiar with MiApp, you can apply that knowledge to the development of another application even if it has a completely different networking capability requirement.
- By communicating to the Microchip proprietary protocols, MiApp indirectly talks to the Microchip RF transceivers through the MiMAC interface. As a result, MiApp indirectly enables the wireless application developers to switch between Microchip RF transceivers through MiMAC. This flexibility, in turn, further reduces the development risk of the wireless application project.

## FEATURES

The MiApp programming interface has the following features:

- Easy to learn and use
- Powerful interface to meet most requirements from wireless applications
- Little or no extra effort to migrate the wireless application between Microchip proprietary wireless protocols
- Minimum footprint impact

## CONSIDERATIONS

The MiApp specification is designed to support Microchip proprietary wireless communication protocols. Once a wireless application is implemented by the MiApp programming interface, the Microchip RF transceivers are also supported through standardization in MiMAC, the module defined in the Media Access Controller (MAC) layer.

MiMAC standardizes the interface between Microchip wireless protocols and Microchip RF transceivers. MiMAC makes Microchip RF transceivers interchangeable with little or no change in the software application code. For more information on MiMAC, refer to the Application Note "*AN1283 Microchip Wireless MiWi™ Media Access Controller - MiMAC"* (DS00001283).

MiMAC regulates the lower interface of the Microchip proprietary wireless protocols, while MiApp regulates the higher interface of the Microchip proprietary wireless protocols. Working together, both MiMAC and MiApp provide wireless application developers the maximum flexibility to choose the RF transceivers and wireless communication protocols at any stage of software development, thus further minimizing the risk of software development.The block diagram in Figure 1 shows the Microchip Wireless MiWi™ stack offerings.

There are three layers of configurations for application, protocol stacks and RF transceivers. Application configuration may change between devices in the same application according to their hardware design, role in the application and network. Wireless application developers tend to do the majority of the configuration in the application layer. Protocol configurations fine-tune the behavior of the protocol stack. The majority of protocol stack configurations define the timing and routing mechanism for the chosen wireless protocol. Transceiver configurations define the frequency band, data rate and other RF related features of the RF transceiver. The default settings for the protocol and RF transceiver configurations may work with the application without any modification. The application configurations, however, usually need to be changed to fit the needs of different wireless applications.

## MiApp OVERVIEW

As discussed earlier, there are two parts defined in the MiApp specification:

- Configuration parameters defined in the configuration file
- Signatures of function calls to the Microchip proprietary wireless protocols

The configuration file contains parameters that must be set before compilation. Generally, there are two types of parameters defined in the configuration file:

- **Hardware Definitions** – these includes MCU hardware resources, peripherals definition and the definition for the RF transceiver control pins. The default hardware definitions are defined in several Microchip standard demo boards that support Microchip RF transceivers. In these cases, the definition of demo boards automatically introduces all hardware definitions.
- **Software Definitions** – these definitions control the code sections to be compiled into the firmware hex file. The software definitions include selections of Microchip proprietary wireless protocol, choice of Microchip RF transceiver, and individual functionalities. Proper definitions in this category ensure the minimum firmware footprint with the intended protocol capabilities.

Application Programming Interfaces (APIs) are the function calls between the Microchip proprietary wireless communication protocols with the wireless application. As a rule, the application interface must be clean, concise, easy to understand, and powerful.

There are five categories of interfaces for the APIs:

- The initialization interface allows wireless application developers to properly initialize the selected Microchip proprietary wireless protocol in the configuration file.
- The handshaking interface allows the wireless nodes to discover and get connected with their peers, or to join the network.
- Interfaces to send messages which enable application developers to transmit information from the current node to an intended audience over-the-air.
- Interfaces to receive messages which enable application developers to receive information over- the-air from other devices.
- Special functionalities which ensure the optimal operating condition for wireless nodes through environment noise control and power saving.

## MiApp CONFIGURATION FILE

The hardware definitions depends on the choices of the Demo/Development boards, the MCUs and the RF transceivers. The hardware definitions can be divided into the following subcategories:

- I/Os on the demo board – these includes the push buttons, LEDs, serial ports, and so on.
- MCU system resources – these includes the timers, interrupts, and so on.
- Interconnections between MCU and RF transceiver

Hardware definitions are mainly associated with hardware selections of the wireless application system design. They depend more on the hardware than the software and vary across different designs. As a result, MiApp does not have a set of standards for those hardware definitions.

Selective compilation configurations select the features among the list of available ones. Using the selective compilation, application developers are able to configure Microchip proprietary wireless protocols to perform the desired functionality with the least possible system resources. Table 1 describes the possible selective compilation configurations and its scope, value, and functionality.

