BM innovations GmbH
incremental changes in thinking

BlueRobin®

Wireless Low-power Sensor Network

BM-USBRTXx USB Transceiver Unit
BM-USBDx USB Transceiver Dongle

Integration Guide

Date: 29th of July 2014
Revision: 1.1

BM innovations GmbH · Mainburger Str. 3 · 85413 Hoergertshausen · Germany · Phone +49 8764 949142 · Fax +49 8764 949141 · www.bm-innovations.com

1 Introduction

A BlueRobin interface DLL is available providing all functions required for a PC application to communicate with a BM-USBRTX3/BM-USBRTX4 transceiver unit or a BM-USBD1/BM-USBD4 transceiver dongle via a USB interface. It supports the following operating systems:

- Windows: WinXP, Windows 7, Windows 8, Windows 8.1
- MacOS: >= 10.7
- Linux: Kernel 2.9x

Currently only a 32-bit version is supported for Windows Operating Systems.

The System Development Kit (SDK) also includes code examples based on Microsoft Visual Studio MFC (Microsoft Visual Studio 2008).

2 *BlueRobin®* Support in BM-USBRTXx and BM-USBDx

The transceivers provide full ***BlueRobin®*** receiver (RX mode) support with messaging system and high speed data download.

3 Interface Function Overview

Function	Description
rtxDll_GetVersion	Get BlueRobin DLL Version
rtxDll_Initalize	Initialize BlueRobin DLL
rtxDll_DeInitialize	Deinitialize BlueRobin DLL
rtxDll_IsInitialized	Check if BlueRobin DLL is initialized
rtxInterface_ScanForDevices	Scan interface for all connected devices
rtxInterface_GetDeviceInfo	Get device interface information
rtxInterface_UnplugDetectionStart	Start device unplug detection
rtxInterface_UnplugDetectionStop	Stop device unplug detection
rtxInterface_UnplugDetected	Device unplug detection status function
rtxDevice_Open	Open port the device is connected to
rtxDevice _Close	Close previously opened port
rtxDevice _IsOpen	Check if port has already been opened
rtxDevice _InitHardware	Initialize device
rtxDevice_GetInfo	Get device configuration and version info
rtxConfig_RxMode	Configure BlueRobin RX mode operation
rtxConfig_RxLinkFailureLimit	Set limit for consecutive lost data packets in RX mode
rtxConfig_RxSearchTimeout	Set timeout when trying to find a transmitter in RX mode
rtxConfig_RxPairingSensitivity	Set receiver sensitivity for transmitter pairing mode (ID set to 0) in RX mode
rtxConfig_RxSearchRatio	Set ratio between active search time and search pause in RX mode
rtxRX_ChannelReset	Reset RX channel to default settings
rtxRX_ChannelSetProfile	Set profile of transmitter to be received on RX channel
rtxRX_ChannelGetProfile	Get profile of RX channel
rtxRX_ChannelSetID	Set ID of transmitter to be received on RX channel
rtxRX_ChannelGetID	Get ID of RX channel
rtxRX_ChannelStart	Start search or pairing for transmitter on RX channel
rtxRX_ChannelStop	Stop RX channel
rtxRX_ChannelGetState	Get state of RX channel

rtxRX_ChannelSendMessage	Send message on RX channel
rtxRX_ChannelSendMessageData	Send message data on RX channel
rtxRX_ChannelRequestDownload	Request high speed data download on RX channel
rtxRX_ChannelClear	Clear any pending message and data download request on RX channel
rtxRX_GetState	Update internal buffers and return RX mode state
rtxRX_GetData	Get new data received on any RX channel
rtxRX_GetMessageData	Get requested message data received on any RX channel
rtxRX_GetDownload	Get requested high speed data download received on any RX channel

4 API Function Details

4.1 DLL Functions

4.1.1 Get version of DLL

This function returns the main and sub version number of the rtxBlueRobin DLL.

```
unsigned short rtxDll_GetVersion(void);
```

Return Value:
Returns the main version number in the upper byte and the sub version number in the lower byte.

4.1.2 Initialize DLL

This function initializes the rtxBlueRobin DLL.

```
bool rtxDll_Initialize(void);
```

Return Value:
Returns `true` if DLL could be initialized successfully, otherwise returns `false`

Remarks:
Has to be called first before accessing any other function of the rtxBlueRobin DLL API.

4.1.3 Deinitialize DLL

This function unloads the rtxBlueRobin DLL.

```
bool rtxDll_DeInitialize(void);
```

Return Value:
Returns `true` if DLL could be unloaded successfully, otherwise returns `false`

Remarks:
Has to be called when closing the application so that all object threads and ports are released.

