MiWi(TM) P2P Stack Application Programming Interfaces

Table of Contents

Symbol Reference 1

Functions 1

ActiveScan Function 2

AddConnection Function 3

BroadcastPacket Function 3

CheckForData Function 4

CreateNewConnection Function 4

DiscardPacket Function 4

DumpConnection Function 5

InitChannelHopping Function 5

initMRF24J40 Function 6

isSameAddress Function 6

MRF24J40Sleep Function 6

MRF24J40Wake Function 7

OptimalChannel Function 7

P2PInit Function 8

PHYGetLongRAMAddr Function 8

PHYGetShortRAMAddr Function 8

PHYSetLongRAMAddr Function 9

PHYSetShortRAMAddr Function 9

ReceivedPacket Function 10

ResyncConnection Function 10

SetChannel Function 11

StartChannelHopping Function 11

UnicastConnection Function 11

UnicastLongAddress Function 12

Structs, Records, Enums 12

ACTIVE_SCAN_RESULT Structure 13

INDIRECT_MESSAGE Structure 13

MRF24J40_IFREG Union 14

P2P_CAPACITY Union 15

P2P_CONNECTION_ENTRY Structure 15

P2P_STATUS Union 16

RECEIVED_FRAME Structure 17

Variables 19

AckFailureTimes Variable 19

ActiveScanResults Variable 19

AdditionalConnectionPayload Variable 19
CCAFailureTimes Variable 20
currentChannel Variable 20
indirectMessages Variable 20
myLongAddress Variable 20
myPANID Variable 20
P2PConnections Variable 20
rxFrame Variable 21

Macros 21
ClearReadyToSleep Macro 21
DataRequesting Macro 21
DisableAcknowledgement Macro 22
DisableNewConnection Macro 22
EnableAcknowledgement Macro 22
EnableNewConnection Macro 22
FlushTx Macro 22
isDataRequesting Macro 22
isReadyToSleep Macro 23
ReadyToSleep Macro 23
WriteData Macro 23

Index a

1 Symbol Reference

1.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	ActiveScan (see page 2)	BYTE ActiveScan(INPUT BYTE ScanDuration, INPUT DWORD ChannelMap) This function do an active scan and returns the number of PANs catched during the scan. The scan results are stored in global variable ActiveScanResults (see page 19)
	AddConnection (see page 3)	BYTE AddConnection(void) This function create a new P2P connection entry
	BroadcastPacket (see page 3)	BOOL BroadcastPacket(INPUT WORD_VAL DestinationPANID, INPUT BOOL isCommand, INPUT BOOL SecurityEnabled) This function broadcast a packet
	CheckForData (see page 4)	BOOL CheckForData(void) This function sends out a Data Request to the peer device of the first P2P connection.
	CreateNewConnection (see page 4)	BYTE CreateNewConnection(INPUT BYTE RetryInterval, INPUT DWORD ChannelMap) This function create a new P2P connection between two devices
	DiscardPacket (see page 4)	void DiscardPacket(void) This function needs to be called to discard, clear and reset the Rx module of the MRF24J40. This function needs to be called after the user is done processing a received packet
	DumpConnection (see page 5)	void DumpConnection(INPUT BYTE index) This function prints out the content of the connection with the input index of the P2P Connection Entry
	InitChannelHopping (see page 5)	BOOL InitChannelHopping(INPUT DWORD ChannelMap) This function try to start the process of channel hopping
	initMRF24J40 (see page 6)	void initMRF24J40(void) This function initializes the MRF24J40 and is required before stack operation is available
	isSameAddress (see page 6)	BOOL isSameAddress(INPUT BYTE *Address1, INPUT BYTE *Address2) This function compares two long address and returns the boolean to indicate if they are the same
	MRF24J40Sleep (see page 6)	void MRF24J40Sleep(void) Put the MRF24J40 radio into sleep
	MRF24J40Wake (see page 7)	void MRF24J40Wake(void) This function wakes up the MRF24J40 radio