**TABLE 1: APPLICATION LAYOUT SOFTWARE DEFINITIONS IN A CONFIGURATION FILE**

| Application Definitions | Functionality | Comments |
|---|---|---|
| `#define PROTOCOL_P2P`<br>`#define PROTOCOL_STAR`<br>`#define PROTOCOL_MESH` | Selects the Microchip proprietary wireless protocol to be used in the wireless application. | Only one protocol can be defined at any one time. |
| `#define MRF24J40`<br>`#define MRF89XA` | Selects the Microchip RF transceiver to be used in the wireless application. | Only one transceiver can be defined at any one time. |
| `#define TX_BUFFER_SIZE 64` | Defines the maximum size of the application payload to be transmitted, excluding all protocol headers. | There may be RF transceiver hardware restrictions on the size of buffer that can be transmitted. The hardware restriction includes all protocol headers. |
| `#define RX_BUFFER_SIZE 64` | The maximum size of application payload to be received, excluding all protocol headers. | There may be RF transceiver hardware restrictions on the size of buffer that can be received. The hardware restriction includes all protocol headers. |
| `#define CONNECTION_SIZE 10` | The size of connection table. Determines the maximum number of devices that the node can connect to. | Depends upon available MCU RAM. |
| `#define ADDITIONAL_NODE_ID_SIZE 1` | Defines the size of additional information attached to the packets in the handshake process. Primarily used to identify the node in the application layer. | The additional node identifier plays no role in Microchip's proprietary protocols. However, it may play an important role in the application. In a simple case of light and switch, two lights may not be interested in connecting to each other, and the same applies to two switches. Using the additional node identifier enables the application to identify the role of the node in the application so that switches only connect with lights. |
| `#define ENABLE_PA_LNA` | Enables the RF transceiver to use an external power amplifier or a low noise amplifier, or both. | For RF transceivers that can control an external PA or LNA, or both. |
| `#define ENABLE_HAND_SHAKE` | Enables the Microchip proprietary wireless protocol to automatically establish connections with peers. | Handshake process enables two wireless nodes to know each other. In other protocols, this process is also called *Pairing*. Applications without handshake only use broadcast to exchange messages. |
| `#define ENABLE_SLEEP` | Enables the RF transceiver to go to sleep when idle to save power. | Sleep mode depends on the capability of the RF transceiver. |
| `#define ENABLE_ED_SCAN` | Enables the Microchip proprietary wireless protocol and RF transceiver to perform an energy detection scan. | The energy scan depends on the capability of the RF transceiver. |
| `#define ENABLE_ACTIVE_SCAN` | Enables the Microchip proprietary wireless protocol to perform an active scan to discover nodes and networks in the neighborhood. | Active Scan is used to search for existing wireless devices of the same kind in the neighborhood. Active Scan can be used to decide which device to connect to. |
| `#define ENABLE_SECURITY` | Enables the Microchip proprietary wireless protocol to secure packets that are transferred. | The security engine, security mode, and keys are defined in a configuration file for the RF transceiver, as security is defined as part of MiMAC. |

**TABLE 1: APPLICATION LAYOUT SOFTWARE DEFINITIONS IN A CONFIGURATION FILE**

| Application Definitions | Functionality | Comments |
|---|---|---|
| `#define ENABLE_INDIRECT_MESSAGE` | Enables the wireless node to cache messages for sleeping devices and to deliver them once the sleeping device wakes up and asks for the messages. | Only wireless nodes that do not go to sleep can cache message for sleeping nodes. The number of messages which can be cached depends on the available MCU RAM. |
| `#define RFD_WAKEUP_INTERVAL 5` | Defines, in seconds, the RFD devices' wake-up time interval. | Only effective when indirect message is enabled. This definition is used for devices that are always awake to keep track of time-outs for indirect messages. The sleeping time of sleeping devices depends on the WDT setting of the host MCU. |
| `#define ENABLE_BROADCAST` | Enables the wireless node to handle broadcast messages for sleeping devices. | Only wireless nodes that do not go to sleep can cache messages for sleeping nodes. This macro is supported *only* for MiWi P2P and Star networks. Mesh supports caching only through the Personal Area Network (PAN) Coordinator. |
| `#define ENABLE_FREQUENCY_AGILITY` | Enables the Microchip proprietary wireless protocol to perform frequency agility procedures. | Only supported by P2P and Star networks. |
| `#define HARDWARE_SPI` | Enables the MCU to use the hardware SPI to communicate with the transceiver. | Defining of HARDWARE_SPI enables the MCU to use the hardware SPI to communicate with the transceiver. Otherwise, the MCU can use bit-bang to simulate SPI communication with transceiver. |
| `#define TARGET_SMALL` | Minimizes the footprint of Microchip proprietary wireless protocols. | Some features of the Microchip proprietary wireless protocol may not be supported when minimizing the footprint of the protocol. This macro is supported *only* for MiWi P2P and Star networks. |
| `#define ENABLE_NETWORK_FREEZER` | Enables the Microchip proprietary wireless protocol to store critical network parameters and to recover from power loss to the original network setting. | Requires nonvolatile memory of either MCU data EEPROM, external EEPROM or programming space. Network size and chosen wireless protocol decides the total amount of nonvolatile memory required. This macro is supported *only* for MiWi P2P and Star networks. The Mesh does not take care of this network freezer functionality due to the dynamic connections, routing, and self-healing features. |
| `#define ENABLE_APP_LAYER_ACK` | Enables the acknowledgment feature to receive a software-based acknowledgment from the end device receiving the packet. | Only supported by MiWi Star protocol. |