4.1.4 Check if DLL has been initialized

This function can be used to check if the rtxBlueRobin DLL has already been successfully initialized.

```
bool rtxDll_IsInitialized(void);
```

Return Value:
Returns `true` if DLL has already been initialized successfully, otherwise returns `false`

4.2 Interface Functions

4.2.1 Scan for all BlueRobin devices on ports

This function checks all ports for connected BlueRobin devices having the specified device type, it returns the number of connected devices and it generates an internal list with device details which can be accessed then using function `rtxInterface_GetInfo()`.

```
bool rtxInterface_ScanForDevices(bmiDeviceType DeviceType,
                                 int           &NoOfDevicesFound);
```

Parameters:

`DeviceType`	Scanning for BM-USBD1:	bmiDEVTYPE_USBD1
	Scanning for BM-USBRTX3:	bmiDEVTYPE_USBRTX3
`NoOfDevicesFound`	Number of devices found with specified device type	

Return Value:
Returns `true` if at least one device could be found, otherwise returns `false`

4.2.2 Get device information for a found BlueRobin device

After having scanned all ports for connected BlueRobin devices of a specific device type using function `rtxInterface_ScanForDevices()` this function has to be used to get details for the found devices so that the desired device to be opened can be selected.

```
bool rtxInterface_GetInfo(unsigned char    DeviceIndex,
                          rtxUsbDeviceInfo *DeviceInfo);
```

Parameters:

`DeviceIndex` — Index into the device list generated by function `rtxInterface_ScanForDevices()`; starts at 0 and can be up to the number of found devices less 1 returned when scanning

`DeviceInfo` — Contains the device information for the selected device

```
typedef struct rtxUsbDeviceInfo
{
  unsigned short DeviceType;        // Device type identifier
  char           PortName[256];     // Port name
  char           PhysName[256];     // Physical name
  char           DevDescr[256];     // Friendly name
  char           EnumName[256];     // Enumerator name
  char           LocationName[256]; // USB bus location identifier
  char           SerialNumber[256]; // Serial number
  unsigned short Vid;               // Vendor ID
  unsigned short Pid;               // Product ID
} rtxUsbDeviceInfo;
```

Return Value:
Returns `true` if device could be found and its information is available, otherwise returns `false`

4.2.3 Start to detect if a BlueRobin device has been unplugged

This function starts background checking if the specified BlueRobin device has been unplugged from the specified port.

```
bool rtxInterface_UnplugDetectionStart(const char *PortName);
```

Parameters:

`PortName` — String specifying the port name; will be returned as a part of the USB device information when calling function `rtxInterface_GetInfo()`

Return Value:
Returns `true` if BlueRobin device and port has been found, otherwise returns `false`

Remarks:
This function is in experimental state!

4.2.4 Stop to detect if a BlueRobin device has been unplugged

This function stops any previously started check for an unplug event.

```
bool rtxInterface_UnplugDetectionStop(void);
```

Return Value:
Returns `true` if detection could have been stopped, otherwise returns `false`

Remarks:
This function is in experimental state!

4.2.5 Signal an unplugged BlueRobin device

This function should be periodically checked to see if an unplug event happened.

```
bool rtxInterface_UnplugDetected(void);
```

Return Value:
Returns `true` if an unplug event has been detected, otherwise returns `false`

Remarks:
This function is in experimental state!

4.3 Device Functions

4.3.1 Open port

Opens the port the BlueRobin transceiver hardware is connected to. The BM-USBRX3 hardware is powered up when opening the port. Some parts of the hardware require a start up time of up to 2 seconds so it could take up to that time to open the port. No start up time and therefore no delay can be seen for BM-USBD1 transceiver dongles.

```
bool rtxDevice_Open(const char *PortName);
```

Parameters:

`PortName`	String specifying the port name; will be returned as a part of the device information when calling function `rtxInterface_GetDeviceInfo()`

Return Value:
Returns `true` if port could be opened and transceiver hardware could be found, otherwise returns `false`

Remarks:
Do not try to open more than one port for a BM-USBRTX3 transceiver unit at a time. Always close the port first before trying to open the port again!

