BlackBerry Software Development Kit

Version 2.5

Radio API Reference Guide (Mobitex)

BlackBerry Software Development Kit Version 2.5 Radio API Reference Guide
Last modified: 18 March 2002
Part number: PDF-04638-001

At the time of printing, this documentation complies with RIM Wireless Handheld Version 2.5.

Research In Motion Limited
295 Phillip Street
Waterloo, ON N2L 3W8
Canada

Produced in Canada

Contents

About this guide

This guide provides a detailed reference for the Radio Application Programming Interface (API).

The Radio API provides packet-level access to the Mobitex network using function calls to send and receive data. You do not need extensive knowledge of the network to use these functions.

This guide assumes you have experience with C++ programming.

Other resources

Before using this guide, you should be familiar with the following documentation. These other resources can help you develop C++ applications for the BlackBerry Wireless Handheld.

All RIM documentation is available at `http://developers.rim.net`.

- *BlackBerry SDK Developer Guide*

  This guide explains how to use the BlackBerry SDK and contains sample code for the wireless handheld's general functions.

- *BlackBerry SDK Message API Reference Guide*

  The Radio API provides packet-level access to the radio network. If your application needs to send messages, such as email or fax, use the Messaging API.

- `README.txt`

  The `README.txt` file is installed with the BlackBerry Software Developer Kit (SDK). It provides information on any known issues and workarounds, as well as last-minute documentation updates and release notes.

Chapter 1
Getting started

This chapter provides an overview of radio communications over the Mobitex network, including these topics:

- overview of the Mobitex network
- how data is routed in the network

Note: This chapter provides background information on radio communications on the Mobitex network. This information is not intended to be comprehensive. Contact your network operator for complete documentation on network operation.

Understanding the Mobitex network

Mobitex is a packet-switched, narrowband personal communications service (PCS) network designed for wide-area wireless data communications. Mobitex networks are operated by service providers such as Cingular Interactive in the United States, Rogers AT&T in Canada, and other companies in Asia, Australia, Europe, and South America.

Wireless applications typically send short amounts of data in bursts, with fairly long delays between each transmission. Packet switching uses limited radio frequency resources efficiently by enabling multiple users to share channels.

Mobitex provides highly reliable, 2-way digital data transmission. The network provides error detection and correction, including transmission acknowledgement, to maintain the integrity of the data being sent.

Packet-switching technology provides flexibility and efficiency for wireless data transmission, especially when the application involves messaging, dispatching, remote queries, or other situations in which only small amounts of data are transferred.

Routing

Each device on the Mobitex network is assigned a unique, 24-bit Mobitex access number (MAN).

Data is routed through the network from sender to receiver in the form of Mobitex packets (MPAKs). When sending MPAKs, you can have the Radio API automatically format the MPAK and send it. The application fills in the appropriate elements in the `HEADER` structure and then calls the `RadioSendMpak` function.

MPAKs are typically assembled and formatted by the operating system. Applications fill in elements of the header structure, and then call `RadioSendMpak()` to submit this structure, with the data, to the operating system for assembly into an MPAK. Refer to "Structures" on page 9 for more information on the `MPAK_HEADER` structure.

Alternatively, applications can assemble and format raw MPAKs.

The Radio API supports the following MPAK types: `TEXT`, `DATA`, `STATUS`, and `HPDATA`. In addition, the Radio API does not support MPAKs with address lists to send to multiple destinations.

If another MPAK type is required, the application must assemble and format a raw MPAK.

An MPAK structure contains the following information:

- 24-bit sender MAN
- 24-bit addressee MAN
- 24-bit time stamp of one-minute intervals
- 2 bytes of type information
- up to 512 bytes of payload data

If it is necessary to send more than 512 bytes of data, the application must divide the data into two or more MPAKs.

Note: At the data link layer between the handheld and the network base station, each MPAK is divided into smaller units of data, called radio-oriented synchronous information (ROSI) blocks. Contact your network operator for more information.

Chapter 2
Radio API reference

This chapter provides information on Radio API structures, functions, and error codes.

Mobitex Radio API

The Radio API provides access to the radio network using simple API function calls to send and receive data. You do not need extensive knowledge of the radio network to use these function calls.

Radio events are announced to applications through the message system and provide information on the status of incoming and outgoing packet communications. Refer to "Radio events" on page 22 for more information.

