Networked Receiver Controller

NRC-2.0.1

Programmer's Guide

11th October 2005

DUNS: 826771508
CAGE: 1RKF1

www.aegis-inc.net
Telecom Engineering Services & Products
8610 Washington Blvd. Suite 213 Jessup, MD 20794
240.568.9006 240.568.9008 *[fax]*

Proprietary Statement

Disclaimer

Information furnished in this manual is believed to accurate and reliable. However, no responsibility is assumed for its use, or for any infringements of patents or other rights of third parties that may result from its use.

Warning

This equipment utilizes voltages which are potentially dangerous and may be fatal if contacted. Exercise caution when working with the equipment any time the protective cover is removed.

Table of Contents

Welcome 5
Contents 5
System Requirements 6
Sample Applications 7
Building the samples 7
Running the samples 7
Understanding the samples 7
NRC C++ SDK API Reference 13
Class Channel 13
Class NRC 30
Class SampleProcessor 38
Class NRCRPMConstants 40
NRC Java SDK API Reference 62
Class ChannelBean 62
Class NRCBean 83
Class NRCException 95
Interface SampleProcessor 98
Interface RPMConstants 99
Appendix A - Interface Control Document 126
RPM Protocol 126
Commands 127
Errors 142

Welcome

Contents

The NRC software development kit (SDK) contains many tools to help develop client applications designed to utilize the NRC. These include the following:

Applications

- *NRC Java Client*
  This application provides an example of an NRC client. Refer to the User's Guide for detailed documentation on this application.

Documentation

- *Networked Receiver Controller Users' Guide* (`/docs/UsersGuide.rtf`)
  The users' guide contains general information and specifications for the NRC as well as detailed instructions for its installation and operation.
- *Networked Receiver Controller Programmers' Guide* (`/docs/ProgrammersGuide.rtf`)
  This document helps the programmer in developing client software for the NRC.
- *NRC C++ SDK API Reference* (`/docs/sdk/cpp/index.html`)
  This document is the API reference for the NRC C++ SDK. The C++ SDK encapsulates communication with the NRC in a set of well documented classes. It is generated using cppdoc[1].
- *NRC Common Library API Reference* (`/docs/common/index.html`)
  This document is the API reference for the NRC Aegis common libraries. The common libraries are a low level set of classes which may be useful in developing your applications. The documentation was generated by cppdoc.
- *NRC Java SDK API Reference* (`/docs/sdk/java/index.html`)
  This document is the API reference for the NRC Java SDK. The Java SDK encapsulates communication with the NRC in a set of well documented classes. It is generated using javadoc[2].

NRC Client SDK Libraries

- *NRC C++ SDK Library* (`/lib/libNRC.a`)
  This static library should be linked to C++ NRC client applications in the standard manner. See the sample Makefile for an example.
- *NRC Java SDK Library* (`/lib/NRCBean.jar`)
  This Java archive should be included in the `CLASSPATH` during compilation and execution of Java-based NRC client applications.

[1] Refer to www.cppdoc.com
[2] Refer to http:// java.sun.com/j2se/javadoc

Client SDK Header files

Client applications developed using the NRC C++ SDK require the use of header files during compilation. Refer to the `/include` directory of the SDK for available header files. Additional documentation on these files can be found in the NRC C++ SDK API Reference section of this document.

Sample client applications

The SDK provides two sample client applications; one written in C++, the other in Java. Both applications provide the same functionality represented in each programming language. The samples are contained in the `/samples` directory of the SDK. For a detailed look at the samples, refer to the Sample Applications chapter below.

System Requirements

C++ client applications

The following libraries are required to use the NRC C++ SDK:

- libstdc++-libc6.2-2.so.3
- libpthread.so.0
- libm.so.6
- libc.so.6

Java client applications

The Java SDK Library has been developed and tested using the Sun Java 2 SDK, version 1.4.1.

Sample Applications

Two sample applications are provided in the SDK to illustrate the use of the NRC client application programming interface (API). The two applications perform the same operations; one sample is written in C++ and contained in the file `Sample1.cpp`, the other is in Java and is found in `Sample1.java`. The source code for the samples can be found in the /samples directory of the SDK.

Building the samples

Before building the samples, be sure the system conforms to the requirements outlined in the System Requirements section above. In addition, the `PATH` environment variable should point to the `bin` directory of the Java 2 SDK, as well as to the C++ compiler.
The samples can be built using the standard make utility. From /samples directory, run:

```
make
```

To individually make the C++ sample application, run:

```
make cppsample
```

To individually make the Java sample application, execute:

```
make javasample
```

Running the samples

The Java sample application can be run using the following command:

```
java -classpath .:../lib/NRCBean.jar Sample1
```

The C++ sample application can be run by executing the following command:

```
./Sample1
```

Understanding the samples

While the sample applications do not exercise all of the available APIs, they do attempt to touch on the various types of calls that are available. This section will explain the samples step-by-step.
For a complete list of available APIs, refer to the API reference section of this document.

Accessing the NRC SDK libraries

Sample1.cpp	Sample1.java
`#include "../include/nrc/NRC.h"`	`import com.aegis.NRC.sdk.java.*;`
C++ applications should include the appropriate header files found in the `/include/nrc` directory of the SDK. See the NRC C++ SDK API Reference for the list of available APIs and the required header files for each API.	Java applications should import the `com.aegis.NRC.sdk.java` package and include `/lib/NRCBean.jar` in the `CLASSPATH` during compilation.

Connecting to the NRC

Sample1.cpp

```
NRC* nrc = new NRC;

/**
 * Obtain the NRC host from the user
 */
char in[256];
in[0] = 0;
std::cout << "Enter NRC address: ";
std::cin >> in;




/**
 * Connect to the NRC
 */
if (!nrc->connectToNRC(in))
{
  std::cerr << "Could not connect to NRC:"
            << in << std::endl;
  return 1;
}
```

Connections to the NRC are controlled through an instance of the `NRC` class. Use the `connectToNRC()` method with an appropriate host name or IP address to connect to an NRC.

Sample1.java

```
NRCBean nrc = new NRCBean();

/**
 * Obtain the NRC host from the user
 */
BufferedReader br = new BufferedReader(
    new InputStreamReader(System.in));

System.out.print("Enter NRC address: ");
String in = "";
try {
  in = br.readLine();
} catch (java.io.IOException ioe) {
  System.out.println("Could not read
input");
}

/**
 * Connect to the NRC
 */
if (!nrc.connectToNRC(in)) {
  System.out.println(
    "Could not connect to NRC: " + in);
  return;
}
```

Connections to the NRC are controlled through an instance of the `NRCBean` class. Use the `connectToNRC()` method with an appropriate host name or IP address to connect to an NRC.

Connecting to a channel

Connecting to a channel is often a prerequisite to accessing or modifying properties on that channel. Only one client application can be connected to a channel at any instance. When one client disconnects from a channel, another client can then connect to that channel.

Sample1.cpp

```
/**
 * Connect to the next available channel
 */
Channel* c = nrc->connectToChannel();
if (c == NULL)
{
  std::cerr << "No channels available."
            << std::endl;
  return 0;
}

std::cout << "Connected to channel "
          << c->getID()
          << "." << std::endl;
```

Sample1.java

```
/**
 * Connect to the next available channel
 */
ChannelBean c;
try {
  c = nrc.connectToChannel();
} catch (NRCException nrce) {
  System.out.println(
    "Could not connect to a channel");
  return;
}

if (c == null) {
  System.out.println("No channels
available.");
  return;
```

	} System.out.println("Connected to channel" + c.getID() + ".");
NRC channels are controlled through an instance of the `Channel` class. Use the `connectToChannel()` method to connect to the next available channel and obtain a pointer to that channel. However, if the call returns `NULL`, no channels are available. See the API reference for connecting to channels based on specific criteria.	NRC channels are controlled through an instance of the `ChannelBean` class. Use the `connectToChannel()` method to connect to the next available channel and obtain a reference to that channel. However, if the call returns `null`, no channels are available. In addition, the Java API uses exception handling to report error conditions. Details on the error can be obtained by referencing the exception. See the API reference for connecting to channels based on specific criteria.

Getting channel settings

Sample1.cpp

```
/**
 * Get the list of available detection
modes
 */
std::string* list;
int length;
std::cout << "Available detection modes:";
if (c->getDetectionModeList(list, length))
{
  for (int i = 0; i < length; i++)
    std::cout << list[i] << " ";

  delete[] list;
  std::cout << std::endl;
}
else
  std::cout << "none." << std::endl;









/**
 * Get the existing receiver settings
 */
std::cout << "Current receiver settings: "
          << std::endl;
if ( c->isPiggybacking() )
        std::cout << " – Piggybacking"
                  << std::endl;
std::cout << " - Frequency: "
          << c->getFrequency()
          << std::endl;
std::cout << " - Detection mode: "
```

Sample1.java

```
/**
 * Get the list of available detection
modes
 */
String[] list;
int length;
try {
  list = c.getDetectionModeList();
} catch (NRCException nrce) {
  System.out.println(
    "Could not get detection mode list");
  return;
}

System.out.print(
    "Available detection modes: ");
if (list != null) {
  for (int i = 0; i < list.length; i++)
    System.out.print(list[i] + " ");

  System.out.println();
}
else
  System.out.println("none.");

/**
 * Get the existing receiver settings
 */
try {
  System.out.println(
    "Current receiver settings: ");
  if ( !c.isWritable() )
    System.out.println(" - Piggybacking");
  System.out.println(
    " - Frequency: " + c.getFrequency());
  System.out.println(
```



```
          << c->getDetectionMode()
          << std::endl;
std::cout << " - BFO: " << c->getBFO()
          << std::endl;
std::cout << " - IFBW: "
          << c->getIFBandwidth()
          << std::endl;
std::cout << " - Receiver status: "
          << c->getReceiverStatus()
          << std::endl;
```

```
    " - Detection mode: "
    + c.getDetectionMode());
  System.out.println(
    " - BFO: " + c.getBFO());
  System.out.println(
    " - IFBW: " + c.getIFBandwidth());
  System.out.println(
    " - Receiver status: "
    + c.getReceiverStatus());
} catch (NRCException nrce) {
  System.out.println(
    "Error getting receiver settings.");
  return;
}
```

`NRC`and `Channel` instances provide methods to get many of the current operating properties of the NRC, channel, and receiver. Refer to the API reference for the list of the available properties.

`NRCBean`and `ChannelBean` instances provide methods to get many of the current operating properties of the NRC, channel, and receiver. Refer to the API reference for a list of the various properties available.
Note again the use of exception handling in the Java code.

Adjusting channel settings

Sample1.cpp

```
/**
 * Tune the receiver
 */
double freq;
std::cout << "Please enter new frequency \
             (0.00-30.00 MHz): ";
std::cin >> freq;

if (!c->setFrequency(freq))
{
  std::cerr << "Unable to change
frequency"
            << std::endl;
  return 1;
}
else
  std::cout << "Frequency changed." <<
std::endl;
```

Sample1.java

```
/**
 * Tune the receiver
 */
double freq;
System.out.print("Please enter new
frequency: ");
String freqStr = "";
try {
  freqStr = br.readLine();
} catch (java.io.IOException ioe) {
  System.out.println("Could not read
input");
}

try
{
  freq = Double.parseDouble(freqStr);
  c.setFrequency(freq);
} catch (NumberFormatException nfe) {
  System.out.println("Unable to change \
                      frequency: bad
number");
  return;
} catch (NRCException nrce) {
  System.out.println("Unable to change \
                      frequency");
  return;
}
System.out.println("Frequency changed.");
```

In addition, the same instances can be used to adjust NRC, channel, and receiver properties.

In addition, the same instances can be used to adjust NRC, channel, and receiver properties.

This portion shows how to change the receiver frequency through the Channel instance. Please refer to the API reference for the list of available properties.

This portion shows how to change the receiver frequency through the ChannelBean instance. Please refer to the API reference for the list of available properties.

Capturing audio data

APIs are provided which capture audio and forward it to a specific destination, such as a file or socket. The example demonstrates capturing audio data to a file. In addition to specifying the destination, capture APIs allow the client to specify the sample rate (8/16 KHz), the number of samples to receive at a time, a data filter option, and a time stamp option. For example, if you sample at 16KHz and specify 16000 samples to receive at a time, with FILTERING_OFF you can expect to receive 1 packet per second from the NRC of data of raw un-processed receiver data. Adjusting the sample rate/number of samples settings is useful in time-sensitive applications, while adjusting the filter option allows pre-filtered data to be forwarded. Using the TIMESTAMP_ON option causes the NRC to prefix each data packet with a TAI64N timestamp (12 bytes, nanosecond precision) representing the time that the last data sample was taken. However, these time stamps are not saved to the data file when using captureDataToFile.

Sample1.cpp

```
/**
 * Capture some audio data from this
 * channel
 */
std::ostringstream oss;
oss << "AudioChannel" << c->getID()
    << ".dat";
std::string filepath = oss.str();

std::cout << "Capturing 5 seconds of \
             audio data to file: "
          << filepath
          << " ..." << std::endl;
if (!c->captureDataToFile(filepath,
         Channel::SAMPLE_RATE_16KHZ, 512,
         Channel::FILTERING_OFF,
         Channel::TIMESTAMP_OFF))
{
  std::cerr << "Failure to capture \
               audio data."
            << std::endl;
  return 1;
}




/**
 * Sleep for 5 seconds to capture data
 */
sleep(5);
```

Sample1.java

```
/**
 * Capture some audio data from this
 * channel
 */
System.out.println("Capturing 5 seconds \
                    of audio data to \
                    file:" +
                    "AudioChannel"
                    + c.getID() +
                    ".dat" + " ...");
try {
  if (!c.captureDataToFile(
         new File("AudioChannel"
           + c.getID() + ".dat"),
         ChannelBean.SAMPLE_RATE_16KHZ,
         512, ChannelBean.FILTERING_OFF,
         ChannelBean.TIMESTAMP_OFF))
  {
    System.out.println("Failure to \
                        capture \
                        audio data.");
    return;
  }
} catch (NRCException nrce) {
  System.out.println("Could not capture \
                      audio data");
  return;
}

/**
 * Sleep for 5 seconds to capture data
 */
try {
  Thread.sleep(5000);
} catch (InterruptedException ie) {
  System.out.println("5 second wait \
                      interrupted.");
```



```
/**
 * Stop audio capture
 */
if (!c->stopDataCapture())
  std::cout << "Failure to stop audio \
               data capture."
            << std::endl;
else
  std::cout << "Audio data capture \
               complete."
            << std::endl;
```

```
}

/**
 * Stop audio capture
 */
try {
  if (!c.stopDataCapture())
    System.out.println("Failure to stop \
                        audio \
                        data capture.");
  else
    System.out.println("Audio data \
                        capture \
                        complete.");
} catch (NRCException nrce) {
  System.out.println("Failure to stop \
                      audio \
                      data capture.");
}
```

Use the `Channel` instance to start and stop data capture. See the API reference for methods of capturing data to various destinations.

Use the `ChannelBean` instance to start and stop data capture. See the API reference for methods of capturing data to various destinations.