**TABLE 1: APPLICATION LAYOUT SOFTWARE DEFINITIONS IN A CONFIGURATION FILE**

| Application Definitions | Functionality | Comments |
|---|---|---|
| `#define ENABLE_PERIODIC_CONNECTION_TABLE_SHARE` | Used to enable a connection table share feature on the PAN Coordinator. The PAN Coordinator periodically broadcasts the connection table information to the end devices in the network. This feature is added to ensure the Full-Function Devices (FFDs) are notified of all the other end devices that join or leave the network. The frequency of the periodic connection table broadcast is a compile option. | Only supported by MiWi Star protocol. |
| `#define ENABLE_LINK_STATUS` | Provides the means for the PAN Coordinator to monitor the status of the network. By default, each end device sends a link status message to the PAN Coordinator every 15 seconds indicating it is alive/active and communicating on the network. The frequency of the link status is a compile time option. | Only supported by MiWi Star protocol. |
| `#define MAKE_ENDDEVICE_PERMANENT` | Enables the Reduced-Function Device (RFD) sleeping device to be deemed permanently active in the network. PAN Coordinator do not expect any link status being sent from the end device which has enabled this feature. | Only supported by MiWi Star protocol. |
| `#define RADIO_BUFFER_SIZE 64` | Enables the size of the Radio Buffer allocated and used along with the TX and RX buffers. | Only supported by MiWi Mesh protocol. |
| `#define MY_ADDRESS_LENGTH 8` | Defines the address length based on the supported radio devices (in IEEE or non IEEE format). | This can be 2 to 8 bytes in size. |
| `#define MEM_ACK_INFO_SIZE 5` | Displays the maximum number of acknowledgments that must be handled after a TX. | — |
| `#define ACTIVE_SCAN_RESULT_SIZE 5` | Defines the number of results to be managed/accounted for based on the Active Scan responses. | — |
| `#define PAYLOAD_START 0` | Initializes the initial size of the payload to zero . | — |
| `#define PAN_ID 0x1234` | Defines the PAN ID number when a PAN Coordinator starts the network. | The PAN ID can be dynamically assigned in P2P and Star networks when the PAN ID is defined as `0xFFFF`. |

**Note:** Some of the values shown along with the definitions point to the default values that are part of the latest MiWi stack.

## MiApp FUNCTION INTERFACES

Other than the options used as Macro definitions as part of the configuration file which refers to the hardware definitions, the application layer also uses the software definitions through the function calls to communicate with the Microchip proprietary wireless protocol layer, thus controlling the transceiver indirectly to perform wireless communication. There are five categories of function calls to the protocol layers from the application layer:

- Initialization
- Handshaking
- Sending Messages
- Receiving Messages
- Special Functionality

The following sections describe the function interfaces in detail, as well as associated structure definitions.

### Initialization

To initialize the RF transceiver and protocol stack, the application layer only needs to trigger the initialization process by calling the function *ProtocolInit*. The full function signature is as follows:

```
bool MiApp_ProtocolInit (bool bNetworkFreezer);
```

There is only one parameter for the initialization. The input boolean decides if the network freezer feature is performed during the initialization. When the network freezer feature is performed, the old network settings that are stored in nonvolatile memory are restored. The return value is a boolean to indicate if the operation is successful.

Other than the normal initialization process, wireless applications may need to change the transmit or receive frequency during operation. MiApp defines the following function to change the operating frequency of the RF transceiver according to the predefined channel. Each channel defines the frequency either according to the specification, or the RF transceiver settings under different operating frequency bands. The function signature is as follows:

```
bool MiApp_SetChannel (uint8_t Channel);
```

The only input parameter is the channel to be set. The return value indicates if the operation is successful. The possible channel numbers are from 0 to 31. Depending on the RF transceiver, frequency band and data rate, not all channels from 0 to 31 may be valid under all conditions. If the input channel is invalid under current conditions, the operating channel is unchanged and the return value is FALSE to indicate failure.

#### PERSONAL AREA NETWORK IDENTIFIER (PAN ID)

The PAN ID is used by the Coordinator in the Mesh network to start a PAN network. Any device in the vicinity can identify the PAN and join based on the requirements. The PAN ID is defined by a macro in the file `miwi_config.h` with the default PAN ID as number 0x1234.

```
#define PAN_ID 0x1234
```

The function related to PAN ID, specifically for the Mesh network, is MiAPP_SetAddressPan. This function sets the device address and the PAN ID and has no return value. The input parameters to this function are the short address of the node and the PAN ID.

The short address is a unique address for each wireless node in the network. The PAN ID value must be the same for all the wireless nodes in a chosen network.

### Handshaking

Unless hard coded in manufacturing, in most applications, the two communication endpoints need an introduction before they can unicast messages between a pair of wireless nodes. The introduction for a networking protocol is sometimes called *joining the network*. For the P2P protocol, this process is called *pairing*. Since this strategy does not focus on any particular topology or protocol, this process is generally called the *handshaking phase*. Without a handshaking process, wireless nodes can only use broadcast, which treats every wireless node in the source radio range as the audience, to communicate with each other.

The following function calls for handshaking are available to the application layer:

- MiApp_StartConnection
- MiApp_SearchConnection
- MiApp_EstablishConnection
- MiApp_ConnectionMode
- MiApp_RemoveConnection

#### MiApp_StartConnection

The function call MiApp_StartConnection enables a wireless node to start operating in different ways. There are three ways to start a PAN: start a PAN directly on a particular channel, or start a PAN after either of the two channel assessments. The full function signature is as follows:

```
bool  MiApp_StartConnection  (uint8_t  Mode,
uint8_t ScanDuration, uint32_t ChannelMap);
```

The return value of the function call indicates if the operation is successful.

The input parameter mode specifies the mode of starting the PAN. The modes are as follows:

- **START_CONN_DIRECT** – start the connection at the current channel without any channel assessment.
- **START_CONN_ENERGY_SCN** – start the connection after an energy detection scan and the PAN start at the channel with the lowest energy.
- **START_CONN_CS_SCN** – start the connection after a carrier sense scan and the PAN start at the channel with the lowest carrier sense detected.

For the transceivers that do not support energy detection or carrier sense scan, or both, those modes are not valid and the function should start the PAN without any channel assessment if such a mode is specified in the input parameter.

The input parameter ScanDuration specifies the maximum time to perform the channel assessment. The max-and-hold method must be applied for the scan period, if multiple scans can be performed. In case the starting mode specifies no channel assessment, this input parameter is discarded. The value of the input parameter ScanDuration complies with the definition in the IEEE 802.15.4™ specification. Its range is from 1 to 14. Equation 1 is the formula to calculate the scan duration time.