Structures

The Radio API uses the following structures.

MPAK_HEADER

This structure represents the header data in an Mobitex packet (MPAK).

```
typedef struct {
    long Sender;
    long Destination;
    int MpakType;
    int HPID;
    int Flags;
    TIME Timestamp;
    long lTime;
    int TrafficState;
} MPAK_HEADER;
```

Field	Description
Sender	The Sender field specifies the source MAN, which identifies the originator of the MPAK. The MAN is specified in most significant bit (MSB) to least significant bit (LSB) format. This field is filled in automatically when sending MPAKs. This field tells the network where to send a failure status or positive acknowledgment. If the MPAK is returned to the sender, the source and destination MANs do not change. The returned MPAK is an exact duplicate of the original MPAK that was sent, with the exception of the traffic state bits indicating its failed status.
Destination	The Destination field specifies the destination MAN, which identifies the intended recipient of the MPAK. The MAN is specified in MSB to LSB format. To send an MPAK, your application must know the MAN of the destination device. When replying to a message, applications can use the source MAN from the header of the received MPAK to send a reply. The RadioSendMPAK function can contain a pointer to an MPAK_HEADER structure, in which header.DESTINATION contains a DWORD indicating the destination MAN. If the MPAK is returned to the sender, the source and destination MANs do not change. The returned MPAK is an exact duplicate of the original MPAK that was sent, with the exception of the traffic state bits indicating its failed status.
MpakType	The MpakType field specifies the type of MPAK. This can be one of the following: MPAK_TEXT MPAK_DATA MPAK_STATUS MPAK_HPDATA For all other MPAK types, this value is set to 0.
HPID	The HPID field specifies the HPID value for HPDATA MPAK types

Field	Description
Flags	The Flags field specifies one or both of the following values: • FLAG_MAILBOX: If the destination is not reachable, the network stores the MPAK in a mailbox and returns a copy to the sender with the TrafficState field state set to TS_MESSAGE_IN_MAILBOX. • FLAG_POSACK: The network sends a copy of the original MPAK to the sender after it has been delivered.
Timestamp	The Timestamp field specifies the parsed Mobitex time stamp for a received MPAK. The Radio API sets the timestamp to 0x000000 when sending an MPAK; the time stamp is set by the network when it receives the packet.
lTime	The lTime field specifies the raw MPAK Mobitex timestamp, in minutes since January 1, 1985.
TrafficState	The TrafficState field specifies the MPAK traffic state. This can be one of the following: • TS_MESSAGE_OK • TS_MESSAGE_FROM_MAILBOX • TS_MESSAGE_IN_MAILBOX • TS_CANNOT_BE_REACHED • TS_ILLEGAL_MESSAGE • TS_NETWORK_CONGESTED • TS_TECHNICAL_ERROR • TS_DESTINATION_BUSY This field must be set to TS_MESSAGE_OK when sending an MPAK.

RADIO_INFO

This structure stores general information about the state of the radio.

```
typedef struct {
    int   RadioOn;
    DWORD LocalMAN;
    DWORD ESN;
    int   Base;
    int   Area;
    int   RSSI;
    WORD  NetworkID;
    DWORD FaultBits;
    BOOL  Active;
    BOOL  PowerSaveMode;
    BOOL  LiveState;
    BOOL  TransmitterEnabled;
} RADIO_INFO;
```

Field	Description
RadioOn	One of RADIO_ON or RADIO_OFF
LocalMAN	The handheld's MAN number, as a 32-bit value
ESN	The electronic serial number, as a 32-bit value
Base, Area	Current base and Area ID where the handheld is located or where the handheld was last in network coverage; the Base and Area together uniquely identify a base station in the Mobitex network
RSSI	RSSI value in the range of –113 to –40, or RSSI_NO_COVERAGE; values above –90 are generally reliable network coverage
NetworkID	Network ID of the network on which the handheld is located; this value is 0xB433 in the US
FaultBits	Flags indicating various problems with the handheld
Active	Network flow control flag that indicates whether the handheld is receiving messages. The handheld can send an ACTIVE or INACTIVE MPAKs to the network. 1 = "flow on" state; network will forward packets 0 = "flow off" state; network will not forward packets Applications should not send INACTIVE or ACTIVE packets directly to the network; instead, they should call RadioStopReception() and RadioResumeReception().
PowerSaveMode	Status of the low-power mode; always 1 on the handheld: 1 = powersave mode 0 = express
LiveState	Status of the live state mode: 1 = LIVE state, 0 = DIE state
TransmitterEnabled	Status of the transmitter mode: 1 = Tx enabled, 0 = Tx disabled