4.3.2 Close port

A successfully opened port has to be closed at the end of the application runtime to release all allocated memory and the port itself.

```
bool rtxDevice_Close(void);
```

Return Value:
Returns `true` if port could be found and closed, otherwise returns `false`

4.3.3 Check if port has been opened

This function checks if a port to a BlueRobin device has already been opened.

```
bool rtxDevice_IsOpen(void);
```

Return Value:
Returns `true` if port is open, otherwise returns `false`

4.3.4 Initialize BlueRobin hardware

To initialize the connected transceiver hardware the following function has to be called once after opening the port and before any further access to the hardware is done. It also returns the number of available channels in RX mode.

```
bool rtxDevice_InitHardware(unsigned char &MaxChannels);
```

Parameters:

`MaxChannels` Number of available channels

Return Value:
Returns `true` if the initialization was successful and `false` if accessing the hardware failed.

Remarks:
BlueRobin RX mode has to be configured first.

4.3.5 Get transceiver unit configuration and version info

To get the number of configured channels in RX mode and the profile, ID and descriptor in TX mode (currently not implemented) the following function can be used. Also the serial number and the software and hardware revisions will be returned.

```
bool rtxDevice_GetInfo(rtxDeviceInfo *DeviceInfo);
```

Parameters:

`DeviceInfo` Device info if read successfully, passed in the following structure:

```
typedef struct rtxDevInfo
{
  unsigned char  MaxChannels;  // Number of available channels in RX mode
  unsigned char  Profile;      // Profile in TX mode (not used)
  unsigned long  ID;           // ID in TX mode (not used)
  unsigned char  Descriptor;   // Descriptor in TX mode (not used)
  unsigned long  SerialNumber; // Device serial number
  unsigned short SwRev;        // Device software revision
  unsigned short HwRev;        // Device hardware revision
} rtxDevInfo;
```

Return Value:
Returns `true` if the requested information could be read successfully and `false` if accessing the hardware failed.

4.4 Configuration Functions

4.4.1 Configure BlueRobin RX mode operation

In case the maximum possible number of channels is not required the number of channels to be used can be configured with this function to increase the system performance. Optionally this function can also be used to preset channels with profiles. There is no need to call this function if the maximum available number of channels has to be used without any preset.

```
bool rtxConfig_RxMode(unsigned char           MaxChannels,
                      rtxPresetChannelProfile *ProfileList,
                      unsigned char           NoOfProfiles);
```

Parameters:

`MaxChannels` Number of required channels

`ProfileList` Pointer to array of structures (`ProfileList[NoOfProfiles]`) containing channel and profile pairs to be preset; if no preset channels are required this parameter has to be set to NULL

```
typedef struct rtxPresetChannelProfile
{
  unsigned char Channels; // Number of channels to be preset with same profile
  unsigned char Profile;  // Profile to be preset
} rtxPresetChannelProfile;
```

`NoOfProfiles` Number of channel and profile pairs contained in the array

Return Value:
Returns `true` if both setting number of channels and presetting channels was successful and `false` if configuring failed.

Remarks:
Reducing the number of used channels results in an increased system performance.

4.4.2 Set limit for consecutive lost data packets

If an active channel does not get valid data packets over a certain period the synchronisation to the transmitter could get lost due to timing deviations. This function sets the number of consecutive lost data packets after which an active channel will be switched off or set to seach mode automatically. The default link failure limit is 8 packets, the valid range is 1 to 200 packets.

```
bool rtxConfig_LinkFailureLimit(unsigned char Limit);
```

Parameters:

`Limit`	Number of missed packets in a row causing a link failure

Return Value:
Returns `true` if the link failure limit could be changed and `false` if the change failed.

Remarks:
If the search timeout is set to 0 (see function `rtxConfig_SearchTimeout()`) a transmitter search will be started automatically if the link failure limit is reached.

4.4.3 Set timeout when trying to find a transmitter

The time for trying to find a transmitter when starting a channel in both pairing and normal search mode can be set with this function. It is specified in seconds. If the transmitter could be found within this time the channel changes from search to active mode, otherwise it switches off. The default search timeout is 8 seconds, the valid range is 3 to 200 seconds. Setting the timeout value to 0 results in an infinite search and an automatic search start in case the link failure limit is reached.

```
bool rtxConfig_SearchTimeout(unsigned char Timeout);
```

Parameters:

`Timeout`	Number of seconds the search on a channel will be stopped if the specified transmitter could not be found or infinite search with automatic search start on link failure if timeout set to 0.

Return Value:
Returns `true` if the search timeout could be changed and `false` if the change failed.

Remarks:
In search mode the average current consumption is more than 200 times higher than in active mode, so if current consumption is an issue the search timeout should be set to the smallest value acceptable for the application.