	OptimalChannel (see page 7)	BYTE OptimalChannel(INPUT BYTE ScanDuration, INPUT DWORD ChannelMap, OUTPUT BYTE *RSSIValue) This function finds out the optimal channel with least noise after doing energy scan on all available channels supplied.
	P2PInit (see page 8)	FUNCTION PROTOTYPES void P2PInit(void) This function initializes the P2P stack and is required before stack operation is available
	PHYGetLongRAMAddr (see page 8)	BYTE PHYGetLongRAMAddr(INPUT WORD address) This function reads a value from a long RAM address
	PHYGetShortRAMAddr (see page 8)	BYTE PHYGetShortRAMAddr(INPUT BYTE address) This function reads a value from a short RAM address
	PHYSetLongRAMAddr (see page 9)	void PHYSetLongRAMAddr(INPUT WORD address, INPUT BYTE value) This function writes a value to a LONG RAM address
	PHYSetShortRAMAddr (see page 9)	void PHYSetShortRAMAddr(INPUT BYTE address, INPUT BYTE value) This function writes a value to a short RAM address
	ReceivedPacket (see page 10)	BOOL ReceivedPacket(void) This function returns the boolean to indicate if a new packet has been received by the stack
	ResyncConnection (see page 10)	BOOL ResyncConnection(INPUT BYTE *DestinationAddress, INPUT DWORD ChannelMap) This function try to resynchronize the connection with its peer.
	SetChannel (see page 11)	void SetChannel(INPUT BYTE channel) This function sets the current operating channel of the MRF24J40
	StartChannelHopping (see page 11)	void StartChannelHopping(INPUT BYTE OptimalChannel (see page 7)) This function broadcast the channel hopping command and after that, change operating channel to the input optimal channel
	UnicastConnection (see page 11)	BOOL UnicastConnection(INPUT BYTE ConnectionIndex, INPUT BOOL isCommand, INPUT BOOL SecurityEnabled) This is one of ways to unicast a packet to the peer device
	UnicastLongAddress (see page 12)	BOOL UnicastLongAddress(INPUT WORD_VAL DestinationPANID, INPUT BYTE *DestinationAddress, INPUT BOOL isCommand, INPUT BOOL SecurityEnabled) This is the way to unicast a packet to the peer device represented by teh destination long address

1.1.1 ActiveScan Function

BYTE ActiveScan(INPUT BYTE ScanDuration, INPUT DWORD ChannelMap)

This function do an active scan and returns the number of PANs catched during the scan. The scan results are stored in global variable ActiveScanResults (see page 19)

C

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

Parameters

Parameters	Description
INPUT BYTE ScanDuration	The time period to do the active scan on each channel

INPUT DWORD ChannelMap	The bit map of the channels to do active scan

Returns

The totoal number of PANs that have been acquired during active scan

Remarks

None

Conditions

P2P stack has been initialized

1.1.2 AddConnection Function

BYTE AddConnection(void)

This function create a new P2P connection entry

C

```
BYTE AddConnection();
```

Returns

The index of the P2P Connection Entry for the newly added connection

Remarks

A new P2P Connection Entry, the search connection operation ends if an entry is added successfully

Conditions

A P2P Connection Request or Response has been received and stored in rxFrame (see page 21) structure

1.1.3 BroadcastPacket Function

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

This function broadcast a packet

C

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

Parameters

Parameters	Description
INPUT WORD_VAL DestinationPANID	Destination PAN Identifier
INPUT BOOL isCommand	If packet to send is a command packet
INPUT BOOL SecurityEnabled	If packet to send needs encryption

Returns

If operation successful

Remarks

MRF24J40 is triggered to broadcast a packet

Conditions

MRF24J40 is initialized

1.1.4 CheckForData Function

BOOL CheckForData(void)

This function sends out a Data Request to the peer device of the first P2P connection.

C

```
BOOL CheckForData();
```

Returns

None

Remarks

The P2P stack is waiting for the response from the peer device. A data request timer has been started. In case there is no response from the peer device, the data request will time-out itself

Conditions

MRF24J40 is initialized and fully waken up

1.1.5 CreateNewConnection Function

BYTE CreateNewConnection(INPUT BYTE RetryInterval, INPUT DWORD ChannelMap)

This function create a new P2P connection between two devices

C

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

Parameters

Parameters	Description
INPUT BYTE RetryInterval	The interval between two tries of sending P2P Connection request in seconds
INPUT DWORD ChannelMap	Bitmap of available channels. Bit 11 means channel 11 and bit 26 means channel 26

Returns

The index of the P2P Connection Entry of the peer device for the new connection

Remarks

A new P2P Connection Entry

1.1.6 DiscardPacket Function

void DiscardPacket(void)

This function needs to be called to discard, clear and reset the Rx module of the MRF24J40. This function needs to be called after the user is done processing a received packet

C

```
void DiscardPacket();
```

Returns

None

Remarks

The RxBuffer gets flushed and the MRF24J40 is allowed to receive again.