SKIPNUM_INFO

This structure is used to query the radio about R14N skipnum settings.

```
typedef struct {
    BYTE  SkipNum;
    BYTE  ProtocolRevision;
    BYTE  SkipTrans;
    BYTE  Mode;
} SKIPNUM_INFO;
```

NETWORKS_INFO

Field	Description
SkipNum	Current Skipnum value used by the handheld
ProtocolRevision	Mobitex protocol revision (0 for pre-R14N)
SkipTrans	SkipTrans value of the Mobitex network
Mode	Status of the low-power mode: 1 = low-power mode, 0 = express mode

This structure is used to query the radio for supported network IDs.

```
typedef struct {
    int   DefaultNetworkIndex;
    int   CurrentNetworkIndex;
    int   NumValidNetworks;
    struct {
                WORD         NetworkId;
                BYTE     NetworkName[10];
    } Networks[10];
} NETWORKS_INFO;
```

Field	Description
DefaultNetworkIndex	Default network
CurrentNetworkIndex	Current network
NumValidNetworks	Number of valid networks
NetworkId	Network frame synchronization word (0xB433 in the US) Part of the Networks substructure in NETWORKS_INFO
NetworkName[10]	Name of the network Part of the Networks substructure in NETWORKS_INFO

Functions

The following functions are listed alphabetically.

RadioAccelerateRetries

Causes the radio to retry transmitting more aggressively.

```
void RadioAccelerateRetries(int mpakTag)
```

Parameters

`mpakTag`	Tag of the MPAK for which the radio should accelerate transmission (not currently used).

Description When the radio has difficulty transmitting an MPAK to the base station due to network congestion or poor network coverage, it normally increases the interval between transmission retries to allow conditions to improve. `RadioAccelerateRetries` causes the radio to retry sending the MPAK in the handheld more aggressively. This decreases battery life in exchange for stronger attempts to send the MPAK. `RadioAccelerateRetries` should normally only be called based on user action that indicates that the user is waiting for a packet to be sent (such as the user selecting Resend for data that has already been submitted by an application).

RadioCancelSendMpak

Cancels a submitted MPAK.

```
int RadioCancelSendMpak(int mpakTag)
```

Parameters

`mpakTag`	This parameter is the tag assigned by the application server when the packet is submitted to the handheld for transmission. A value of -1 cancels all MPAKs that are queued for transmission by the calling application.

Returns This function returns the number of MPAKs that were cancelled. It returns a negative value if an error occurs.

Description This function attempts to cancel a submitted packet that is identified by the tag number. If this function is called before the MPAK is transmitted, the MPAK is returned to the application as cancelled, provided that it has not already been sent. There is no guarantee, however, that a cancelled MPAK was not already received by the Mobitex network.

RadioChangeNetworks

Changes the current radio network (Rogers AT&T in Canada, Cingular Interactive in the United States).

```
void RadioChangeNetworks(DWORD NetworkId, BYTE * NetworkName)
```

Parameters

`NetworkId`	The new network's ID number.
`NetworkName`	Name of the network to which the current network is being changed.

Returns No return value.

Description This function changes the current network to the specified network. This could be necessary if the application requires access to networks in both Canada and the United States.

RadioDeregister

Deregisters applications from receiving radio events.

```
void RadioDeregister(void)
```

Returns No return value.

Description This function deregisters the current application so that it no longer receives `RADIO` events. Any MPAKs that the deregistering application has pending for transmission are cancelled and returned to the application. Therefore, it is still possible for the application to receive some radio events after de-registering.

RadioGetAvailableNetworks

Programs the available networks into the handheld.

```
void RadioGetAvailableNetworks(NETWORKS_INFO * info)
```

Parameters

`info`	Pointer to a NETWORKS_INFO structure (refer to "Structures" on page 9).

Returns No return value.

Description Enables you to query the handheld and determine which networks have been programmed.

RadioGetDetailedInfo

Retrieves the current state of the radio.

```
void RadioGetDetailedInfo(RADIO_INFO * info)
```

Returns No return value.