4.4.4 Set receiver sensitivity for transmitter pairing mode

This function has to be used to reduce the receiver sensitivity for a channel when pairing to a transmitter (ID set to 0) to be able to only get transmitters close to the receiver. As soon as a transmitter could be found the sensitivity of the corresponding channel will be set back automatically to its maximum value. Default setting is no sensitivity reduction, the valid reduction range is 0 to 40 dB.

```
bool rtxConfig_PairingSensitivity(unsigned char Reduction);
```

Parameters:

`Reduction`	Reduction of sensitivity in dB.

Return Value:
Returns `true` if the sensitivity reduction could be changed and `false` if the change failed.

4.4.5 Set ratio between active search time and search pause

To reduce the average current consumption when searching for a transmitter the search can be paused in regular intervals. This function allows setting the search/pause ratio. Default ratio is 1, currently only a ratio of 0 for continuous search and 1 for a search/pause ratio of ~2/1 are supported.

```
bool rtxConfig_SearchRatio(unsigned char Ratio);
```

Parameters:

`Ratio`	Search/pause ratio, only 0 and 1 allowed.

Return Value:
Returns `true` if the search ratio could be changed and `false` if the change failed.

4.5 RX Channel Functions

4.5.1 Reset RX channel to default settings

To reset a channel to its default values (both profile and ID to 0) this function can be used.

```
bool rtxRX_ChannelReset(unsigned char Channel);
```

Parameters:

`Channel`	Index of channel the reset has to be performed on; starts at 0 rtxALL_CHANNELS can be used to reset all channels

Return Value:
Returns `true` if channel(s) could be reset and `false` if the reset failed.

4.5.2 Set profile of transmitter to be received on RX channel

To specify the profile of a transmitter to be received on a channel this function can be used. In combination with the ID either only a unique transmitter will be received or if the ID is set to 0 any transmitter with the specified profile and not already active on any other channel will be received (= pairing mode). The default channel profile setting is 0. It has to be changed before being able to start a channel as 0 is an invalid profile. The valid profile range is 1 to 250. A channel has to be switched off before its profile can be changed.

```
bool rtxRX_ChannelSetProfile(unsigned char Channel,
                             unsigned char Profile);
```

Parameters:

`Channel`	Index of channel the profile has to be set on; starts at 0 rtxALL_CHANNELS can be used if the profile has to be set on all channels
`Profile`	Profile to be set

Return Value:
Returns `true` if channel(s) could be set to the specified profile and `false` if the setting failed.

Remarks:
The profiles can also be set when configuring the BlueRobin RX mode operation using `rtxConfig_RxMode()`.

4.5.3 Get profile of RX channel

The currently set profile can be read from a channel using this function.

```
bool rtxRX_ChannelGetProfile(unsigned char Channel,
                             unsigned char &Profile);
```

Parameters:

`Channel`	Index of channel the profile has to be read from; starts at 0
`Profile`	Profile

Return Value:
Returns `true` if the profile could be read and `false` if accessing the hardware failed.

4.5.4 Set ID of transmitter to be received on RX channel

To specify the ID of a transmitter to be received on a channel this function can be used. In combination with the profile either only a unique transmitter will be received or if the ID is set to 0 any transmitter with the specified profile and not already active on any other channel will be received (= pairing mode). The default channel ID setting is 0. The valid ID range is 1 to 16000000. A channel has to be switched off before its ID can be changed.

```
bool rtxRX_ChannelSetID(unsigned char Channel,
                        unsigned long ID);
```

Parameters:

`Channel`	Index of channel the ID has to be set on; starts at 0
`ID`	ID to be set

Return Value:
Returns `true` if channel could be set to the specified ID and `false` if the setting failed.

Remarks:
The ID can also be set when configuring the BlueRobin RX mode operation using `rtxConfig_RxMode()`.

4.5.5 Get ID of RX channel

The currently set ID can be read from a channel using this function. After pairing to a transmitter the ID of the found transmitter can be read and then preset before any further channel start so that only the paired transmitter will be received on this channel later on.

```
bool rtxRX_ChannelGetID(unsigned char Channel,
                        unsigned long &ID);
```

Parameters:

`Channel`	Index of channel the profile and ID has to be read from; starts at 0
`ID`	ID

Return Value:
Returns `true` if the ID could be read and `false` if accessing the hardware failed.

4.5.6 Start search or pairing for transmitter on RX channel

If the ID is set to a value not equal to zero a search for a transmitter with this ID will be started, otherwise a search for any transmitter not already active on any other channel will be started. If a channel is already active, starting it does not have any effect.

```
bool rtxRX_ChannelStart(unsigned char Channel);
```

Parameters:

`Channel`	Index of channel to be started; starts at 0 rtxALL_CHANNELS can be used to start all configured channels at once

Return Value:
Returns `true` if the channel(s) could be started or was already active and `false` if the function failed.

