BioRadio Software Development Kit
BioRadio 150 DLL

Telephone: (216) 791-6720 or Toll-free 1-877-CleveMed (1-877-253-8363)
9:00 a.m. - 5:00 p.m. EST
Monday - Friday
Fax: (216) 791-6739

E-Mail: Customer Support: support@CleveMed.com
Sales: sales@CleveMed.com
Web: http://www.CleveMed.com

Mailing Address: Cleveland Medical Devices Inc.
4415 Euclid Avenue, Fourth Floor
Cleveland, Ohio 44103

Table of Contents

1. Introduction

The BioRadio 150 DLL (Dynamic Link Library) provides a software interface to the BioRadio 150 wireless physiological monitor, accommodating all necessary data acquisition and device configuration functionality. The library facilitates straightforward development of custom applications using the device.

This document outlines for the application developer the DLL interface and the steps involved in controlling and communicating with a BioRadio 150.

2. Example Use Cases

2.1. Programming the Device Configuration

2.1.1. Overview

Since all data interaction between the Computer Unit and User Unit is performed via the radio link, communication with the device must be underway in order for it to be programmed. Using the DLL interface, settings are programmed from a configuration file, an example of which is found later in this document. When the DLL is instructed to program the device with a new configuration, it first queries the device's current settings and determines what changes are necessary, to ensure minimal time is spent programming the device. Once the required changes are determined, each setting is sent and verified individually over the radio link. This verification ensures agreement between the loaded configuration file and the programmed device settings.

During device programming, the DLL optionally manages an update dialog detailing the current configuration progress.

2.1.2. Steps

1. Plug the Computer Unit into the computer's USB port.
2. Turn on the User Unit, ensuring the green LED on the front of the device is illuminated.
3. Create the BioRadio object by calling the `CreateBioRadio()` function, saving its returned handle to supply to later functions.
4. Start BioRadio communication by calling the `StartAcq()` function. In standard C-string format (such as "`COM1`"), provide to the `StartAcq()` function the name of the serial port to which the device is connected. This function returns 1 upon success and 0 upon failure. Upon success, the port will be opened and the device will be ready for operation.
5. Program the device by calling the `ProgramConfig()` function. If the parameter `displayProgressDialog` is greater than zero, the DLL will display an update screen detailing the current configuration progress. The parameter `Filename` should contain, in standard C-string format, the full path and filename of the configuration file with which the device is to be programmed.
6. Stop BioRadio communication by calling the `StopAcq()` function.

7. Delete the previously allocated the BioRadio object by calling the `DestroyBioRadio()` function.

2.2. Acquiring a Stream of Data

2.2.1. Overview

The BioRadio150 DLL allows the user to acquire on a periodic basis an interleaved stream of data points. These data points are reported as double-precision values measured in volts (V) and are scaled according to the currently-loaded device configuration (for more information on device configuration, see Programming the Device Configuration above). The device will return to the user up to 6,553,600 samples per request. It is important that the application using the DLL have its own timing mechanism to periodically request data from the device. Ideally, the device should be queried at a rate of between once every 25 milliseconds to once every second in order to minimize system overhead and to ensure available headroom in the data buffer.

2.2.2. Steps

1. Plug the Computer Unit into the computer's USB port.
2. Turn on the User Unit, ensuring the green LED on the front of the device is illuminated.
3. Create the BioRadio object by calling the `CreateBioRadio()` function, saving its returned handle to supply to later functions.
4. Start BioRadio communication by calling the `StartAcq()` function. In standard C-string format (such as "`COM1`"), provide to the `StartAcq()` function the name of the serial port to which the device is connected. This function returns 1 upon success and 0 upon failure. Upon success, the port will be opened and the device will be ready for operation.
5. Start the application's timer to control iterative data queries.
6. Call the `TransferBuffer()` function to instruct the device object to prepare its data buffer for reading. The return value will be the number of points in the buffer.
7. Dump the data to the application's data buffer using the `ReadScaled()` function. The parameter `Data` should be an allocated array of doubles; `Size` should be the size of the array `Data`; `NumRead` will be populated by the device with the actual number of samples read. Note that `Size` must greater than or equal to the value returned by `TransferBuffer()` in step 6.
8. Until data collection is finished, return to step 6. When data collection is finished, stop the application's timer.
9. Stop BioRadio communication by calling the `StopAcq()` function.
10. Delete the previously allocated the BioRadio object by calling the `DestroyBioRadio()` function.