**EQUATION 1: SCAN DURATION CALCULATION**

$$\text{ScanTime(us)} = 960 * (2^{\text{ScanDuration}} + 1)$$

As the formula shows, a ScanDuration of 10 is roughly one second. An increase by one roughly doubles the time, while a decrease by one roughly cuts the time in half.

The input parameter ChannelMap specifies the channels to be scanned in the process. ChannelMap is defined as a 4 byte double word. It uses bit map to represent channel 0 to channel 31. When a bit is set in the double word, it means that the corresponding channel performs the channel assessment. For instance, if bit `0` of the input parameter ChannelMap is set, channel 0 performs the channel assessment. To perform channel assessment on all available channels, set the input parameter ChannelMap to `0xFFFFFFFF`.

### MiApp_SearchConnection

The function call MiApp_SearchConnection searches for and discovers the existing peer wireless nodes in the neighborhood. This procedure is also known as active scan. In some applications, this step informs the device whether it should start a PAN or choose a PAN to join. If a PAN is started, this procedure can be used to decide which PAN identifier to chose. If the device joins a PAN, this procedure is used to choose which PAN and which device to join.

The full function signature is as follows:

```
uint8_t    MiApp_SearchConnection(uint8_t
ScanDuration, uint32_t ChannelMap);
```

The return value of this function indicates the total number of returned PANs. The result of the return PAN is stored in the global variable in the format of structure ACTIVE_SCAN_RESULT, which is defined as follows:

```
typedef struct
{
     uint8_t Channel;
    uint8_t Address  [MY ADDRESS LENGTH];
     API_UINT16_UNION    ;
     uint8_t RSSIValue;
     uint8_t LQIValue;
     union
     {
        uint8_t Val;
        struct
        {
          uint8_t Role:       2;
          uint8_t Sleep:      1;
          uint8_t SecurityEn: 1;
          uint8_t RepeatEn:   1;
          uint8_t AllowJoin:  1;
          uint8_t Direct:     1;
          uint8_t altSrcAddr: 1;
        } bits;
     }Capability
         #if ADDITIONAL_NODE_ID_SIZE>0
        uint8_t PeerInfo[ADDITIONAL_NODE
         _ID_SIZE];
         #endif
} ACTIVE_SCAN_RESULT;
```

In this structure, element address indicates the address of the device that responded to the active scan.

Element PAN ID indicates the PAN identifier, if available. The PAN identifier is used to specify the network ID.

Elements RSSI and LQI indicate the strength and quality of the responding signal, respectively. This information may not be available for all RF transceivers.

Element Capability contains information regarding the capability of the device that sends back the response. It is a bit map of capabilities, which is defined in the union. Depending upon the protocol used under the application layer, the capability information may not be available.

### MiApp_EstablishConnection

The function call MiApp_EstablishConnection establishes a connection with one or more devices. The full function signature is as follows:

```
uint8_t MiApp_EstablishConnection(INPUT
uint8_t ActiveScanIndex, INPUT uint8_t
Mode) ;
```

This function call returns a byte to indicate the index of the new peer node in the connection table. If the return value is `0xFF`, it means the procedure to establish a connection has failed after attempting the number of predefined retries. If there are multiple connections established during the procedure, the return value is the index of the connection table for one of the connections.

The parameter ActiveScanIndex is the index in the active scan result table for the node to establish connection. If the value is `0xFF`, the protocol tries to establish a connection with any device. For this reason, multiple connections may be established during the process.

The parameter mode specifies the connection mode. There are two modes defined:

- **MODE_DIRECT** – this mode directly establishes a connection in the radio range. The P2P stack uses this mode to establish a connection, while a network protocol uses it to establish a connection with a parent to join the network.
- **MODE_INDIRECT** – this mode is used by a network protocol to establish a connection across the network with one or more hops. The connected devices may or may not be in the radio range of the requesting node. In this case, the input parameter ActiveScanIndex has the value `0xFF`.

### MiApp_ConnectionMode

The function call MiApp_ConnectionMode sets the connection mode that regulates whether the current wireless node is able to accept direct connections from new devices. The full function signature is as follows:

```
void   MiApp_ConnectionMode(INPUT   uint8_t
Mode);
```

There is no return value for this function. The input parameter mode indicates the mode of the operation. The available modes of operation are as follows:

- **ENABLE_ALL_CONN** – this mode enables the connection under any condition. This is the default mode when the application starts to run.
- **ENABLE_PREV_CONN** – this mode only enables old connections. Connection requests from nodes that are already on the connection table is allowed. Otherwise, the request is ignored.
- **ENABLE_ACTIVE_SCAN_RSP** – this mode enables the current node to respond to any active scan request to identify itself.
- **DISABLE_ALL_CONN** – this mode disables all connection requests, including active scan.

The connection privilege decreases from ENABLE_-CONN to DISABLE_ALL_CONN. Any higher privilege has all the rights for the lower one.

### MiApp_RemoveConnection

The function call MiApp_RemoveConnection allows the current node to disconnect certain connections. The full function signature is as follows:

```
void      MiApp_RemoveConnection(INPUT
uint8_t ConnectionIndex);
```

There is no return value for this function. The input parameter ConnectionIndex specifies the index in the connection table for the peer node to be removed. If the ConnectionIndex is `0xFF`, the device removes all connections and leave the network. In a network protocol, this also means that all the device's children leaves the network. In case that the ConnectionIndex points to the parent node in a network protocol, the current node and all of its children must leave the network. If the connection index points to a node that is not the parent of the current node, the connection is removed and the device stays in the PAN.