Conditions

None

1.1.7 DumpConnection Function

void DumpConnection(INPUT BYTE index)

This function prints out the content of the connection with the input index of the P2P Connection Entry

C

```
void DumpConnection(INPUT BYTE index);
```

Parameters

Parameters	Description
INPUT BYTE index	The index of the P2P Connection Entry to be printed out

Returns

None

Remarks

The content of the connection pointed by the index of the P2P Connection Entry will be printed out

1.1.8 InitChannelHopping Function

BOOL InitChannelHopping(INPUT DWORD ChannelMap)

This function try to start the process of channel hopping

C

```
BOOL InitChannelHopping(INPUT DWORD ChannelMap);
```

Parameters

Parameters	Description
INPUT DWORD ChannelMap	The bit map of the candicate channels to hop to

Returns

boolean to indicate if channel hopping is initiated

Remarks

The operating channel will change to the optimal channel with least noise

Conditions

MRF24J40 has been initialized

1.1.9 initMRF24J40 Function

void initMRF24J40(void)

This function initializes the MRF24J40 and is required before stack operation is available

C

```
void initMRF24J40();
```

Returns

None

Remarks

MRF24J40 is initialized

Conditions

BoardInit (or other initialzation code is required)

1.1.10 isSameAddress Function

BOOL isSameAddress(INPUT BYTE *Address1, INPUT BYTE *Address2)

This function compares two long address and returns the boolean to indicate if they are the same

C

```
BOOL isSameAddress(INPUT BYTE * Address1, INPUT BYTE * Address2);
```

Parameters

Parameters	Description
INPUT BYTE * Address1	Pointer to the first long address to be compared
INPUT BYTE * Address2	Pointer to the second long address to be compared

Returns

If the two address are the same

1.1.11 MRF24J40Sleep Function

void MRF24J40Sleep(void)

Put the MRF24J40 radio into sleep

C

```
void MRF24J40Sleep();
```

Returns

None

Remarks

None

Conditions

BoardInit (or other initialzation code is required)

1.1.12 MRF24J40Wake Function

void MRF24J40Wake(void)

This function wakes up the MRF24J40 radio

C

```
void MRF24J40Wake();
```

Returns

None

Remarks

None

Conditions

MRF24J40 radio is in sleeping mode

1.1.13 OptimalChannel Function

BYTE OptimalChannel(INPUT BYTE ScanDuration, INPUT DWORD ChannelMap, OUTPUT BYTE *RSSIValue)

This function finds out the optimal channel with least noise after doing energy scan on all available channels supplied.

C

```
BYTE OptimalChannel(INPUT BYTE ScanDuration, INPUT DWORD ChannelMap, OUTPUT BYTE *
RSSIValue);
```

Parameters

Parameters	Description
INPUT BYTE ScanDuration	The scan duration of the energy scan. The scan duration is defined in IEEE902.15.4. The actual time is 60 * (2^(ScanDuration) + 1) symbols
INPUT DWORD ChannelMap	The bitmap of the available channels. Bit 11 represents channel 11 and Bit 26 represents channel 26 etc.
OUTPUT BYTE * RSSIValue	(OUTPUT) The maximum RSSI value for the optimal channel.

Returns

The optimal channel with least noise

Remarks

None

Conditions

MRF24J40 has been initialized

1.1.14 P2PInit Function

FUNCTION PROTOTYPES

void P2PInit(void)

This function initializes the P2P stack and is required before stack operation is available

C

```
void P2PInit();
```

Returns

None

Remarks

the P2P stack is initialized

Conditions

BoardInit (or other initialzation code is required)

1.1.15 PHYGetLongRAMAddr Function

BYTE PHYGetLongRAMAddr(INPUT WORD address)

This function reads a value from a long RAM address

C

```
BYTE PHYGetLongRAMAddr(INPUT WORD address);
```

Parameters

Parameters	Description
INPUT WORD address	the address of the long RAM address that you want to read from.

Returns

the value read from the specified Long register

Remarks

None

Conditions

Communication port to the MRF24J40 initialized

1.1.16 PHYGetShortRAMAddr Function

BYTE PHYGetShortRAMAddr(INPUT BYTE address)

This function reads a value from a short RAM address

C

```
BYTE PHYGetShortRAMAddr(INPUT BYTE address);
```

Parameters

Parameters	Description
INPUT BYTE address	the address of the short RAM address that you want to read from. Should use the READ_ThisAddress definition in the MRF24J40 include file.

Returns

None

Remarks

Interrupt from radio is turned off before accessing the SPI and turned back on after accessing the SPI

Conditions

Communication port to the MRF24J40 initialized

1.1.17 PHYSetLongRAMAddr Function

void PHYSetLongRAMAddr(INPUT WORD address, INPUT BYTE value)

This function writes a value to a LONG RAM address

C

```
void PHYSetLongRAMAddr(INPUT WORD address, INPUT BYTE value);
```

Parameters

Parameters	Description
INPUT WORD address	the address of the LONG RAM address that you want to write to
INPUT BYTE value	the value that you want to write to that register

Returns

None

Remarks

The register value is changed

Conditions

Communication port to the MRF24J40 initialized

1.1.18 PHYSetShortRAMAddr Function

void PHYSetShortRAMAddr(INPUT BYTE address, INPUT BYTE value)

This function writes a value to a short RAM address

C

```
void PHYSetShortRAMAddr(INPUT BYTE address, INPUT BYTE value);
```

Parameters

Parameters	Description
INPUT BYTE address	the address of the short RAM address that you want to write to. Should use the WRITE_ThisAddress definition in the MRF24J40 include file.

