

---

## Microchip MiWi™ P2P Wireless Protocol

---

<p><i>Author: Yifeng Yang Microchip Technology Inc.</i></p>
---

### INTRODUCTION

The demand is growing for more and more applications to move to wireless communication.

The benefits are reduced costs and ease of implementation. Wireless communication does not require cabling and other hardware, and the associated installation costs. It also can be implemented in locations where cabling would be hard, if not impossible, to install.

Since the IEEE released the Wireless Personal Area Network (WPAN) specification (IEEE 802.15.4™) in 2003, it has become the de facto industry standard for low-rate WPANs (LR-WPAN). The specification applies to low data rate applications with low-power and low-cost requirements.

The Microchip MiWi™ P2P Wireless Protocol is a variation of IEEE 802.15.4, using Microchip's MRF24J40 2.4 GHz transceiver and any Microchip 8, 16 or 32-bit microcontroller with a Serial Peripheral Interface (SPI).

The protocol provides reliable direct wireless communication via an easy-to-use programming interface. It has a rich feature set that can be compiled in and out of the stack to meet a wide range of customer needs – while minimizing the stack footprint.

This application note describes the Microchip Wireless (MiWi™) Peer-to-Peer (P2P) Protocol and its differences from IEEE 802.15.4. The document details the supported features and how to implement them. Simple, application-level data structures and programming interfaces also are described.

This application note assumes that readers know C programming. It is strongly recommended that readers review the IEEE 802.15.4 specification before starting this application note or working with the MiWi P2P wireless protocol.

### Protocol Overview

The MiWi P2P protocol modifies the IEEE 802.15.4 specification's Media Access Control (MAC) layer by adding commands that simplify the handshaking process. It simplifies link disconnection and channel hopping by providing supplementary MAC commands.

However, application-specific decisions, such as when to perform an energy detect scan or when to jump channels, are not defined in the protocol. Those issues are left to the application developer.

### Protocol Features

The MiWi™ P2P Wireless Protocol:

- Provides 16 channels in the 2.4 GHz spectrum (using an MRF24J40 transceiver)
- Operates on Microchip PIC18, PIC24, dsPIC33 and PIC32 platforms
- Supports Microchip C18, C30 and C32 compilers
- Functions as a state machine (not RTOS-dependent)
- Supports a sleeping device at the end of the communication
- Enables Energy Detect (ED) scanning to operate on the least-noisy channel
- Provides active scan for detecting existing connections
- Supports all of the security modes defined in IEEE 802.15.4.
- Enables frequency agility (channel hopping)

### Protocol Considerations

The MiWi P2P protocol is a variation of IEEE 802.15.4 and supports both peer-to-peer and star topologies. It has no routing mechanism, so the wireless communication coverage is defined by the radio range.

Guaranteed Time Slot (GTS) and beacon networks are not supported, so both sides of the communication cannot go to Sleep at the same time.

If the application requires wireless routing instead of P2P communication; or interoperability with other vendors' devices; or a standard-based solution, for marketability, see AN1066 "MiWi™ Wireless Protocol Stack" or AN965, "Microchip Stack for the ZigBee™ Protocol".

## IEEE 802.15.4™ SPECIFICATION AND MIWI™ P2P WIRELESS PROTOCOL

After the initial 2003 release of the IEEE specification, a 2006 revision was published to clarify a few issues. Referred to as IEEE 802.15.4b or 802.15.4-2006, the revision added two PHY layer definitions in the sub-GHz spectrum and modified the security module.

Most of the market's current products, however, use the original, IEEE 802.15.4a specification – also called IEEE 802.15.4-2003 or Revision A. The Microchip MRF24J40 radio supports Revision A of the specification.

In this document, references to IEEE 802.15.4 will mean Revision A of the specification.

### PHY Layers

The MiWi P2P stack uses only a portion of the IEEE 802.15.4 specification's rich PHY and MAC layers' definitions.

The specification defines three PHY layers, operating on a spectrum of 868 MHz, 915 MHz and 2.4 GHz. The MRF24J40 radio operates on the 2.4 GHz, Industrial, Scientific and Medical (ISM) band – freely available worldwide. That spectrum has 16 available channels and a maximum packet length of 127 bytes, including a two-byte Cyclic Redundancy Check (CRC) value.

The total bandwidth for the IEEE 802.15.4, 2.4 GHz ISM band is, theoretically, 250 kbps. In reality, for reliable communication, the bandwidth is 20-30 kbps.

### Device Types

The MiWi P2P protocol categorizes devices based on their IEEE definitions and their role in making the communication connections (see Table 1 and Table 2).

The MiWi P2P stack supports all of these device types.

**TABLE 1: IEEE 802.15.4™ DEVICE TYPES – BASED ON FUNCTIONALITY**

Functional Type	Power Source	Receiver Idle Configuration	Data Reception Method
Full Function Device (FFD)	Mains	On	Direct
Reduced Function Device (RFD)	Battery	Off	Poll from the associated device

**TABLE 2: IEEE 802.15.4™ DEVICE TYPES – BASED ON ROLE**

Role Type	Functional Type	Role Description
Personal Area Network (PAN) Coordinator	FFD	The device starts first and waits for a connection.
End Device	FFD or RFD	The device starts after the PAN coordinator has started to establish a connection.

## Supported Topologies

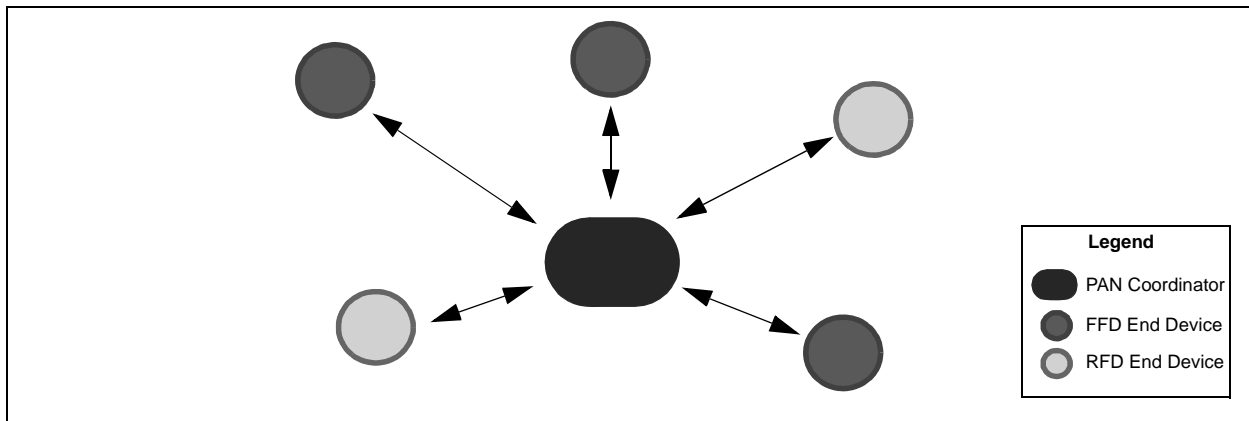
IEEE 802.15.4 and the MiWi P2P stack support two topologies: Star and Peer-to-Peer.

### STAR TOPOLOGY

A typical star topology is shown in Figure 1. From a device role perspective, the topology has one Personal Area Network (PAN) coordinator that initiates communications and accepts connections from other devices. It has several end devices that join the communication. End devices can establish connections only with the PAN coordinator.

As to functionality type, the star topology's PAN coordinator is a Full Function Device (FFD). An end device can be an FFD with its radios on all the time, or a Reduced Function Device (RFD) with its radio off when it is Idle. Regardless of its functional type, end devices can only talk to the PAN coordinator.

**FIGURE 1: STAR TOPOLOGY**

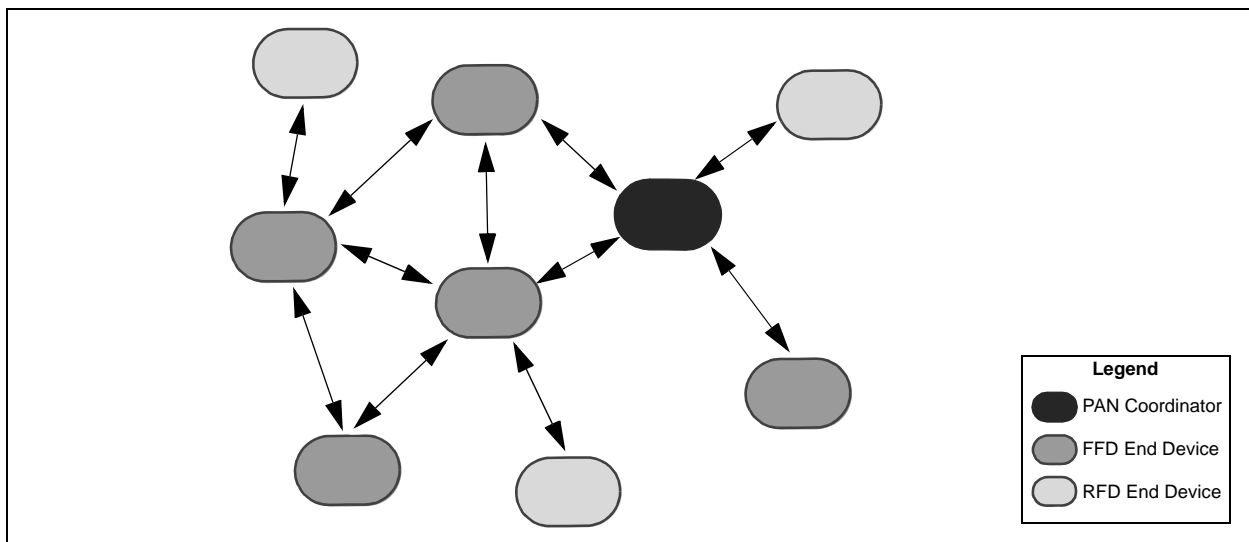


### PEER-TO-PEER (P2P) TOPOLOGY

A typical P2P topology is shown in Figure 2. From a device role perspective, this topology also has one PAN coordinator that starts communication and the end devices. When joining the network, however, end devices do not have to establish their connection with the PAN coordinator.

As to functional types, the PAN coordinator is an FFD and the end devices can be FFDs or RFDs. In this topology, however, end devices that are FFDs can have multiple connections. Each of the end device RFDs, however, can connect to only one FFD and cannot connect to another RFD.

**FIGURE 2: PEER-TO-PEER TOPOLOGY**



## Network Types

The IEEE 802.15.4 specification has two types of networks: beacon and non-beacon.

In a beacon network, devices can transmit data only during their assigned time slot. The PAN coordinator assigns the time slots by periodically sending out a superframe (beacon frame). All devices are supposed to synchronize with the beacon frame and transmit data only during their assigned time slot.

In a non-beacon network, any device can transmit data at any time, as long as the energy level (noise) is below the predefined level.

Beacon networks reduce all devices' power consumption because all of the devices have the opportunity to turn off their radios periodically.