## Sending Messages

The most important functionality of a wireless node is to communicate, or send and receive data. All protocols have reserved buffers for the data transfer, with the size equal or larger than TX_BUFFER_SIZE defined in the configuration file. Two macro functions are defined to manage the TX buffer in the stack:

```
#define MiApp_FlushTx();
#define MiApp_WriteData(a)
TxBuffer[TxData++] = a
```

The function MiApp_FlushTx is used to reset the pointer of the transmission buffer in the stack. It has no parameter and no return value.

The function MiApp_WriteData is used to fill one byte of data to the transmission buffer in the stack. The only input parameter is the one byte of data to be filled into the transmission buffer.

Usually, MiApp_FlushTx is called first to reset the buffer pointer and then MiApp_WriteData is called multiple times to fill the transmission buffer, one byte at a time.

After the transmission buffer is filled, the next step is to trigger the message to be transmitted by the protocol layer. There are three ways to transmit a message:

- Broadcast
- Unicast to the node by its index in the connection table
- Unicast to the node by its address, either the permanent address or the alternative network address.

Broadcasting a message targets all devices regardless of their addresses. The full function signature for a broadcast is as follows:

```
bool MiApp_BroadcastPacket(bool SecEn);
```

The return value of this function call indicates if the transmission is successful. The only input parameter, SecEn, is a boolean to specify if the payload needs to be secured.

Unicast targets a single device as a destination. There are two ways to unicast a message:

- The destination is represented by an index on the connection table
- The destination address is clearly given, either the permanent address or a network address.

The full function signature for unicast with an index of the connection table is as follows:

```
bool      MiApp_UnicastConnection(uint8_t
ConnectionIndex, bool SecEn);
```

The return value of this function call indicates if the transmission is successful. The input parameter ConnectionIndex is the index of the destination node in the connection table. The input parameter SecEn is a boolean to indicate if the payload needs to be secured.

The full function signature for unicast with a destination address is as follows:

```
bool         MiApp_UnicastAddress(uint8_t
*DestinationAddress, bool PermanentAddr,
bool SecEn);
```

The return value of this function call indicates if the transmission is successful.

The input parameter address is the pointer that points to the destination address.

The input boolean parameter PermanentAddr indicates if the destination address is a permanent address or an alternative network address. For P2P or Star protocol, only the permanent address is used, thus the input parameter PermanentAddr has no effect.

The input parameter SecEn indicates if the payload needs to be secured.

## Receiving Messages

The other important functionality of the transceiver is to receive messages. The application layer needs to know when a message is received, the content of the message and occasionally on how the message is received. The application layer also needs to discard the message to release the resources and enable to receive and process new messages. To work with the flow as described, there are two function calls and one structure to define.

### MiApp_MessageAvailable

The function call MiApp_MessageAvailable has no input parameter and returns a boolean to indicate if a new message is received and is available for processing in the application layer. The full function signature is as follows:

```
bool MiApp_MessageAvailable(void);
```

### DATA STRUCTURE FOR RECEIVED MESSAGES

All received messages that are forwarded to the application layer are stored in a global variable defined in the format of RECEIVED_MESSAGE as follows:

```
typedef struct
{
     union
     {
        uint8_t Val;
        struct
        {
           uint8_t broadcast: 2;
           uint8_t ackReq: 1;
           uint8_t secEn: 1;
           uint8_t repeat: 1;
           uint8_t command: 1;
           uint8_t srcPrsnt: 1;
           uint8_t altSrcAddr: 1;
        } bits
     } flags;

    API_UINT16_UNION SourcePANID;
     uint8_t *SourceAddress;
     uint8_t *Payload;
     uint8_t PayloadSize;
     uint8_t PacketRSSI;
     uint8_t PacketLQI;

} RECEIVED_MESSAGE;
```

Depending upon the transceiver and the Microchip proprietary protocol used, not all elements in the structure are valid.

### MiApp_DiscardMessage

The function call MiApp_DiscardMessage has no input parameter and returns no value. The application layer calls this function to notify the Microchip proprietary wireless protocol layer that the current packet is done processing and it is ready to process the next packet. The full function signature is as follows:

```
void MiApp_DiscardMessage(void);
```

In a MiWi wireless network for RFD sleeping devices there needs to be a mechanism for these devices to wake-up and receive a data packet. The function MiApp_RequestData is used by RFD sleeping devices to request for cached data from their parent device.

```
void MiApp_RequestData(void);
```

This function has no input parameter and returns no value. The `rxMessage.SourceAddress` holds the received data.

| **Note:** | This function *only* applies for Sleeping RFD devices. |
|---|---|

## Special Functionality

Some transceivers have special functionalities that enable the protocol stack to be more robust and adaptable to the environment.

### NOISE DETECTION SCAN

The noise detection scan enables the transceiver to detect the noise level in the environment. It is valuable to start a new PAN at a quiet frequency, as well as deciding whether channel hopping is necessary and to which channel to hop.

The full function signature is as follows:

```
uint8_t MiApp_NoiseDetection(INPUT
uint32_t ChannelMap, INPUT uint8_t
ScanDuration, INPUT uint8_t DetectionMode,
OUTPUT uint8_t *RSSIValue);
```

The function call MiApp_NoiseDetection returns the channel with the least amount of noise. The function has four function parameters:

- **ChannelMap** – this input parameter defines the bit map of channels to be scanned. For each transceiver, the supported number of channels is different; therefore, not all bit maps in the input parameter ChannelMap are valid.
- **ScanDuration** – this input parameter defines the total times of noise detection on each channel. The max-and-hold mechanism is used to detect the noise level on each channel. Input parameter ScanDuration follows the IEEE 802.15.4™ specification as detailed earlier in this application note with a formula to calculate real time.
- **DetectionMode** – this input parameter defines the detection mode to be used: energy detection or carrier sense detection. Not all detection modes are supported by all RF transceivers.
- **NoiseLevel** – this output parameter returns the noise level on the best channel, or the channel of the return value of this function call. This output parameter enables the application layer to view the noise level on the best possible channel. The higher the NoiseLevel parameter value, the noisier the environment.