Disconnecting from the NRC

Sample1.cpp

```
/**
 * Disconnect from channel and NRC
 */
nrc->disconnectFromChannel();
nrc->disconnectFromNRC();
delete nrc;
```

Sample1.java

```
/**
 * Disconnect from channel and NRC
 */
try {
  nrc.disconnectFromChannel();
} catch (NRCException nrce) {
  System.out.println("Could not \
                      disconnect \
                      from channel");
}

nrc.disconnectFromNRC();
```

Release a channel using the `disconnectFromChannel()` method of the `NRC` class. The client can then attach to a different channel, if desired. Disconnect from the NRC using the `disconnectFromNRC()` method.

Release a channel using the `disconnectFromChannel()` method of the `NRCBean` class. The client can then attach to a different channel, if desired. Disconnect from the NRC using the `disconnectFromNRC()` method.

NRC C++ SDK API Reference

Class Channel

class **Channel**

The `Channel` class encapsulates a single channel in the NRC. A channel is defined as a duplex data stream that can be connected to one of the NRC receivers. The receiver can be controlled through the channel, and its digitized audio data can be received through the channel. The Channel class provides methods that allow this functionality to occur. Instances of the Channel class can be obtained via the `getChannel()` or `connectToChannel()` methods of the `NRC` class.
Please note that most operations on a channel are only allowed if that channel is connected to the client using the `NRC::connectToChannel()` methods.

Field Summary

`static const int`	**`FILTERING_OFF`** Constant: defines No Channel Data Filtering
`static const int`	**`LPF_4KHZ_ON`** Constant: defines Channel Data filtering through a LPF with cut-off frequency of 4KHz
`static const int`	**`SAMPLE_RATE_16KHZ`** Constant: defines 16KHz sampling rate
`static const int`	**`SAMPLE_RATE_8KHZ`** Constant: defines 8KHz sampling rate
`static const int`	**`TIMESTAMP_OFF`** Constant: defines when TAI64N time stamp is off
`static const int`	**`TIMESTAMP_ON`** Constant: defines when TAI64N time stamp is on

Constructor Summary

`private`	**`Channel`**`( int ID, NRC* owner )` Constructor.
`private`	**`~Channel`**`()` Destructor

Method Summary

`bool`	**`captureDataToFile`**`( std::string file, int sampleRate,`

	`int samplesPerPacket, int filterOption = LPF_4KHZ_ON, int timeStampOption = TIMESTAMP_OFF )` Starts capturing digitized audio data from the receiver attached to this channel and sends it to the specified file.
`bool`	**`captureDataToProcessor`**`( SampleProcessor* p, int sampleRate, int samplesPerPacket, int filterOption = LPF_4KHZ_ON, int timeStampOption = TIMESTAMP_OFF )` Starts capturing digitized audio data from the receiver attached to this channel and sends it to the specified processor.
`bool`	**`captureDataToSocket`**`( std::string hostname, int port, int sampleRate, int samplesPerPacket, int filterOption = LPF_4KHZ_ON, int timeStampOption = TIMESTAMP_OFF )` Starts capturing digitized audio data from the receiver attached to this channel and forwards it to the server socket specified by the hostname and port.
`std::string`	**`changeAntenna`**`( std::string antenna )` Attempts to change the name of the antenna that is attached to this channel.
`std::string`	**`changeReceiverModel`**`( std::string model )` Attempts to change the receiver that is attached to this channel to a different manufacturer's model.
`std::string`	**`getAGC`**`()` Returns the AGC setting of the receiver attached to this channel.
`std::string`	**`getAntenna`**`()` Returns the name of the antenna currently attached to this channel.
`int`	**`getBFO`**`()` Returns the BFO setting (in Hz) of the receiver attached to this channel.
`std::string`	**`getCaptureFile`**`()` Returns the name of the file that data is currently being captured to.
`std::string`	**`getCaptureHost`**`()` Returns the hostname of the computer that is receiving captured data.
`int`	**`getCapturePort`**`()` Returns the socket port of the computer that is receiving captured data.
`std::string`	**`getClientLocation`**`()` Returns the location of the channel client.
`std::string`	**`getDetectionMode`**`()` Returns the detection mode setting of the channel receiver.

bool	**getDetectionModeList**(std::string*& modes, int& length) Returns the list of valid detection modes of the receiver attached to this channel.
int	**getFilterOption**() Returns the filter option associated with the channel's data collection Refer to the class constants for the range of valid return values.
double	**getFrequency**() Returns the frequency setting in MHz of the channel receiver.
int	**getID**() Returns the channel ID.
int	**getIFBandwidth**() Returns the current IF bandwidth setting (in Hz) of the receiver attached to this channel.
int	**getMemoryCapacity**()
NRC*	**getNRC**() Returns the NRC class that contains this channel
int	**getOnlinePollInterval**() Returns the online polling interval, in seconds, of the channel receiver.
std::string	**getReceiverConfiguration**() Returns a string that contains general information about the capabilities of the receiver attached to the channel.
std::string	**getReceiverModel**() Returns the manufacturer's model designation of the receiver currently attached to this channel.
bool	**getReceiverModelList**(std::string*& models, int& length) Returns the list of valid receiver models that can be attached to this channel of the NRC.
std::string	**getReceiverStatus**() Returns a string that defines the status of the receiver attached to this channel.
int	**getSampleRate**() Returns a constant representing the capture sample rate (in Hz) setting for this channel.
int	**getSamplesPerPacket**() Returns the number of samples per packet that should be received when capturing digitized receiver audio data.
int	**getSamplesSkippedOnTune**()

	Returns the number of samples to skip following a change in receiver settings, such as a change in frequency, detection mode, IF bandwidth, and BFO.
int	**getTimeStampOption**() Returns the filter option associated with the channel's data collection Refer to the class constants for the range of valid return values.
bool	**getWritable**() Function checks to see if the receiver's settings may be changed
bool	**isCapturing**() Returns the state of data capture for this channel.
bool	**isOnline**() Returns true if this channel receiver is online, false otherwise.
bool	**isPiggybacking**() This function is just a wrapper for `Channel::getWritable()`
bool	**loadMemoryLocation**(int loc) Load the receiver settings stored at a memory location
std::string	**queryMemoryFields**()
std::string	**queryMemoryLocation**(int loc) Return the settings stored in a receiver memory location
bool	**resetReceiver**() Resets the receiver to the default state from the manufacture.
bool	**sendReceiverCommand**(std::string cmd) Sends the given command directly to the channel receiver.
std::string	**sendReceiverRequest**(std::string cmd) Sends the given command directly to the channel receiver and returns a response.
bool	**setAGCMode**(const std::string& agcmode) Adjusts the AGC mode setting of the channel receiver.
bool	**setBFO**(int bfo) Adjusts the BFO setting (in Hz) of the channel receiver.
bool	**setDataOptions**(int sampleRate, int samplesPerPacket, int filterOption = LPF_4KHZ_ON, int timeStampOption = TIMESTAMP_OFF) Adjust options for digitized audio data capture.
bool	**setDetectionMode**(std::string detmode) Adjusts the detection mode setting of the channel receiver.

bool	**setFrequency**(double freq) Tunes the channel receiver to the specified frequency (in MHz).
bool	**setIFBandwidth**(int ifbw) Adjusts the IF bandwidth setting (in Hz) of the receiver attached to this channel.
bool	**setOnlinePollInterval**(int interval) Sets the interval in seconds at which this channel receiver will be tested to see if it is online.
bool	**setSamplesSkippedOnTune**(int skip) Sets the number of samples to skip following a change in receiver settings, such as a change in frequency, detection mode, IF bandwidth, and BFO.
bool	**stopDataCapture**() Stops capturing digitized audio data from the receiver attached to this channel.
bool	**writeMemoryLocation**(int loc) Write the current receiver settings to a memory location

Field Detail

FILTERING_OFF

```
public static const int FILTERING_OFF;
```

Constant: defines No Channel Data Filtering

LPF_4KHZ_ON

```
public static const int LPF_4KHZ_ON;
```

Constant: defines Channel Data filtering through a LPF with cut-off frequency of 4KHz

---

SAMPLE_RATE_16KHZ

```
public static const int SAMPLE_RATE_16KHZ;
```

Constant: defines 16KHz sampling rate

SAMPLE_RATE_8KHZ

```
public static const int SAMPLE_RATE_8KHZ;
```

Constant: defines 8KHz sampling rate

TIMESTAMP_OFF

```
public static const int TIMESTAMP_OFF;
```

Constant: defines when TAI64N time stamp is off

TIMESTAMP_ON

```
public static const int TIMESTAMP_ON;
```

Constant: defines when TAI64N time stamp is on

Constructor Detail

Channel

```
private Channel( int ID, NRC* owner );
```

Constructor. Instances of this class should not be created by the client, but rather by the NRC class.

Parameters:

`ID` - channel ID

`owner` - NRC that owns this channel

~Channel

```
private ~Channel();
```

Destructor

Method Detail

captureDataToFile

```
public bool captureDataToFile( std::string file, int sampleRate, int
samplesPerPacket, int filterOption = LPF_4KHZ_ON, int timeStampOption =
TIMESTAMP_OFF );
```

Starts capturing digitized audio data from the receiver attached to this channel and sends it to the specified file.

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Parameters:

`file` - the path of the file to capture to

`sampleRate` - constant representing the sample rate (in Hz) for capture (see class constants)

`samplesPerPacket` - the number of samples the NRC should collect before writing them to the file.

`filterOption` - the filter option associated with the channel data

`timeStampOption` - the time stamp option associated with the channel data

Returns:

true, if successful, false otherwise

---

captureDataToProcessor

```
public bool captureDataToProcessor( SampleProcessor* p, int sampleRate,
int samplesPerPacket, int filterOption = LPF_4KHZ_ON, int
timeStampOption = TIMESTAMP_OFF );
```

Starts capturing digitized audio data from the receiver attached to this channel and sends it to the specified processor.

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Parameters:

`p` - the processor to push samples to

`sampleRate` - the sample rate to use (see class constants)

`samplesPerPacket` - the samples per packet to request

`filterOption` - the filter option associated with the channel data

`timeStampOption` - the time stamp option associated with the channel data

Returns:

true, if successful, false otherwise

---

captureDataToSocket

```
public bool captureDataToSocket( std::string hostname, int port, int
sampleRate, int samplesPerPacket, int filterOption = LPF_4KHZ_ON, int
timeStampOption = TIMESTAMP_OFF );
```

Starts capturing digitized audio data from the receiver attached to this channel and forwards it to the server socket specified by the hostname and port. Note: multi-byte samples are received in network byte order.

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Parameters:

`hostname` - the host that contains the destination server socket

`port` - the port of the destination server socket

`sampleRate` - constant representing the sample rate (in Hz) for capture (see class constants)

`samplesPerPacket` - the number of samples the NRC should collect before writing them to the socket

`filterOption` - the filter option associated with the channel data

`timeStampOption` - the time stamp option associated with the channel data

Returns:

true, if successful, false otherwise

---

changeAntenna

```
public std::string changeAntenna( std::string antenna );
```

Attempts to change the name of the antenna that is attached to this channel.
This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Parameters:

`antenna` - the name of the antenna

Returns:

the new antenna name connected to the receiver of this channel.

---

changeReceiverModel

```
public std::string changeReceiverModel( std::string model );
```

Attempts to change the receiver that is attached to this channel to a different manufacturer's model. If the NRC supports that model, it will connect to the receiver and this channel will begin operating on the new receiver.
This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.
Note: the specified receiver model must match a model name known by the NRC. For a list of known receiver models, use `getReceiverModelList()`.

Parameters:

`model` - the manufacturer's model name of the new receiver.

Returns:

the new receiver model name.

---

getAGC

```
public std::string getAGC();
```

Returns the AGC setting of the receiver attached to this channel.
This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Returns:

the BFO setting (in Hz)

---

getAntenna

```
public std::string getAntenna();
```

Returns the name of the antenna currently attached to this channel.

Returns:

the antenna name connected to the receiver of this channel.

---

getBFO

```
public int getBFO();
```

Returns the BFO setting (in Hz) of the receiver attached to this channel.
This can only be performed if this channel has been acquired by the client using

an `NRC::connectToChannel()` call.

Returns:

the BFO setting (in Hz)

---

getCaptureFile

`public std::string` **`getCaptureFile`**`();`

Returns the name of the file that data is currently being captured to.

Returns:

the data capture file name, if capturing, null otherwise.

---

getCaptureHost

`public std::string` **`getCaptureHost`**`();`

Returns the hostname of the computer that is receiving captured data.

Returns:

the hostname, if data is being captured, null otherwise.

---

getCapturePort

`public int` **`getCapturePort`**`();`

Returns the socket port of the computer that is receiving captured data.

Returns:

the remote socket port, if capturing, null otherwise.

---

getClientLocation

`public std::string` **`getClientLocation`**`();`

Returns the location of the channel client. This is typically the hostname or IP address of the client application.

Returns:

the location of the client

---

getDetectionMode

`public std::string` **`getDetectionMode`**`();`

Returns the detection mode setting of the channel receiver.

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Returns:

the current detection mode

---

getDetectionModeList

`public bool` **`getDetectionModeList`**`( std::string*& modes, int& length );`

Returns the list of valid detection modes of the receiver attached to this channel.

Caller is responsible to delete modes array.
This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Parameters:

`modes` - [out] the returned modes as an array of strings
`length` - [out] the length of the array of modes

---

getFilterOption

```
public int getFilterOption();
```

Returns the filter option associated with the channel's data collection Refer to the class constants for the range of valid return values.

---

getFrequency

```
public double getFrequency();
```

Returns the frequency setting in MHz of the channel receiver.
This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Returns:

the receiver frequency, in MHz

---

getID

```
public int getID();
```

Returns the channel ID.

Returns:

the channel ID

---

getIFBandwidth

```
public int getIFBandwidth();
```

Returns the current IF bandwidth setting (in Hz) of the receiver attached to this channel.
This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Returns:

the current IF bandwidth setting (in Hz)

Throws:

`NRCException`

---

getMemoryCapacity

```
public int getMemoryCapacity();
```

Returns:

the number of memory locations the receiver supports

---

getNRC

```
public NRC* getNRC();
```

Returns the NRC class that contains this channel

Returns:

the NRC that contains this channel

---

getOnlinePollInterval

```
public int getOnlinePollInterval();
```

Returns the online polling interval, in seconds, of the channel receiver. A value of 0 means that polling is currently disabled. A negative value denotes that an error has occurred.

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Returns:

the online polling interval in seconds, or 0 if disabled

---

getReceiverConfiguration

```
public std::string getReceiverConfiguration();
```

Returns a string that contains general information about the capabilities of the receiver attached to the channel. For the format of this message, see the Network Receiver Controller Interface Control Document.

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Returns:

receiver status string

---

getReceiverModel

```
public std::string getReceiverModel();
```

Returns the manufacturer's model designation of the receiver currently attached to this channel.