INPUT BYTE value	the value that you want to write to that register

Returns

None

Remarks

The register value is changed

Conditions

Communication port to the MRF24J40 initialized

1.1.19 ReceivedPacket Function

BOOL ReceivedPacket(void)

This function returns the boolean to indicate if a new packet has been received by the stack

C

```
BOOL ReceivedPacket();
```

Returns

Boolean to indicate if a new packet has been received by the stack

Remarks

Stack state machine has been called

Conditions

None

1.1.20 ResyncConnection Function

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

This function try to resynchronize the connection with its peer.

C

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

Parameters

Parameters	Description
INPUT BYTE * DestinationAddress	The pointer to the long address of the destination address
INPUT DWORD ChannelMap	• The bit map of the candicate channel to resynchronize connection

Returns

boolean to indicate of connection has been resynchronized

Remarks

The operating channel will change to the operating channel of the destination node, if successful

Conditions

MRF24J40 has been initialized

1.1.21 SetChannel Function

void SetChannel(INPUT BYTE channel)

This function sets the current operating channel of the MRF24J40

C

```
void SetChannel(INPUT BYTE channel);
```

Parameters

Parameters	Description
INPUT BYTE channel	this is the channel that you wish to operate on. This should be CHANNEL_11, CHANNEL_12, ..., CHANNEL_26.

Returns

None

Remarks

the MRF24J40 now operates on that channel

Conditions

MRF24J40 is initialized

1.1.22 StartChannelHopping Function

void StartChannelHopping(INPUT BYTE OptimalChannel (see page 7))

This function broadcast the channel hopping command and after that, change operating channel to the input optimal channel

C

```
void StartChannelHopping(INPUT BYTE OptimalChannel);
```

Parameters

Parameters	Description
INPUT BYTE OptimalChannel	The channel to hop to

Returns

None

Remarks

The operating channel for current device will change to the specified channel

Conditions

MRF24J40 has been initialized

1.1.23 UnicastConnection Function

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

This is one of ways to unicast a packet to the peer device

C

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

Parameters

Parameters	Description
INPUT BYTE ConnectionIndex	The index of P2P Connection Entry for the destination device
INPUT BOOL isCommand	If packet to send is a command packet
INPUT BOOL SecurityEnabled	If packet to send needs encryption

Returns

If operation successful

Remarks

MRF24J40 is triggered to unicast a packet

Conditions

MRF24J40 is initialized

1.1.24 UnicastLongAddress Function

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

This is the way to unicast a packet to the peer device represented by teh destination long address

C

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

Parameters

Parameters	Description
INPUT WORD_VAL DestinationPANID	Destination PAN Identifier
INPUT BYTE * DestinationAddress	Pointer to the destination long address
INPUT BOOL isCommand	If packet to send is a command packet
INPUT BOOL SecurityEnabled	If packet to send needs encryption

Returns

If operation successful

Remarks

MRF24J40 is triggered to unicast a packet

Conditions

MRF24J40 is initialized

1.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Structures

Name	Description
ACTIVE_SCAN_RESULT (see page 13)	The structure that store the active scan result
INDIRECT_MESSAGE (see page 13)	The structure to store indirect messages for devices turn off radio when idle
P2P_CONNECTION_ENTRY (see page 15)	The record to store the information of peer device in the MiWi(TM) P2P Connection
RECEIVED_FRAME (see page 17)	The structure that stores all information about the received frame, provided to the upper layer of the MiWi(TM) P2P stack

Unions

Name	Description
MRF24J40_IFREG (see page 14)	The interpolation of MRF24J40 radio interrupts. Details of interrupts can be found in MRF24J40 data sheet
P2P_CAPACITY (see page 15)	The capacity information for a MiWi(TM) P2P device. It is the definition of the first byte of PeerInfo defined in P2P_CONNECTION_ENTRY (see page 15). The highest bit also be used to indicate if the P2P connection entry is a valid entry
P2P_STATUS (see page 16)	structure to indicate the status of P2P stack

1.2.1 ACTIVE_SCAN_RESULT Structure

The structure that store the active scan result

C

```
typedef struct {
  BYTE Channel;
  BYTE RSSIValue;
  WORD_VAL PANID;
} ACTIVE_SCAN_RESULT;
```

Members

Members	Description
BYTE Channel;	The channel of the PAN that is operating on
BYTE RSSIValue;	The signal strength of the PAN. If there are more than one devices that belongs to the same PAN and respond to the active scan, the one with the strongest signal will be recorded
WORD_VAL PANID;	The PAN identifier of the PAN