Non-beacon networks increase the power consumption by FFD devices because they must have their radios on all the time. These networks reduce the power consumption of RFD devices, however, because the RFDs do not have to perform the frequent synchronizations.

The MiWi P2P stack supports only non-beacon networks.

## Network Addressing

The IEEE 802.15.4 specification defines two kinds of addressing mechanisms:

- Extended Organizationally Unique Identifier (EUI) or long address – An eight-byte address that is unique for each device, worldwide.

The upper three bytes are purchased from IEEE by the company that releases the product. The lower five bytes are assigned by the device manufacturer as long as each device's full EUI is unique.

- Short Address – A two-byte address that is assigned to the device by its parent when it joins the network.

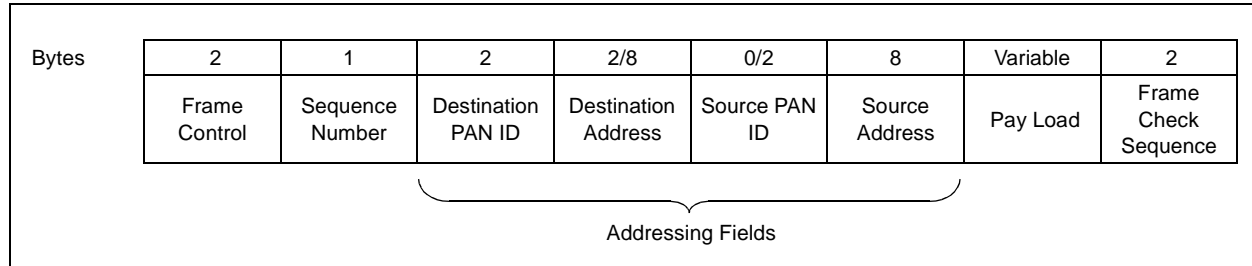
The short address must be unique within the network.

The MiWi P2P stack supports only one-hop communication, so it transmits messages via EUI – or long address – addressing. Short addressing is used only when the stack transmits a broadcast message. This is because there is no predefined broadcast long address defined in the IEEE 802.15.4 specification.

## Message Format

The message format of the MiWi P2P stack is a subset of the IEEE 802.15.4 specification's message format. Figure 3 shows the stack's packet format and its fields.

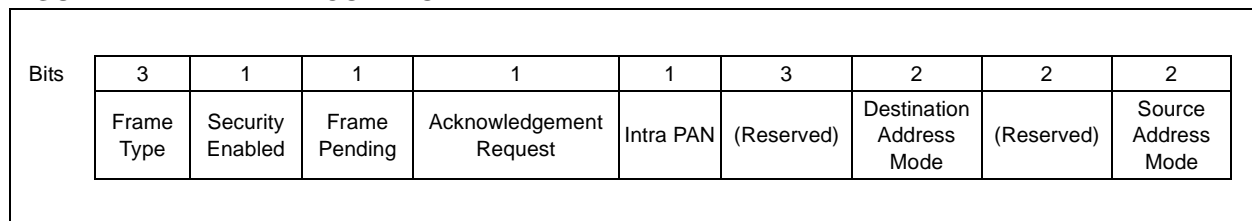
**FIGURE 3: MIWI™ P2P WIRELESS PROTOCOL STACK PACKET FORMAT**



## FRAME CONTROL

Figure 4 shows the format of the two-byte frame control field.

**FIGURE 4: FRAME CONTROL**



The three-bit frame type field defines the type of packet. The values can be:

- Data frame = 001
- Acknowledgement = 010
- Command frame = 011

In the stack, however, the Acknowledgement frame is not used, since all Acknowledgement packets are handled by hardware in the MRF24J40 radio.

The security enabled bit indicates if the current packet is encrypted. If encryption is used, there will be an additional security header which will be detailed in later sections on the security feature.

The frame pending bit is used only in the Acknowledgement packet handled by the MRF24J40 radio hardware. The bit indicates if an additional packet will follow the Acknowledgement after a data request packet is received from a RFD end device.

The intra PAN bit indicates if the message is within the current PAN. If this bit is set to '1', the source PAN ID field in the addressing fields will be omitted. In the stack, this bit is always set to '1', but it could be set to '0' to enable inter-PAN communication. Resetting the bit to '0' can be done in the application layer, if necessary.

The Destination Address mode can be either:

- 16-Bit Short Address mode = 10
- 64-Bit Long Address mode = 11

In the MiWi P2P stack, the Destination Address mode is usually set to the Long Address mode. The Short Address mode is used only for a broadcast message. For broadcast messages, the destination address field in the addressing fields will be fixed to 0xFFFF.

The Source Address mode for the MiWi P2P stack can only be the 64-Bit Long Address mode.

## SEQUENCE NUMBER

The sequence number is eight bits long. It starts with a random number and increases by one each time a data or command packet has been sent. The number is used in the Acknowledgement packet to identify the original packet.

The sequence number of the original packet and the Acknowledgement packet must be the same.

## DESTINATION PAN ID

This is the PAN identifier for the destination device. If the PAN identifier is not known, or not required, the broadcast PAN identifier (0xFFFF) can be used.

## DESTINATION ADDRESS

The destination address can either be a 64-bit long address or a 16-bit short address. The destination address must be consistent with the Destination Address mode defined in the frame control field.

If the 16-bit short address is used, it must be the broadcast address of 0xFFFF.

## SOURCE PAN ID

The source PAN identifier is the PAN identifier for the source device and must match the intra-PAN definition in the frame control field. The source PAN ID will exist in the packet only if the intra-PAN value is '0'.

In the current MiWi P2P stack implementation, all communication is intra-PAN. As a result, all packets do not have a source PAN ID field.

However, the stack reserves the capability for the application layer to transmit the message inter-PAN. If a message needs to transmit inter-PAN, the source PAN ID will be used.

## SOURCE ADDRESS

The source address field is fixed to use the 64-bit extended address of the source device.

## Transmitting and Receiving

The MiWi P2P stack transmits and receives packets according to the IEEE 802.15.4 specification, with a few exceptions.

### TRANSMITTING MESSAGES

There are two ways to transmit a message: broadcast and unicast.

Broadcast packets have all devices in the radio range as their destination. IEEE 802.15.4 defines a specific short address as the broadcast address, but has no definition for the long address. As a result, broadcasting is the only situation when the MiWi P2P stack uses a short address.

There is no Acknowledgement for broadcasting messages.

Unicast transmissions have only one destination and use the long address as the destination address. The MiWi P2P stack requires Acknowledgement for all unicast messages.

If the transmitting device has at least one device that turns off its radio when Idle, the transmitting device will save the message in RAM and wait for the sleeping device to wake-up and request the message. This kind of data transmitting is called indirect messaging.

If the sleeping device fails to acquire the indirect message, it will expire and be discarded. Usually, the indirect message time-out needs to be longer than the pulling interval for the sleeping device.

### RECEIVING MESSAGES

In the MiWi P2P stack, only the messaged device will be notified by the radio. If the messaged device turns off its radio when Idle, it can only receive a message from the device to which it is connected.

For the idling device with the turned off radio to receive the message, the device must send a data request command to its connection peer. Then, it will acquire the indirect message if there is one.

## VARIATIONS FOR HANDSHAKING

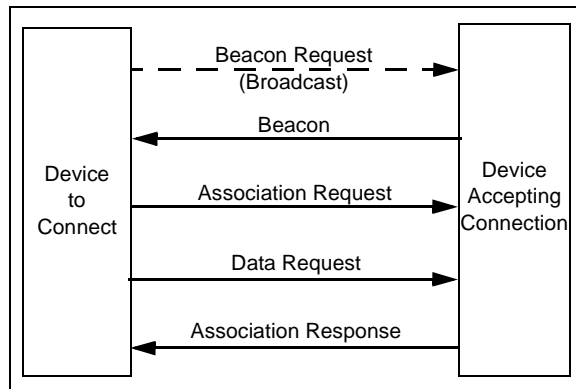
The MiWi™ P2P Wireless Protocol's major difference from the IEEE 802.15.4 specification is in the process of handshaking.

Under IEEE 802.15.4, a device's first step after powering up is to do a handshake with the rest of the world.

The specification's handshaking process, shown in Figure 5, is defined as:

1. The device seeking to communicate sends out a beacon request.
2. All devices capable of connecting to other devices respond with a beacon message.
3. The initiating device collects all of the beacons. (To accommodate multiple responses, the device waits until the active scan request times out.) The device decides which beacon to use to establish the handshake and sends out an association request command.
4. After a predefined time, the initiating device issues a data request command to get the association response from the other side of the intended connection.
5. The device on the other side of the connection sends the association response.

**FIGURE 5: TYPICAL HANDSHAKING IN IEEE 802.15.4™**



Handshaking is the complex process of joining a network. A device can join only a single device as its parent, so the initial handshaking actually is the process of choosing a parent.

Choosing the parent requires:

1. Listing all of the possible parents.
2. Choosing the right one as its parent.

The beacon frames do not use CSMA-CA detection before transmitting to meet the timing requirement of the active scan time-out. As a result, the beacon frames may be discarded due to packet collision.

The MiWi P2P protocol is designed for simplicity and direct connections in star and P2P communication topologies. Some IEEE 802.15.4 requirements obstruct that design:

- The five-step handshaking process – plus two time-outs – requires a more complex stack.
- The association process uses one-connection communication rather than the multi-connection concept of peer-to-peer topology

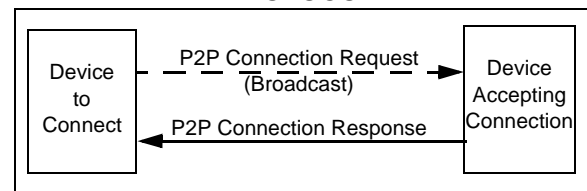
For the preceding reasons, the MiWi P2P protocol uses its own, two-step handshaking process. In that process, shown in Figure 6:

1. The initiating device sends out a P2P connection request command.
2. Any device within radio range responds with a P2P connection response command that finalizes the connection.

This is a one-to-many process that may establish multiple connections, where possible, to establish a Peer-to-Peer topology. Since this handshaking process uses a MAC layer command, CSMA-CA is applied for each transmission. This reduces the likelihood of packet collision.

RFDs may receive the Connection Request command from several FFDs, but can connect to only one FFD. An RFD chooses the FFD, from which it receives the first P2P connection response, as its peer.

**FIGURE 6: HANDSHAKING PROCESS FOR MIWI™ P2P WIRELESS PROTOCOL**



## Custom MAC Commands for MiWi™ P2P Wireless Protocol

The MiWi P2P protocol extends the IEEE 802.15.4 specification's functionality by using custom MAC commands for removing the connection between two devices. All of the protocol's custom MAC commands are listed in Table 3.