4.5.7 Stop RX channel

If a channel is in active mode it can be stopped with this function. Any pending message will be sent before, the channel configuration (profile and ID) will be kept. If a channel is already switched off it does not have any effect.

```
bool rtxRX_ChannelStop(unsigned char Channel);
```

Parameters:

`Channel`	Index of channel to be stopped; starts at 0 rtxALL_CHANNELS can be used to stop all active channels at once

Return Value:
Returns `true` if the channel(s) could be stopped or has already been switched off and `false` if the function failed.

4.5.8 Get state of RX channel

To get the current state of a channel this function has to be used.

```
bool rtxRX_ChannelGetState(unsigned char Channel,
                           unsigned char &ChannelState,
                           unsigned char &MessageState,
                           unsigned char &DownloadState);
```

Parameters:

`Channel` — Index of channel the state is required; starts at 0

`ChannelState` — Channel state if reading was successful:

`rtxCH_STATE_OFF`	Channel switched off
`rtxCH_STATE_RESTART`	Channel restarted after high speed data download
`rtxCH_STATE_SEARCH`	Channel in search mode with specified ID
`rtxCH_STATE_PAIRING`	Channel in pairing mode, ID set to 0
`rtxCH_STATE_ACTIVE`	Channel in active mode, no data or information available
`rtxCH_STATE_DATA`	Channel in active mode and new regular data or information received; can be read using `rtxRX_GetData()`
`rtxCH_STATE_DATA_MESSAGE`	Channel in active mode and new regular data and new message data received; can be read using `rtxRX_GetData()` and `rtxRX_GetMessageData()`

`MessageState` — Message state if reading was successful:

`rtxCH_STATE_MSG_IDLE`	No message pending
`rtxCH_STATE_MSG_PENDING`	Message sent, waiting for acknowledgement
`rtxCH_STATE_MSG_ACK`	Sent message acknowledged
`rtxCH_STATE_MSG_FAILED`	Sending message failed

`DownloadState` — Data download state if reading was successful:

`rtxCH_STATE_DOWNLOAD_IDLE`	No data download pending
`rtxCH_STATE_DOWNLOAD_PENDING`	Data download in progress, number of processed bytes can be read using `rtxRX_GetDownload()`
`rtxCH_STATE_DOWNLOAD_READY`	Requested data download received, can be read using `rtxRX_GetDownload()`
`rtxCH_STATE_DOWNLOAD_FAILED`	Data download failed

Return Value:
Returns `true` if the state of the channel could be determined and `false` if the function failed.

4.6 RX Message Functions

4.6.1 Send message on RX channel

This function can be used to send a message including message info bytes to a transmitter after the next received data packet.

```
bool rtxRX_ChannelSendMessage(unsigned char Channel,
                              unsigned char Message,
                              unsigned char *Data);
```

Parameters:

`Channel` — Index of channel the message has to be sent on; starts at 0

`Message` — Message identifier

`Data` — Pointer to 5 bytes containing message info

Return Value:
Returns `true` if the message for the channel could be prepared to be sent (but not already sent!) and `false` if the function failed.

Remarks:
Sending a message can also be requested if a channel is not already started, so e.g. a wake-up message could be sent to a transmitter immediately after the first standby packet has been received. The periodically called function `rtxRX_GetState()` signals any progress or failure in the message handling. Function `rtxRX_ChannelGetState()` can be used to get more details of the current message handling state.

4.6.2 Send message data on RX channel

This function can be used to send message data to a transmitter after the next received data packet.

```
bool rtxRX_ChannelSendMessageData(unsigned char  Channel,
                                  unsigned short Descriptor,
                                  unsigned char  Size,
                                  unsigned char  *Data);
```

Parameters:

`Channel`	Index of channel the message data have to be sent on; starts at 0
`Descriptor`	Data descriptor
`Size`	Number of data bytes to be sent; valid range is 1 to 48
`Data`	Pointer to data to be sent

Return Value:
Returns `true` if the message data for the channel could be prepared to be sent (but not already sent!) and `false` if the function failed.

Remarks:
There is no built-in acknowledgement process for message data. If an acknowledgement is required it is recommended to read back and verify the sent data or any other indicator showing a successful transfer.