1.2.2 INDIRECT_MESSAGE Structure

The structure to store indirect messages for devices turn off radio when idle

C

```
typedef struct {
  TICK TickStart;
  WORD_VAL DestPANID;
  union {
    BYTE DestLongAddress[8];
    BYTE DestIndex[P2P_CONNECTION_SIZE];
  } DestAddress;
  union {
    BYTE Val;
```

```
      struct {
        BYTE isValid : 1;
        BYTE isBroadcast : 1;
        BYTE isCommand : 1;
        BYTE isSecured : 1;
      } bits;
    } flags;
    BYTE PayLoadSize;
    BYTE PayLoad[TX_BUFFER_SIZE-21];
  } INDIRECT_MESSAGE;
```

Members

Members	Description
TICK TickStart;	start time of the indirect message. Used for checking indirect message time out
WORD_VAL DestPANID;	the PAN identifier for the destination node

union {
BYTE DestLongAddress[8];
BYTE DestIndex[P2P_CONNECTION_SIZE];
} DestAddress;
	destination address for the indirect message. Can either for unicast or broadcast

BYTE DestLongAddress[8];	unicast destination long address
BYTE DestIndex[P2P_CONNECTION_SIZE];	broadcast index of the P2P Connection Entries for destination RFD devices

union {
BYTE Val;
struct {
BYTE isValid : 1;
BYTE isBroadcast : 1;
BYTE isCommand : 1;
BYTE isSecured : 1;
} bits;
} flags;
	flags for indirect message

BYTE Val;	value for the flags

struct {
BYTE isValid : 1;
BYTE isBroadcast : 1;
BYTE isCommand : 1;
BYTE isSecured : 1;
} bits;
	bit map of the flags

BYTE isValid : 1;	if this indirect message is valid
BYTE isBroadcast : 1;	if this indirect message is for broadcasting
BYTE isCommand : 1;	if this indirect message a command
BYTE isSecured : 1;	if this indirect message requires encryption
BYTE PayLoadSize;	the indirect message pay load size
BYTE PayLoad[TX_BUFFER_SIZE-21];	the indirect message pay load

1.2.3 MRF24J40_IFREG Union

The interpolation of MRF24J40 radio interrupts. Details of interrupts can be found in MRF24J40 data sheet

C

```
typedef union {
  BYTE Val;
  struct {
    BYTE RF_TXIF : 1;
```

```
      BYTE RF_RXIF : 1;
      BYTE SECIF : 1;
    } bits;
  } MRF24J40_IFREG;
```

Members

Members	Description
BYTE Val;	value of interrupts

struct {
BYTE RF_TXIF : 1;
BYTE RF_RXIF : 1;
BYTE SECIF : 1;
} bits;
	bit map of interrupts

BYTE RF_TXIF : 1;	transmission finish interrupt
BYTE RF_RXIF : 1;	receiving a packet interrupt
BYTE SECIF : 1;	receiving a secured packet interrupt

1.2.4 P2P_CAPACITY Union

The capacity information for a MiWi(TM) P2P device. It is the definition of the first byte of PeerInfo defined in P2P_CONNECTION_ENTRY (see page 15). The highest bit also be used to indicate if the P2P connection entry is a valid entry

C

```
  typedef union {
    BYTE Val;
    struct _P2P_CAPACITY_BITS {
      BYTE RXOnWhileIdel : 1;
      BYTE DataRequestNeeded : 1;
      BYTE TimeSynchronization : 1;
      BYTE SecurityCapacity : 1;
      BYTE filler : 3;
      BYTE isValid : 1;
    } bits;
  } P2P_CAPACITY;
```

Members

Members	Description
BYTE Val;	the value of the P2P capacitity
BYTE RXOnWhileIdel : 1;	if device turns on radio when idle
BYTE DataRequestNeeded : 1;	if data request is required when device turns off radio when idle. It is used to decide if an indirect message is necessary to be stored.
BYTE TimeSynchronization : 1;	reserved bit for future development
BYTE SecurityCapacity : 1;	if the device is capable of handling encrypted information
BYTE isValid : 1;	use this bit to indicate that this entry is a valid entry

1.2.5 P2P_CONNECTION_ENTRY Structure

The record to store the information of peer device in the MiWi(TM) P2P Connection

C

```
  typedef struct {
    BYTE PeerLongAddress[8];
    DWORD_VAL IncomingFrameCounter;
    BYTE PeerInfo[1+ADDITIONAL_CONNECTION_PAYLOAD];
  } P2P_CONNECTION_ENTRY;
```

Members

Members	Description
BYTE PeerLongAddress[8];	the long address of the peer device
DWORD_VAL IncomingFrameCounter;	the incoming frame counter. Used to check frame freshness to avoid repeat attack only valid if security is enabled
BYTE PeerInfo[1+ADDITIONAL_CONNECTION_PAYLOAD];	The peer information. The first byte is the standard peer information. The detailed definition of the first byte can be found in P2P_CAPACITY (see page 15) definition. Additional info can be added according to the specific application