### TRANSCEIVER POWER STATE

To enable a wireless node powered by a battery, it is necessary to set the radio transceiver to a different power state, or to put it into sleep and wake it up periodically. The function call MiApp_TransceiverPowerState is defined to achieve this goal:

```
uint8_t MiApp_TransceiverPowerState(INPUT
uint8_t Mode);
```

The only input parameter for this function call is the operation mode. The predefined operation modes are as follows:

- **POWER_STATE_SLEEP** – puts the transceiver into Sleep mode.
- **POWER_STATE_WAKEUP** – wakes up the transceiver without sending any data request.
- **POWER_STATE_WAKEUP_DR** – wakes up the transceiver and then sends out a data request to its main associated device to ask for incoming data.

The function call MiApp_TransceiverPowerState returns a byte to indicate the status of the operation. The predefined operation status return values are as follows:

- **SUCCESS** – indicates that every operation is successful.
- **ERR_TRX_FAIL** – indicates that the request to sleep or wake-up the transceiver failed.
- **ERR_TXFAIL** – indicates that the request to send out data failed. This option is only available when `WAKE_DR` is the operation mode.
- **ERR_RXFAIL** – indicates that the request to receive data from the parent failed. This option is only available when `WAKE_DR` is the operation mode.

### FREQUENCY AGILITY

The frequency agility is the capability to hop channels during operation to bypass persistent noise at certain frequency.

Not all transceivers and protocols support frequency agility. Frequency agility functions are optional for application interfaces.

There are two functions to establish frequency agility. One function is used to initiate the frequency agility procedure. The other function is used to synchronize the connection if communication is lost due to frequency agility performed at the other end of the communication.

The full function signature to initiate the frequency agility procedure is as follows:

```
bool MiApp_InitChannelHopping(INPUT
uint32_t ChannelMap);
```

The return value of the function call MiApp_InitChannelHopping indicates if the channel hopping operation is successful. The ChannelMap input parameter indicates available channels to move to. The ChannelMap parameter is a bit map of possible channels. If a channel is available, the corresponding bit (nth bit for channel n) is set; otherwise, it is cleared.

The MiApp specification does not define when to start channel hopping. The trigger event can be continuous transmission or receiving failures, or just periodically searching for the optimal frequency to operate the wireless application. It is up to the wireless application to decide when to start the channel hopping process. The MiApp specification provides the proper interface to the Microchip proprietary wireless protocols to perform these actions as dictated by the application layer.

Once the channel hopping procedure is done, it is possible that some of the wireless nodes, especially those in Sleep mode, do not know that the network moved to a different channel. It is necessary to define a function to resynchronize the connection:

```
bool       MiApp_ResyncConnection(INPUT
uint8_t ConnectionIndex, INPUT uint32_t
ChannelMap);
```

The return value of the function MiApp_ResynConnection indicates if the resynchronization procedure is successful. There are two input parameters:

- **ConnectionIndex** – this is the index of the device to be synchronized in the connection table.
- **ChannelMap** – this is the bit map of the possible channels to synchronize connection.

### SECURITY

The network security in MiWi protocol is enabled by the `ENABLE_SECURITY`. However, the Broadcast and the Unicast functions enable the packet security through their respective MiApp functions. For more information, refer to **"Sending Messages"** section.

| **Note:** | The macro `ENABLE_SECURITY` must be enabled in all the nodes to use this security feature. |
|---|---|

## PROTOCOL SPECIFIC FUNCTIONS AND DEFINITIONS

### Functions and Definitions Specific to MiWi Star Protocol

#### BROADCAST CONNECTION TABLE

The function MiApp_BroadcastConnectionTable is used by the PAN Coordinator. This function is used only in a Star network by the PAN Coordinator and is a command type packet. The PAN Coordinator broadcasts Connection Table as part the network protocol function for all the End Devices in the network whenever an End Device joins the network. The function checks for the defined macro ENABLE_PERIODIC_-CONNECTION_TABLE_SHARE. Therefore, whenever a new device joins the Star network, the PAN Coordinator sends an updated connection table to the connected nodes in its PAN network. The full function signature used in the MiWi Star is as follows:

```
void     MiApp_BroadcastConnectionTable
(void);
```

This function does not have any of the input and return parameters.

#### UNICAST MESSAGE

The function MiApp_UnicastStar is used by the End Device in the Star network. This function call expects two variables , one is the EndDevice_Connection_Index and the other is a Boolean input to check for security if enabled or disabled. EndDevice_Connection_Index is the index of the Destination End Device in the END_DEVICES_Unique_Short_Address. END_DEVICES_Unique_Short_Address is the structure that stores the 3 bytes short addresses of End Devices connected to the PAN Coordinator. The function returns `true` if the transmission is successful.

```
bool MiApp_UnicastStar(bool SecEn);
```

The return value of the function call indicates if the operation is successful.

#### LEAVE NETWORK

The function MiApp_Leave_Network is used by the End Device in the Star network. This function is used by End Devices to indicate that it is leaving of the network and is related only to the Star network. The device is considered inactive when the command is acknowledged by PAN Coordinator. However, the End Device can still join the same PAN when required. The full function signature used in the MiWi Star is as follows:

```
void MiApp_Leave_Network(void);
```

This function does not have any of the input and return parameters.

## Functions and Definitions Specific to MiWi Mesh Protocol

### SET PAN ID AND DEVICE ADDRESS

This function must be used by all device roles - PAN Coordinator (FFD), Coordinator (FFD), and Sleeping RFDs/Sleeping End Devices.