3. Available Functions

3.1. Multiple BioRadio 150s and the Object Handle

The BioRadio 150 DLL allows for operation of and acquisition from multiple BioRadio 150s, simultaneously. Each time the `CreateBioRadio` function is called, a new device object is allocated and initiated, and a handle reference to the object is returned. Each subsequent function call that operates upon this object must be provided the object's handle. Each BioRadio 150 device should be connected to only a single object.

3.2. List of Fuctions

Below is a description of all functions available in the DLL for use by the application developer.

`DWORD` `CreateBioRadio()` – Allocates memory for a BioRadio150 object and returns a handle referencing its location.

`int` `DestroyBioRadio(DWORD BioRadio150Handle)` – Destroys the allocated BioRadio150 object.

`int` `StartAcq(DWORD BioRadio150Handle, char displayProgressDialog, char *PortName)` – Opens the COM port and starts communication with the BioRadio150 device. `displayProgressDialog` is a boolean (`>0` : true) flag indicating whether a progress dialog should be displayed while starting, `PortName` should be a C-string containing the name of the port to which the device is connected (e.g. "`COM1`").

`int` `StopAcq(DWORD BioRadio150Handle)` – Closes the COM port and stops communication with the BioRadio150 device.

`int` `LoadConfig(DWORD BioRadio150Handle, char *Filename)` – Assigns a new configuration to the BioRadio150 object, but does not program that configuration into the device. Filename should be a null-terminated C-string containing the complete path and file name of the configuration file.

`int` `PingConfig(DWORD BioRadio150Handle, char displayProgressDialog)` – Acquires current device configuration from a BioRadio150 with which communications/acquisition has been started. The BioRadio150 software object is populated with this information. `displayProgressDialog` is a boolean (`>0` : true) flag indicating whether a progress dialog should be displayed while starting. . `Filename` should be a null-terminated C-string containing the complete path and file name of the configuration file.

`int` `ProgramConfig(DWORD BioRadio150Handle, char displayProgressDialog, const char *Filename)` – Instructs the BioRadio150 object to load a configuration

from a file then program the device with the loaded configuration. `displayProgressDialog` is a boolean (>0 : true) flag indicating whether a progress dialog should be displayed while starting. `Filename` should be a null-terminated C-string containing the complete path and file name of the configuration file.

int `ReadScaled(DWORD BioRadio150Handle, double *Data, int Size, int *NumRead)` – Instructs the BioRadio150 object to copy its current data buffer into the application's data buffer, pointed to by the parameter `Data`. `Size` should be the capacity of the buffer `Data`. `NumRead` will be populated by the object with the number of samples actually copied. Note that `ReadScaled()` must be preceded by `TransferBuffer()`, and that `Size` must be greater than or equal to the value returned by `TransferBuffer()`.

int `TransferBuffer(DWORD BioRadio150Handle)` – Instructs the BioRadio150 object to prepare its data buffer for reading.

double `SetBadDataValue(DWORD BioRadio150Handle, double BadDataValue)` – Sets the default value with which the device's data stream is padded when packets are lost.

long `GetNumChannels(DWORD BioRadio150Handle)` – Returns the number of enabled acquisition channels in the currently loaded configuration.

char `GetEnabledChannels(DWORD BioRadio150Handle)` – Returns a byte-size boolean array, each bit of which indicating whether the corresponding channel is enabled in the currently loaded configuration. Bits 0 through 7 correspond to BioRadio channels 1 though 8 (the flag for channel 1 is the resultant's least significant bit).

int `GetFreqHoppingMode(DWORD BioRadio150Handle)` – Indicates whether or not the device is in Frequency-Hopping mode. (`1`: Frequency-Hopping mode *on*, `0`: Frequency-Hopping mode *off*, `-1`: error)

int `GetFreqHoppingModeIndicator()` – Returns the integer value that `GetRFChannel` will return if the device is in Frequency-Hopping mode (`-3`).

int `SetFreqHoppingMode(DWORD BioRadio150Handle, bool HoppingEnabled)` – Enabled or disable Frequency-Hopping mode. Supplying a `true` value for the `HoppingEnabled` parameter will enable Frequency-Hopping mode, while sending `false` will disable frequency-hopping and set the device to communicate in narrowband mode.

long `GetSampleRate(DWORD BioRadio150Handle)` – Returns the rate, in samples per second, at which data is being acquired, defined by the currently loaded configuration.

long `GetBitResolution(DWORD BioRadio150Handle)` – Returns the bit resolution (4, 8, 12 or 16 bits) defined by the currently loaded configuration.