**TABLE 3: CUSTOM MAC COMMANDS FOR MIWI™ P2P WIRELESS PROTOCOL**

Command Identifier	Command Name	Description
0x81	P2P Connection Request	Request to establish a P2P connection. Usually broadcast to seek P2P connection after powering up. Alternately, unicast to seek an individual connection. Also used for active scan functionality. (See “ <b>Active Scan</b> ” on Page 13.)
0x82	P2P Connection Removal Request	Removes the P2P connection with the other end device.
0x83	Data Request	Similar to the IEEE 802.15.4™ specification's data request command, a request for data from the other end of a P2P connection if the local node had its radio turned off. Reserved for the previously sleeping device to request the other node to send the missed message (indirect messaging).
0x84	Channel Hopping	Request to change operating channel to a different channel. Usually used in the feature of frequency agility.
0x91	P2P Connection Response	Response to the P2P connection request. Also can be used in active scan process.
0x92	P2P Connection Removal Response	Response to the P2P connection removal request.

### P2P CONNECTION REQUEST

The P2P connection request (0x81) is broadcasted to establish a P2P connection with other devices after powering up. The request can also be unicast to a specific device to establish a single connection.

When the transmitting device receives a P2P connection response (0x91) from the other end, a P2P connection is established.

The P2P connection request custom command can also start an active scan which is done to determine what devices are available in the neighborhood.

When a P2P connection request command is sent out for active scan purposes, the capability information and optional payload will not be attached. The receiving device uses the attachment, or absence of capability

information, and an optional payload to determine if the command is a request to establish a connection or just an active scan.

The MiWi P2P stack can enable or disable a device to allow other devices to establish connections. Once a device is disabled from making connections, any new P2P connection request will be discarded, except under the following conditions:

- The P2P connection request is coming from a device with which the receiving end already has had an established connection.
- The P2P connection request is an active scan.

The format of the P2P connection request command frame is shown in Figure 7.

**FIGURE 7: P2P CONNECTION REQUEST COMMAND FORMAT**

Octets	15/21	1	1	1 (Optional)	Various (Optional)
	MAC Header	Command Identifier (0x81)	Operating Channel	Capability Information	Optional payload to identify the node. It is not required for the stack, but may be useful for applications.



The operating channel is used to bypass the effect of subharmonics that may come from another channel. It will avoid the false connections with devices that operate on different channels.

The capability information byte, shown in Figure 7, is formatted as shown in Figure 8.

**FIGURE 8: CAPABILITY INFORMATION FORMAT**

Bits	0	1	2	3	4-7
	Receiver on when Idle	Will do Data Request Once Wake-up	Need Time Synchronization (Reserved)	Security Capable	(Reserved)

The P2P connection request's optional payload is provided for specific applications. A device may need additional information to identify itself – either its unique identifier or information about its capabilities in the application. With the optional payload, no additional packets are required to introduce or identify the device after the connection is established.

The optional payload will not be used in the stack itself.

#### P2P CONNECTION REMOVAL REQUEST

The P2P connection removal request (0x82) is sent to the other end of the connection to remove the P2P connection. The request's format is shown in Figure 9.

**FIGURE 9: P2P CONNECTION REMOVAL REQUEST FORMAT**

Octets	15/21	1
	MAC Header: Send to the other end of the P2P connection to cut the communication	Command Identifier (0x82)

#### DATA REQUEST

The data request (0x83) command is the same as the IEEE 802.15.4 specification's data request (0x04) command. Its format is shown in Figure 10.

If one side of a P2P connection node is able to Sleep when Idle, and that node could receive a message while in Sleep, the always active side of the connection

must store the message in its RAM. The always active side delivers the message when the sleeping device wakes up and requests the message.

If an application involves such conditions, the feature, `ENABLE_INDIRECT_MESSAGE`, needs to be activated. The sleeping node must send the data request command after it wakes up.

**FIGURE 10: DATA REQUEST FORMAT**

Octets	21	1
	MAC Header: Unicast from extended source address to extended destination address	Command Identifier (0x83 or 0x04)

## CHANNEL HOPPING

The channel hopping command (0x84) requests the destination device to change the operating channel to another one. The command's format is shown in Figure 11.

This command is usually sent out by the frequency agility initiator, which decides when to change channels and which channel to select.

This command usually is broadcasted to notify all devices, with their radios on when Idle, to switch channels. To ensure that every device receives this message, the frequency agility initiator will broadcast three times and all FFD devices will rebroadcast this message.

When the channel hopping sequence is carried out and all FFDs hop to a new channel, RFDs have to perform resynchronization to restore connection to their respective FFD peers.

**FIGURE 11: CHANNEL HOPPING FORMAT**

Octets	15/21	1	1	1
	MAC Header: Broadcast or unicast from the Frequency Agility Starter	Command Identifier (0x84)	Current Operating Channel	Destination Channel to Jump to

## P2P CONNECTION RESPONSE

The P2P connection response (0x91) command is used to respond to the P2P connection request. The command's format is shown in Figure 12.

The P2P connection response command can be used to establish a connection. Alternately, the command can be used by a device responding to an active scan, identifying itself as active in the neighborhood.

If the P2P connection request command that was received had a capability information byte and an optional payload attached, it is requesting a connection. The capability information and optional payload, if any, would be attached to the P2P connection response.

Once the response is received by the other end of the connection, a P2P connection is established. The two ends of the connection now can exchange packets.

If the P2P connection request command received did not have a capability information byte and optional payload, the command is an active scan. The P2P connection response, therefore, would have no capability information or optional payload attached.

In the case of the active scan connection request, no connection would be established after the message exchange.

**FIGURE 12: P2P CONNECTION RESPONSE FORMAT**

Octets	21	1	1	1 (Optional)	Various (Optional)
	MAC Header: Unicast from extended source address to extended destination address.	Command Identifier (0x91)	Status. 0x00 means successful. All other values are error codes.	Capability Information	Optional payload to identify the node. Not required for the stack, but possibly useful for applications.

The format of the response's capability information is shown in Figure 8.

The optional payload is provided for specific applications. Its format and usage is the same as the optional payload attached to P2P connection request command (see Page 9).

## P2P CONNECTION REMOVAL RESPONSE

The P2P connection removal response command (0x92) is used to respond to the P2P connection removal request. It notifies the other end of the P2P connection that a P2P connection request had been received early and whether the resulting connection has been removed.

The command's format is shown in Figure 13.

**FIGURE 13: P2P CONNECTION REMOVAL RESPONSE FORMAT**

Octets	21	1	1
	MAC Header: Unicast from extended source address to extended destination address	Command Identifier (0x92)	Status. <ul style="list-style-type: none"> <li>• 0x00 means successful.</li> <li>• All other values are error codes</li> </ul>

## MIWI™ P2P WIRELESS PROTOCOL'S UNIQUE FEATURES

The MiWi P2P protocol supports a reduced functionality, point-to-point, direct connection and a rich set of features. All features can be enabled or disabled and compiled in and out of the stack, according to the needs of the wireless application.

Microchip's ZENA™ software application can facilitate configuration of the stack and generation of the header file. For more information, see the "*ZENA™ Wireless Network Analyzer User's Guide*" (DS51606).

This section describes the unique features of the MiWi P2P protocol. They include:

- Small programming size
- Support for Idle devices to turn off radio
- Indirect messaging
- Special security features
- Active scan for finding existing PANs on different channels
- Energy scan for finding the channel with the least noise
- Frequency agility (channel hopping)

### Small Programming Size

To address many wireless applications' cost constraints, the MiWi P2P stack is as small as possible. Enabling the stack to target the smallest programming size can reduce the code size to be just over 3 Kbytes. A simple application could easily fit into a microcontroller with only 4 Kbytes of programming memory.

To activate this feature, "TARGET\_SMALL" must be defined in the file, `P2PDefs.h`. The file can be generated automatically by the ZENA software or created manually by an experienced programmer.

The feature supports bidirectional communication between devices, but communication between PANs is disabled. If the security feature is used, the freshness check will be disabled. (For more details about the freshness check, see "**Security Features**" on Page 12.)

### Idle Devices Turning Off Radios

For those devices operating on batteries, reducing power consumption is essential. This can be done by having the devices turn off their radios when they are not transmitting data. The MiWi P2P stack includes features for putting radios into Sleep and waking them up.

To activate this feature, "ENABLE\_SLEEP" must be defined in the file, `P2PDefs.h`. The file can be generated automatically by the ZENA software or created manually by an experienced user.

The decision as to when a device is put into Sleep is made by the specific application. Possible triggers could include:

- Length of radio Idle time
- Receipt of a packet from a connected FFD, requesting the device to go to Sleep

The conditions for awakening a device can also be decided by the specific application. Possible triggers include:

- An external event, such as a button being pressed
- Expiration of a predefined timer

While a device is sleeping, its peer device may need to send it a message. If no message needs to be sent, no additional feature must be enabled by the peer device.

If the peer device needs to send a message to the sleeping device, the peer device must store the message in its volatile memory until the sleeping device wakes up and acquires the message. Since the message is not being delivered directly to the sleeping device, this process is called an indirect message.

If an indirect message needs to be delivered, the peer device of the sleeping node needs to define "ENABLE\_INDIRECT\_MESSAGE" in the file, `P2PDefs.h`. The file can be generated automatically by the ZENA tool or created manually by the user.

If indirect messaging is enabled, there must be a specified maximum number of indirect messages that can be stored in the volatile memory. That message maximum depends on the free RAM memory available in the peer device and from the number of RFDs connected to the same parent FFD.

The maximum number of indirect messages is defined by "INDIRECT\_MESSAGE\_SIZE" in the file, `P2PDefs.h`. The file can be generated automatically by the ZENA software or created manually by the user.

For indirect messaging, the time-out period for the indirect messages also needs to be defined. If a time-out period was not defined and an RFD device was dead, the indirect message would remain forever in the volatile memory.

The indirect message time-out period is defined by "INDIRECT\_MESSAGE\_TIMEOUT" in the file, `P2PDefs.h`, with seconds as the unit of measurement.

Broadcasting may be useful for some applications, but it requires more effort for peer devices. When a peer device can broadcast a message to an RFD, "ENABLE\_BROADCAST" must be defined in the file, `P2PDefs.h`.

## Security Features

Wireless communication requires additional security considerations as a simple sniffer can intercept a message if its data is not encrypted. The MiWi P2P stack provides the IEEE 802.15.4 specification's seven security modes to fit a variety of security needs. The modes can be categorized into three groups:

- AES-CTR mode – Encrypts the message with a 16-byte security key. The mode has no built-in message integrity or frame freshness check, so it is susceptible to repeated attacks.

- AES-CBC-MAC modes – Ensures the integrity of the message with 4/8/16 bytes of Message Integrity Code (MIC) attached to each packet. This assures that the packet, including header and payload, has not been modified during transmission. The payload, however, is not encrypted, so the message is exposed.

The size of the MIC is determined by the particular mode, with the larger MICs providing stronger protection.

- AES-CCM modes – Combines the previous two security modes to ensure both the secrecy and integrity of the message.

The specification's seven security modes are described in Table 4.