4.6.3 Request high speed data download on RX channel

To send a request for a memory download to a transmitter after the next received data packet this function has to be used. All active channels will be switched off temporary during download and set to restart state (search after high speed data download) afterwards.

```
bool rtxRX_ChannelRequestDownload(unsigned char Channel,
                                  unsigned char Flags,
                                  unsigned long Address,
                                  unsigned long Size);
```

Parameters:

`Channel`	Index of channel the data download has to be requested; starts at 0
`Flags`	Data download options (currently not used, to be set to 0)
`Address`	Start address for data to be downloaded
`Size`	Number of bytes to be downloaded

Return Value:
Returns `true` if the request for the channel could be prepared to be sent (but not already sent!) and `false` if the function failed.

Remarks:
The periodically called function `rtxRX_GetState()` signals any progress or failure in the data download handling. Function `rtxRX_ChannelGetState()` can be used to get more details of the current download state.

4.6.4 Clear any pending message and data download request on RX channel

This function removes any pending message and data download request and clears all corresponding channel buffers.

```
bool rtxRX_ChannelClear(unsigned char Channel);
```

Parameters:

`Channel`	Index of channel to be cleared; starts at 0

Return Value:
Returns `true` if the channel could be cleared and `false` if the function failed.

4.7 RX Data Functions

4.7.1 Update internal buffers and return RX mode state

The following function has to be called periodically with the update rate required by the application (typically one second). It parses all data automatically sent by the transceiver hardware from the USB port buffer to the DLL buffers and signals the availability of new data or information to the application.

```
bool rtxRX_GetState(unsigned short &State);
```

Parameters:

State	Current state if reading was successful, signaled with the following flags:	
	rtxSTATE_RX_DATA	New channel data or information received; for details see rtxRX_GetData()
	rtxSTATE_RX_MESSAGE	Requested channel message data or information received; for details see rtxRX_GetMessageData()
	stxSTATE_RX_DOWNLOAD	New channel data download or progress info received; for details see rtxRX_GetDownload()

Return Value:
Returns `true` if updating the buffers and the transceiver state was successful and `false` if accessing the hardware failed.

Remarks:
It is recommended to call this function at least every two seconds to avoid buffer overflows.

4.7.2 Get new data received on any RX channel

This function has to be used to get received regular data on any channel together with signal strength and sequence counter. To pass new data a FIFO buffer is used. This buffer is big enough to store data for at least 3 seconds, so calling `BR_GetState()` to check for new data is required at least every 2 seconds to ensure no data will get lost. Also a lost packet, a lost transmitter, a not successful search and an automatic search start will be indicated.

```
bool rtxRX_GetData(unsigned char &Channel,
                   unsigned char &Type,
                   unsigned char &SignalStrength,
                   unsigned char &SequenceCounter,
                   unsigned char *Data);
```

Parameters:

Channel	Index of the channel the data have been received on; starts at 0	
Type	Type of data or information	
	rtxCH_TYPE_DATA	Data packet with 5 data bytes received
	rtxCH_TYPE_STANDBY	Standby packet with one info byte received
	rtxCH_TYPE_DATA_LOST	Data packet lost
	rtxCH_TYPE_TX_LOST	Transmitter lost, channel switched off
	rtxCH_TYPE_TX_SEARCH	Transmitter lost, new search started
	rtxCH_TYPE_SEARCH_STOPPED	Search for transmitter not successful, channel switched off
	rtxCH_TYPE_STANDBY_SENT	Transmitter successfully set to standby mode
SignalStrength	RSSI level of the received data	
SequenceCounter	Sequence counter state when data have been received	
Data	Pointer to received data bytes or standby info byte	

Return Value:
Returns `true` if new data are available and `false` if no new data have been received.

Remarks:
It is recommended to call this function as many times as it returns `true` after `rtxRX_GetState()` indicates new data are available.

4.7.3 Get requested message data received on any RX channel

This function has to be used to get requested message data on any channel together with the signal strength. There is only one message data buffer per channel, so received data should be read before requesting further message data on a channel. Also indicates a failed message data request.

```
bool rtxRX_GetMessageData(unsigned char &Channel,
                          unsigned char &State;
                          unsigned char &SignalStrength,
                          unsigned char &Size,
                          unsigned char *Data);
```

Parameters:

`Channel`	Index of the channel the message data have been received on; starts at 0	
`State`	Message state:	
	`rtxCH_STATE_MSG_OFF`	Channel not in active mode, message will be sent as soon as channel is in active mode
	`rtxCH_STATE_MSG_PENDING`	Message sent, waiting for acknowledgement
	`rtxCH_STATE_MSG_ACK`	Sent message acknowledged
	`rtxCH_STATE_MSG_FAILED`	Sending message failed
`SignalStrength`	RSSI level of the received message data, set to 0 if message data request failed	
`Size`	Number of passed bytes	
`Data`	Pointer to buffer for passing data; has to be big enough to receive the number of requested bytes (max. 48 bytes of message data are possible)	

Return Value:
Returns `true` if new message data are available and `false` if no new message data have been received.