Returns:

the receiver model name

---

getReceiverModelList

```
public bool getReceiverModelList( std::string*& models, int& length );
```

Returns the list of valid receiver models that can be attached to this channel of the NRC. The caller is responsible for deallocating the storage for the models array.

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Parameters:

`modes` - [out] the returned modes as an array of strings

`length` - [out] the length of the array of modes

getReceiverStatus

`public std::string **getReceiverStatus**();`

Returns a string that defines the status of the receiver attached to this channel. The format of the string is defined by the receiver. Please refer to the receiver operating manual for more information on how to use this string.
This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Returns:

receiver status string

---

getSampleRate

`public int **getSampleRate**();`

Returns a constant representing the capture sample rate (in Hz) setting for this channel. Refer to the class constants for the range of return values.

Returns:

the sample rate

---

getSamplesPerPacket

`public int **getSamplesPerPacket**();`

Returns the number of samples per packet that should be received when capturing digitized receiver audio data. Since each samples is two bytes, the actual size of the packet in bytes will be twice the number of samples per packet.

Returns:

the number of samples to be sent in each data capture packet

---

getSamplesSkippedOnTune

`public int **getSamplesSkippedOnTune**();`

Returns the number of samples to skip following a change in receiver settings, such as a change in frequency, detection mode, IF bandwidth, and BFO. This helps eliminate invalid audio data during the receiver transition.
This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Returns:

the number of samples to skip following receiver tune

---

getTimeStampOption

`public int **getTimeStampOption**();`

Returns the filter option associated with the channel's data collection Refer to the class constants for the range of valid return values.

---

getWritable

`public bool **getWritable**();`

Function checks to see if the receiver's settings may be changed

Returns:

true if the channel's configuration may be changed by this client. False indicates the client is "piggybacking".

---

isCapturing

`public bool **isCapturing**();`

Returns the state of data capture for this channel.

Returns:

true, if capturing, false otherwise.

---

isOnline

`public bool **isOnline**();`

Returns true if this channel receiver is online, false otherwise.

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Returns:

true, if online, false otherwise

---

isPiggybacking

`public bool **isPiggybacking**();`

This function is just a wrapper for `Channel::getWritable()`

Returns:

True if the client is piggybacking the channel, false otherwise.

---

loadMemoryLocation

`public bool **loadMemoryLocation**( int loc );`

Load the receiver settings stored at a memory location

Parameters:

`loc` - the location to load

---

queryMemoryFields

`public std::string **queryMemoryFields**();`

Returns:

the settings the receiver stores for each memory location

---

queryMemoryLocation

`public std::string **queryMemoryLocation**( int loc );`

Return the settings stored in a receiver memory location

Parameters:

`loc` - the location to query

---

resetReceiver

```
public bool resetReceiver();
```

Resets the receiver to the default state from the manufacture. It may take a couple of seconds before the receiver is internally re-initialized and is ready to accept tuning commands again.

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Returns:

true, if successful, false otherwise

---

sendReceiverCommand

```
public bool sendReceiverCommand( std::string cmd );
```

Sends the given command directly to the channel receiver. No response will be reported. Note: no translation of this command will occur; it is delivered to the receiver as-is.

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Parameters:

`cmd` - the receiver command

Returns:

true, if successful, false otherwise

---

sendReceiverRequest

```
public std::string sendReceiverRequest( std::string cmd );
```

Sends the given command directly to the channel receiver and returns a response. If the command does not generate a response, the receiver will briefly go offline and come back online. This function should only be called when a response from the receiver is needed by the application. Otherwise, use the `Channel::sendReceiverCommand` function Note: no translation of this command will occur; it is delivered to the receiver as-is.

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Parameters:

`cmd` - the receiver command

Returns:

the response from the receiver, or "" if an error occured

---

setAGCMode

```
public bool setAGCMode( const std::string& agcmode );
```

Adjusts the AGC mode setting of the channel receiver. The valid AGC modes are "SLOW", "MEDIUM", and "FAST".

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Parameters:

`agcmode` - the AGC mode

Returns:

true, if successful, false otherwise

---

setBFO

```
public bool setBFO( int bfo );
```

Adjusts the BFO setting (in Hz) of the channel receiver. The range of valid BFO settings is defined by each receiver's capability.

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Parameters:

`bfo` - the BFO to set (in Hz)

Throws:

`NRCException`

---

setDataOptions

```
public bool setDataOptions( int sampleRate, int samplesPerPacket, int
filterOption = LPF_4KHZ_ON, int timeStampOption = TIMESTAMP_OFF );
```

Adjust options for digitized audio data capture. These settings are only effective if made prior to data capture. Setting the data options during data capture will fail.

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Parameters:

`sampleRate` - a constant representing the sample rate (in Hz) at which to capture audio data. See class constants.

`samplesPerPacket` - the number of samples to collect before forwarding them to the client

`filterOption` - the filter option associated with the channel data

`timeStampOption` - the time stamp option associated with the channel data

---

setDetectionMode

```
public bool setDetectionMode( std::string detmode );
```

Adjusts the detection mode setting of the channel receiver. The valid range of detection modes are defined by each receiver. A list of valid detection modes for the channel receiver can be obtained by calling `getDetectionModeList()`.

This can only be performed if this channel has been acquired by the client using

an `NRC::connectToChannel()` call.

Parameters:

`detmode` - the detection mode

Returns:

true, if successful, false otherwise

---

setFrequency

`public bool` **`setFrequency`**`( double freq );`

Tunes the channel receiver to the specified frequency (in MHz). The range of valid frequencies is defined by the capability of each receiver.

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Parameters:

`freq` - the frequency to tune to (in MHz)

---

setIFBandwidth

`public bool` **`setIFBandwidth`**`( int ifbw );`

Adjusts the IF bandwidth setting (in Hz) of the receiver attached to this channel. The range of valid IF bandwidths is defined by each receiver's capability.

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Parameters:

`ifbw` - the new IF bandwidth setting (in Hz)

---

setOnlinePollInterval

`public bool` **`setOnlinePollInterval`**`( int interval );`

Sets the interval in seconds at which this channel receiver will be tested to see if it is online. An interval of 0 disables online polling on this channel. This is helpful when performing a lot of receiver operations as the online polling can slow down receiver performance.

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Parameters:

`interval` - the new polling interval in seconds, or 0 to stop

Returns:

true, if successful, false otherwise

---

setSamplesSkippedOnTune

`public bool` **`setSamplesSkippedOnTune`**`( int skip );`

Sets the number of samples to skip following a change in receiver settings, such as a change in frequency, detection mode, IF bandwidth, and BFO. This helps eliminate invalid audio data during the receiver transition.

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Parameters:

`skip` - the number of samples to skip following receiver tune

Returns:

true, if successful, false otherwise

---

stopDataCapture

```
public bool stopDataCapture();
```

Stops capturing digitized audio data from the receiver attached to this channel. The file, socket, or processor destination is also closed.

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Returns:

true, if successful, false otherwise

---

writeMemoryLocation

```
public bool writeMemoryLocation( int loc );
```

Write the current receiver settings to a memory location

Parameters:

`loc` - the location to write to

Class NRC

class **NRC**

This class is a representation of a connection to an NRC. This class is used to connect and disconnect from an NRC as well as the channels within an NRC. This class is also used to set and access NRC properties.

Inner Classes, Typedefs, and Enums

struct	**NRC::FileInfo**

Field Summary

static const int	**DEFAULT_CONNECT_PORT** The default NRC socket port
static const int	**DEFAULT_RESPONSE_TIMEOUT** The default NRC socket response timeout (in seconds)

Constructor Summary

NRC() Default constructor
~NRC() Destructor

Method Summary

Channel*	**connectToChannel**(int channel) Attempts to connect to the channel with the specified ID.
Channel*	**connectToChannel**() Attempts to connect to the next available channel.
bool	**connectToNRC**(std::string IPAddress, int port = DEFAULT_CONNECT_PORT) Establishes a connection to an NRC located at the given IP address (dot-notation) or DNS-resolvable host name and socket port.
bool	**disconnectFromChannel**() Disconnects from the currently connected channel
bool	**disconnectFromNRC**() terminates the socket connection to the NRC.

`int*`	**`getADCSettings`**`()` Retrieves the current ADC (Audio/Digital Converter) settings from the NRC.
`std::string`	**`getAntenna`**`( int channel )` Returns the name of the antenna that is connected to the receiver attached to the channel with the given channel ID.
`Channel*`	**`getChannel`**`( int channel )` Returns the channel with the given integer identifier.
`int`	**`getChannelCount`**`()` Returns the number of channels contained in the NRC.
`std::string`	**`getClientLocation`**`( int channel )` Returns the location of the client connected to the channel with the given ID.
`Channel*`	**`getConnectedChannel`**`()` Returns the channel that the client is currently connected to, or NULL, if the client is not connected to a channel.
`std::string`	**`getIPAddress`**`()`
`int`	**`getMonitorInfo`**`( void* buffer, int maxSize )` Gets a packet containing NRC monitor data (see the programmers guide for data format description)
`std::string`	**`getNRCConfiguration`**`()` Causes the NRC to return a configuration message with general information about the NRC and the availability of each receiver.
`int`	**`getPort`**`()` Returns the NRC socket port value if the client has connected to the NRC.
`std::string`	**`getReceiverModel`**`( int channel )` Returns the manufacturer and model name of receiver attached to the channel with the given channel ID.
`int`	**`getResponseTimeout`**`()` Returns the current timeout (in seconds) for a expected response from the NRC server.
`std::vector< NRC::FileInfo >`	**`getVersionInformation`**`()` Return a vector of File information about files on the server
`bool`	**`isConnected`**`()` Returns true if the client is connected to an NRC, false otherwise.
`bool`	**`ping`**`()` Sends a ping command to the connected NRC and waits for an appropriate response.

bool	**rebootNRC**() Causes the Network Receiver Controller software to perform a warm restart.
bool	**setADCSettings**(int flags, int ch0Cfg, int ch1Cfg, int ch2Cfg, int ch3Cfg, int ch4Cfg, int ch5Cfg, int ch6Cfg, int ch7Cfg) Changes the ADC (Audio/Digital Converter) settings in the NRC.
void	**setResponseTimeout**(int secs) Sets the timeout (in seconds) for a expected response from the NRC server.

Field Detail

DEFAULT_CONNECT_PORT

```
public static const int DEFAULT_CONNECT_PORT;
```

The default NRC socket port

DEFAULT_RESPONSE_TIMEOUT

```
public static const int DEFAULT_RESPONSE_TIMEOUT;
```

The default NRC socket response timeout (in seconds)

Constructor Detail

NRC

```
public NRC();
```

Default constructor

~NRC

```
public ~NRC();
```

Destructor

Method Detail

connectToChannel

```
public Channel* connectToChannel( int channel );
```

Attempts to connect to the channel with the specified ID. If successful, returns the channel record.

Parameters:

`channel` - the ID of the channel to connect to (0 .. `getChannelCount()` - 1)

Returns:

the record of the newly connected channel, or NULL on failure

---

connectToChannel

```
public Channel* connectToChannel();
```

Attempts to connect to the next available channel. If successful, returns the channel record.

Returns:

the record of the newly connected channel, or NULL on failure

---

connectToNRC

```
public bool connectToNRC( std::string IPAddress, int port =
DEFAULT_CONNECT_PORT );
```

Establishes a connection to an NRC located at the given IP address (dot-notation) or DNS-resolvable host name and socket port. Note: The default port is correct unless using a custom NRC server with a different port.

Parameters:

`IPAddress` - location of NRC (host name or dot-separated IP address)

`port` - (optional) the socket port to connect to

Returns:

true on success, false otherwise

---

disconnectFromChannel

```
public bool disconnectFromChannel();
```

Disconnects from the currently connected channel

Returns:

true if successful, false otherwise

---

disconnectFromNRC

```
public bool disconnectFromNRC();
```

terminates the socket connection to the NRC.

Returns:

true if successful, false otherwise

---

getADCSettings

```
public int* getADCSettings();
```

Retrieves the current ADC (Audio/Digital Converter) settings from the NRC.

Returns:

the current settings. Constants beginning with "FLAG_ADC" from NRCRPMConstants.h have been ORed together to get the reported setting.

---

getAntenna

```
public std::string getAntenna( int channel );
```

Returns the name of the antenna that is connected to the receiver attached to the channel with the given channel ID. A zero-length string is returned if no antenna name is given.

Parameters:

`channel` - the ID of the channel (0 .. `getChannelCount()` - 1)

Returns:

the antenna connected to the receiver at the given channel

---

getChannel

```
public Channel* getChannel( int channel );
```

Returns the channel with the given integer identifier.

Parameters:

`channel` - the ID of the channel to return [0 .. `getChannelCount()`]

Returns:

the channel with the given ID.

---

getChannelCount

```
public int getChannelCount();
```

Returns the number of channels contained in the NRC.

Returns:

channel count

---

getClientLocation

```
public std::string getClientLocation( int channel );
```

Returns the location of the client connected to the channel with the given ID. This is typically the IP address of the attached client. If no client is connected, "Not connected" is returned.

Parameters:

`channel` - the ID of the channel to connect to (0..channelcount-1)

Returns:

the location of the client connected to the given channel

---

getConnectedChannel

```
public Channel* getConnectedChannel();
```

Returns the channel that the client is currently connected to, or NULL, if the client is not connected to a channel.

Returns:

current connected channel, or NULL, if no connection

---

getIPAddress

```
public std::string getIPAddress();
```

Returns:

the IP address of the connected NRC, or a zero-length string if the client is not connected to an NRC.

---

getMonitorInfo

```
public int getMonitorInfo( void* buffer, int maxSize );
```

Gets a packet containing NRC monitor data (see the programmers guide for data format description)

Parameters:

`buffer` - a data buffer to copy the monitor data in to

`maxSize` - the size of the data buffer

Returns:

the number of bytes of data copied into buffer. If the buffer was not big enough, a negative number will be returned indicating the buffer size required.

---

getNRCConfiguration

```
public std::string getNRCConfiguration();
```

Causes the NRC to return a configuration message with general information about the NRC and the availability of each receiver.

Returns:

the current configuration. Format of string can be found in the Network Receiver Controller Interface Control Document.

---

getPort

```
public int getPort();
```

Returns the NRC socket port value if the client has connected to the NRC. Otherwise, 0 is returned.

Returns:

the socket port value of connection to the NRC, otherwise 0.

---

getReceiverModel

```
public std::string getReceiverModel( int channel );
```


Returns the manufacturer and model name of receiver attached to the channel with the given channel ID. A zero-length string is returned if no receiver is connected.

Parameters:

`channel` - the ID of the channel (0 .. `getChannelCount()` - 1)

Returns:

the receiver model at the given channel

---

getResponseTimeout

```
public int getResponseTimeout();
```

Returns the current timeout (in seconds) for a expected response from the NRC server.

Returns:

the current response timeout, in seconds

---

getVersionInformation

```
public std::vector< NRC::FileInfo > getVersionInformation();
```

Return a vector of File information about files on the server

---

isConnected

```
public bool isConnected();
```

Returns true if the client is connected to an NRC, false otherwise.