DWORD `GetGoodPackets(DWORD BioRadio150Handle)` – Returns the number of valid data packets seen so far.

DWORD `GetBadPackets(DWORD BioRadio150Handle)` – Returns the number of invalid data packets seen so far.

DWORD `GetDroppedPackets(DWORD BioRadio150Handle)` – Returns the number of data packets missed so far.

int `GetUpRSSI(DWORD BioRadio150Handle)` – Returns the Received Signal Strength Indicator of the radio uplink. This value is scaled to between zero and 100.

int `GetDownRSSI(DWORD BioRadio150Handle)` – Returns the Received Signal Strength Indicator of the radio downlink. This value is scaled to between zero and 100.

int `GetLinkBufferSize(DWORD BioRadio150Handle)` – Returns the number of packets currently in the buffer on board the User Unit.

int `GetBitErrCount(DWORD BioRadio150Handle)` – Returns the bit error count of the radio link.

int `GetBitErrRate(DWORD BioRadio150Handle)` – Returns the bit error rate of the radio link.

int `GetRFChannel(DWORD BioRadio150Handle)` – Returns the narrowband RF channel on which the BioRadio 150 is communicating, or, if the device is in Frequency-Hopping mode, an integer indication of this state (see `GetFreqHoppingModeIndicator()`). A `-1` return value indicates an error in discovering the mode or channel.

int `SetRFChannel(DWORD BioRadio150Handle, int RFChannel)` – If the RFChannel parameter supplied is a valid channel choice, the BioRadio 150 is configured to communicate on this channel. If `RFChannel` is equal to the Frequency-Hopping mode indicator (see `GetFreqHoppingModeIndicator()`), the device is set to Frequency-Hopping mode. A `-1` return value indicates an improper channel, an error in setting the channel, or an error in enabling or disabling Frequency-Hopping mode.

int `GetUsableRFChannelList(int *UsableRFChannelList, int Size)` -
Populates the provided UsableRFChannelList integer array with the list of possible narrowband RF channel values, in increasing order. The `Size` parameter should indicate the size of the `UsableRFChannelList` array. The function returns the number of usable channels filled in `UsableRFChannelList`. If the `UsableRFChannelList` is not large enough to fit the entire list, the provided space will be filled and the function's return will be equal to `Size`.

4. Configuration Files

4.1. Fields in the Configuration File

4.1.1. System Settings

These settings are reported under the `[System]` section of the configuration file. They apply to the data acquisition system as a whole but may be overridden by individual channel settings, as typical with `BitResolution`.

`SystemSampleRate` – This refers to the number of samples per second acquired by the device, and will be between 5 and 1000.

`BitResolution` – This is the default resolution of acquired data samples. Though normally this will be equal to the `BitResolution` field on each of the DAQ Board settings, if there is a discrepancy between any channel and this value, the channel's value will take precedence.

`SweepsPerPacket` – This defines the number of sweeps of all channels per packet. For the BioRadio150, this value should always be 10.

`Format` – Unused. Set to 0.

4.1.2. DAQ Board Settings

These settings are reported under the `[DAQ Board]` section of the configuration file. Each channel has a string of settings labeled as "`Channel_XX`" where XX is the channel number counting up from zero.

`InputName` – The name to be used in displaying this channel; not stored in device hardware: exists in (with lifespan of) BioRadio150 software object.

`BitResolution` – The resolution, in bits, of samples recorded on this channel.

`Index` – Unused. Set to 0.

`SampleRate` – Unused. Set to `SystemSampleRate`.

`UpperScale` – The value in Volts corresponding to the maximum ADC value that can be reported by the device on this channel.

`LowerScale` - The value in Volts corresponding to the minimum ADC value that can be reported by the device on this channel. Usually equal to `UpperScale` * -1.0.

`ADCUpper` – The base-10 value of the maximum ADC value that can be reported by the device on this channel. Normally, this will be $2^{BitResolution} - 1$.

`ADCLower` – The base-10 value of the minimum ADC value that can be reported by the device on this channel. Normally, this will be 0.