The function call MiApp_SetAddressPan has two parameters:

- **Address** – this input parameter is the short address input for the wireless node. This address must be unique for each node.
- **PAN ID** – this input parameter is the 2 byte network identifier (PAN ID) on which the wireless node operates on.

**Note:** This function is used *only* by the Coordinator.

The MiApp_SetAddressPan sets the 2 or 8 byte address for the device based on the IEEE802.15.4 compliance. The macro MY_ADDRESS_LENGTH must enabled to execute the function. The full function signature used in the MiWi Mesh protocol stack is as follows:

```
void    MiApp_SetAddressPan    (uint8_t
*address, uint16_t panid );
```

The input parameters are the pointer to the 8 byte or the 2 byte address and the 2 bytes of the PAN ID. There is no return value for this function.

### GET PARENT ADDRESS

The function MiApp_GetParentAddress is used by the Coordinator or by a sleeping RFD to identify the short address of its parent. Usually, wireless Mesh network consists of multiple nodes connected to each other. In order to understand the network mapping, it is important to know the relationship between the nodes. The full function signature used in the MiWi Mesh is as follows:

```
addr_t MiApp_GetParentAddress (void);
```

The function MiApp_GetParentAddress has no input parameter and the return value is an address. All addresses that are received to the application layer are stored in a global variable defined in the format of addr_t as follows:

```
typedef  union addr_t_def
{
     uint8_t bytes[ADDRESS_LEN];
     uint16_t even_addr_pack;
} addr_t;
```

`addr_t` is a structure to store the address of the parent node to which the Coordinator or the RFD Sleeping device is connected as a child.

**Note:** This function is *only* used by the Coordinator or a sleeping RFD in a Mesh network.

### NETWORK MEMBER

In a Mesh Network, if a wireless node loses its connection with its parent node or if it is not part of network, there must be a way to find out the lost connection from the application layer. The function MiApp_IsMemberOfNetwork can be used to identify whether a network exists and the node is part of the existing network. The full function signature used in the MiWi Mesh protocol stack is as follows:

```
bool MiApp_IsMemberOfNetwork(void);
```

There is no input parameter for this function and the return value is a boolean `true` if the node is part of an existing network. When called by the PAN Coordinator device, this function always returns the value `true` (the PAN Coordinator creates the network). When used in a RFD sleeping device or a Coordinator, this function helps find the status of the wireless node in the network. If a boolean, `false` is returned by the function call which indicates that the node is not part of any network. This function must be used by all device roles: PAN Coordinator, Coordinator, and Sleeping RFDs.

### INDIRECT MESSAGE

The function MiApp_InitSleepRFDBuffers is used to initialize the indirect message buffer for the devices in the network. Indirect message buffers define the number of messages cached inside the parent device. The parent device for a RFD sleeping device can be the PAN Coordinator or a Coordinator. Hence, this function is only applicable for PAN Coordinator or Coordinator devices in the network. The full function signature used in the MiWi Mesh protocol stack is as follows:

```
uint16_t MiApp_InitSleepRFDBuffers
(uint8_t *Buffer, uint16_t
BufferSize, uint16_t rfdMaxDataSize);
```

The function call MiApp_InitSleepRFDBuffers returns a number of sleeping RFD buffers allocated. The function has three input parameters:

- **Buffer** – this input parameter is the pointer to the buffer array.
- **BufferSize** – this input parameter is the size of the buffer.
- **rfdMaxDataSize** – this input parameter is the maximum length of data to be stored for each sleeping RFDs in the network.

To enable the Sleep Buffers in the Coordinator, the macro MAKE_RFDBUFFLEN must be initialized. The macro defines the size of buffer.

**EQUATION 2: SIZE OF BUFFER**

```
#define MAKE_RFDBUFFLEN(RFDCount,DATALEN)
        ((ADDRESS_LEN+3+DATALEN)*RFDCount)
```

Where,

*RDFCount* = Number of Sleeping RFDs to support

*DATALEN* = Maximum length of packet to store for RFD

*ADDRESS_LEN* = Length of the address

Other overheads are automatically included in this MACRO.

| | |
|---|---|
| **Note 1:** | The macro ENABLE_INDIRECT_MESSAGE must be enabled in the PAN Coordinator for the function to execute. |
| **2:** | This function is applicable *only* for the PAN Coordinator Mesh network. |

## DATA REQUEST FROM SLEEPING DEVICES

In a MiWi wireless network for RFD sleeping devices there must be a mechanism for these devices to wake-up and receive a data packet. The function MiApp_RequestData is used by RFD sleeping devices to request for cached data from their parent device. If data is buffered by the network, it is delivered following this request. The full function signature used in the MiWi Mesh protocol stack is as follows:

```
void MiApp_RequestData(void);
```

This function has no input parameter and returns no value. The `rxMessage.SourceAddress` holds the received data.

| | |
|---|---|
| **Note:** | This function *only* applies for Sleeping devices used in the Mesh protocol stack. |

## NETWORK SECURITY

The function MiApp_SetNetworkSecure is used to secure all the messages in the network. The following packet types are secured:

- Data packet
- Command packets
- Status packets
- Control packets
- Network maintenance

This function can be called by all nodes (PAN Coordinator, Coordinator, RFD Sleeping Device) in the network. The network security function enables or disables the security of the network for any new device communicating in the Mesh network. User can already enable/disable user level security by using the boolean in MiApp_UnicastAddress function, however the network level messages (that user does not normally deal with) can also be secured. The MiApp_SetNetworkSecure allows for such operation but the necessary precaution must be taken with this function because the joining device must know the key of the network it attempts to join. Otherwise, the device is unable to join the network.

```
void MiApp_SetNetworkSecure(bool isSecure)
```