Returns:

true if the client is connected to NRC, false otherwise.

---

ping

```
public bool ping();
```

Sends a ping command to the connected NRC and waits for an appropriate response. True is returned if the correct response is received; otherwise, false is returned.

Returns:

true if successful, false otherwise

---

rebootNRC

```
public bool rebootNRC();
```

Causes the Network Receiver Controller software to perform a warm restart. Any currently active socket connections will be terminated and all parameters will be resotred to their initial power up conditions. A restart of the NRC will take about one minute to complete. Any attempts to establish a socket connection with the NRC during the reboot will fail. Use with care.

---

setADCSettings

```
public bool setADCSettings( int flags, int ch0Cfg, int ch1Cfg, int
ch2Cfg, int ch3Cfg, int ch4Cfg, int ch5Cfg, int ch6Cfg, int ch7Cfg );
```

Changes the ADC (Audio/Digital Converter) settings in the NRC. If any specified settings are different than the existing set, the NRC data acquisition process is "rebooted", which will disrupt data capture on all channels. Use with care.

Parameters:

`flags` - ADCSetting - class to access: Clock & Diff/Single Ended mode settings - NRCRPMConstants.h should be ORed together to get the desired setting.
`ch0Cfg` - ch<0-7>Cfg the ADC channel settings. - bit 0-1: Gain: Gain=1: (00), Gain=2: (01), Gain=4: (10), Gain=8: (11) - bit 8: RESERVED - for Slow bit setting: Slow on (1), Slow off (0)

---

setResponseTimeout

```
public void setResponseTimeout( int secs );
```

Sets the timeout (in seconds) for a expected response from the NRC server.

Parameters:

`secs` - the new response timeout (in seconds)

Class SampleProcessor

class **SampleProcessor**

This interface is to be implemented in classes that want to process captured audio data. The derived class should be used in conjunction with a call to `Channel::captureDataToProcessor()`.

Constructor Summary

`virtual ~SampleProcessor()`

Method Summary

`virtual bool`	**`processDone`**`()= 0` Callback method is called when the sample capture stream has been closed
`virtual bool`	**`processSamples`**`( const char* signal, int numchars )= 0` Method is called to process samples.

Constructor Detail

~SampleProcessor

```
public virtual ~SampleProcessor();
```

Method Detail

processDone

```
public virtual bool processDone()= 0;
```

Callback method is called when the sample capture stream has been closed

Returns:

true on success, false otherwise

processSamples

```
public virtual bool processSamples(const char* signal, int numchars)=0;
```

Method is called to process samples. Sample data is provided in bytes, so it is important to reconstruct multibyte samples before processing.

Parameters:

`signal` - byte samples array

`numchars` - number of bytes (= num samples * 2)

Returns:

true on success, false otherwise

Class NRCRPMConstants

class **NRCRPMConstants**

This file includes various NRC constants.
Copyright (c) 2005 Aegis, Inc.

Field Summary

static const unsigned short	**CMD_CHANNEL_COUNT_REQUEST** Request command: get channel count
static const unsigned short	**CMD_CHANNEL_COUNT_RESPONSE** Response command: get channel count
static const unsigned short	**CMD_CHANNEL_INFO_REQUEST** **Deprecated.** *@deprecated*
static const unsigned short	**CMD_CHANNEL_INFO_RESPONSE** **Deprecated.** *@deprecated*
static const unsigned short	**CMD_CHANNEL_SUMMARY_REQUEST** Request command: get channel summary
static const unsigned short	**CMD_CHANNEL_SUMMARY_RESPONSE** Response command: get channel summary
static const unsigned short	**CMD_CHANNEL_WRITABLE** Notification command: client may now alter the channel/receiver configuration
static const unsigned short	**CMD_CLIENT_LOCATION_REQUEST** Request command: get channel client location
static const unsigned short	**CMD_CLIENT_LOCATION_RESPONSE** Response command: get channel client location
static const unsigned short	**CMD_CONNECT_REQUEST** Request command: connect to channel
static const unsigned short	**CMD_CONNECT_RESPONSE** Response command: connect to channel
static const unsigned short	**CMD_DATA_OPTIONS_REQUEST** Request command: set data options (sample rate, packet size, etc)
static const unsigned short	**CMD_DATA_OPTIONS_RESPONSE** Response command: set data options (sample rate, packet size, etc)
static const unsigned short	**CMD_DATA_PACKET** Data command: data packet

static const unsigned short	**CMD_DETMODE_LIST_REQUEST** Request command: get valid detection modes
static const unsigned short	**CMD_DETMODE_LIST_RESPONSE** Response command: get valid detection modes
static const unsigned short	**CMD_DISCONNECT_REQUEST** Request command: disconnect from channel
static const unsigned short	**CMD_DISCONNECT_RESPONSE** Response command: disconnect from channel
static const unsigned short	**CMD_ERROR_RESPONSE** Error command
static const unsigned short	**CMD_GET_ADC_SETTINGS_REQUEST** Request command: get A/D converter settings
static const unsigned short	**CMD_GET_ADC_SETTINGS_RESPONSE** Response command: get A/D converter settings
static const unsigned short	**CMD_GET_ANTENNA_NAME_REQUEST** Request command: get antenna name
static const unsigned short	**CMD_GET_ANTENNA_NAME_RESPONSE** Response command: get antenna name
static const unsigned short	**CMD_GET_ONLINE_POLL_INTERVAL_REQUEST** Request command: get receiver online poll interval
static const unsigned short	**CMD_GET_ONLINE_POLL_INTERVAL_RESPONSE** Response command: get receiver online poll interval
static const unsigned short	**CMD_GET_WRITABLE_REQUEST** Request command: is the channel/receiver writable by this device
static const unsigned short	**CMD_GET_WRITABLE_RESPONSE** Response command: is the channel/receiver writable by this device
static const unsigned short	**CMD_LOADMEM_REQUEST** Request command: Load receiver memory location
static const unsigned short	**CMD_LOADMEM_RESPONSE** Response command: Load receiver memory location
static const unsigned short	**CMD_MEMCAPACITY_REQUEST** Request command: Query receiver memory capacity
static const unsigned short	**CMD_MEMCAPACITY_RESPONSE** Response command: Query receiver memory capacity
static const unsigned short	**CMD_MEMFIELDS_REQUEST** Request command: Query receiver memory fields
static const unsigned short	**CMD_MEMFIELDS_RESPONSE** Response command: Query receiver memory fields
static const	**CMD_MEMLOC_REQUEST**

unsigned short	Request command: Query receiver memory location
static const unsigned short	**CMD_MEMLOC_RESPONSE** Response command: Query receiver memory location
static const unsigned short	**CMD_MODEL_NAME_REQUEST** Request command: get receiver model
static const unsigned short	**CMD_MODEL_NAME_RESPONSE** Response command: get receiver model
static const unsigned short	**CMD_MONITOR_REQUEST** Request command: read monitor information
static const unsigned short	**CMD_MONITOR_RESPONSE** Response command: read monitor information
static const unsigned short	**CMD_NRC_CONFIGURATION_REQUEST** **Deprecated.** *Use CMD_MONITOR_REQUEST instead*
static const unsigned short	**CMD_NRC_CONFIGURATION_RESPONSE** **Deprecated.** *Use CMD_MONITOR_REQUEST instead*
static const unsigned short	**CMD_NRC_REBOOT_REQUEST** Request command: reboot NRC
static const unsigned short	**CMD_PASSTHRU_RESPONSE_REQUEST** Request command: Send a passthrough command that expects a response from the receiver
static const unsigned short	**CMD_PASSTHRU_RESPONSE_RESPONSE** Response command: Send a passthrough command that expects a response from the receiver
static const unsigned short	**CMD_PING** Request command: ping
static const unsigned short	**CMD_PONG** Response command: ping
static const unsigned short	**CMD_READ_DATAOPTS_REQUEST** Request command: read data options for the current channel
static const unsigned short	**CMD_READ_DATAOPTS_RESPONSE** Response command: read data options for the current channel
static const unsigned short	**CMD_RECEIVER_BFO_REQUEST** Request command: set receiver BFO
static const unsigned short	**CMD_RECEIVER_BFO_RESPONSE** Response command: set receiver BFO
static const unsigned short	**CMD_RECEIVER_CONFIGURATION_REQUEST** Request command: get Receiver configuration
static const unsigned short	**CMD_RECEIVER_CONFIGURATION_RESPONSE** Response command: get Receiver configuration
static const unsigned short	**CMD_RECEIVER_DETMODE_REQUEST** Request command: set receiver detection mode

static const unsigned short	**CMD_RECEIVER_DETMODE_RESPONSE** Response command: set receiver detection mode
static const unsigned short	**CMD_RECEIVER_GETAGC_REQUEST** Request command: get agc
static const unsigned short	**CMD_RECEIVER_GETAGC_RESPONSE** Response command: get agc
static const unsigned short	**CMD_RECEIVER_GETBFO_REQUEST** Request command: get receiver BFO
static const unsigned short	**CMD_RECEIVER_GETBFO_RESPONSE** Response command: get receiver BFO
static const unsigned short	**CMD_RECEIVER_GETDETMODE_REQUEST** Request command: get receiver detection mode
static const unsigned short	**CMD_RECEIVER_GETDETMODE_RESPONSE** Response command: get receiver detection mode
static const unsigned short	**CMD_RECEIVER_GETFREQ_REQUEST** Request command: get receiver frequency
static const unsigned short	**CMD_RECEIVER_GETFREQ_RESPONSE** Response command: get receiver frequency
static const unsigned short	**CMD_RECEIVER_GETIFBANDWIDTH_REQUEST** Request command: get receiver IF bandwidth
static const unsigned short	**CMD_RECEIVER_GETIFBANDWIDTH_RESPONSE** Response command: get receiver IF bandwidth
static const unsigned short	**CMD_RECEIVER_GETSAMPLESSKIPPEDONTUNE_REQUEST** Request command: get number of samples skipped on tune
static const unsigned short	**CMD_RECEIVER_GETSAMPLESSKIPPEDONTUNE_RESPONSE** Response command: get number of samples skipped on tune
static const unsigned short	**CMD_RECEIVER_IFBANDWIDTH_REQUEST** Request command: set receiver IF bandwidth
static const unsigned short	**CMD_RECEIVER_IFBANDWIDTH_RESPONSE** Response command: set receiver IF bandwidth
static const unsigned short	**CMD_RECEIVER_MODEL_LIST_REQUEST** Request command: get valid receiver model list
static const unsigned short	**CMD_RECEIVER_MODEL_LIST_RESPONSE** Response command: get valid receiver model list
static const unsigned short	**CMD_RECEIVER_OFFLINE** Notification command: receiver is offline
static const unsigned short	**CMD_RECEIVER_ONLINE** Notification command: receiver is online
static const unsigned short	**CMD_RECEIVER_PASSTHRU_REQUEST** Request command: send receiver passthrough command
static const	**CMD_RECEIVER_PASSTHRU_RESPONSE**

unsigned short	Response command: send receiver passthrough command
static const unsigned short	**CMD_RECEIVER_REBOOT_REQUEST** Request command: reboot Receiver
static const unsigned short	**CMD_RECEIVER_REBOOT_RESPONSE** Response command: reboot Receiver
static const unsigned short	**CMD_RECEIVER_RESET_REQUEST** Request command: reset Receiver
static const unsigned short	**CMD_RECEIVER_RESET_RESPONSE** Response command: reset Receiver
static const unsigned short	**CMD_RECEIVER_SET_MODEL_REQUEST** Request command: set channel receiver model
static const unsigned short	**CMD_RECEIVER_SET_MODEL_RESPONSE** Response command: set channel receiver model
static const unsigned short	**CMD_RECEIVER_SETAGC_REQUEST** Request command: set AGC
static const unsigned short	**CMD_RECEIVER_SETAGC_RESPONSE** Response command: set AGC
static const unsigned short	**CMD_RECEIVER_SETMODE_REQUEST** Request command: set samples skipped on tune
static const unsigned short	**CMD_RECEIVER_SETMODE_RESPONSE** Response command: set samples skipped on tune
static const unsigned short	**CMD_RECEIVER_STATUS_REQUEST** Request command: receiver status
static const unsigned short	**CMD_RECEIVER_STATUS_RESPONSE** Response command: receiver status
static const unsigned short	**CMD_RECEIVER_TUNE_REQUEST** Request command: set receiver frequency
static const unsigned short	**CMD_RECEIVER_TUNE_RESPONSE** Response command: set receiver frequency
static const unsigned short	**CMD_SET_ADC_SETTINGS_REQUEST** Request command: set A/D converter settings
static const unsigned short	**CMD_SET_ADC_SETTINGS_RESPONSE** Response command: set A/D converter settings
static const unsigned short	**CMD_SET_ANTENNA_NAME_REQUEST** Request command: set antenna name
static const unsigned short	**CMD_SET_ANTENNA_NAME_RESPONSE** Response command: set antenna name
static const unsigned short	**CMD_SET_ONLINE_POLL_INTERVAL_REQUEST** Request command: set receiver online poll interval
static const unsigned short	**CMD_SET_ONLINE_POLL_INTERVAL_RESPONSE** Response command: set receiver online poll interval

static const unsigned short	**CMD_START_DATA_REQUEST** Request command: start data capture
static const unsigned short	**CMD_STOP_DATA_REQUEST** Request command: stop data capture
static const unsigned short	**CMD_STOP_DATA_RESPONSE** Response command: stop data capture
static const unsigned short	**CMD_VERSION_REQUEST** Request command: Query NRC version information
static const unsigned short	**CMD_VERSION_RESPONSE** Response command: Query NRC version information
static const unsigned short	**CMD_WRITEMEM_REQUEST** Request command: Write receiver memory location
static const unsigned short	**CMD_WRITEMEM_RESPONSE** Response command: Write receiver memory location
static const unsigned short	**EALREADYCON** Error data: A receiver is already connected
static const unsigned short	**EARGUMENT** Error data: The command argument is invalid
static const unsigned short	**ENORESOURCE** Error data: No more receivers are available
static const unsigned short	**ENOTCONNECTED** Error data: No receiver is connected
static const unsigned short	**ERBADCMDTRANSLATION** Error data: The command could not be translated
static const unsigned short	**ERCANNOTATTACHRECEIVER** Error data: Cannot attach the receiver
static const unsigned short	**ERCANNOTREMOVERECEIVER** Error data: Cannot remove the receiver
static const unsigned short	**ERCAPTURING** Error data: Error while capturing audio data
static const unsigned short	**ERCHANALRDYCON** Error data: The channel requested was already connected to another socket
static const unsigned short	**ERCHANSELFTESTPENDING** Error data: The channel requested is has a self-test pending
static const unsigned short	**ERCHANSELFTESTRUNNING** Error data: The channel requested is has a self-test running
static const unsigned short	**ERINVALIDCHAN** Error data: An invalid channel value was used in the packet message
static const	**ERINVALIDCMD**

unsigned short	Error data: An invalid command value was used in the packet message
static const unsigned short	**ERINVALIDLEN** Error data: An invalid length value was used in the packet message
static const unsigned short	**ERINVALIDSAMPLERATE** Error data: The requested sample rate is invalid
static const unsigned short	**ERINVALIDSAMPLESPERPACKET** Error data: An invalid number of samples per packet was specified
static const unsigned short	**ERINVALIDVGCFILTEROPTION** Error data: An invalid VGC Filter option for audio data
static const unsigned short	**ERNODATAWRITTEN** Error data: Zero bytes of data written to the socket
static const unsigned short	**ERNORECEIVERDEF** Error data: A receiver definition file could not be found
static const unsigned short	**ERREADONLY** Error data: The requested command requires write access to the channel.
static const unsigned short	**ERSOCKALRDYCON** Error data: The socket already has a channel connected to it
static const unsigned short	**ERSOCKNOTCON** Error data: The socket and a channel have not yet been connected
static const unsigned short	**ERTIMEOUT** Error data: The command timed out
static const unsigned short	**ERUNKNOWN** Error data: An unknown error occurred
static const unsigned short	**ERWRITEERROR** Error data: An unspecified error has occurred
static const unsigned short	**ESERIALOVLD** Error data: The serial port is overloaded
static const unsigned short	**FLAG_ADC_USE_DIFFERENTIAL_MODE** Flag: Use differential input mode for A/D converter.
static const unsigned short	**FLAG_ADC_USE_EXT_CLOCK** Flag: Use external clock signal for A/D converter.
static const unsigned short	**SUCCESS** Error data: success (no error)

Method Summary

static bool	**IsReadCommand**(unsigned short cmd) Check to see if a command is "read only." (That is, check to see if a command only reads status information and does not change any settings which may effect other clients connected to this channel.)