**TABLE 4: IEEE 802.15.4™ SECURITY MODES**

Security Mode		Security Services				MIC (Bytes)
Identifier	Name	Access Control	Data Encryption	Message Integrity	Sequential Freshness	
01h	AES-CTR	X	X	—	X	0
02h	AES-CCM-128	X	X	X	X	16
03h	AES-CCM-64	X	X	X	X	8
04h	AES-CCM-32	X	X	X	X	4
05h	AES-CBC-MAC-128	X	—	X	—	16
06h	AES-CBC-MAC-64	X	—	X	—	8
07h	AES-CBC-MAC-32	X	—	X	—	4

To activate the security feature, "ENABLE\_SECURITY" must be defined in the file, `P2PDefs.h`. That file is generated automatically by the ZENA tool or can be manually produced.

When security is activated, additional elements must be defined in file, `P2PDefs.h`. Those elements are the 16-byte security key, key sequence number and the security level. The ZENA software assists with inputting these items and writes them to the file. Alternately, an expert user could manually define these elements.

Besides encrypting and decrypting the data, the security module checks the freshness of the message using the frame counter. If the value of a message's frame

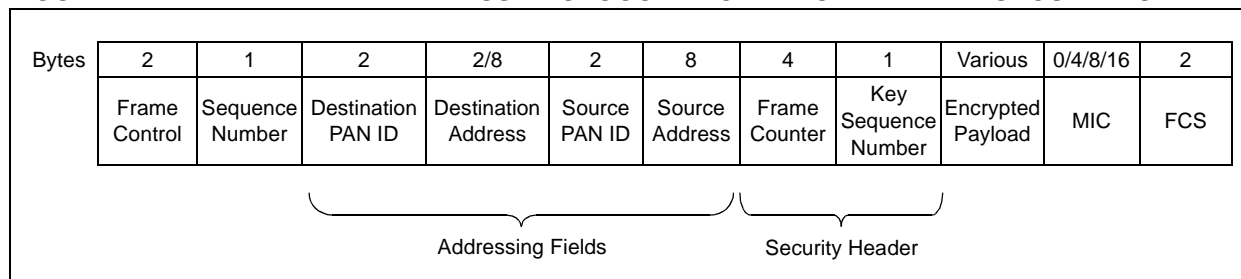
counter is lower than the value of record, the message will be discarded. This feature can prevent repeated attacks.

When a smaller programming size is preferred, the checking of the message's freshness can be disabled. This is done by defining the field `TARGET_SMALL` in the file, `P2PDefs.h`.

When security is enabled, an additional security header is added before the standard MAC header. This header contains a four-byte frame counter and a one-byte security key sequence number.

The packet format, when security is enabled, is illustrated in Figure 14.

**FIGURE 14: MIWI™ P2P WIRELESS PROTOCOL PACKET FORMAT WITH SECURITY ON**



## Active Scan

Active scan is the process of acquiring information about the local Personal Area Network (PAN). The active scan determines:

- The device's operating channel
- The device's signal strength in the PAN
- The PAN's identifier code

Active scan is particularly useful if there is no predefined channel or PAN ID for the local devices.

The maximum number of PANs that an active scan can acquire is defined, in the stack, as `ACTIVE_SCAN_RESULT_SIZE`.

The scan duration and channels to be scanned are determined before the active scan begins.

The scan duration is defined by the IEEE 802.15.4 specification and its length of time, measured in symbols, is calculated with the formula shown in Equation 1. (One second equals 62,500 symbols.)

### EQUATION 1:

$$\text{Scan Time Period} \equiv 60 \cdot (2^{\text{ScanDuration}} + 1)$$

**Note:** ScanDuration = The user-designated input parameter for the scan. An interger is from 1 to 14.

A scan duration of 10 would result in a scan time period of 61,500 symbols or about 1 second. A scan duration of 9 is about half second.

The scan channels are defined by a bitmap with each channel number represented by its comparable bit number in the double word. Channel 11 would be b'0000 0000 0000 0000 0000 1000 0000 0000. Channels 11 to 26, supported in the 2.4 GHz spectrum, would be b'0000 0111 1111 1111 1111 1000 0000 0000 or 0x07FFF800.

When an active scan broadcasts a P2P connection request command, it expects any device in radio range to answer with a P2P connection response command. The active scan will determine only what PANs are available in the neighborhood, not how many individual devices are available for new connections. That is because every device responds to the scan – even those that will not allow new connections.

To activate the active scan feature, "ENABLE\_ACTIVE\_SCAN" must be defined in the file, `P2PDefs.h`.

## Energy Scan

The IEEE 802.15.4 specification defines 16 channels in the 2.4 GHz spectrum, but a PAN must operate on one. The best channel to use is the one with the least amount of energy or noise.

Energy scan is used to scan all available channels and determine the channel with the least noise.

The scan duration and channels to be scanned are determined before the energy scan is performed.

The scan duration is defined by the IEEE 802.15.4 specification and its length of time, measured in symbols, is calculated with the formula shown in Equation 1.

See "Active Scan" on Page 13 for an explanation of the measurement.

After the scan is complete, the channel identifier with the least noise will be returned.

To activate the Energy Scan feature, "ENABLE\_ENERGY\_SCAN" must be defined in the file, `P2PDefs.h`.

## Frequency Agility

Frequency agility enables the MiWi P2P PAN to move to a different channel if operating conditions so require.

In implementing this feature, the affected devices fall into one of two roles:

- Frequency agility initiators – The devices that decide whether channel hopping is necessary and which new channel to use
- Frequency agility followers – Devices that change to another channel when so directed

### FREQUENCY AGILITY INITIATORS

Each PAN can have one or more devices as a frequency agility initiator; an initiator must be an FFD.

Each initiator must have the energy scanning feature enabled. That is because the initiator must do an energy scan to determine the optimal channel for the hop. Then, the initiator broadcasts a channel hopping command to the other devices on the PAN.

### FREQUENCY AGILITY FOLLOWERS

A frequency agility follower can be an FFD or an RFD device.

The FFD makes the channel hop by doing one of the following:

- Receiving the channel hopping command from the initiator
- Resynchronizing the connection, if data transmissions fail continuously

An RFD device makes the hop using the resynchronization method – reconnecting to the PAN when communication fails.

## IMPLEMENTING, ACTIVATING FEATURE

When to perform a frequency agility operation is decided by the application. The MiWi P2P stack, however, provides two global variables to help the application make that decision.

- “CCAFailureTimes” – Defines the number of continuous data transmission failures due to Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) failure.

This condition usually means there is excessive noise on the current channel.

- “AckFailureTimes” – Defines the number of continuous data transmission failures due to no Acknowledgement.

A large AckFailureTimes usually means there is a high noise level at the receiving end.

Alternatively, it means the destination device is not available because it is malfunctioning or has hopped to another channel. If the destination device has changed channels, a resynchronization operation is necessary.

To activate the frequency agility feature, “ENABLE\_FREQUENCY\_AGILITY” must be defined in the file, P2PDefs.h. To enable devices to act as frequency agility initiators, “FREQUENCY\_AGILITY\_STARTER” must be defined in that file.

## APPLICATION PROGRAMMING INTERFACES (APIs)

The MiWi™ P2P Wireless Protocol stack provides application developers with a set of easy-to-use APIs. Simple P2P applications may require less than 30 lines of code in the application layer.

The stack also gives the flexibility of supporting a rich feature by expanding the application layer with additional API calls.

The APIs are listed – alphabetized by function name – in the “**List of Application Programming Interfaces (APIs)**” on Page 21. This list only gives each API's basic functional characteristics.

This section gives more detailed information on each API, grouping them by the following basic processes:

<b>Creating a Peer-to-Peer Connection</b> .....	16
<b>Receiving a Packet</b> .....	17
<b>Transmitting a Packet</b> .....	18
<b>Doing an Active Scan</b> .....	19
<b>Doing an Energy Scan</b> .....	19
<b>Using Frequency Agility</b> .....	20

### Simple P2P Application

Example 1 gives code for a simple P2P application. The code establishes a connection, receives a packet and transmits a packet.

Example 1 displays the key portion of the MiWi P2P programming interface. In this document's following sections, this sample code will be used to demonstrate how the application layer uses the stack to interconnect with a peer device.

#### EXAMPLE 1: SIMPLE P2P APPLICATION

1.	void main(void)
2.	{
3.	Initiazation();// Initialize hardware platform and P2P stack
4.	SetChannel(myChannel);// set the channel to operate
5.	EnableNewConnection();// accept new P2P connection
6.	CreateNewConnection(2);// establish a connection with another device
7.	
8.	while(1)
9.	{
10.	if( ReceivedPacket() )// check if a packet has been received
11.	{
12.	LED_1 = rxFrame.PayLoad[0]     // processing the packet, setting the LED according to received packet
13.	DiscardPacket();   // discard the packet to ready to receive the next
14.	}
15.	else
16.	{
17.	BYTE PressedButton = ButtonPressed();     // check if a button pressed
18.	if (PressedButton)                         // if a button has been pressed
19.	{
20.	FlushTx();// reset the transmitting buffer
21.	for(i = 0; i < 65; i++)
22.	{
23.	WriteData(P2P[i]);// fill the transmitting buffer
24.	}
25.	
26.	UnicastConnection(0, FALSE, TRUE);     // unicast the packet
27.	}
28.	}
29.	}

## Creating a Peer-to-Peer Connection

This section gives some details on APIs that create peer-to-peer connections.

**Note:** All APIs are listed, alphabetized by function name, in the “**List of Application Programming Interfaces (APIs)**” on Page 21. That list gives only basic functional elements.

In Example 1, code lines 5-6 demonstrate the two ways to establish a connection

- Passively accept a new connection
- Actively pursue a new connection

To accept a new connection, the device:

1. Waits for the P2P connection request.
2. Responds with the P2P connection response command.

To pursue a new connection, the device:

1. Sends out a P2P connection request command to look for device with which to connect.
2. Receives the responding P2P connection response command to finish the connection.

Either way, after a new connection has been established, the information about the device on the other end of the connection will be stored in both devices' P2P connection entries. Those entries represent the existing connections and may be used to transmit the packet to the peer device of the connection. (For details, see “**Transmitting a Packet**” on Page 18.)

The APIs for making peer-to-peer connections follow.

### **void EnableNewConnection(void)**

The function call, `EnableNewConnection`, allows the stack to accept new connections by responding to a P2P connection request command with P2P connection response.

### **void DisableNewConnection(void)**

`DisableNewConnection` prevents the stack from accepting new connections by responding to a P2P connection request command. In the following two cases, this setting will be ignored, even if the new connection is disabled:

- The P2P connection request command is actually an active scan without any peer information attached.
- A P2P connection response command without peer information attached will be sent out to respond.
- The P2P connection request command is coming from a device that already has a connection with the current device.
- This feature is usually used when one end of the connection is used to re-establish the old connections after power-down.

### **BYTE CreateNewConnection(BYTE RetryInterval)/BYTE CreateNewConnection(BYTE RetryInterval, DWORD ChannelMap)**

The function call, `CreateNewConnection`, actively pursues a new connection. Before the function, there is no need to enable the stack to accept new connections. The function enables them automatically.