Remarks:
It is recommended to call this function as many times as it returns `true` after `rtxRX_GetState()` indicates new message data are available. To avoid a buffer overflow for “unexpected” pending messages it is recommended to always use a 48 byte buffer even if less bytes have been requested.

4.7.4 Get requested high speed data download received on any RX channel

This function has to be used to get a requested data download on any channel. There is only one data download buffer for all channels, so received data should be read before requesting the next data download.

```
bool rtxRX_GetDownload(unsigned char &Channel,
                       unsigned char &State;
                       unsigned char &SignalStrength,
                       unsigned long &Address,
                       unsigned long &Size,
                       unsigned char *Data);
```

Parameters:

`Channel`	Index of the channel the data download has been received on; starts at 0	
`State`	Data download state:	
	`rtxCH_STATE_DOWNLOAD_OFF`	Channel not in active mode, data download will be started as soon as channel is in active mode
	`rtxCH_STATE_DOWNLOAD_PENDING`	Data download in progress; received bytes available in `Size`
	`rtxCH_STATE_DOWNLOAD_READY`	Data download completed
	`rtxCH_STATE_DOWNLOAD_FAILED`	Data download failed
`SignalStrength`	RSSI level of the received data download, set to 0 if data download failed	
`Address`	Start address of downloaded data	
`Size`	Number of already processed bytes; has to be set to application buffer size to allow buffer overflow check; max supported buffer size is 512kBytes	
`Data`	Pointer to buffer for passing data; has to be big enough to receive the number of requested data bytes. Data will only be copied to buffer if download completed	

Return Value:
Returns `true` if a new data download is available and false if no new data have been received.

Remarks:
This function is required to be called only once after `rtxRX_GetState()` indicates new data or information as not more than one data download can be processed at the same time.

5 Integration Steps

5.1 Initialization

First of all the rtxBlueRobin DLL has to be initialized. Next steps are to scan for connected BlueRobin transceiver units or dongles, to get the USB device info of the device to be used, to open the port to that device and to initialize the device.

Here is an example of how to search for and then initialize either a BM-USBRTX4 unit or if that one cannot be found a BM-USBD1 dongle:

```
int FoundDevices;
rtxUsbDeviceInfo UsbDeviceInfo;

// BlueRobin DLL not already initialized?
if (!rtxDll_IsInitialized())
{
     // Initializing DLL required before calling any further DLL function
     rtxDll_Initialize();
}

// Check if BM-USBD1 or BM-USBRTX4 device connected
rtxInterface_ScanForDevices(bmiDEVTYPE_USBRTX4, FoundDevices);
// No BM-USBRTX4 connected?
if (FoundDevices == 0)
{
     rtxInterface_ScanForDevices(bmiDEVTYPE_USBD1, FoundDevices);
}
// Also no BM-USBD1 connected?
if (FoundDevices == 0)
{
     // No BlueRobin transceiver device found
     ...
}

// USB device info for first found device not available?
if (!rtxInterface_GetInfo(0, &UsbDeviceInfo))
{
     // BlueRobin transceiver device info not found
     ...
}

// Port cannot be opened?
if (!rtxDevice_Open(UsbDeviceInfo.PortName))
{
     rtxDevice_Close();
     // Opening port for BlueRobin transceiver device failed
     ...
}

// Initializing transceiver to x channels failed?
if (!rtxDevice_InitHardware(x))
{
     rtxDevice_Close();
     // Could not initialize BlueRobin transceiver device
     ...
}

// Start unplugg detection of transceiver device
rtxInterface_UnplugDetectionStart(UsbDeviceInfo.PortName);
```

Next the device information can be read from the opened transceiver, channel profiles have to be set and further parameters affecting all channels can be set:

```
rtxDevInfo DeviceInfo;
rtxPresetChannelProfile ProfileList[1];

// Get device information
rtxDevice_GetInfo(&DeviceInfo);
SerialNumber     = DeviceInfo.SerialNumber;
SoftwareRevision = DeviceInfo.SwRev;
HardwareRevision = DeviceInfo.HwRev;
MaxChannels      = DeviceInfo.MaxChannels;

// Set number of required channels to 20 and preset 10 of them with profile 33 and 10 with profile 99
ProfileList[0].Channels = 10;
ProfileList[0].Profile  = 33;
ProfileList[1].Channels = 10;
ProfileList[1].Profile  = 99;
rtxConfig_RxMode(20, ProfileList, 2);

// Infinite search on started channels
rtxConfig_SearchTimeout(0);
// Set number of lost packets in a row from a transmitter causing to search for it again to 6
rtxConfig_LinkFailureLimit(6);
```