Field Detail

CMD_CHANNEL_COUNT_REQUEST

```
public static const unsigned short CMD_CHANNEL_COUNT_REQUEST;
```

Request command: get channel count

CMD_CHANNEL_COUNT_RESPONSE

```
public static const unsigned short CMD_CHANNEL_COUNT_RESPONSE;
```

Response command: get channel count

CMD_CHANNEL_INFO_REQUEST

```
public static const unsigned short CMD_CHANNEL_INFO_REQUEST;
```

Deprecated. *@deprecated*

Request command: get channel info (deprecated)

CMD_CHANNEL_INFO_RESPONSE

```
public static const unsigned short CMD_CHANNEL_INFO_RESPONSE;
```

Deprecated. *@deprecated*

Response command: get channel info (deprecated)

CMD_CHANNEL_SUMMARY_REQUEST

```
public static const unsigned short CMD_CHANNEL_SUMMARY_REQUEST;
```

Request command: get channel summary

CMD_CHANNEL_SUMMARY_RESPONSE

```
public static const unsigned short CMD_CHANNEL_SUMMARY_RESPONSE;
```

Response command: get channel summary

CMD_CHANNEL_WRITABLE

```
public static const unsigned short CMD_CHANNEL_WRITABLE;
```


Notification command: client may now alter the channel/receiver configuration

---

CMD_CLIENT_LOCATION_REQUEST

```
public static const unsigned short CMD_CLIENT_LOCATION_REQUEST;
```

Request command: get channel client location

CMD_CLIENT_LOCATION_RESPONSE

```
public static const unsigned short CMD_CLIENT_LOCATION_RESPONSE;
```

Response command: get channel client location

CMD_CONNECT_REQUEST

```
public static const unsigned short CMD_CONNECT_REQUEST;
```

Request command: connect to channel

CMD_CONNECT_RESPONSE

```
public static const unsigned short CMD_CONNECT_RESPONSE;
```

Response command: connect to channel

CMD_DATA_OPTIONS_REQUEST

```
public static const unsigned short CMD_DATA_OPTIONS_REQUEST;
```

Request command: set data options (sample rate, packet size, etc)

CMD_DATA_OPTIONS_RESPONSE

```
public static const unsigned short CMD_DATA_OPTIONS_RESPONSE;
```

Response command: set data options (sample rate, packet size, etc)

CMD_DATA_PACKET

```
public static const unsigned short CMD_DATA_PACKET;
```

Data command: data packet

CMD_DETMODE_LIST_REQUEST

```
public static const unsigned short CMD_DETMODE_LIST_REQUEST;
```

Request command: get valid detection modes

CMD_DETMODE_LIST_RESPONSE

```
public static const unsigned short CMD_DETMODE_LIST_RESPONSE;
```

Response command: get valid detection modes

CMD_DISCONNECT_REQUEST

```
public static const unsigned short CMD_DISCONNECT_REQUEST;
```

Request command: disconnect from channel

CMD_DISCONNECT_RESPONSE

```
public static const unsigned short CMD_DISCONNECT_RESPONSE;
```

Response command: disconnect from channel

CMD_ERROR_RESPONSE

```
public static const unsigned short CMD_ERROR_RESPONSE;
```

Error command

CMD_GET_ADC_SETTINGS_REQUEST

```
public static const unsigned short CMD_GET_ADC_SETTINGS_REQUEST;
```

Request command: get A/D converter settings

CMD_GET_ADC_SETTINGS_RESPONSE

```
public static const unsigned short CMD_GET_ADC_SETTINGS_RESPONSE;
```

Response command: get A/D converter settings

CMD_GET_ANTENNA_NAME_REQUEST

```
public static const unsigned short CMD_GET_ANTENNA_NAME_REQUEST;
```

Request command: get antenna name

CMD_GET_ANTENNA_NAME_RESPONSE

```
public static const unsigned short CMD_GET_ANTENNA_NAME_RESPONSE;
```

Response command: get antenna name

CMD_GET_ONLINE_POLL_INTERVAL_REQUEST

```
public static const unsigned short
CMD_GET_ONLINE_POLL_INTERVAL_REQUEST;
```

Request command: get receiver online poll interval

CMD_GET_ONLINE_POLL_INTERVAL_RESPONSE

```
public static const unsigned short
CMD_GET_ONLINE_POLL_INTERVAL_RESPONSE;
```

Response command: get receiver online poll interval

CMD_GET_WRITABLE_REQUEST

```
public static const unsigned short CMD_GET_WRITABLE_REQUEST;
```

Request command: is the channel/receiver writable by this device

---

CMD_GET_WRITABLE_RESPONSE

```
public static const unsigned short CMD_GET_WRITABLE_RESPONSE;
```

Response command: is the channel/receiver writable by this device

---

CMD_LOADMEM_REQUEST

```
public static const unsigned short CMD_LOADMEM_REQUEST;
```

Request command: Load receiver memory location

CMD_LOADMEM_RESPONSE

```
public static const unsigned short CMD_LOADMEM_RESPONSE;
```

Response command: Load receiver memory location

CMD_MEMCAPACITY_REQUEST

```
public static const unsigned short CMD_MEMCAPACITY_REQUEST;
```

Request command: Query receiver memory capacity

CMD_MEMCAPACITY_RESPONSE

```
public static const unsigned short CMD_MEMCAPACITY_RESPONSE;
```

Response command: Query receiver memory capacity

CMD_MEMFIELDS_REQUEST

```
public static const unsigned short CMD_MEMFIELDS_REQUEST;
```

Request command: Query receiver memory fields

CMD_MEMFIELDS_RESPONSE

```
public static const unsigned short CMD_MEMFIELDS_RESPONSE;
```

Response command: Query receiver memory fields

CMD_MEMLOC_REQUEST

```
public static const unsigned short CMD_MEMLOC_REQUEST;
```

Request command: Query receiver memory location

CMD_MEMLOC_RESPONSE

```
public static const unsigned short CMD_MEMLOC_RESPONSE;
```

Response command: Query receiver memory location

CMD_MODEL_NAME_REQUEST

```
public static const unsigned short CMD_MODEL_NAME_REQUEST;
```

Request command: get receiver model

CMD_MODEL_NAME_RESPONSE

```
public static const unsigned short CMD_MODEL_NAME_RESPONSE;
```

Response command: get receiver model

CMD_MONITOR_REQUEST

```
public static const unsigned short CMD_MONITOR_REQUEST;
```

Request command: read monitor information

CMD_MONITOR_RESPONSE

```
public static const unsigned short CMD_MONITOR_RESPONSE;
```

Response command: read monitor information

CMD_NRC_CONFIGURATION_REQUEST

```
public static const unsigned short CMD_NRC_CONFIGURATION_REQUEST;
```

Deprecated. *Use CMD_MONITOR_REQUEST instead*

Request command: get NRC configuration

CMD_NRC_CONFIGURATION_RESPONSE

```
public static const unsigned short CMD_NRC_CONFIGURATION_RESPONSE;
```

Deprecated. *Use CMD_MONITOR_REQUEST instead*

Response command: get NRC configuration

CMD_NRC_REBOOT_REQUEST

```
public static const unsigned short CMD_NRC_REBOOT_REQUEST;
```

Request command: reboot NRC

CMD_PASSTHRU_RESPONSE_REQUEST

```
public static const unsigned short CMD_PASSTHRU_RESPONSE_REQUEST;
```

Request command: Send a passthrough command that expects a response from

the receiver

CMD_PASSTHRU_RESPONSE_RESPONSE

```
public static const unsigned short CMD_PASSTHRU_RESPONSE_RESPONSE;
```

Response command: Send a passthrough command that expects a response from the receiver

---

CMD_PING

```
public static const unsigned short CMD_PING;
```

Request command: ping

CMD_PONG

```
public static const unsigned short CMD_PONG;
```

Response command: ping

CMD_READ_DATAOPTS_REQUEST

```
public static const unsigned short CMD_READ_DATAOPTS_REQUEST;
```

Request command: read data options for the current channel

CMD_READ_DATAOPTS_RESPONSE

```
public static const unsigned short CMD_READ_DATAOPTS_RESPONSE;
```

Response command: read data options for the current channel

CMD_RECEIVER_BFO_REQUEST

```
public static const unsigned short CMD_RECEIVER_BFO_REQUEST;
```

Request command: set receiver BFO

CMD_RECEIVER_BFO_RESPONSE

```
public static const unsigned short CMD_RECEIVER_BFO_RESPONSE;
```

Response command: set receiver BFO

CMD_RECEIVER_CONFIGURATION_REQUEST

```
public static const unsigned short CMD_RECEIVER_CONFIGURATION_REQUEST;
```

Request command: get Receiver configuration

CMD_RECEIVER_CONFIGURATION_RESPONSE

```
public static const unsigned short CMD_RECEIVER_CONFIGURATION_RESPONSE;
```

Response command: get Receiver configuration

CMD_RECEIVER_DETMODE_REQUEST

```
public static const unsigned short CMD_RECEIVER_DETMODE_REQUEST;
```

Request command: set receiver detection mode

CMD_RECEIVER_DETMODE_RESPONSE

```
public static const unsigned short CMD_RECEIVER_DETMODE_RESPONSE;
```

Response command: set receiver detection mode

CMD_RECEIVER_GETAGC_REQUEST

```
public static const unsigned short CMD_RECEIVER_GETAGC_REQUEST;
```

Request command: get agc

CMD_RECEIVER_GETAGC_RESPONSE

```
public static const unsigned short CMD_RECEIVER_GETAGC_RESPONSE;
```

Response command: get agc

CMD_RECEIVER_GETBFO_REQUEST

```
public static const unsigned short CMD_RECEIVER_GETBFO_REQUEST;
```

Request command: get receiver BFO

CMD_RECEIVER_GETBFO_RESPONSE

```
public static const unsigned short CMD_RECEIVER_GETBFO_RESPONSE;
```

Response command: get receiver BFO

CMD_RECEIVER_GETDETMODE_REQUEST

```
public static const unsigned short CMD_RECEIVER_GETDETMODE_REQUEST;
```

Request command: get receiver detection mode

CMD_RECEIVER_GETDETMODE_RESPONSE

```
public static const unsigned short CMD_RECEIVER_GETDETMODE_RESPONSE;
```

Response command: get receiver detection mode

CMD_RECEIVER_GETFREQ_REQUEST

```
public static const unsigned short CMD_RECEIVER_GETFREQ_REQUEST;
```

Request command: get receiver frequency

CMD_RECEIVER_GETFREQ_RESPONSE

```
public static const unsigned short CMD_RECEIVER_GETFREQ_RESPONSE;
```

Response command: get receiver frequency

CMD_RECEIVER_GETIFBANDWIDTH_REQUEST

```
public static const unsigned short CMD_RECEIVER_GETIFBANDWIDTH_REQUEST;
```

Request command: get receiver IF bandwidth

CMD_RECEIVER_GETIFBANDWIDTH_RESPONSE

```
public static const unsigned short
CMD_RECEIVER_GETIFBANDWIDTH_RESPONSE;
```

Response command: get receiver IF bandwidth

CMD_RECEIVER_GETSAMPLESSKIPPEDONTUNE_REQUEST

```
public static const unsigned short
CMD_RECEIVER_GETSAMPLESSKIPPEDONTUNE_REQUEST;
```

Request command: get number of samples skipped on tune

CMD_RECEIVER_GETSAMPLESSKIPPEDONTUNE_RESPONSE

```
public static const unsigned short
CMD_RECEIVER_GETSAMPLESSKIPPEDONTUNE_RESPONSE;
```

Response command: get number of samples skipped on tune

CMD_RECEIVER_IFBANDWIDTH_REQUEST

```
public static const unsigned short CMD_RECEIVER_IFBANDWIDTH_REQUEST;
```

Request command: set receiver IF bandwidth

CMD_RECEIVER_IFBANDWIDTH_RESPONSE

```
public static const unsigned short CMD_RECEIVER_IFBANDWIDTH_RESPONSE;
```

Response command: set receiver IF bandwidth

CMD_RECEIVER_MODEL_LIST_REQUEST

```
public static const unsigned short CMD_RECEIVER_MODEL_LIST_REQUEST;
```

Request command: get valid receiver model list

CMD_RECEIVER_MODEL_LIST_RESPONSE

```
public static const unsigned short CMD_RECEIVER_MODEL_LIST_RESPONSE;
```

Response command: get valid receiver model list

CMD_RECEIVER_OFFLINE

```
public static const unsigned short CMD_RECEIVER_OFFLINE;
```

Notification command: receiver is offline

CMD_RECEIVER_ONLINE

```
public static const unsigned short CMD_RECEIVER_ONLINE;
```

Notification command: receiver is online

CMD_RECEIVER_PASSTHRU_REQUEST

```
public static const unsigned short CMD_RECEIVER_PASSTHRU_REQUEST;
```

Request command: send receiver passthrough command

CMD_RECEIVER_PASSTHRU_RESPONSE

```
public static const unsigned short CMD_RECEIVER_PASSTHRU_RESPONSE;
```

Response command: send receiver passthrough command

CMD_RECEIVER_REBOOT_REQUEST

```
public static const unsigned short CMD_RECEIVER_REBOOT_REQUEST;
```

Request command: reboot Receiver

CMD_RECEIVER_REBOOT_RESPONSE

```
public static const unsigned short CMD_RECEIVER_REBOOT_RESPONSE;
```