Description Retrieves the current state of the radio, such as MAN number, RSSI, on/off, powersave/express, base, and area, into a `RADIO_INFO` structure (refer to "Structures" on page 9 for details)

RadioGetMpak

Retrieves the data of a received MPAK

```
int RadioGetMpak(int mpakTag,
    MPAK_HEADER * header,
    BYTE * data )
```

Parameters

`mpakTag`	This parameter is the `MPAK_TAG` value from the `MESSAGE_RECEIVED` message. The `MPAK_TAG` value has a limited life span. For received MPAKs, the tag must be used before getting the next message or yielding control.
`header`	This parameter is a pointer to an `MPAK_HEADER` structure (refer to "Structures" on page 9 for details). The information extracted from the MPAK header is placed in this structure. If this pointer is NULL, no header is extracted, and the raw MPAK is placed in the buffer.
`data`	This parameter is a pointer to a buffer large enough to contain the MPAK. The amount of space required can be determined by calling `RadioGetMpak`. It is recommended that this parameter always point to a buffer of at least 512 bytes. If the header pointer is NULL, the raw MPAK is placed in the buffer. In this case, the buffer should be at least 560 bytes. If the data pointer is NULL, the MPAK is not copied.

Returns If `header` is not NULL, the number of data bytes in the data portion of the MPAK (0 to 512) is returned if successful. If `header` is NULL, the length of the entire MPAK is returned.

The function has a return value of -1 if it is unsuccessful.

Description When an MPAK is received, the `MPAK_TAG` value is contained in the message. This tag value is used to obtain subsequent information about the MPAK.

This function can also be used to get copies of MPAKs that are queued for transmission. MPAKs that are queued for transmission can be recalled at any time until `RimTaskYield` or `RimGetMessage` are called.

`RadioGetMpak` can be used in several ways:

- You can obtain only the header, by setting the data pointer to NULL.
- You can obtain both the header and the MPAK. Both pointers point to their respective data areas. Only the data portion of the MPAK is copied into the data buffer.
- You can obtain the raw MPAK. The header pointer is set to NULL, while the data pointer points to a buffer for the raw MPAK. The entire MPAK, including header information, is copied into the data buffer.

The MPAK is only guaranteed to be available until the application yields control to the system (via `RimGetMessage` or `RimYield`). The MPAK remains available until all applications that have registered to receive MPAKs have received the `RADIO_MESSAGE_RECIEVED` message. After all registered applications have received this message, the MPAK is released the next time that control is yielded to the system (through `RimGetMessage` or `RimTaskYield`).

RadioGetSignalLevel

Gets the current signal strength.

```
int RadioGetSignalLevel()
```

Returns Radio signal level in dBm, if the handheld is in an area of wireless network coverage; the value is typically between -121 dBm and -40 dBm.

If the handheld is out of network coverage, the return value is -256 (RSSI_NO_COVERAGE) or less.

Description The return value is always negative. A higher number (closer to 0) indicates greater strength of the received signal. For example, –90 dBm. indicates greater coverage than -93 dBm.

Example

```
// Displays the strength of the received radio signal
int level = RadioGetSignalLevel();

if (level > RSSI_NO_COVERAGE){
    sprintf( buffer, “Level = %d dBm”, level );
} else {
    sprintf( buffer, “No coverage”);
}
```

RadioOnOff

Checks/changes radio status (on/off).

```
int RadioOnOff(int mode)
```

Parameters

`mode`	Specifies the new state of the radio; the `mode` parameter can be one of the following values: `radio_on` — turns on the radio `radio_off` — turns off the radio `radio_get_onoff` — returns the current state

Returns The function returns the state of the radio before `RadioOnOff` was called, and can be one of the following values:

`radio_on`	The radio is on.
`radio_off`	The radio is off, or turning off.
`radio_lowbatt`	The radio is on, but the battery is too low for it to be operational.

Description This function enables the applications to check and modify the on/off state of the radio. The radio must be explicitly turned on if applications want to use it, as its default state is off if any applications are loaded.

Note: Refer to `RadioGetDetailedInfo` to check other details of the radio's state.

RadioRegister

Registers applications for radio events.

```
void RadioRegister()
```

Returns No return value.

Description Applications must call this function to receive notification of `RADIO` events (including received MPAKs). Applications that have not registered for radio events cannot send or receive MPAKs. After calling `RadioRegister`, the application receives a `SIGNAL_LEVEL` message if the radio is on or receives a `RADIO_TURNED_OFF` message if the radio is off.