`Truncate` – Unused. Set to 0.

`RangeIndex` – The index of the available input ranges by which the data on this channel should be scaled.

`Enabled` – Boolean value indicating whether or not this channel should report data.

`InputType` – Unused. Set to 0.

4.2. Sample Configuration File

```
[System]
SystemSampleRate=600
BitResolution=12
SweepsPerPacket=10
Format=0
[DAQ Board]
Channel_00=InputName=Ch1,BitResolution=12,Index=0,SampleRate=600,UpperScale=0.000835,L
owerScale=-
0.000835,ADCUpper=4095,ADCLower=0,Truncate=0,RangeIndex=0,Enabled=1,InputType=0
Channel_01=InputName=Ch2,BitResolution=12,Index=0,SampleRate=600,UpperScale=0.000835,L
owerScale=-
0.000835,ADCUpper=4095,ADCLower=0,Truncate=0,RangeIndex=0,Enabled=1,InputType=0
Channel_02=InputName=Ch3,BitResolution=12,Index=0,SampleRate=600,UpperScale=0.000835,L
owerScale=-
0.000835,ADCUpper=4095,ADCLower=0,Truncate=0,RangeIndex=0,Enabled=1,InputType=0
Channel_03=InputName=Ch4,BitResolution=12,Index=0,SampleRate=600,UpperScale=0.000835,L
owerScale=-
0.000835,ADCUpper=4095,ADCLower=0,Truncate=0,RangeIndex=0,Enabled=1,InputType=0
Channel_04=InputName=Ch5,BitResolution=12,Index=0,SampleRate=600,UpperScale=0.000835,L
owerScale=-
0.000835,ADCUpper=4095,ADCLower=0,Truncate=0,RangeIndex=0,Enabled=1,InputType=0
Channel_05=InputName=Ch6,BitResolution=12,Index=0,SampleRate=600,UpperScale=0.000835,L
owerScale=-
0.000835,ADCUpper=4095,ADCLower=0,Truncate=0,RangeIndex=0,Enabled=1,InputType=0
Channel_06=InputName=Ch7,BitResolution=12,Index=0,SampleRate=600,UpperScale=0.000835,L
owerScale=-
0.000835,ADCUpper=4095,ADCLower=0,Truncate=0,RangeIndex=0,Enabled=1,InputType=0
Channel_07=InputName=Ch8,BitResolution=12,Index=0,SampleRate=600,UpperScale=0.000835,L
owerScale=-
0.000835,ADCUpper=4095,ADCLower=0,Truncate=0,RangeIndex=0,Enabled=1,InputType=0
```