Response command: reboot Receiver

CMD_RECEIVER_RESET_REQUEST

```
public static const unsigned short CMD_RECEIVER_RESET_REQUEST;
```

Request command: reset Receiver

CMD_RECEIVER_RESET_RESPONSE

```
public static const unsigned short CMD_RECEIVER_RESET_RESPONSE;
```

Response command: reset Receiver

CMD_RECEIVER_SET_MODEL_REQUEST

```
public static const unsigned short CMD_RECEIVER_SET_MODEL_REQUEST;
```

Request command: set channel receiver model

CMD_RECEIVER_SET_MODEL_RESPONSE

```
public static const unsigned short CMD_RECEIVER_SET_MODEL_RESPONSE;
```

Response command: set channel receiver model

CMD_RECEIVER_SETAGC_REQUEST

```
public static const unsigned short CMD_RECEIVER_SETAGC_REQUEST;
```

Request command: set AGC

CMD_RECEIVER_SETAGC_RESPONSE

```
public static const unsigned short CMD_RECEIVER_SETAGC_RESPONSE;
```

Response command: set AGC

CMD_RECEIVER_SETMODE_REQUEST

```
public static const unsigned short CMD_RECEIVER_SETMODE_REQUEST;
```

Request command: set samples skipped on tune

CMD_RECEIVER_SETMODE_RESPONSE

```
public static const unsigned short CMD_RECEIVER_SETMODE_RESPONSE;
```

Response command: set samples skipped on tune

CMD_RECEIVER_STATUS_REQUEST

```
public static const unsigned short CMD_RECEIVER_STATUS_REQUEST;
```

Request command: receiver status

CMD_RECEIVER_STATUS_RESPONSE

```
public static const unsigned short CMD_RECEIVER_STATUS_RESPONSE;
```

Response command: receiver status

CMD_RECEIVER_TUNE_REQUEST

```
public static const unsigned short CMD_RECEIVER_TUNE_REQUEST;
```

Request command: set receiver frequency

CMD_RECEIVER_TUNE_RESPONSE

```
public static const unsigned short CMD_RECEIVER_TUNE_RESPONSE;
```

Response command: set receiver frequency

CMD_SET_ADC_SETTINGS_REQUEST

```
public static const unsigned short CMD_SET_ADC_SETTINGS_REQUEST;
```

Request command: set A/D converter settings

CMD_SET_ADC_SETTINGS_RESPONSE

```
public static const unsigned short CMD_SET_ADC_SETTINGS_RESPONSE;
```

Response command: set A/D converter settings

CMD_SET_ANTENNA_NAME_REQUEST

```
public static const unsigned short CMD_SET_ANTENNA_NAME_REQUEST;
```

Request command: set antenna name

CMD_SET_ANTENNA_NAME_RESPONSE

```
public static const unsigned short CMD_SET_ANTENNA_NAME_RESPONSE;
```

Response command: set antenna name

CMD_SET_ONLINE_POLL_INTERVAL_REQUEST

```
public static const unsigned short
CMD_SET_ONLINE_POLL_INTERVAL_REQUEST;
```

Request command: set receiver online poll interval

CMD_SET_ONLINE_POLL_INTERVAL_RESPONSE

```
public static const unsigned short
CMD_SET_ONLINE_POLL_INTERVAL_RESPONSE;
```

Response command: set receiver online poll interval

CMD_START_DATA_REQUEST

```
public static const unsigned short CMD_START_DATA_REQUEST;
```

Request command: start data capture

CMD_STOP_DATA_REQUEST

```
public static const unsigned short CMD_STOP_DATA_REQUEST;
```

Request command: stop data capture

CMD_STOP_DATA_RESPONSE

```
public static const unsigned short CMD_STOP_DATA_RESPONSE;
```

Response command: stop data capture

CMD_VERSION_REQUEST

```
public static const unsigned short CMD_VERSION_REQUEST;
```

Request command: Query NRC version information

CMD_VERSION_RESPONSE

```
public static const unsigned short CMD_VERSION_RESPONSE;
```

Response command: Query NRC version information

CMD_WRITEMEM_REQUEST

```
public static const unsigned short CMD_WRITEMEM_REQUEST;
```

Request command: Write receiver memory location

CMD_WRITEMEM_RESPONSE

```
public static const unsigned short CMD_WRITEMEM_RESPONSE;
```

Response command: Write receiver memory location

EALREADYCON

```
public static const unsigned short EALREADYCON;
```

Error data: A receiver is already connected

EARGUMENT

```
public static const unsigned short EARGUMENT;
```

Error data: The command argument is invalid

ENORESOURCE

```
public static const unsigned short ENORESOURCE;
```

Error data: No more receivers are available

ENOTCONNECTED

```
public static const unsigned short ENOTCONNECTED;
```

Error data: No receiver is connected

ERBADCMDTRANSLATION

```
public static const unsigned short ERBADCMDTRANSLATION;
```

Error data: The command could not be translated

ERCANNOTATTACHRECEIVER

```
public static const unsigned short ERCANNOTATTACHRECEIVER;
```

Error data: Cannot attach the receiver

ERCANNOTREMOVERECEIVER

```
public static const unsigned short ERCANNOTREMOVERECEIVER;
```

Error data: Cannot remove the receiver

ERCAPTURING

```
public static const unsigned short ERCAPTURING;
```

Error data: Error while capturing audio data

ERCHANALRDYCON

```
public static const unsigned short ERCHANALRDYCON;
```

Error data: The channel requested was already connected to another socket

---

ERCHANSELFTESTPENDING

```
public static const unsigned short ERCHANSELFTESTPENDING;
```

Error data: The channel requested is has a self-test pending

ERCHANSELFTESTRUNNING

```
public static const unsigned short ERCHANSELFTESTRUNNING;
```

Error data: The channel requested is has a self-test running

ERINVALIDCHAN

```
public static const unsigned short ERINVALIDCHAN;
```

Error data: An invalid channel value was used in the packet message

---

ERINVALIDCMD

```
public static const unsigned short ERINVALIDCMD;
```

Error data: An invalid command value was used in the packet message

---

ERINVALIDLEN

```
public static const unsigned short ERINVALIDLEN;
```

Error data: An invalid length value was used in the packet message

---

ERINVALIDSAMPLERATE

```
public static const unsigned short ERINVALIDSAMPLERATE;
```

Error data: The requested sample rate is invalid

ERINVALIDSAMPLESPERPACKET

```
public static const unsigned short ERINVALIDSAMPLESPERPACKET;
```

Error data: An invalid number of samples per packet was specified

---

ERINVALIDVGCFILTEROPTION

```
public static const unsigned short ERINVALIDVGCFILTEROPTION;
```

Error data: An invalid VGC Filter option for audio data

ERNODATAWRITTEN

```
public static const unsigned short ERNODATAWRITTEN;
```

Error data: Zero bytes of data written to the socket

ERNORECEIVERDEF

```
public static const unsigned short ERNORECEIVERDEF;
```

Error data: A receiver definition file could not be found

ERREADONLY

```
public static const unsigned short ERREADONLY;
```

Error data: The requested command requires write access to the channel.

ERSOCKALRDYCON

```
public static const unsigned short ERSOCKALRDYCON;
```

Error data: The socket already has a channel connected to it

ERSOCKNOTCON

```
public static const unsigned short ERSOCKNOTCON;
```

Error data: The socket and a channel have not yet been connected

---

ERTIMEOUT

```
public static const unsigned short ERTIMEOUT;
```

Error data: The command timed out

ERUNKNOWN

```
public static const unsigned short ERUNKNOWN;
```

Error data: An unknown error occurred

ERWRITEERROR

```
public static const unsigned short ERWRITEERROR;
```

Error data: An unspecified error has occurred

ESERIALOVLD

```
public static const unsigned short ESERIALOVLD;
```

Error data: The serial port is overloaded

FLAG_ADC_USE_DIFFERENTIAL_MODE

```
public static const unsigned short FLAG_ADC_USE_DIFFERENTIAL_MODE;
```

Flag: Use differential input mode for A/D converter. Omit this flag to specify single-ended input mode.

FLAG_ADC_USE_EXT_CLOCK

```
public static const unsigned short FLAG_ADC_USE_EXT_CLOCK;
```

Flag: Use external clock signal for A/D converter. Omit this flag to specify internal clocking.

SUCCESS

```
public static const unsigned short SUCCESS;
```

Error data: success (no error)

Method Detail

IsReadCommand

```
public static bool IsReadCommand( unsigned short cmd );
```

Check to see if a command is "read only." (That is, check to see if a command only reads status information and does not change any settings which may effect other clients connected to this channel.)

Parameters:

`cmd` - The command to check

Returns:

True if the command is "Read Only"

NRC Java SDK API Reference

Class ChannelBean

```
java.lang.Object
  |
  +--com.aegis.NRC.sdk.java.ChannelBean
```

---

public class **ChannelBean**
extends java.lang.Object

The `ChannelBean` class encapsulates a single channel in the NRC. A channel is defined as a duplex data stream that can be connected to one of the NRC receivers. The receiver can be controlled through the channel, and its digitized audio data can be received through the channel. The `ChannelBean` class provides methods that allow this functionality to occur. Instances of the `ChannelBean` class can be obtained via the `getChannel()` or `connectToChannel()` methods of the `NRC` class. Please note that most operations on a channel are only allowed if that channel is connected to the client using the NRC::connectToChannel() methods.

Field Summary

static int	**FILTERING_OFF** Constant: defines when the vgc filter is OFF
static int	**LPF_4KHZ_ON** Constant: defines when the 4KHz LPF is ON
static java.lang.String	**PROP_ANTENNA** The property name attached to events describing the antenna connected to the receiver attached to this channel.
static java.lang.String	**PROP_CLIENT_LOCATION** The property name attached to events describing the location of the client which has acquired this channel.
static java.lang.String	**PROP_DATA_CAPTURE** The property name attached to events describing the status of data capture on this channel.
static java.lang.String	**PROP_DATA_CAPTURE_FAILED** The property name attached to events describing the failure of data capture on this channel.
static java.lang.String	**PROP_RECEIVER_MODEL** The property name attached to events describing the model of the receiver attached to this channel.

static java.lang.String	**PROP_RECEIVER_ONLINE** The property name attached to events describing the receiver's online status.
static java.lang.String	**PROP_WRITABLE** The property name attached to events describing the write-ablity of the channel
static int	**SAMPLE_RATE_16KHZ** Constant: defines 16KHz sampling rate
static int	**SAMPLE_RATE_8KHZ** Constant: defines 8KHz sampling rate
static int	**TIMESTAMP_OFF** Constant: defines when TAI64N time stamp is off
static int	**TIMESTAMP_ON** Constant: defines when TAI64N time stamp is on

Method Summary

void	**addPropertyChangeListener**(java.beans.PropertyChangeListener listener) Adds a property change listener to this bean.
boolean	**captureDataToFile**(java.io.File file, int sampleRate, int samplesPerPacket, int filterOption, int timeStamp) Starts capturing digitized audio data from the receiver attached to this channel and sends it to the specified file.
boolean	**captureDataToProcessor**(com.aegis.NRC.sdk.java.SampleProcessor processor, int sampleRate, int samplesPerPacket, int filterOption, int timeStampOpt) Starts capturing digitized audio data from the receiver attached to this channel and sends it to the specified processor.
boolean	**captureDataToSocket**(java.lang.String hostname, int port, int sampleRate, int samplesPerPacket, int filterOption, int timeStamp) Starts capturing digitized audio data from the receiver attached to this channel and forwards it to the server socket specified by the hostname and port.
boolean	**changeAntenna**(java.lang.String newAntenna) Attempts to change the name of the antenna connected to the receiver that is attached to this channel.
boolean	**changeReceiverModel**(java.lang.String model) Attempts to change the receiver that is attached to this channel to a different manufacturer's model.
java.lang.String	**getAGCMode**()

	Returns the agc mode setting of the channel receiver.
java.lang.String	**getAntenna**() Returns the name of the antenna connected to the receiver currently attached to this channel.
java.lang.String	**getAntenna**(boolean forceQuery) Returns the name of the antenna connected to the receiver currently attached to this channel.
int	**getBFO**() Returns the BFO setting (in Hz) of the receiver attached to this channel.
java.io.File	**getCaptureFile**() Returns the name of the file that data is currently being captured to.
java.lang.String	**getCaptureHost**() Returns the hostname of the computer that is receiving captured data.
int	**getCapturePort**() Returns the socket port of the computer that is receiving captured data.
java.lang.String	**getClientLocation**() Returns the location of the channel client.
boolean	**getDataOptionsFromServer**() Get the current data options on this channel from the server
java.lang.String	**getDetectionMode**() Returns the detection mode setting of the channel receiver.
java.lang.String[]	**getDetectionModeList**() Returns the list of valid detection modes of the receiver attached to this channel.
int	**getFilterOption**() Returns the filter option for the channel data stream.
double	**getFrequency**() Returns the frequency setting in MHz of the channel receiver.
int	**getID**() Returns the numeric identifier for this channel.
int	**getIFBandwidth**() Returns the current IF bandwidth setting (in Hz) of the receiver attached to this channel.
int	**getMemoryCapacity**()
com.aegis.NRC.sdk.java.NRCBean	**getNRCBean**() Gets the bean for the NRC that this channel belongs to.

int	**getOnlinePollInterval**() Returns the online polling interval, in seconds, of the channel receiver.
java.lang.String	**getReceiverConfiguration**() Returns a string that contains configuration information of the receiver attached to this channel.
java.lang.String	**getReceiverModel**() Returns the manufacturer's model designation of the receiver currently attached to this channel.
java.lang.String	**getReceiverModel**(boolean forceQuery) Returns the manufacturer's model designation of the receiver currently attached to this channel.
java.lang.String[]	**getReceiverModelList**() Returns the list of valid receiver models that can be attached to this channel of the NRC.
java.lang.String	**getReceiverStatus**() Returns a string that defines the status of the receiver attached to this channel.
int	**getSampleRate**() Returns a constant representing the capture sample rate (in Hz) setting for this channel.
int	**getSamplesPerPacket**() Returns the number of samples per packet that should be received when capturing digitized receiver audio data.
int	**getSamplesSkippedOnTune**() Returns the number of samples to skip following a change in receiver settings, such as a change in frequency, detection mode, IF bandwidth, and BFO.
int	**getTimeStampOption**() Returns the time stamp option for the channel data stream.
boolean	**isCapturing**() Returns true if this client is capturing audio samples from this channel, otherwise false.
boolean	**isOnline**() Returns true if this channel receiver is online, false otherwise.
boolean	**isWritable**() Returns true if this client can change properties of the receiver, false otherwise
boolean	**loadMemoryLocation**(int loc) Load the receiver settings stored at a memory location
java.lang.String	**queryMemoryFields**()

	Return the settings the receiver stores for each memory location
java.lang.String	**queryMemoryLocation**(int loc) Return the settings stored in a receiver memory location
void	**rebootReceiver**() Re-boot the channel receiver to rectify fault condition It may take a few seconds for this command to take effect.
void	**removePropertyChangeListener**(java.beans.PropertyChangeListener listener) Removes a property change listener from this bean.
void	**resetReceiver**() Re-initialize the channel receiver to the factory default conditions.
void	**sendReceiverCommand**(java.lang.String cmd) Sends the given command directly to the channel receiver.
java.lang.String	**sendReceiverCommand**(java.lang.String cmd, boolean expectResponse) Sends the given command directly to the channel receiver.
void	**setAGCMode**(java.lang.String agcmode) Adjusts the agc mode setting of the channel receiver.
void	**setBFO**(int bfo) Adjusts the BFO setting (in Hz) of the channel receiver.
void	**setDetectionMode**(java.lang.String detmode) Adjusts the detection mode setting of the channel receiver.
void	**setFrequency**(double freq) Tunes the channel receiver to the specified frequency (in MHz).
void	**setIFBandwidth**(int ifbw) Adjusts the IF bandwidth setting (in Hz) of the receiver attached to this channel.
boolean	**setOnlinePollInterval**(int interval) Sets the interval in seconds at which this channel receiver will be tested to see if it is online.
void	**setSamplesSkippedOnTune**(int skip) Sets the number of samples to skip following a change in receiver settings, such as a change in frequency, detection mode, IF bandwidth, and BFO.
void	**setWritable**(boolean val)
boolean	**stopDataCapture**() Stops capturing digitized audio data from the receiver

	attached to this channel.
boolean	**writeMemoryLocation**(int loc) Write the current receiver settings to a memory location

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

PROP_RECEIVER_ONLINE

public static final java.lang.String **PROP_RECEIVER_ONLINE**

The property name attached to events describing the receiver's online status.