RadioRequestSkipnum

Sets and requests the skipnum parameters used in R14N networks

```
int RadioRequestSkipnum(SKIPNUM_INFO * SkipInfo,
    int Skipnum)
```

Parameters

SkipInfo	This parameter is a pointer to a SKIPNUM_INFO structure (refer to "Structures" on page 9 for details). This structure is filled with the current parameters of the R14N skipping algorithm. These parameters reflect the value of Skipnum that was used at the time that RadioRequestSkipnum was called. If Info is a NULL pointer, the structure will not be filled in.
Skipnum	This parameter represents the new value of Skipnum to use. Legal values are 1, 2, 4, 8, and 16. A value of 0 indicates an information request without a change request.

Returns This function returns the value of Skipnum that was used before RadioRequestSkipnum was called. It returns negative if an error occurs.

Description This function is used to set and request the surface velocity profilers (SVP) skip parameters used in networks that support the R14N level of firmware or higher, such as the Cingular Interactive network in the United States.

The skipnum value determines how many 10-second intervals the handheld waits before turning on to see if the network has traffic to address to it. Thus, an additional delay of up to 10 times the value of skipnum can be introduced on unsolicited traffic from the base station. Skipnum does not affect the timing of traffic going into the network.

When the skipnum value changes, the handheld transmits to the network which skipnum interval is used. A skipnum value of 1 provides the fastest delivery of unsolicited traffic from the network, while larger values of skipnum save battery power because the receiver is not turned on as often. Skipnum values of 1, 2, or 4 are recommended, and 4 is the default. The gain in battery life of settings greater than 4 is small, so values above 4 are not recommended.

Refer to Ericsson's documentation on R14N for more details on skipnum and how it affects various aspects of the Mobitex protocol.

RadioResumeReception

Indicates that the application is ready to receive MPAKs again.

```
void RadioResumeReception()
```

Returns No return value.

Description This function is used to indicate that the application is ready to receive MPAKs again after `RadioStopReception` is called. If `RadioStopReception` is used to save an MPAK, the MPAKis again received with a `MESSAGE_RECEIVED` message, as if it had just been received by the radio.

This function must be called by the same task or thread that calls `RadioStopReception`. Each task that calls `RadioStopReception` must call this function before more MPAKs can be received.

RadioSendMpak

Submits an MPAK for transmission by the radio.

```
int RadioSendMpak(MPAK_HEADER * header,
    BYTE * data,
    int length)
```

Parameters

`header`	This parameter is a pointer to an `MPAK_HEADER` structure (refer to "Structures" on page 9 for details). This structure contains information for building the MPAK header, including the type of MPAK and the addressee. If the application wants to build the MPAK and header itself, the `header` parameter is set to NULL, and no header is added to the MPAK. The `Sender` field of the header is always filled by the application server to the handheld's MAN, and does not need to be set by the application.
`data`	This parameter is a pointer to a buffer that contains the data bytes that are to be included in the MPAK.

Returns A tag is assigned to the MPAK by the application server. If the sequence identification is negative, the message cannot be queued for sending. The returned tag value is always less than `MAX_QUEUED_MPAKS`, which is currently defined as 7.

Description `RadioSendMpak` submits an MPAK for transmission by the radio. If an MPAK has already been submitted for transmission by this or any other application, the MPAK is queued. If more than four MPAKs are already queued, `RadioSendMpak` fails and returns a negative error code.

`RadioSendMpak` copies the data that is provided. The data that is pointed to when the call is made can be deleted after the call returns.

Example

```
// Send an Hpdata 123 MPAK with data “Hello” to MAN
// 123456. Send a Status 10 MPAK to MAN 123456.
{
    MPAK_HEADER header;
    int temp;

    header.Destination = 123456;
    header.MpakType = MPAK_HPDATA;
    header.Flags = FLAG_POSACK;
    header.TrafficState = TS_MESSAGE_OK;
    header.HPID = 123;

    tag1 = RadioSendMpak(header, “Hello”, 5);
    // Send a status MPAK. Header is mostly set up
    // already.
    header.MpakType = MPAK_STATUS;
    temp = 10;
    tag2 = RadioSendMpak(header, &temp, 1);
}
```

RadioStopReception

Indicates that the radio is not ready to receive MPAKs.

```
void RadioStopReception(int mpakTag)
```

Parameters

`mpakTag`	If `RadioStopReception` is called in response to a `MESSAGE_RECEIVED` message, the `mpakTag` value can be passed into the function as a parameter. After `RadioResumeReception` is called, the saved MPAK is resent to the calling application.