1.2.6 P2P_STATUS Union

structure to indicate the status of P2P stack

C

```
typedef union {
  WORD Val;
  struct {
    BYTE RX_PENDING : 1;
    BYTE RX_BUFFERED : 1;
    BYTE RX_ENABLED : 1;
    BYTE RX_IGNORE_SECURITY : 1;
    BYTE RX_SECURITY : 1;
    BYTE TX_BUSY : 1;
    BYTE TX_PENDING_ACK : 1;
    BYTE PHY_SLEEPING : 1;
    BYTE TimeToSleep : 1;
    BYTE DataRequesting : 1;
    BYTE RxHasUserData : 1;
    BYTE AckRequired : 1;
    BYTE EnableNewConnection : 1;
    BYTE TX_FAIL : 1;
    BYTE SearchConnection : 1;
    BYTE Resync : 1;
  } bits;
} P2P_STATUS;
```

Members

Members	Description
WORD Val;	The value of the P2P status flags

struct {
BYTE RX_PENDING : 1;
BYTE RX_BUFFERED : 1;
BYTE RX_ENABLED : 1;
BYTE RX_IGNORE_SECURITY : 1;
BYTE RX_SECURITY : 1;
BYTE TX_BUSY : 1;
BYTE TX_PENDING_ACK : 1;
BYTE PHY_SLEEPING : 1;
BYTE TimeToSleep : 1;
BYTE DataRequesting : 1;
BYTE RxHasUserData : 1;
BYTE AckRequired : 1;
BYTE EnableNewConnection : 1;
BYTE TX_FAIL : 1;
BYTE SearchConnection : 1;
BYTE Resync : 1;
} bits;
	bit map of the P2P status

BYTE RX_PENDING : 1;	indicate if a frame is received and pending to read
BYTE RX_BUFFERED : 1;	indicate if a frame has been read into the buffer of MCU and waiting to be processed
BYTE RX_ENABLED : 1;	indicate if the radio is enabled to receive the next packet
BYTE RX_IGNORE_SECURITY : 1;	indicate if we need to ignore the current packet with encryption. Usually, the reason is frame counter freshness checking fails
BYTE RX_SECURITY : 1;	indicate if current received packet is encrypted
BYTE TX_BUSY : 1;	indicate if the radio is currently busy transmitting a packet
BYTE TX_PENDING_ACK : 1;	indicate if the current transmitting packet waiting for an acknowledgement from the peer device
BYTE PHY_SLEEPING : 1;	indicate if the device in sleeping state
BYTE TimeToSleep : 1;	used by the application layer to indicate that the RFD device is idle and ready to go to sleep
BYTE DataRequesting : 1;	indicate that device is in the process of data request from its parent. Only effective if device enables sleeping
BYTE RxHasUserData : 1;	indicate if the received frame needs processing from the application layer
BYTE AckRequired : 1;	indicate if the transmitting packet require acknowledgement from the peer device
BYTE EnableNewConnection : 1;	indicate if the current device allow new P2P connection established.
BYTE TX_FAIL : 1;	indicate if the current transmission fails
BYTE SearchConnection : 1;	indicate if the stack is currently in the process of looking for new connection
BYTE Resync : 1;	indicate if the stack is currently in the process of resynchronizing connection with the peer device

1.2.7 RECEIVED_FRAME Structure

The structure that stores all information about the received frame, provided to the upper layer of the MiWi(TM) P2P stack

C

```
typedef struct {
  union {
    BYTE Val;
    struct {
      BYTE commandFrame : 1;
      BYTE security : 1;
      BYTE framePending : 1;
      BYTE intraPAN : 1;
      BYTE broadcast : 1;
    } bits;
  } flags;
  BYTE PacketLQI;
  BYTE PacketRSSI;
  WORD_VAL SourcePANID;
  BYTE SourceLongAddress[8];
  BYTE PayLoadSize;
  BYTE * PayLoad;
} RECEIVED_FRAME;
```

Members

Members	Description

union {
BYTE Val;
struct {
BYTE commandFrame : 1;
BYTE security : 1;
BYTE framePending : 1;
BYTE intraPAN : 1;
BYTE broadcast : 1;
} bits;
} flags;
	received frame flags

BYTE Val;	value of the flags for RECEIVED_FRAME

struct {
BYTE commandFrame : 1;
BYTE security : 1;
BYTE framePending : 1;
BYTE intraPAN : 1;
BYTE broadcast : 1;
} bits;
	bit map of the flags for received frame

BYTE commandFrame : 1;	if the received frame a command frame. data: 0; command 1
BYTE security : 1;	if the received frame is encrypted during transmission. The payload in this structure has already been decrypted
BYTE framePending : 1;	if the frame pending bit has been set in the frame control this bit is hardly used, just a legancy inherited from IEEE 802.15.4
BYTE intraPAN : 1;	if the received frame a intra-PAN frame. Meaning if the the source PAN ID the same as the destination PAN ID
BYTE broadcast : 1;	if the received frame a broadcast message
BYTE PacketLQI;	the link quality indication for the received packet that indicats the quality of the received frame
BYTE PacketRSSI;	the RSSI of for the received packet that indicates the signal strength of the received frame
WORD_VAL SourcePANID;	The PAN identifier of the source device
BYTE SourceLongAddress[8];	The long address of the source device
BYTE PayLoadSize;	The size of the payload
BYTE * PayLoad;	The pointer to the pay load of the received packet. The pay load is the MAC payload without the MAC header

1.3 Variables

The following table lists variables in this documentation.