See Also:

Constant Field Values

PROP_WRITABLE

public static final java.lang.String **PROP_WRITABLE**

The property name attached to events describing the write-ablity of the channel

See Also:

Constant Field Values

PROP_CLIENT_LOCATION

public static final java.lang.String **PROP_CLIENT_LOCATION**

The property name attached to events describing the location of the client which has acquired this channel.

See Also:

Constant Field Values

PROP_DATA_CAPTURE

public static final java.lang.String **PROP_DATA_CAPTURE**

The property name attached to events describing the status of data capture on this channel.

See Also:

Constant Field Values

PROP_DATA_CAPTURE_FAILED

public static final java.lang.String **PROP_DATA_CAPTURE_FAILED**

The property name attached to events describing the failure of data capture on this

channel.

See Also:

Constant Field Values

PROP_RECEIVER_MODEL

`public static final java.lang.String` **`PROP_RECEIVER_MODEL`**

The property name attached to events describing the model of the receiver attached to this channel.

See Also:

Constant Field Values

PROP_ANTENNA

`public static final java.lang.String` **`PROP_ANTENNA`**

The property name attached to events describing the antenna connected to the receiver attached to this channel.

See Also:

Constant Field Values

SAMPLE_RATE_16KHZ

`public static final int` **`SAMPLE_RATE_16KHZ`**

Constant: defines 16KHz sampling rate

See Also:

Constant Field Values

SAMPLE_RATE_8KHZ

`public static final int` **`SAMPLE_RATE_8KHZ`**

Constant: defines 8KHz sampling rate

See Also:

Constant Field Values

FILTERING_OFF

`public static final int` **`FILTERING_OFF`**

Constant: defines when the vgc filter is OFF

See Also:

Constant Field Values

LPF_4KHZ_ON

`public static final int` **`LPF_4KHZ_ON`**

Constant: defines when the 4KHz LPF is ON

See Also:

Constant Field Values

TIMESTAMP_OFF

```
public static final int TIMESTAMP_OFF
```

Constant: defines when TAI64N time stamp is off

See Also:

Constant Field Values

TIMESTAMP_ON

```
public static final int TIMESTAMP_ON
```

Constant: defines when TAI64N time stamp is on

See Also:

Constant Field Values

Method Detail

addPropertyChangeListener

```
public void
addPropertyChangeListener(java.beans.PropertyChangeListener listener)
```

Adds a property change listener to this bean. Refer to the class constants available to discover the types of events fired on this bean.

Parameters:

`listener` - the listener to add

---

removePropertyChangeListener

```
public void
removePropertyChangeListener(java.beans.PropertyChangeListener listener)
```

Removes a property change listener from this bean.

Parameters:

`listener` - the listener to remove

---

getNRCBean

```
public com.aegis.NRC.sdk.java.NRCBean getNRCBean()
```

Gets the bean for the NRC that this channel belongs to.

Returns:

the NRC bean

---

getID

```
public int getID()
```

Returns the numeric identifier for this channel.

Returns:

the channel ID

isWritable

`public boolean **isWritable**()`

Returns true if this client can change properties of the receiver, false otherwise

Returns:

true, if capturing, false otherwise.

---

setWritable

`public void **setWritable**(boolean val)`

---

isCapturing

`public boolean **isCapturing**()`

Returns true if this client is capturing audio samples from this channel, otherwise false.

Returns:

true, if capturing, false otherwise.

---

getDataOptionsFromServer

`public boolean **getDataOptionsFromServer**()`

Get the current data options on this channel from the server

Returns:

true if the options were fetched successfully, false if there was an error

---

getSampleRate

`public int **getSampleRate**()`

Returns a constant representing the capture sample rate (in Hz) setting for this channel. Refer to the class constants for the range of return values.

Returns:

the sample rate

---

getFilterOption

`public int **getFilterOption**()`

Returns the filter option for the channel data stream.

Returns:

the filter option for the data channel packet

---

getTimeStampOption

`public int **getTimeStampOption**()`

Returns the time stamp option for the channel data stream.

Returns:
the time stamp option for the data channel packet

---

getSamplesPerPacket

`public int` **`getSamplesPerPacket`**`()`

Returns the number of samples per packet that should be received when capturing digitized receiver audio data. Since each samples is two bytes, the actual size of the packet in bytes will be twice the number of samples per packet.

Returns:
the number of samples to be sent in each data capture packet

---

getCaptureFile

`public java.io.File` **`getCaptureFile`**`()`

Returns the name of the file that data is currently being captured to.

Returns:
the data capture file, if capturing, null otherwise.

---

getCaptureHost

`public java.lang.String` **`getCaptureHost`**`()`

Returns the hostname of the computer that is receiving captured data.

Returns:
the hostname, if capturing, null otherwise.

---

getCapturePort

`public int` **`getCapturePort`**`()`

Returns the socket port of the computer that is receiving captured data.

Returns:
the remote socket port, if capturing, null otherwise.

---

getReceiverModel

`public java.lang.String` **`getReceiverModel`**`()`
`throws NRCException`

Returns the manufacturer's model designation of the receiver currently attached to this channel. This method does not force a query of the NRC. Rather, the last known receiver model is returned. If the receiver model is unknown, the NRC is queried.

Returns:
the receiver model name

Throws:
`NRCException`

---

getReceiverModel

```
public java.lang.String getReceiverModel(boolean forceQuery)
                                  throws NRCException
```

Returns the manufacturer's model designation of the receiver currently attached to this channel.

Parameters:

`forceQuery` - whether to force a query to the NRC or not.

Returns:

the receiver model name

Throws:

`NRCException`

---

changeReceiverModel

```
public boolean changeReceiverModel(java.lang.String model)
                           throws NRCException
```

Attempts to change the receiver that is attached to this channel to a different manufacturer's model. If the NRC supports that model, it will connect to the receiver and this channel will begin operating on the new receiver.

This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Note: the specified receiver model must match a model name known by the NRC. For a list of known receiver models, use `getReceiverModelList()`.

Parameters:

`model` - the manufacturer's model name of the new receiver.

Returns:

true, if successful, false otherwise

Throws:

`NRCException`

---

getAntenna

```
public java.lang.String getAntenna()
                           throws NRCException
```

Returns the name of the antenna connected to the receiver currently attached to this channel. This method does not force a query of the NRC. Rather, the last known antenna name is returned. If the antenna name is unknown, the NRC is queried.

Returns:

the antenna name

Throws:

`NRCException`

---

getAntenna

```
public java.lang.String getAntenna(boolean forceQuery)
                           throws NRCException
```


Returns the name of the antenna connected to the receiver currently attached to this channel.

Parameters:

`forceQuery` - whether to force a query to the NRC or not.

Returns:

the name of the antenna attached to the receiver

Throws:

`NRCException`

---

changeAntenna

```
public boolean changeAntenna(java.lang.String newAntenna)
                      throws NRCException
```

Attempts to change the name of the antenna connected to the receiver that is attached to this channel.

This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Parameters:

`newAntenna` - the name of the antenna

Returns:

true, if successful, false otherwise

Throws:

`NRCException`

---

getClientLocation

```
public java.lang.String getClientLocation()
                                    throws NRCException
```

Returns the location of the channel client. This is typically the hostname or IP address of the client application.

Returns:

the location of the client

Throws:

`NRCException`

---

getReceiverStatus

```
public java.lang.String getReceiverStatus()
                                    throws NRCException
```

Returns a string that defines the status of the receiver attached to this channel. The format of the string is defined by the receiver. Please refer to the receiver operating manual for more information on how to use this string.

This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Returns:

receiver status string

Throws:

`NRCException`

getReceiverConfiguration

```
public java.lang.String getReceiverConfiguration()
                                                 throws NRCException
```

Returns a string that contains configuration information of the receiver attached to this channel. The format of the string is defined in the NRC Interface Control Document. Please refer to this manual for more information on how to use this string.

This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Returns:

receiver configuration string

Throws:

`NRCException`

---

getFrequency

```
public double getFrequency()
                    throws NRCException
```

Returns the frequency setting in MHz of the channel receiver.

This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Returns:

the receiver frequency, in MHz

Throws:

`NRCException`

---

setFrequency

```
public void setFrequency(double freq)
                  throws NRCException
```

Tunes the channel receiver to the specified frequency (in MHz). The range of valid frequencies is defined by the capability of each receiver.

This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Parameters:

`freq` - the frequency to tune to, in MHz

Throws:

`NRCException`

---

getOnlinePollInterval

```
public int getOnlinePollInterval()
                           throws NRCException
```

Returns the online polling interval, in seconds, of the channel receiver. A value of 0 means that polling is currently disabled. A negative value denotes that an error has occurred.

This can only be performed if this channel has been acquired by the client using an `NRC::connectToChannel()` call.

Returns:

the online polling interval in seconds, or 0 if disabled

`NRCException`

---

setOnlinePollInterval

```
public boolean setOnlinePollInterval(int interval)
                              throws NRCException
```

Sets the interval in seconds at which this channel receiver will be tested to see if it is online. An interval of 0 disables online polling on this channel. This is helpful when performing a lot of receiver operations as the online polling can slow down receiver performance.

This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Parameters:

`interval` - the new polling interval in seconds, or 0 to stop

Returns:

true, if successful, false otherwise

`NRCException`

---

getSamplesSkippedOnTune

```
public int getSamplesSkippedOnTune()
                            throws NRCException
```

Returns the number of samples to skip following a change in receiver settings, such as a change in frequency, detection mode, IF bandwidth, and BFO. This helps eliminate invalid audio data during the receiver transition.

This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Returns:

the number of samples to skip

Throws:

`NRCException`

---

setSamplesSkippedOnTune

```
public void setSamplesSkippedOnTune(int skip)
                             throws NRCException
```

Sets the number of samples to skip following a change in receiver settings, such as a change in frequency, detection mode, IF bandwidth, and BFO. This helps eliminate invalid audio data during the receiver transition.

This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Parameters:

`skip` - the number of samples following receiver tune

Throws:
`NRCException`

---

getAGCMode

```
public java.lang.String getAGCMode()
                                throws NRCException
```

Returns the agc mode setting of the channel receiver.
This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Returns:
the agc mode

Throws:
`NRCException`

---

setAGCMode

```
public void setAGCMode(java.lang.String agcmode)
                throws NRCException
```

Adjusts the agc mode setting of the channel receiver. The valid range of agc modes are "SLOW", "MEDIUM", and "FAST".
This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Parameters:
`agcmode` - the agc mode

Throws:
`NRCException`

---

getDetectionMode

```
public java.lang.String getDetectionMode()
                                     throws NRCException
```

Returns the detection mode setting of the channel receiver.
This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Returns:
the detection mode

Throws:
`NRCException`

---

setDetectionMode

```
public void setDetectionMode(java.lang.String detmode)
                      throws NRCException
```

Adjusts the detection mode setting of the channel receiver. The valid range of detection modes are defined by each receiver. A list of valid detection modes for the channel receiver can be obtained by calling `getDetectionModeList()`.
This can only be performed if this channel has been acquired by the client using

an NRCBean.connectToChannel() call.

Parameters:

detmode - the detection mode

Throws:

NRCException

---

getDetectionModeList

```
public java.lang.String[] getDetectionModeList()
                                        throws NRCException
```

Returns the list of valid detection modes of the receiver attached to this channel. This can only be performed if this channel has been acquired by the client using an NRCBean.connectToChannel() call.

Returns:

the detection mode list

Throws:

NRCException

---

getReceiverModelList

```
public java.lang.String[] getReceiverModelList()
                                        throws NRCException
```

Returns the list of valid receiver models that can be attached to this channel of the NRC. This can only be performed if this channel has been acquired by the client using an NRC::connectToChannel() call.
This can only be performed if this channel has been acquired by the client using an NRCBean.connectToChannel() call.

Returns:

the valid receiver model list

Throws:

NRCException

---

getBFO

```
public int getBFO()
           throws NRCException
```

Returns the BFO setting (in Hz) of the receiver attached to this channel.
This can only be performed if this channel has been acquired by the client using an NRCBean.connectToChannel() call.

Returns:

the BFO setting, in Hz

Throws:

NRCException

---

setBFO

```
public void setBFO(int bfo)
            throws NRCException
```


Adjusts the BFO setting (in Hz) of the channel receiver. The range of valid BFO settings is defined by each receiver's capability.
This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Parameters:

`bfo` - the BFO to set, in Hz

Throws:

`NRCException`

---

getIFBandwidth

```
public int getIFBandwidth()
                  throws NRCException
```

Returns the current IF bandwidth setting (in Hz) of the receiver attached to this channel.
This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Returns:

the IF bandwidth (in Hz)

Throws:

`NRCException`

---

setIFBandwidth

```
public void setIFBandwidth(int ifbw)
                   throws NRCException
```

Adjusts the IF bandwidth setting (in Hz) of the receiver attached to this channel. The range of valid IF bandwidths is defined by each receiver's capability.
This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Parameters:

`ifbw` - the IF bandwidth to set (in Hz)

Throws:

`NRCException`

---

resetReceiver

```
public void resetReceiver()
                  throws NRCException
```

Re-initialize the channel receiver to the factory default conditions. It may take a few seconds for this command to take effect.
This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Throws:

`NRCException`

---

rebootReceiver

```
public void rebootReceiver()
                    throws NRCException
```

Re-boot the channel receiver to rectify fault condition It may take a few seconds for this command to take effect.

This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Throws:

`NRCException`

---

sendReceiverCommand

```
public void sendReceiverCommand(java.lang.String cmd)
                         throws NRCException
```

Sends the given command directly to the channel receiver. Note: no translation of this command will occur; it is delivered to the receiver as-is.