Description The `RadioStopReception` function stops the handheld from receiving MPAKs. It is intended for use when all buffers for receiving MPAKs are full. This function should be used only if no more memory can be allocated to save received data. `RadioStopReception` causes the radio to eventually stop receiving MPAKs for all applications running on the handheld.

Further MPAKs can still be received after `RadioStopReception` is called, as they might already be in the calling task’s message queue. These MPAKs can still be saved by calling `RadioStopReception` again.

Radio events

When any of the following events occur, the `Device` member of the `MESSAGE` structure is equal to `DEVICE_RADIO`.

Event	Description
`MESSAGE_RECEIVED`	This event is sent to all applications that have registered to receive radio events (`RadioRegister`). This event indicates that a data packet was received from the Mobitex network. The `SubMsg` field contains a tag value to be passed into `RadioGetMpak`. `Data[0]` contains the type of the MPAK received, and `Data[1]` contains the HPID (if the MPAK is of type hpdata). Applications should call `RadioGetMpak` to receive the message data.
`MESSAGE_SENT`	This event is an acknowledgement that a transmitted packet was received by the Mobitex network. This event is sent to the application that sent the packet, whether that application is in the foreground or the background. The `SubMsg` field contains the tag value that was returned by `RadioGetMpak`.
`MESSAGE_NOT_SENT`	This event indicates that an attempt to transmit information to the Mobitex network failed. This event is sent to the task that submitted the packet when coverage is too poor for transmission or when an invalid data package is sent. The `SubMsg` field contains the tag value returned by `RadioGetMpak`. The `Data[0]` contains the error number.
`SIGNAL_LEVEL`	This event is sent to all registered applications to indicate that the received signal level has changed. The `SubMsg` field contains a negative value that represents the level of the signal in dBm. A more positive value (closer to zero) indicates a stronger signal. A value of -256 dBm (`RSSI_NO_COVERAGE`) indicates that the modem is out of coverage.
`NETWORK_STARTED`	This event is sent to all registered applications to indicate that the radio modem has been turned on or has just switched to a new network.

Event	Description
BASE_STATION_CHANGE	This event is sent when the handheld switches base stations. It has no other effect and requires no action on the part of the application.
RADIO_TURNED_OFF	This event is sent to all registered applications to indicate that the radio modem has been turned off, either by the user or as a result of a low battery.
MESSAGE_STATUS	An MPAK sent to the Radio API might not be transmitted immediately. The sender of an MPAK is notified of that MPAK transmission status through this event. The Data[0] field of the message structure contains one of the following status subcodes: • MPAK_TRANSMITTING : MPAK is being sent by the radio • MPAK_TX_PENDING: The radio is not transmitting the MPAK because of transmission difficulties; it will try again later

Error codes

The following error codes pertain to radio function return values.

Code	Description	Description
-1	RADIO_APP_NOT_REGISTERED	Applications must be registered for radio events to be allowed to send MPAKs. Attempting to send MPAKs without being radio- registered returns this error code.
-2	RADIO_MPAK_NOT_FOUND	Attempting to fetch an MPAK with a tag value that has expired produces this error code. MPAKs must be fetched before the task yields control to other tasks.
-3	RADIO_NO_FREE_BUFFERS	Attempting to send an MPAK with all the radio's outgoing buffers full produces this error code.
-4	RADIO_BAD_DATA	Attempting to send an MPAK with format data that cannot be used in an MPAK produces this error code.
-5	RADIO_BAD_TAG	Attempting to fetch an MPAK with a tag value outside the legal range produces this error code.
-6	RADIO_ERROR_GENERAL	This is a generic radio error.
-7	RADIO_ILLEGAL_SKIPNUM	Attempting to set the Skipnum value with `RimRequestSkipnum` to any value other than 1, 2, 4, 8, or 16 produces this error.
-8	RADIO_ILLEGAL_WSM_PACKET	Not implemented.

Index

Functions

A

B

C

M

N

P

R

S

Index