5. Sample Program

The following sample code will use the BioRadio150DLL to acquire data from an attached BioRadio 150 once every half second. The program sets (obviously editable) presumptions of the presence of a BioRadio 150 on serial port COM4, as well as a valid configuration file "`testcfg1.ini`" located in the directory "`C:\CleveMed\BioRadio\`". Device configuration is pinged, then programmed to the configuration file if the number of channels is less than 4.

```
//---------------------------------------------------------------------------

#include <vcl.h>
#include <iostream.h>
#include <conio.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused

#ifndef _COMMAND_LINE_DLL_TEST_H_
#define _COMMAND_LINE_DLL_TEST_H_

extern "C" __declspec(dllimport) DWORD _stdcall CreateBioRadio();
extern "C" __declspec(dllimport) int _stdcall DestroyBioRadio(DWORD);
extern "C" __declspec(dllimport) int _stdcall StartAcq(DWORD, char, char *);
extern "C" __declspec(dllimport) int _stdcall StopAcq(DWORD);
extern "C" __declspec(dllimport) int _stdcall LoadConfig(DWORD, char *);
extern "C" __declspec(dllimport) int _stdcall PingConfig(DWORD, char);
extern "C" __declspec(dllimport) int _stdcall ProgramConfig(DWORD, char, const char *);
extern "C" __declspec(dllimport) int _stdcall ReadScaled(DWORD, double *, int, int *);
extern "C" __declspec(dllimport) int _stdcall TransferBuffer(DWORD);
extern "C" __declspec(dllimport) double _stdcall SetBadDataValue(DWORD, double);
extern "C" __declspec(dllimport) long _stdcall GetNumChannels(DWORD);
extern "C" __declspec(dllimport) char _stdcall GetEnabledChannels(DWORD);
extern "C" __declspec(dllimport) int _stdcall GetFreqHoppingMode(DWORD);
extern "C" __declspec(dllimport) int _stdcall GetFreqHoppingModeIndicator();
extern "C" __declspec(dllimport) int _stdcall SetFreqHoppingMode(DWORD, bool);
extern "C" __declspec(dllimport) long _stdcall GetSampleRate(DWORD);
extern "C" __declspec(dllimport) long _stdcall GetBitResolution(DWORD);
extern "C" __declspec(dllimport) DWORD _stdcall GetGoodPackets(DWORD);
extern "C" __declspec(dllimport) DWORD _stdcall GetBadPackets(DWORD);
extern "C" __declspec(dllimport) DWORD _stdcall GetDroppedPackets(DWORD);
extern "C" __declspec(dllimport) int _stdcall GetUpRSSI(DWORD);
extern "C" __declspec(dllimport) int _stdcall GetDownRSSI(DWORD);
extern "C" __declspec(dllimport) int _stdcall GetLinkBufferSize(DWORD);
extern "C" __declspec(dllimport) int _stdcall GetBitErrCount(DWORD);
extern "C" __declspec(dllimport) int _stdcall GetBitErrRate(DWORD);
extern "C" __declspec(dllimport) int _stdcall GetRFChannel(DWORD);
extern "C" __declspec(dllimport) int _stdcall SetRFChannel(DWORD, int);
extern "C" __declspec(dllimport) int _stdcall GetUsableRFChannelList(int *, int);

#endif //_COMMAND_LINE_DLL_TEST_H_
//---------------------------------------------------------------------------
```

```
int main()
{
  /* DEFINE DEVICE SETTINGS AND METRICS */
  char *configFilePath = "C:\\CleveMed\\BioRadio\\testcfg1.ini";
  char *portName = "COM4";
  double bufSizePerChannel = 2000;
  const int BUF_SIZE_MAX = 2000*8;
  DWORD BioRadio150Handle;

  BioRadio150Handle = CreateBioRadio(); /* CREATE DEVICE OBJECT */
  cout << endl << "CreateBioRadio... (" << BioRadio150Handle << ")" << endl;
  if (BioRadio150Handle) {
    bool SuccessFlag = StartAcq(BioRadio150Handle, 1, portName); /* START ACQUISITION */
    cout << "StartAcq... (" << SuccessFlag << ")" << endl;
    if (SuccessFlag)
    {
      SuccessFlag = PingConfig(BioRadio150Handle, 1); /* PING DEVICE CONFIGURATION */
      cout << "PingConfig... (" << SuccessFlag << ")" << endl;
      if (SuccessFlag)
      {
        long numChannels = GetNumChannels(BioRadio150Handle);
        cout << "Channels = " << numChannels << endl;
        if (numChannels <= 4)
        {
               /* PROGRAM DEVICE CONFIG */
          SuccessFlag = ProgramConfig(BioRadio150Handle, 1, configFilePath);
          cout << "ProgramConfig... (" << SuccessFlag << ")\n" << endl;
        }
        if (SuccessFlag)
        {
          long numChannels = GetNumChannels(BioRadio150Handle);
          cout << "Channels = " << numChannels << endl;
          cout << "Sample Rate = " << GetSampleRate(BioRadio150Handle) << endl;

          double buffer[BUF_SIZE_MAX];

          int readSize, readScaledReturn;

          /* REQUEST DATA ONCE EVERY HALF SECOND UNTIL A KEY IS PRESSED */
          while (!kbhit())
          {
            cout << "TsfrBuf... (" << TransferBuffer(BioRadio150Handle) << ")" << endl;
            readScaledReturn = ReadScaled(BioRadio150Handle, buffer,
                                  bufSizePerChannel*numChannels, &readSize);
            cout << "ReadScaled... (" << readScaledReturn << ")" << endl;
            cout << "readSize = " << readSize << endl;
            Sleep(80);
          }
        }
      }
      /* STOP ACQUISITION */
      cout << endl << "StopAcq... (" << StopAcq(BioRadio150Handle) << ")" <<  endl;
    }

    /* CLEAN UP INSTANCE OF BIORADIO OBJECT */
    SuccessFlag = DestroyBioRadio(BioRadio150Handle);
    cout << "DestroyBioRadio... (" << SuccessFlag << ")\n" << endl;
  }
  return 1;
}
```