For receiving a transmitter on a channel it is recommended to first reset the channel, to set its ID to the transceiver ID to be received if a specific transmitter has to be received and then to start the channel:

```
// Channel to default settings
rtxRX_ChannelReset(Channel);
// Set transmitter ID to be received, not needed if any transmitter has to be received
rtxRX_ChannelSetID(Channel, ID);
// Start channel
rtxRX_ChannelStart(Channel);
```

To stop receiving a transmitter on a channel the channel just has to be stopped:

```
// Stop channel
rtxRX_ChannelStop(Channel);
```

As soon as at least one channel has been started the state of the transceiver device should be checked periodically

```
WORD    wState;
BYTE    bChannel;
BYTE    bType;
BYTE    bSignalLevel;
BYTE    bSequenceCounter;
BYTE    bData[rtxMAX_MESSAGE_DATA_BYTES];
DWORD   dwID;

// Getting state successful?
if (rtxRX_GetState(wState))
{
     // New data available?
     if ((wState & rtxSTATE_RX_DATA) > 0)
     {
          // Process all new regular data or information
          while (rtxRX_GetData(bChannel, bType, bSignalLevel, bSequenceCounter, bData))
          {
               // Next action dependent on received data/information type
               switch (bType)
               {
                    // Data received
                    case rtxCH_TYPE_DATA:
                         // First get current ID of channel if not already read
                         if (...)
                         {
                              // Get ID of found device
                              if (rtxRX_ChannelGetID(bChannel, dwID))
                              {
                                   // Store ID
                                   ...
                              }
                         }

                         // Process received data
                         ...
                         break;

                    // Data packet lost
                    case rtxCH_TYPE_DATA_LOST:
                         // Process packet lost information
                         ...
                         break;

                    // Transmitter lost
                    case rtxCH_TYPE_TX_LOST:
                         // Does not happen when automatic search is set
                         break;

                    // Automatic search started again
                    case rtxCH_TYPE_TX_SEARCH:
                         // Process search start information
                         ...
               }
          }
     }
}
else // Access to port failed
{
     // Hardware error handling
     ...
}
```

When ending the application the opened port has to be closed and the rtxBlueRobin DLL has to be released:

```
// DLL already initialized?
if (rtxDll_IsInitialized())
{
     // Device already opened?
     if (rtxDevice_IsOpen())
     {
          // Stop all channels
          rtxRX_ChannelStop(rtxALL_CHANNELS);
          // Close COM port
          rtxDevice_Close();
     }
     // Unload DLL
     rtxDll_DeInitialize();
}
```

6 FCC and IC Statements

FCC § 15.19
This device complies with Part 15 of the FCC rules. Operation is subject to the following two conditions: (1) This device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

FCC § 15.21 (Warning Statement)
[Any] changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

FCC § 15.105
Note: This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:
—Reorient or relocate the receiving antenna.
—Increase the separation between the equipment and receiver.
—Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
—Consult the dealer or an experienced radio/TV technician for help.

Canada CNR-Gen Section 7.1.3
This device complies with Industry Canada licence-exempt RSS standard(s). Operation is subject to the following two conditions:(1) this device may not cause interference, and (2) this device must accept any interference, including interference that may cause undesired operation of the device.
Le présent appareil est conforme aux CNR d'Industrie Canada applicables aux appareils radio exempts de licence. L'exploitation est autorisée aux deux conditions suivantes : (1) l'appareil ne doit pas produire de brouillage, et (2) l'utilisateur de l'appareil doit accepter tout brouillage radioélectrique subi, même si le brouillage est susceptible d'en compromettre le fonctionnement.

ICES-003
This Class B digital apparatus complies with Canadian ICES-003.
Cet appareil numérique de la classe B est conforme à la norme NMB-003 du Canada.

RSS-Gen. 7.1.2:
The radio transmitter (IC: 8288A-BMUSBRTX4) has been approved by Industry Canaca to operate with the antenna types listed below with the maximum permissible gain and required antenna impedance for each antenna type indicated. Antenna types not included in this list, having a gain greater than the maximum gain indicated for that type, are strictly prohibited for use with this device.

Antenna: PSKN3-925RS Rev SMA Plug (Male), Mobile Mark UNITYGAIN (0 dB) or less

7 Change Log

Revision	Content	Made by	Date
1.0	Initial revision	RM	14 Jul 2014
1.1	RSS Gen Antenna use	HGB	29 Jul 2014