When the function call returns, the status for accepting new connections will be set back to its previous state.

This function call proceeds as follows:

1. The P2P connection request command is sent out with peer information attached.
2. The stack waits for the P2P connection response command to establish a connection.
3. When the first connection is made, the function call returns the index of the connection in the P2P connection entry.

There are two formats of the function, `CreateNewConnection`.

- When the energy scan feature is not activated, the only parameter for this function call is the interval (in seconds) to send out the P2P connection requests.
- When energy scan is activated, there is an additional parameter, `ChannelMap`, that specifies the channels to scan to find the new connection.

The double-word parameter of `ChannelMap` uses a bitmap to specify the channels to scan. Each channel number is represented by its comparable bit number in the double word. Channel 11 would be `b'0000 1000 0000 0000`. In order to scan all channels in the 2.4 GHz band (11 to 26), the `ChannelMap` should be set to `0x07FFF800`.



## Receiving a Packet

This section gives details on the received packet API.

**Note:** All APIs are listed, alphabetized by function name, in the “List of Application Programming Interfaces (APIs)” on Page 21. That list gives only basic functional elements.

In Example 1, code lines 10-14 demonstrate how to receive and process a packet.

### Boolean ReceivedPacket(void)

The function, `ReceivedPacket`, returns a boolean expression indicating whether a packet has been received by the MiWi P2P stack. If the function is called, the whole stack is called internally before returning the status which indicates whether a packet has been received.

If a packet has been received by the stack, the return value of `ReceivedPacket` is `TRUE`. All information regarding the received packet will be stored in the global variable of `rxFrame`, a variable of the structure `RECEIVED_FRAME`. The definition of the structure, `RECEIVED_FRAME`, is given in Example 2.

### EXAMPLE 2: RECEIVED\_FRAME STRUCTURE

1.	<code>typedef struct _RECEIVED_FRAME</code>
2.	<code>{</code>
3.	<code>union _RECEIVED_FRAME_FLAG</code>
4.	<code>{</code>
5.	<code>    BYTE    Val;</code>
6.	<code>    struct _RECEIVED_FRAME_FLAG_BITS</code>
7.	<code>    {</code>
8.	<code>        BYTE    commandFrame    : 1; // data: 0; command: 1</code>
9.	<code>        BYTE    security        : 1;</code>
10.	<code>        BYTE    framePending    : 1;</code>
11.	<code>        BYTE    intraPAN        : 1;</code>
12.	<code>        BYTE    broadcast       : 1;</code>
13.	<code>    } bits;</code>
14.	<code>    } flags;</code>
15.	
16.	<code>#ifndef TARGET_SMALL</code>
17.	<code>    BYTE        PacketLQI;</code>
18.	<code>    BYTE        PacketRSSI;</code>
19.	<code>    WORD_VAL    SourcePANID;</code>
20.	<code>#endif</code>
21.	<code>    BYTE        SourceLongAddress[8];</code>
22.	<code>    BYTE        PayloadSize;</code>
23.	<code>    BYTE        *Payload;</code>
24.	<code>    } RECEIVED_FRAME;</code>

This structure shows that virtually all information regarding the received packet has been categorized and stored. The definition of `RECEIVED_FRAME` shows that:

- The parameters, `PayloadSize` and `Payload`, are for the MAC payload only. The MAC header is not included.
- If the stack is configured to use minimum programming space, some of the parameters, such as `PacketLQI`, `PacketRSSI` and `SourcePANID`, are not available.

Under such conditions, the signal quality and strength and inter-PAN communication capabilities are not supported.

- If a packet is encrypted during transmission, the information will be already decrypted in the `RECEIVED_PACKET` structure.

Whether a packet was encrypted originally can be found in the corresponding settings in the flags (`flags.bits.security`).

How a packet is handled after it is received is up to the specific application. After the packet has been processed and before it is returned to process the stack, the function, `DiscardPacket`, needs to be called. The function removes the current received packet and prepares for the next packet.

If the function, `DiscardPacket`, is not called, the old packet will occupy the space and the new packet cannot be received.

## Transmitting a Packet

This section gives details on the APIs associated with transmitting a packet.

**Note:** All APIs are listed, alphabetized by function name, in the “**List of Application Programming Interfaces (APIs)**” on Page 21. That list gives only basic functional elements.

In Example 1, code lines 20-26 demonstrate one method of transmitting a packet.

When transmitting a packet, the MAC header will be handled by the stack. The application layer only takes care of the MAC payload.

**void FlushTx(void)**

The function call, `FlushTx`, resets the transmitting buffer and prepares it to receive the information to be transferred.

**void WriteData(BYTE data)**

The function call, `WriteData`, will fill the transmitter buffer one byte at a time. If there are 20 bytes to be transmitted in the packet, the function will be called 20 times.

Once the transmitting buffer is filled, a packet is ready to be transmitted. There are three ways to transmit a packet:

- Unicast with an index of connections in the P2P connection entries
- Unicast with a long address
- Broadcast

**BOOL UnicastConnection(BYTE ConnectionIndex, BOOL isCommand, BOOL SecurityEnabled)**

In Example 1, line 26 uses the method of unicast with an index of connections through the function, `UnicastConnection`.

A record in the P2P connection entries contains information on the peer device of a P2P connection and the stack uses it to send the packet to its destination.

There are three parameters for the function:

- Index of the record in P2P connection entries
- Boolean expression indicating if the packet is a command packet.

As previously discussed, the MiWi P2P stack supports only the data and command frame, defined in the frame control bytes of the MAC header. (For more detail, see “**Message Format**” on Page 5 or the IEEE 802.15.4 specification.)

- Boolean expression indicating if encryption is required for the packet.

The security feature must be enabled or encryption will not be performed.

**BOOL UnicastLongAddress(WORD\_VAL DestinationPANID, BYTE \*DestinationAddress, BOOL isCommand, BOOL SecurityEnabled)**

A message can also be unicast by calling the function, `UnicastLongAddress`. By specifying the long address and the PAN identifier, the stack can determine the destination.

There are four parameters for this function:

- Destination PAN identifier
- Destination long address
- Boolean expression indicating if the packet is a command packet
- Boolean expression indicating if encryption is required

**BOOL BroadcastPacket(WORD\_VAL DestinationPANID, BOOL isCommand, BOOL SecurityEnabled)**

Unicast transmits a packet to a single destination. When a message needs to be sent to all devices in the radio range, broadcasting – using the function call, `BroadcastPacket`, is used.

There are three parameters for the function, `BroadcastPacket`:

- Destination PAN identifier  
If the broadcast targets all PANs, an application may choose to use the identifier, 0xFFFF, to remove the restriction of inter-PAN communication.
- Boolean expression to indicate if the packet is a command packet
- Boolean expression to indicate if encryption is required

## Doing an Active Scan

This section gives details on the Active Scan API.

**Note:** All APIs are listed, alphabetized by function name, in the “**List of Application Programming Interfaces (APIs)**” on Page 21. That list gives only basic functional elements.

It can be helpful to know if any MiWi P2P PANs are operating in the area, and if so, what their PAN identifiers are. This enables a PAN coordinator to choose a PAN to join, or start a new PAN with a unique PAN identifier.

To accomplish this, the MiWi P2P stack provides the function call, `ActiveScan`.

**BYTE ActiveScan(BYTE ScanDuration, DWORD ChannelMap)**

This function call is available only if the active scan feature has been activated.

There are two parameters for the function, `ActiveScan`:

- Duration of scan performed on each channel.  
The scan duration is defined by the IEEE 802.15.4 specification. For details, see “**Active Scan**” on Page 13.
- Channel map specifying the channels to be scanned.  
For details on the channel map setting, see “**Active Scan**” on Page 13.

The `ActiveScan` function call returns the total number of PANs found. All the scan results are stored in the global variable, `ActiveScanResults`, of the structure, `ACTIVE_SCAN_RESULT`. The structure of `ACTIVE_SCAN_RESULT` is defined as shown in Example 3.

### EXAMPLE 3: ACTIVE\_SCAN\_RESULT

1.	<code>typedef struct _ACTIVE_SCAN_RESULT</code>
2.	<code>{</code>
3.	<code>    BYTE                Channel;</code>
4.	<code>    BYTE                RSSIValue;</code>
5.	<code>    WORD_VAL            PANID;</code>
6.	<code>    } ACTIVE_SCAN_RESULT;</code>

As shown, the active scan result provides the channel number, signal strength and identifier for the PAN.

The maximum number of PANs that can be acquired is defined as `ACTIVE_SCAN_RESULT_SIZE` in the stack. As RAM resources permit, application developers can modify this setting.

The default value is 16.

## Doing an Energy Scan

This section gives details on the Energy Scan API.

**Note:** All APIs are listed, alphabetized by function name, in the “**List of Application Programming Interfaces (APIs)**” on Page 21. That list gives only basic functional elements.

There are 16 available channels in the 2.4 GHz band. Since the MiWi P2P PAN can operate on any of them, it can search for the channel with the least noise.

The energy scan feature does this. To call the programming interface to operate the energy scan, the energy scan feature must be activated.

**BYTE OptimalChannel(BYTE ScanDuration, DWORD ChannelMap, BYTE \*RSSIValue)**

The function, `OptimalChannel`, needs three parameters:

- Duration of scan on each channel.  
The definition of scan duration follows the IEEE 802.15.4 specification. For more detail, see “**Energy Scan**” on Page 13.
- Channel map specifying the channels to be scanned.  
For details on the channel map setting, see “**Energy Scan**” on Page 13.
- `RSSIValue` – The output parameter that returns the maximum energy reading on the optimal channel that returns.

`OptimalChannel` returns the channel that has the least amount of noise during the scan.

## Using Frequency Agility

The frequency agility feature makes the MiWi P2P stack compatible with the wireless environment. Sometimes called channel hopping, this feature enables the MiWi P2P PAN to migrate across channels to avoid the varying noise levels in the 2.4 GHz band.

Devices on a MiWi P2P PAN have one of two frequency/agility roles: starter or follower. The frequency agility starter starts the channel hopping process by calling the function, `InitChannelHopping`.

### **BOOL InitChannelHopping** (**DWORD ChannelMap**)

This function needs one parameter: the bitmap of the channels to which the PAN may move. For details on this setting, see “**Frequency Agility**” on Page 14.

`InitChannelHopping` will return a boolean expression to indicate if the operation is successful. If, after checking other channels, the current channel is found to have the least amount of noise, the initialization process is stopped and the value `FALSE` is returned.

If the channel is changed, frequency/agility followers either hop to the new channel by receiving the request from the frequency agility starter or by resynchronizing their connection.

To resynchronize the connection, frequency/agility followers call the function, `ResyncConnection`.