Variables

Name	Description
AckFailureTimes (see page 19)	Continuous failure times because of no acknowledgement received
ActiveScanResults (see page 19)	The results for active scan, including the PAN identifier, signal strength and operating channel
AdditionalConnectionPayload (see page 19)	the additional information regarding the device that would like to share with the peer on the other side of P2P connection. This information is applicaiton specific.
CCAFailureTimes (see page 20)	Continuous failure times because of CSMA-CA failure
currentChannel (see page 20)	current operating channel for the device
indirectMessages (see page 20)	structure to store the indirect messages for nodes with radio off duing idle time
myLongAddress (see page 20)	The extended long address for the device
myPANID (see page 20)	the PAN Identifier for the device
P2PConnections (see page 20)	The peer device records for P2P connections
rxFrame (see page 21)	structure to store information for the received packet

1.3.1 AckFailureTimes Variable

Continuous failure times because of no acknowledgement received

C

```
BYTE AckFailureTimes = 0;
```

1.3.2 ActiveScanResults Variable

The results for active scan, including the PAN identifier, signal strength and operating channel

C

```
ACTIVE_SCAN_RESULT ActiveScanResults[ACTIVE_SCAN_RESULT_SIZE];
```

1.3.3 AdditionalConnectionPayload Variable

the additional information regarding the device that would like to share with the peer on the other side of P2P connection. This information is applicaiton specific.

C

```
BYTE AdditionalConnectionPayload[];
```

1.3.4 CCAFailureTimes Variable

Continuous failure times because of CSMA-CA failure

C

```
BYTE CCAFailureTimes = 0;
```

1.3.5 currentChannel Variable

current operating channel for the device

C

```
BYTE currentChannel;
```

1.3.6 indirectMessages Variable

structure to store the indirect messages for nodes with radio off duing idle time

C

```
INDIRECT_MESSAGE indirectMessages[INDIRECT_MESSAGE_SIZE];
```

1.3.7 myLongAddress Variable

The extended long address for the device

C

```
ROM unsigned char myLongAddress[8] = {EUI_0,EUI_1,EUI_2,EUI_3,EUI_4,EUI_5,EUI_6,EUI_7};
```

1.3.8 myPANID Variable

the PAN Identifier for the device

C

```
WORD_VAL myPANID;
```

1.3.9 P2PConnections Variable

The peer device records for P2P connections

C

```
P2P_CONNECTION_ENTRY P2PConnections[P2P_CONNECTION_SIZE];
```

1.3.10 rxFrame Variable

structure to store information for the received packet

C

```
RECEIVED_FRAME rxFrame;
```

1.4 Macros

The following table lists macros in this documentation.

Macros

Name	Description
ClearReadyToSleep (see page 21)	Clear the flag to notify the stack that the device is idle and ready to go to sleep
DataRequesting (see page 21)	Notify the P2P stack that a data request procedure is going on
DisableAcknowledgement (see page 22)	Disable the request of acknowledgement for all unicast message
DisableNewConnection (see page 22)	Disable the MiWi(TM) P2P stack to accept new P2P connection request
EnableAcknowledgement (see page 22)	Enable the request of acknowledgment for all unicast message
EnableNewConnection (see page 22)	Enable the MiWi(TM) P2P stack to accept new P2P connection request
FlushTx (see page 22)	Reset the transmission buffer before writing any data
isDataRequesting (see page 22)	Check if the stack is in the data request process
isReadyToSleep (see page 23)	Check if the stack is idle and ready to go to sleep
ReadyToSleep (see page 23)	Notify the stack that the device is idle and ready to go to sleep now
WriteData (see page 23)	Write one byte into transmission buffer

1.4.1 ClearReadyToSleep Macro

Clear the flag to notify the stack that the device is idle and ready to go to sleep