There is no return value for this function. The only input parameter, isSecure, is a boolean to specify if the payload needs to be secured.

| | |
|---|---|
| **Note:** | The macro ENABLE_SECURITY must be enabled in all the nodes to use this security feature. |

## CONCLUSION

For wireless application developers who are looking for a short range and low-data rate solution, the requirements differ from point-to-point communication to routing messages across several hops. The MiApp specification from Microchip provides a low-cost and low-complexity solution to address nearly all those applications. It enables the wireless application developer to use Microchip's proprietary wireless protocols with little or no modification in the migration path. Working with MiMAC at the lower layer also indirectly enables developers to choose any existing and future RF transceivers supported by Microchip. It is highly recommended for the readers to refer to the Application Note "*AN1283 Microchip Wireless MiWi™ Media Access Controller – MiMAC*" (DS00001283) to understand the total solution available for wireless applications from Microchip. Standardization of the lower MAC layer as MiMAC and the higher application layer as MiApp offers wireless application developers the maximum flexibility in the software development process.

## REFERENCES

- "*Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs)*", IEEE Std 802.15.4™-2003, New York: IEEE, 2003.
- IEEE Std 802.15.4™-2006, (Revision of IEEE Std 802.15.4-2003). New York: IEEE, 2006.
- *"AN1283 Microchip Wireless MiWi™ Media Access Control - MiMAC*" (DS00001283), Yifeng Yang, Pradeep Shamanna, Derrick Lattibeaudiere, and Vivek Anchalia, Microchip Technology Inc., 2009-2017.
- *"AN1204 Microchip MiWi™ P2P Wireless Protocol"* (DS00001204), Yifeng Yang, Pradeep Shamanna, Derrick Lattibeaudiere, and Vivek Anchalia, Microchip Technology Inc., 2008-2017.
- *"AN1066 MiWi™ Wireless Networking Protocol Stack"* (DS01066), David Flowers and Yifeng Yang, Microchip Technology Inc., 2007-2010.
- *"MRF24J40 IEEE 802.15.4™ 2.4GHz RF Transceiver Data Sheet"* (DS39776), Microchip Technology Inc., 2008.
- *""MRF89XA Ultra Low-Power, Integrated ISM Band Sub-GHz Transceiver Data Sheet"* (DS70622), Microchip Technology Inc. 2011.

## APPENDIX A: SOURCE CODE FOR MIWI P2P, STAR, AND MESH WIRELESS NETWORKING PROTOCOL STACK

All of the software covered in this application note are available through Microchip Libraries for Applications (MLA). This MLA suite/archive can be downloaded from the Microchip corporate website at www.microchip.com/mla or www.microchip.com.

## REVISION HISTORY

**Revision A (July 2009)**

- This is the initial release of this document.

**Revision B (August 2017)**

- Updated Figure 1.
- Updated Table 1 and added new definitions.
- Added Personal Area Network Identifier (PAN ID) and Sending Messages to **Section "MiApp Function Interfaces"**.
- Added **Section "Protocol Specific Functions and Definitions"**
- Added **Appendix A: "Source Code for MiWi P2P, Star, and Mesh Wireless Networking Protocol Stack"**.
- Incorporated minor updates to text and corrected formatting throughout the document.

**NOTES:**

ISBN: 978-1-5224-2002-6

MICROCHIP

# Worldwide Sales and Service

## AMERICAS

**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

## ASIA/PACIFIC

**Asia Pacific Office**
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon

**Hong Kong**
Tel: 852-2943-5100
Fax: 852-2401-3431

**Australia - Sydney**
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

**China - Beijing**
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

**China - Chengdu**
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

**China - Chongqing**
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

**China - Dongguan**
Tel: 86-769-8702-9880

**China - Guangzhou**
Tel: 86-20-8755-8029

**China - Hangzhou**
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

**China - Hong Kong SAR**
Tel: 852-2943-5100
Fax: 852-2401-3431

**China - Nanjing**
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

**China - Qingdao**
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

**China - Shanghai**
Tel: 86-21-3326-8000
Fax: 86-21-3326-8021

**China - Shenyang**
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

**China - Shenzhen**
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

**China - Wuhan**
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

**China - Xian**
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

## ASIA/PACIFIC

**China - Xiamen**
Tel: 86-592-2388138
Fax: 86-592-2388130

**China - Zhuhai**
Tel: 86-756-3210040
Fax: 86-756-3210049

**India - Bangalore**
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

**India - New Delhi**
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

**India - Pune**
Tel: 91-20-3019-1500

**Japan - Osaka**
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

**Japan - Tokyo**
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

**Korea - Daegu**
Tel: 82-53-744-4301
Fax: 82-53-744-4302

**Korea - Seoul**
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

**Malaysia - Kuala Lumpur**
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

**Malaysia - Penang**
Tel: 60-4-227-8870
Fax: 60-4-227-4068

**Philippines - Manila**
Tel: 63-2-634-9065
Fax: 63-2-634-9069

**Singapore**
Tel: 65-6334-8870
Fax: 65-6334-8850

**Taiwan - Hsin Chu**
Tel: 886-3-5778-366
Fax: 886-3-5770-955

**Taiwan - Kaohsiung**
Tel: 886-7-213-7830

**Taiwan - Taipei**
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

**Thailand - Bangkok**
Tel: 66-2-694-1351
Fax: 66-2-694-1350

## EUROPE

**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

**France - Saint Cloud**
Tel: 33-1-30-60-70-00

**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: 47-7289-7561

**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

11/07/16