### **BOOL ResyncConnection** (**BYTE \*DestinationAddress**, **DWORD ChannelMap**)

The function, `ResyncConnection`, has two parameters:

- Destination address – The pointer to the long address of the device with which the frequency/agility followers will resynchronize. Usually, this address is that of the node with which the frequency agility follower lost connection.
- ChannelMap – The bitmap of the available new channels.

For more information, see “**Frequency Agility**” on Page 14.

`ResyncConnection` will return a boolean expression to indicate if the operation is successful. If a resynchronized connection is tried unsuccessfully three times on each channel, the function call will return `FALSE`.

## List of Application Programming Interfaces (APIs)

### ActiveScan

This function performs an active scan to find all nearby MiWi P2P PANs.

#### Syntax

```
BYTE ActiveScan(BYTE ScanDuration, DWORD ChannelMap)
```

#### Inputs

- **BYTE – ScanDuration:** The time period for the energy scan on each channel.
- **DWORD – ChannelMap:** The bitmap of the channels for the energy scan.

#### Outputs

**BYTE** – The number of existing MiWi P2P PANs.

#### Notes

The scan result will be stored in the global variable array, `ActiveScanResults`, with the maximum result size of `ACTIVE_SCAN_RESULT_SIZE`.

### BroadcastPacket

This function broadcasts a message to all devices with a destination PAN identifier in the radio range.

#### Syntax

```
BOOL BroadcastPacket(WORD_VAL DestinationPANID, BOOL isCommand, BOOL SecurityEnabled)
```

#### Inputs

- **WORD\_VAL – DestinationPANID:** The PAN identifier of the destination devices.
- **BOOL – isCommand:** The boolean expression to indicate if the transmitting packet is a command.
- **BOOL – SecurityEnabled:** The boolean expression to indicate if the transmitting packet needs encryption.

#### Outputs

**BOOL** – The boolean expression to indicate if the operation is successful.

#### Notes

The message payload already should have been filled before calling this function.

### CreateNewConnection

This function actively pursues a new P2P connection.

#### Syntax

```
BYTE CreateNewConnection(BYTE RetryInterval)  
BYTE CreateNewConnection(BYTE RetryInterval, DWORD ChannelMap)
```

#### Inputs

- **BYTE – RetryInterval:** The time interval, in seconds, between each attempt to request a new connection.
- **DWORD – ChannelMap:** The bitmap of the channels to be tried for establishing a connection. This requires the energy scan feature to be enabled.

#### Outputs

**BYTE** – The index of the new connection in the P2P connection entries.

#### Notes

There is a return only if a new P2P connection is established.

## **DisableNewConnection**

This function disables the stack to accept the new connection.

### **Syntax**

```
void DisableNewConnection(void)
```

### **Inputs**

None

### **Outputs**

None

### **Notes**

After the stack is disabled to accept the new connection, a response is sent if the request is from a device that already is connected or if the request is an active scan.

## **DisableAcknowledgement**

This function disables the stack to request MAC Acknowledgement for each unicast packet.

### **Syntax**

```
void DisableAcknowledgement(void)
```

### **Inputs**

None

### **Outputs**

None

### **Notes**

None

## **DiscardPacket**

This function notifies the stack to discard the current received packet and prepare to receive the next packet.

### **Syntax**

```
void DiscardPacket(void)
```

### **Inputs**

None

### **Outputs**

None

### **Notes**

If this function is not called after processing the current packet, the stack will never be able to receive the next packet.

## **DumpConnection**

This function prints out the information regarding the connection(s) on the HyperTerminal.

### **Syntax**

```
void DumpConnection(BYTE index)
```

### **Inputs**

BYTE Index: The index of the connection in the P2P connection entries. (The value, 0xFF, represents all connections.)

### **Outputs**

None

### **Notes**

This function is usually used during development.

## **EnableAcknowledgement**

This function enables the stack to request MAC Acknowledgement for each unicast packet.

### **Syntax**

```
void EnableAcknowledgement(void)
```

### **Inputs**

None

### **Outputs**

None

### **Notes**

None

## **EnableNewConnection**

This function enables the stack to begin accepting new connections.

### **Syntax**

```
void EnableNewConnection(void)
```

### **Inputs**

None

### **Outputs**

None

### **Notes**

None

## **FlushTx**

This function resets the transmitting buffer.

### **Syntax**

```
void FlushTx(void)
```

### **Inputs**

None

### **Outputs**

None

### **Notes**

This function should be called when the applications starts, before filling the transmitting buffer.

## **InitChannelHopping**

The function is called by a frequency agility starter to begin the process of frequency agility (channel hopping).

### **Syntax**

```
BOOL InitChannelHopping( DWORD ChannelMap)
```

### **Inputs**

DWORD ChannelMap: The bitmap of the candidate channels that the PAN could hop to.

### **Outputs**

BOOL – The boolean expression that indicates the channel hopping request was successful.

### **Notes**

If the current operating channel has the least noise, the channel hopping operation will not be performed and the return value will be FALSE.

## **MRF24J40Sleep**

This function puts the MRF24J40 radio into Sleep mode.

### **Syntax**

```
void MRF24J40Sleep(void)
```

### **Inputs**

None

### **Outputs**

None

### **Notes**

None



## **MRF24J40Wake**

This function wakes the MRF24J40 radio from Sleep mode.

### **Syntax**

```
void MRF24J40Wake(void)
```

### **Inputs**

None

### **Outputs**

None

### **Notes**

None

## **OptimalChannel**

This function scans and finds the channel with the least amount of noise.

### **Syntax**

```
BYTE OptimalChannel(BYTE ScanDuration, DWORD ChannelMap, BYTE *RSSIValue)
```

### **Inputs**

- **BYTE – ScanDuration:** The time period allotted for the energy scan of each channel.
- **DWORD – ChannelMap:** The bitmap of the channels to be subject to the energy scan.

### **Outputs**

- **BYTE – RSSIValue:** The pointer to the energy level for the optimal channel.
- **BYTE –** The optimal channel with the least noise.

### **Notes**

None

## **P2PInit**

This function initializes the MiWi P2P stack.

### **Syntax**

```
void P2PInit(void)
```

### **Inputs**

None

### **Outputs**

None

### **Notes**

None

## ReceivedPacket

This function calls the MiWi P2P stack and checks if a packet has been received.

### Syntax

```
BOOL ReceivedPacket(void)
```

### Inputs

None

### Outputs

BOOL – The boolean expression that indicates a packet has been received by the MiWi P2P stack.

### Notes

None

## ResyncConnection

### Syntax

```
BOOL ResyncConnection(BYTE *DestinationAddress, DWORD ChannelMap)
```

### Inputs

- BYTE – DestinationAddress: The pointer to the long destination address of the peer device in the connection to be resynchronized.
- DWORD – ChannelMap: The bitmap of channels eligible to do the resynchronization.

### Outputs

BOOL – The boolean expression that indicates the resynchronization operation was successful.

### Notes

None

## SetChannel

This function sets the device to operate on one of the 16 channels available in the 2.4 GHz band.

### Syntax

```
void SetChannel(BYTE channel)
```

### Inputs

BYTE – Channel: The channel currently used by a device.

### Outputs

None

### Notes

None

## UnicastConnection

This function unicasts a message to the peer device of the connection in the P2P connection entries field.

### Syntax

```
BOOL UnicastConnection(BYTE ConnectionIndex, BOOL isCommand, BOOL SecurityEnabled)
```

### Inputs

- **BYTE – ConnectionIndex:** The index of the peer device in the P2P connection entries field.
- **BOOL – isCommand:** The boolean expression that indicates a packet is a command packet.
- **BOOL – SecurityEnabled:** The boolean expression that indicates if a packet needs encryption.

### Outputs

**BOOL –** The return to the boolean expression indicating the unicast operation was successful.

### Notes

None

## UnicastLongAddress

This function unicasts a message to the device with the input long address.

### Syntax

```
BOOL UnicastLongAddress(WORD_VAL DestinationPANID, BYTE *DestinationAddress, BOOL isCommand, BOOL SecurityEnabled)
```

### Inputs

- **WORD\_VAL – DestinationPANID:** The PAN identifier of the destination device.
- **BYTE \* – DestinationAddress:** The pointer to the 8-byte long address of the destination device.
- **BOOL – isCommand:** The boolean expression indicating a packet is a command packet.
- **BOOL – SecurityEnabled:** The boolean expression that indicates a packet needs encryption.

### Outputs

**BOOL –** Return the boolean expression indicating this operation was successful.

### Notes

None

## WriteData

This function writes one byte of data to the transmitting buffer.

### Syntax

```
void WriteData(BYTE data)
```

### Inputs

**BYTE – Data:** The data write to the transmitting buffer.

### Outputs

None

### Notes

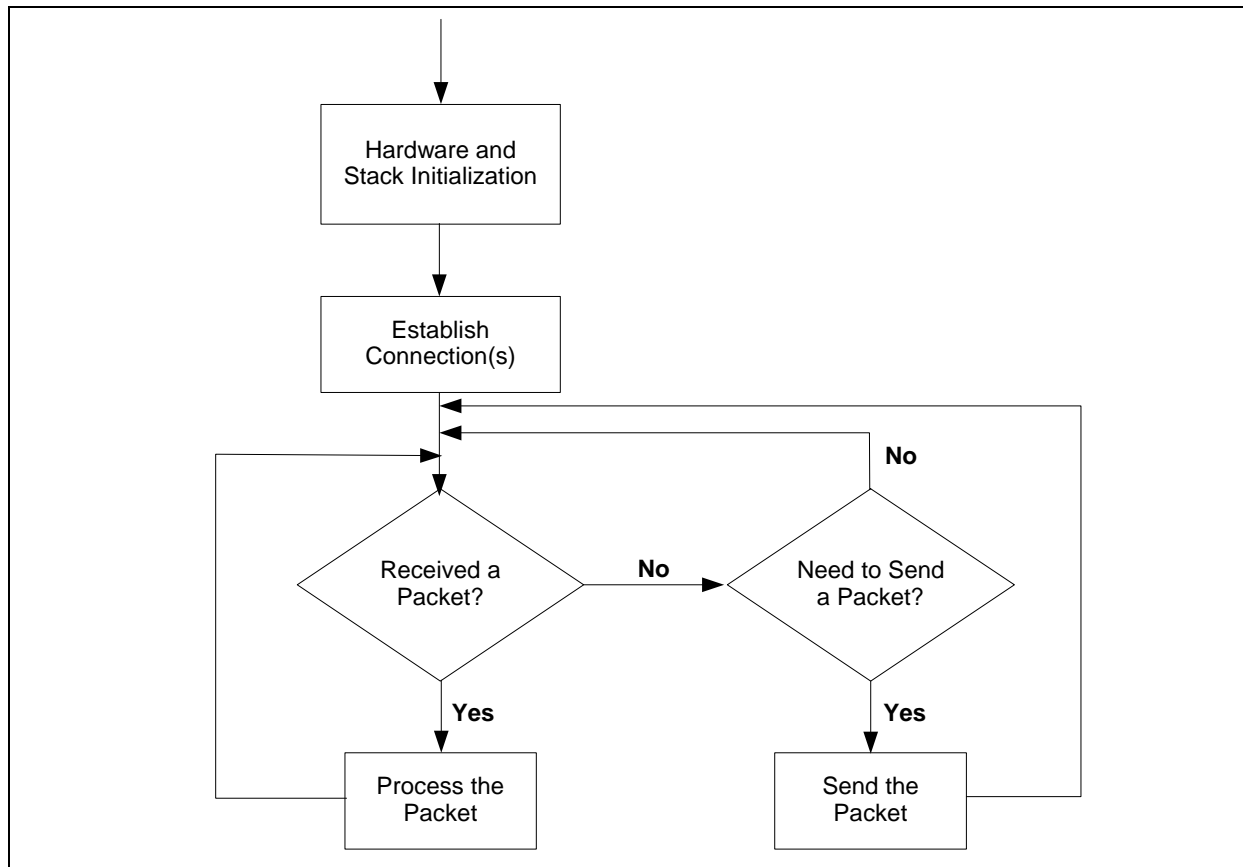
This function should be called only after the `FlushTx` function.