C

```
#define ClearReadyToSleep P2PStatus.bits.TimeToSleep = 0;
```

1.4.2 DataRequesting Macro

Notify the P2P stack that a data request procedure is going on

C

```
#define DataRequesting P2PStatus.bits.DataRequesting = 1;
```

1.4.3 DisableAcknowledgement Macro

Disable the request of acknowledgement for all unicast message

C

```
#define DisableAcknowledgement P2PStatus.bits.AckRequired = 0;
```

1.4.4 DisableNewConnection Macro

Disable the MiWi(TM) P2P stack to accept new P2P connection request

C

```
#define DisableNewConnection P2PStatus.bits.EnableNewConnection = 0;
```

1.4.5 EnableAcknowledgement Macro

Enable the request of acknowledgment for all unicast message

C

```
#define EnableAcknowledgement P2PStatus.bits.AckRequired = 1;
```

1.4.6 EnableNewConnection Macro

Enable the MiWi(TM) P2P stack to accept new P2P connection request

C

```
#define EnableNewConnection P2PStatus.bits.EnableNewConnection = 1;
```

1.4.7 FlushTx Macro

Reset the transmission buffer before writing any data

C

```
#define FlushTx TxData = 0;
```

1.4.8 isDataRequesting Macro

Check if the stack is in the data request process

C

```
#define isDataRequesting P2PStatus.bits.DataRequesting
```

1.4.9 isReadyToSleep Macro

Check if the stack is idle and ready to go to sleep

C

```
#define isReadyToSleep P2PStatus.bits.TimeToSleep
```

1.4.10 ReadyToSleep Macro

Notify the stack that the device is idle and ready to go to sleep now

C

```
#define ReadyToSleep P2PStatus.bits.TimeToSleep = 1;
```

1.4.11 WriteData Macro

Write one byte into transmission buffer

C

```
#define WriteData(a) TxBuffer[TxData++] = a
```

Index

A

AckFailureTimes 19

AckFailureTimes variable 19

ACTIVE_SCAN_RESULT 13

ACTIVE_SCAN_RESULT structure 13

ActiveScan 2

ActiveScan function 2

ActiveScanResults 19

ActiveScanResults variable 19

AddConnection 3

AddConnection function 3

AdditionalConnectionPayload 19

AdditionalConnectionPayload variable 19

B

BroadcastPacket 3

BroadcastPacket function 3

C

CCAFailureTimes 20

CCAFailureTimes variable 20

CheckForData 4

CheckForData function 4

ClearReadyToSleep 21

ClearReadyToSleep macro 21

CreateNewConnection 4

CreateNewConnection function 4

currentChannel 20

currentChannel variable 20

D

DataRequesting 21

DataRequesting macro 21

DisableAcknowledgement 22

DisableAcknowledgement macro 22

DisableNewConnection 22

DisableNewConnection macro 22

DiscardPacket 4

DiscardPacket function 4

DumpConnection 5

DumpConnection function 5

E

EnableAcknowledgement 22

EnableAcknowledgement macro 22

EnableNewConnection 22

EnableNewConnection macro 22

F

FlushTx 22

FlushTx macro 22

Functions 1

I

INDIRECT_MESSAGE 13

INDIRECT_MESSAGE structure 13

indirectMessages 20

indirectMessages variable 20

InitChannelHopping 5

InitChannelHopping function 5

initMRF24J40 6

initMRF24J40 function 6

isDataRequesting 22

isDataRequesting macro 22

isReadyToSleep 23

isReadyToSleep macro 23

isSameAddress 6

isSameAddress function 6

M

Macros 21

MRF24J40_IFREG 14

MRF24J40_IFREG union 14

MRF24J40Sleep 6

MRF24J40Sleep function 6

MRF24J40Wake 7

MRF24J40Wake function 7

myLongAddress 20

myLongAddress variable 20

myPANID 20

myPANID variable 20

O

OptimalChannel 7

OptimalChannel function 7

P

P2P_CAPACITY 15

P2P_CAPACITY union 15

P2P_CONNECTION_ENTRY 15

P2P_CONNECTION_ENTRY structure 15

P2P_STATUS 16

P2P_STATUS union 16

P2PConnections 20

P2PConnections variable 20

P2PInit 8

P2PInit function 8

PHYGetLongRAMAddr 8

PHYGetLongRAMAddr function 8

PHYGetShortRAMAddr 8

PHYGetShortRAMAddr function 8

PHYSetLongRAMAddr 9

PHYSetLongRAMAddr function 9

PHYSetShortRAMAddr 9

PHYSetShortRAMAddr function 9

R

ReadyToSleep 23

ReadyToSleep macro 23

RECEIVED_FRAME 17

RECEIVED_FRAME structure 17

ReceivedPacket 10

ReceivedPacket function 10

ResyncConnection 10

ResyncConnection function 10

rxFrame 21

rxFrame variable 21

S

SetChannel 11

SetChannel function 11

StartChannelHopping 11

StartChannelHopping function 11

Structs, Records, Enums 12

U

UnicastConnection 11

UnicastConnection function 11

UnicastLongAddress 12

UnicastLongAddress function 12

V

Variables 19

W

WriteData 23

WriteData macro 23