## APPLICATION FLOWCHART

A typical MiWi P2P application starts by initializing the hardware and MiWi P2P stack. Then, it tries to establish a connection and enter the normal operation mode of receiving and transmitting data.

Figure 15 gives the typical flow of the MiWi P2P applications.

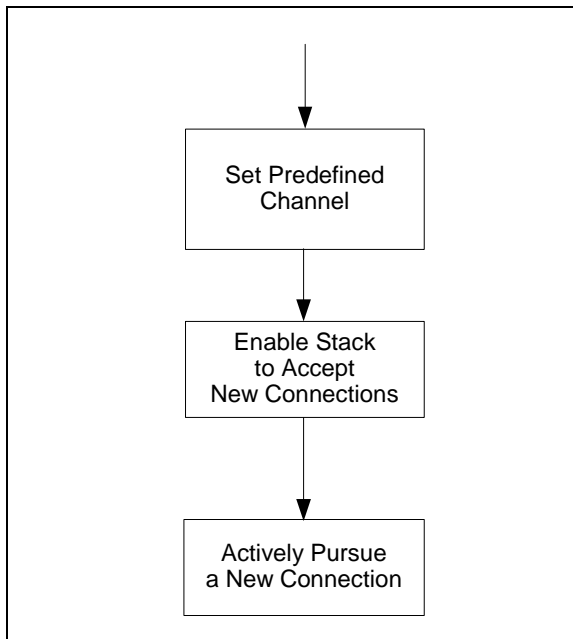
**FIGURE 15: FLOWCHART FOR MIWI™ P2P WIRELESS PROTOCOL APPLICATIONS**



After a connection is established, the procedures for most MiWi P2P applications will be the same. Any variation — due to different stack configurations — takes place during the establishment of connections.

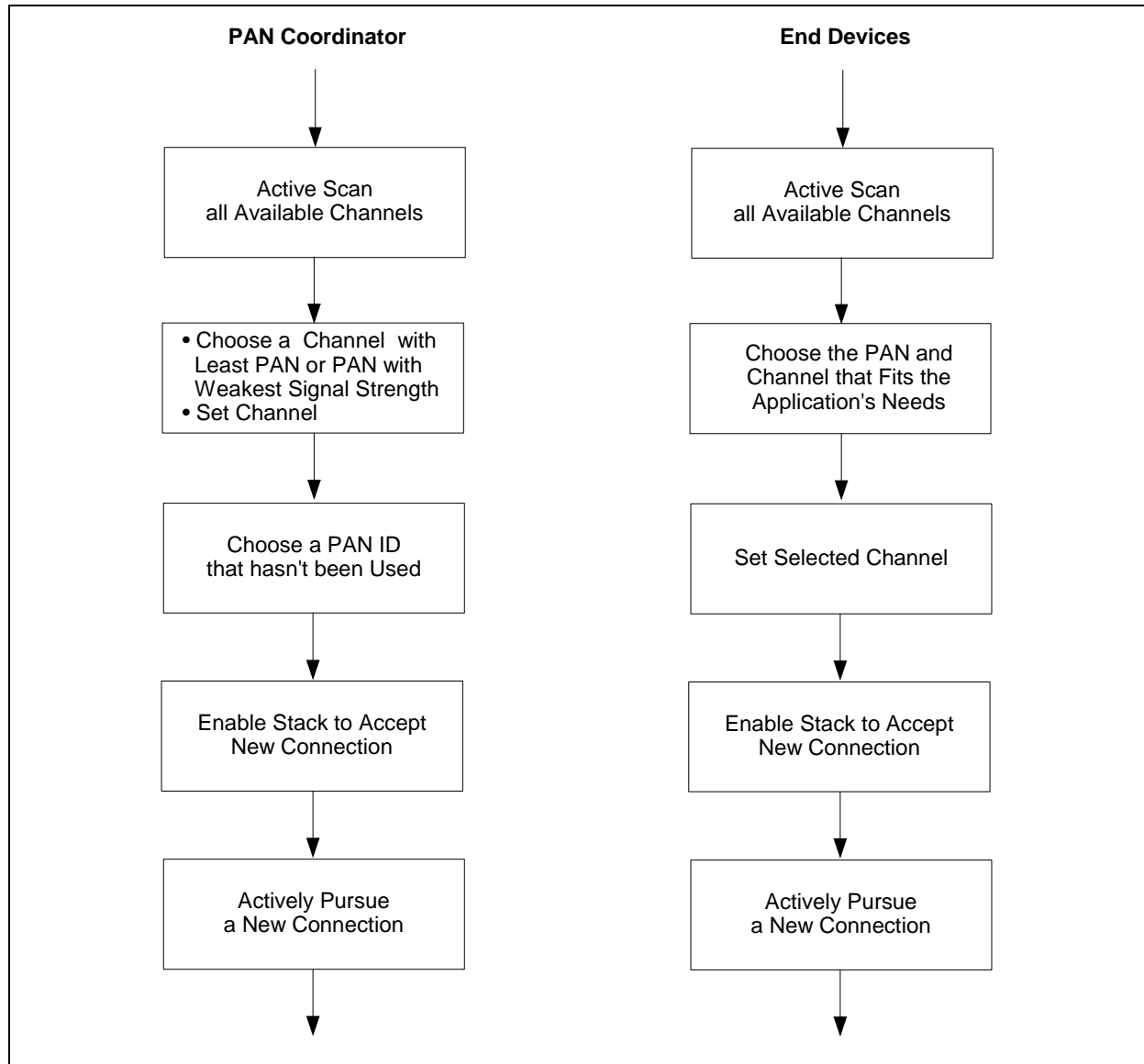
The simplest P2P connection application for establishing connections is shown in Figure 16.

**FIGURE 16: FLOWCHART TO ESTABLISH CONNECTION(S) IN SIMPLE MODE**



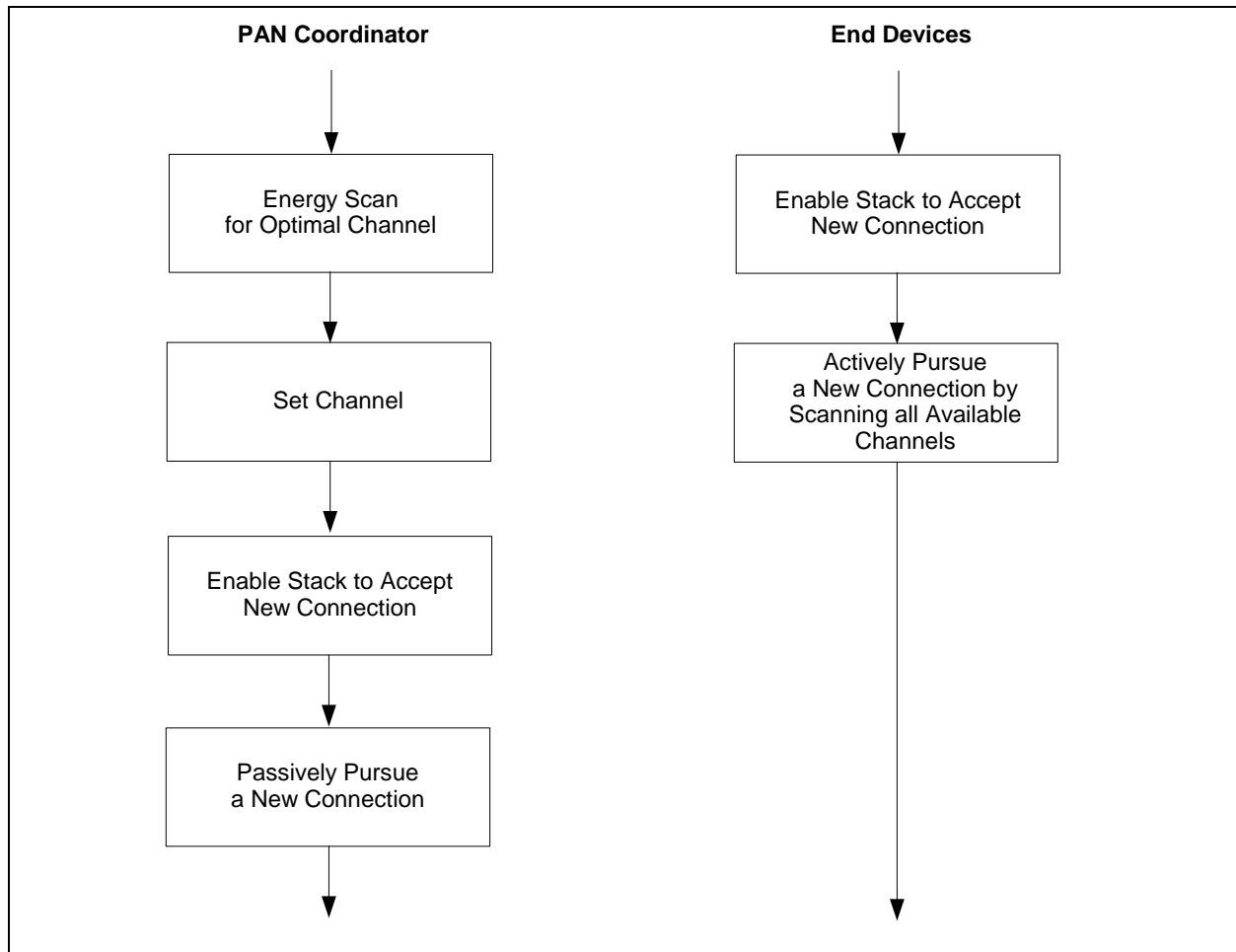
In more complex applications that require active scan capability, the steps for establishing connections differ between the PAN coordinator and end devices. Figure 17 shows the active scan steps for both categories of devices.

**FIGURE 17: FLOWCHART TO ESTABLISH CONNECTIONS WHEN ACTIVE SCAN IS ENABLED**



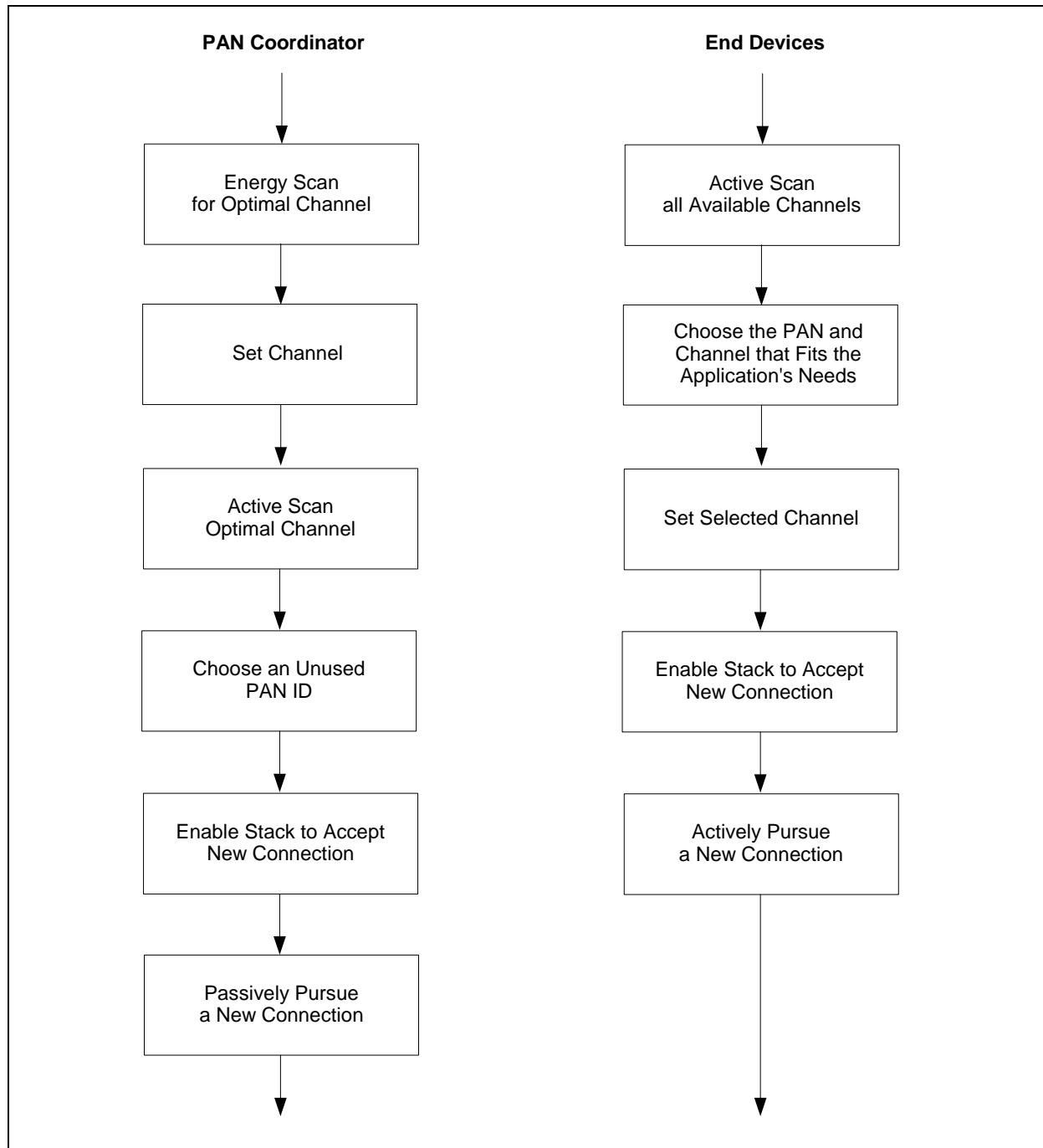
For applications with energy scan enabled, the steps after connection also are different for the PAN coordinator and end devices (see Figure 18).

**FIGURE 18: FLOWCHART TO ESTABLISH CONNECTIONS WHEN ENERGY SCAN IS ENABLED**



The process for establishing connections – with both active scan and energy scan enabled – is shown in Figure 19.

**FIGURE 19: FLOWCHART TO ESTABLISH CONNECTIONS WITH ACTIVE AND ENERGY SCAN**





## SYSTEM RESOURCES REQUIREMENT

The MiWi™ P2P Wireless Protocol stack has a rich set of features. Enabling a feature set will increase the system requirements for the microcontrollers.

Table 5 gives the requirements of a basic configuration.

**TABLE 5: PIC18 MEMORY REQUIREMENTS FOR MIWI™ P2P WIRELESS PROTOCOL BASIC STACK**

Configuration	Program Memory (Bytes)	RAM (Bytes)
Target Small Stack Size	3,336	100 + RX Buffer Size + TX Buffer Size + (9 * P2P Connection Size)

Additional MiWi P2P features require more program memory and RAM. Table 6 lists the system requirements for features above a basic configuration.

**TABLE 6: PIC18 MEMORY REQUIREMENTS FOR MIWI™ P2P WIRELESS PROTOCOL STACK FEATURES†**

Configuration	Additional Program Memory (Bytes)	Additional RAM (Bytes)
Enable Intra-PAN Communication	462	0
Enable Sleep	186	0
Enable Security (Without Frame Freshness Checking)	500	48
Enable Security (With Frame Freshness Checking)	1,488	54
Enable Active Scan	1,070	69
Enable Energy Scan	752	0
Enable Indirect Message	950	Indirect Message Size * TX Buffer Size
Enable Indirect Message with Capability of Broadcasting	1,228	Indirect Message Size * TX Buffer Size

† These requirements are for the PIC18 family of microcontrollers. The stack also supports PIC24, dsPIC33 and PIC32 microcontrollers, but those devices' requirements may vary.

These requirements are for the initial release of the stack and are subject to change.

## CONCLUSION

For wireless applications that require a star or peer-to-peer topology, the MiWi™ P2P Wireless Protocol is a good solution. The stack provides all the benefits of the IEEE 802.15.4 specification with a simple, yet robust, solution.

If an application is more complex, the Microchip MiWi™ Net stack should be considered. That stack provides support for a real network with up to 1,024 active nodes across as many as four hops. For details on this protocol, see *AN1066, "MiWi™ Wireless Networking Protocol Stack"* (DS1066).

For an even more complex network or interoperability, Microchip's implementation of the ZigBee protocol specification is an option. For details on this protocol, see *AN965, "Microchip Stack for the ZigBee™ Protocol"* (DS0965).

## REFERENCES

D. Flowers and Y. Yang, *AN1066, "MiWi™ Wireless Networking Protocol Stack"* (DS1066), Microchip Technology Inc., 2007.

D. Flowers, K. Otten, Nilesh Rajbharti and Y. Yang, *AN965, "Microchip Stack for the ZigBee™ Protocol"* (DS0965), Microchip Technology Inc., 2007.

IEEE Std 802.15.4-2003™, *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs)*, IEEE, 2006.

---

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

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

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

---

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

#### **Trademarks**

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, rPIC and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


FilterLab, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, PICkit, PICDEM, PICDEM.net, PICtail, PIC<sup>32</sup> logo, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rLAB, Select Mode, Total Endurance, UNI/O, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2008, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

**QUALITY MANAGEMENT SYSTEM**  
**CERTIFIED BY DNV**  
**== ISO/TS 16949:2002 ==**

*Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.*



---

## WORLDWIDE SALES AND SERVICE

---

### AMERICAS

#### Corporate Office

2355 West Chandler Blvd.  
Chandler, AZ 85224-6199  
Tel: 480-792-7200  
Fax: 480-792-7277  
Technical Support:  
<http://support.microchip.com>  
Web Address:  
[www.microchip.com](http://www.microchip.com)

#### Atlanta

Duluth, GA  
Tel: 678-957-9614  
Fax: 678-957-1455

#### 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

Farmington Hills, MI  
Tel: 248-538-2250  
Fax: 248-538-2260

#### Kokomo

Kokomo, IN  
Tel: 765-864-8360  
Fax: 765-864-8387

#### Los Angeles

Mission Viejo, CA  
Tel: 949-462-9523  
Fax: 949-462-9608

#### Santa Clara

Santa Clara, CA  
Tel: 408-961-6444  
Fax: 408-961-6445

#### Toronto

Mississauga, Ontario,  
Canada  
Tel: 905-673-0699  
Fax: 905-673-6509

### ASIA/PACIFIC

#### Asia Pacific Office

Suites 3707-14, 37th Floor  
Tower 6, The Gateway  
Harbour City, Kowloon  
Hong Kong  
Tel: 852-2401-1200  
Fax: 852-2401-3431

#### Australia - Sydney

Tel: 61-2-9868-6733  
Fax: 61-2-9868-6755

#### China - Beijing

Tel: 86-10-8528-2100  
Fax: 86-10-8528-2104

#### China - Chengdu

Tel: 86-28-8665-5511  
Fax: 86-28-8665-7889

#### China - Hong Kong SAR

Tel: 852-2401-1200  
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: 86-21-5407-5533  
Fax: 86-21-5407-5066

#### China - Shenyang

Tel: 86-24-2334-2829  
Fax: 86-24-2334-2393

#### China - Shenzhen

Tel: 86-755-8203-2660  
Fax: 86-755-8203-1760

#### China - Wuhan

Tel: 86-27-5980-5300  
Fax: 86-27-5980-5118

#### China - Xiamen

Tel: 86-592-2388138  
Fax: 86-592-2388130

#### China - Xian

Tel: 86-29-8833-7252  
Fax: 86-29-8833-7256

#### China - Zhuhai

Tel: 86-756-3210040  
Fax: 86-756-3210049

### ASIA/PACIFIC

#### India - Bangalore

Tel: 91-80-4182-8400  
Fax: 91-80-4182-8422

#### India - New Delhi

Tel: 91-11-4160-8631  
Fax: 91-11-4160-8632

#### India - Pune

Tel: 91-20-2566-1512  
Fax: 91-20-2566-1513

#### Japan - Yokohama

Tel: 81-45-471- 6166  
Fax: 81-45-471-6122

#### 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: 886-3-572-9526  
Fax: 886-3-572-6459

#### Taiwan - Kaohsiung

Tel: 886-7-536-4818  
Fax: 886-7-536-4803

#### Taiwan - Taipei

Tel: 886-2-2500-6610  
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

#### France - Paris

Tel: 33-1-69-53-63-20  
Fax: 33-1-69-30-90-79

#### Germany - Munich

Tel: 49-89-627-144-0  
Fax: 49-89-627-144-44

#### Italy - Milan

Tel: 39-0331-742611  
Fax: 39-0331-466781

#### Netherlands - Drunen

Tel: 31-416-690399  
Fax: 31-416-690340

#### Spain - Madrid

Tel: 34-91-708-08-90  
Fax: 34-91-708-08-91

#### UK - Wokingham

Tel: 44-118-921-5869  
Fax: 44-118-921-5820