This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Parameters:

`cmd` - the receiver command

Throws:

`NRCException`

---

sendReceiverCommand

```
public java.lang.String sendReceiverCommand(java.lang.String cmd,
                                            boolean expectResponse)
                                     throws NRCException
```

Sends the given command directly to the channel receiver. Note: no translation of this command will occur; it is delivered to the receiver as-is.

This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Parameters:

`cmd` - the receiver command

`expectResponse` - if true, a response string is returned from the receiver

Throws:

`NRCException`

---

isOnline

```
public boolean isOnline()
```

Returns true if this channel receiver is online, false otherwise.

This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Returns:

true, if online, false otherwise

---

getMemoryCapacity

```
public int getMemoryCapacity()
                    throws NRCException
```

Returns:

the number of memory locations the receiver supports

`NRCException`

---

queryMemoryFields

```
public java.lang.String queryMemoryFields()
                                  throws NRCException
```

Return the settings the receiver stores for each memory location

`NRCException`

---

queryMemoryLocation

```
public java.lang.String queryMemoryLocation(int loc)
                                    throws NRCException
```

Return the settings stored in a receiver memory location

Parameters:

`loc` - the location to query

`NRCException`

---

loadMemoryLocation

```
public boolean loadMemoryLocation(int loc)
                           throws NRCException
```

Load the receiver settings stored at a memory location

Parameters:

`loc` - the location to load

`NRCException`

---

writeMemoryLocation

```
public boolean writeMemoryLocation(int loc)
                            throws NRCException
```

Write the current receiver settings to a memory location

Parameters:

`loc` - the location to write to

`NRCException`

---

captureDataToProcessor

```
public boolean
captureDataToProcessor(com.aegis.NRC.sdk.java.SampleProcessor processor,
                                    int sampleRate,
                                    int samplesPerPacket,
                                    int filterOption,
```



```
                                    int timeStampOpt)
                             throws NRCException
```

Starts capturing digitized audio data from the receiver attached to this channel and sends it to the specified processor.
This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Parameters:

`processor` - the processor to capture to
`sampleRate` - the sample rate to use (see class constants)
`samplesPerPacket` - the samples per packet to request
`filterOption` - the filter option associated with the channel data
`timeStampOpt` - the time stamp option associated with the channel data (see class constants)

Returns:

true, if successful, false otherwise

Throws:

`NRCException`

---

captureDataToFile

```
public boolean captureDataToFile(java.io.File file,
                                 int sampleRate,
                                 int samplesPerPacket,
                                 int filterOption,
                                 int timeStamp)
                          throws NRCException
```

Starts capturing digitized audio data from the receiver attached to this channel and sends it to the specified file.
This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Parameters:

`file` - the file to capture to
`sampleRate` - the sample rate to use (see class constants)
`samplesPerPacket` - the samples per packet to request
`filterOption` - the filter option associated with the channel data

Returns:

true, if successful, false otherwise

Throws:

`NRCException`

---

captureDataToSocket

```
public boolean captureDataToSocket(java.lang.String hostname,
                                   int port,
                                   int sampleRate,
                                   int samplesPerPacket,
                                   int filterOption,
                                   int timeStamp)
```



```
                                    throws NRCException
```

Starts capturing digitized audio data from the receiver attached to this channel and forwards it to the server socket specified by the hostname and port. Note: multi-byte samples are received in network byte order.

This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Parameters:

`hostname` - the host to forward data to

`port` - the server socket port on the remote host

`sampleRate` - the sample rate to use (see class constants)

`samplesPerPacket` - the samples per packet to request

`filterOption` - the filter option associated with the channel data

`timeStamp` - the time stamp option associated with the channel data

Returns:

true, if successful, false otherwise

Throws:

`NRCException`

---

stopDataCapture

```
public boolean stopDataCapture()
                             throws NRCException
```

Stops capturing digitized audio data from the receiver attached to this channel. The file, socket, or processor destination is also closed.

This can only be performed if this channel has been acquired by the client using an `NRCBean.connectToChannel()` call.

Returns:

true, if successful, false otherwise

Throws:

`NRCException`

Class NRCBean

```
java.lang.Object
  |
  +--com.aegis.NRC.sdk.java.NRCBean
```

All Implemented Interfaces:

com.aegis.jcommon.protocol.RPMProtocolMgrOwner

---

public class **NRCBean**
extends java.lang.Object
implements com.aegis.jcommon.protocol.RPMProtocolMgrOwner

This class is a representation of a connection to an NRC. This class is used to connect and disconnect from an NRC as well as the channels within an NRC. This class is also used to set and access NRC properties.

Field Summary

static java.lang.String	**DEFAULT_CONNECT_PORT** The default NRC socket port
static int	**DEFAULT_CONNECT_TIMEOUT** The default timeout (in milliseconds) for connecting to the NRC
static int	**DEFAULT_RESPONSE_TIMEOUT** The default NRC socket response timeout (in millisecs)
static int	**DEFAULT_SOCKET_TIMEOUT** The default SO_TIMEOUT (in milliseconds)
static java.lang.String	**PROP_CONNECTED_CHANNEL** Bean property fired when connected channel changes
static java.lang.String	**PROP_NRC_CONNECTION_DROPPED** Bean property fired NRC connection is dropped

Constructor Summary

NRCBean() Default constructor

Method Summary

void	**addPropertyChangeListener**(java.beans.PropertyChangeListener l) Add a property change listener to the bean
com.aegis.NRC.sdk.java.ChannelBean	**connectToChannel**() Attempts to connect to the next available channel.
com.aegis.NRC.sdk.java.ChannelBean	**connectToChannel**(int channel) Attempts to connect to the channel with the specified ID.
boolean	**connectToNRC**(java.lang.String IPAddress) Establishes a connection to an NRC located at the given IP address (dot-notation) or DNS-resolvable host name.
boolean	**connectToNRC**(java.lang.String IPAddress, java.lang.String nport) Establishes a connection to an NRC located at the given IP address (dot-notation) or DNS-resolvable host name and socket port.
boolean	**connectToNRC**(java.lang.String IPAddress, java.lang.String nport, int socketTimeout, int connectTimeout) Establishes a connection to an NRC located at the given IP address (dot-notation) or DNS-resolvable host name and socket port using the specified socket options.
boolean	**disconnectFromChannel**() Disconnects from the currently connected channel.
boolean	**disconnectFromNRC**() Terminates the connection to the NRC.
int[]	**getADCSettings**() Gets the current ADC (Audio/Digital Converter) settings from the NRC.
java.lang.String	**getAntenna**(int channel) Returns the name of the antenna connected to the receiver attached to the channel with the given channel ID.
com.aegis.NRC.sdk.java.ChannelBean	**getChannel**(int channel) Returns the channel with the given integer identifier.
int	**getChannelCount**() Returns the number of channels contained in the NRC.
java.lang.String	**getChannelSummary**() Returns the client location and receiver model for each channel of the NRC.
java.lang.String	**getClientLocation**(int channel) Returns the location of the client connected to the channel with the given ID.

com.aegis.NRC.sdk.java.ChannelBean	**getConnectedChannel**() Returns the channel that the client is currently connected to, or null, if no current connection.
int	**getConnectTimeout**() Returns the initial NRC connection timeout.
java.io.InputStream	**getInputStream**() Returns the socket input stream of the NRC connection.
java.lang.String	**getIPAddress**() Returns the IP address (dot-notation) or DNS-resolvable host name of the connected NRC; or null if no connection.
byte[]	**getMonitorData**() Returns the monitor data fetched from the NRC.
java.lang.String	**getNRCConfiguration**() Returns the NRC configuration.
java.io.OutputStream	**getOutputStream**() Returns the socket output stream of the NRC connection.
java.lang.String	**getPort**() Returns the NRC socket port value if the client has connected to the NRC.
java.lang.String	**getReceiverModel**(int channel) Returns the manufacturer and model name of receiver attached to the channel with the given channel ID.
int	**getResponseTimeout**() Returns the current timeout (in milliseconds) for an expected response from the NRC server.
int	**getSocketTimeout**() Returns the SO_TIMEOUT socket timeout, in milliseconds.
ChannelDocumentHandler.FileInfo[]	**getVersionInfo**() Requests version information from the NRC.
boolean	**isConnected**() Returns true if the client is connected to an NRC, false otherwise.
void	**notifyRPMProtocolMgrClosed**(com.aegis.jcommon.protocol.RPMProtocolMgr mgr) Used internally by the protocol manager to notify NRCBean if the NRC connection is dropped.
boolean	**ping**() Sends a ping command to the connected NRC and waits for an appropriate response.
boolean	**rebootNRC**() Causes the NRC software to perform a warm restart.
void	**removePropertyChangeListener**(java.beans.PropertyChang

	eListener l) Remove a property change listener to the bean
void	**setADCSettings**(int flags, int ch0Cfg, int ch1Cfg, int ch2Cfg, int ch3Cfg, int ch4Cfg, int ch5Cfg, int ch6Cfg, int ch7Cfg) Changes the ADC (Audio/Digital Converter) settings in the NRC.
void	**setResponseTimeout**(int milliseconds) Sets the timeout (in milliseconds) for an expected response from the NRC server.
void	**setWritable**(boolean val) Notify the NRCBean that a channel's write-ability has changed
java.lang.String	**toString**() Returns the IP address (dot-notation) or DNS-resolvable host name of the connected NRC; or null if no connection.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

DEFAULT_SOCKET_TIMEOUT

public static final int **DEFAULT_SOCKET_TIMEOUT**

The default SO_TIMEOUT (in milliseconds)

See Also:

Constant Field Values

DEFAULT_CONNECT_TIMEOUT

public static final int **DEFAULT_CONNECT_TIMEOUT**

The default timeout (in milliseconds) for connecting to the NRC

See Also:

Constant Field Values

DEFAULT_CONNECT_PORT

public static final java.lang.String **DEFAULT_CONNECT_PORT**

The default NRC socket port

See Also:

Constant Field Values

DEFAULT_RESPONSE_TIMEOUT

`public static final int` **`DEFAULT_RESPONSE_TIMEOUT`**

The default NRC socket response timeout (in millisecs)

See Also:

Constant Field Values

PROP_CONNECTED_CHANNEL

`public static final java.lang.String` **`PROP_CONNECTED_CHANNEL`**

Bean property fired when connected channel changes

See Also:

Constant Field Values

PROP_NRC_CONNECTION_DROPPED

`public static final java.lang.String` **`PROP_NRC_CONNECTION_DROPPED`**

Bean property fired NRC connection is dropped

See Also:

Constant Field Values

Constructor Detail

NRCBean

`public` **`NRCBean`**`()`

Default constructor

Method Detail

addPropertyChangeListener

`public void`
`addPropertyChangeListener``(java.beans.PropertyChangeListener l)`

Add a property change listener to the bean

---

removePropertyChangeListener

`public void`
`removePropertyChangeListener``(java.beans.PropertyChangeListener l)`

Remove a property change listener to the bean

---

connectToNRC

`public boolean` **`connectToNRC`**`(java.lang.String IPAddress)`

Establishes a connection to an NRC located at the given IP address (dot-notation) or DNS-resolvable host name.

Parameters:

`IPAddress` - location of NRC

Returns:

true on success, false otherwise

---

connectToNRC

```
public boolean connectToNRC(java.lang.String IPAddress,
                            java.lang.String nport)
```

Establishes a connection to an NRC located at the given IP address (dot-notation) or DNS-resolvable host name and socket port.

Parameters:

`IPAddress` - location of NRC

`nport` - the socket port to connect to

Returns:

true on success, false otherwise

---

connectToNRC

```
public boolean connectToNRC(java.lang.String IPAddress,
                            java.lang.String nport,
                            int socketTimeout,
                            int connectTimeout)
```

Establishes a connection to an NRC located at the given IP address (dot-notation) or DNS-resolvable host name and socket port using the specified socket options.

Parameters:

`IPAddress` - location of NRC (host name or dot-separated IP address)

`nport` - the socket port to connect to

`socketTimeout` - set SO_TIMEOUT with the specified timeout, in milliseconds.

`connectTimeout` - NRC socket connection timeout

Returns:

true on success, false otherwise

---

disconnectFromNRC

```
public boolean disconnectFromNRC()
```

Terminates the connection to the NRC.

Returns:

true if successful, false otherwise

---

getResponseTimeout

```
public int getResponseTimeout()
```

Returns the current timeout (in milliseconds) for an expected response from the NRC server.

Returns:

response timeout, in millisecs

---

setResponseTimeout

`public void` **`setResponseTimeout`**`(int milliseconds)`

Sets the timeout (in milliseconds) for an expected response from the NRC server.

Parameters:

`milliseconds` - the timeout (in millisecs)

---

getSocketTimeout

`public int` **`getSocketTimeout`**`()`

Returns the SO_TIMEOUT socket timeout, in milliseconds.

Returns:

the SO_TIMEOUT socket timeout, in milliseconds.

---

getConnectTimeout

`public int` **`getConnectTimeout`**`()`

Returns the initial NRC connection timeout.

Returns:

the initial NRC connection timeout.

---

getIPAddress

`public java.lang.String` **`getIPAddress`**`()`

Returns the IP address (dot-notation) or DNS-resolvable host name of the connected NRC; or null if no connection.

Returns:

the network location of the connected NRC, or null if no connection.

---

getPort

`public java.lang.String` **`getPort`**`()`

Returns the NRC socket port value if the client has connected to the NRC.

Returns:

the socket port of connection to the NRC.

---

getInputStream

`public java.io.InputStream` **`getInputStream`**`()`

Returns the socket input stream of the NRC connection.

Returns:

the socket input stream

---

getOutputStream

`public java.io.OutputStream` **`getOutputStream`**`()`

Returns the socket output stream of the NRC connection.

Returns:
the socket output stream

---

isConnected

```
public boolean isConnected()
```

Returns true if the client is connected to an NRC, false otherwise.

Returns:
true if connected to NRC, false otherwise

---

ping

```
public boolean ping()
```

Sends a ping command to the connected NRC and waits for an appropriate response. True is returned if the correct response is received; otherwise, false is returned.

Returns:
true if successful, false otherwise

---

getChannelCount

```
public int getChannelCount()
                    throws NRCException
```

Returns the number of channels contained in the NRC.

Returns:
channel count
`NRCException`

---

getChannel

```
public com.aegis.NRC.sdk.java.ChannelBean getChannel(int channel)
                                                  throws NRCException
```

Returns the channel with the given integer identifier.

Parameters:
`channel` - the ID of the channel to return [0 .. `getChannelCount()` -1]

Returns:
the channel with the given ID.
`NRCException`

---

setWritable

```
public void setWritable(boolean val)
```

Notify the NRCBean that a channel's write-ability has changed

---

connectToChannel

```
public com.aegis.NRC.sdk.java.ChannelBean connectToChannel(int channel)
```



```
                                                    throws NRCException
```

Attempts to connect to the channel with the specified ID. If successful, returns the channel record.

Parameters:

`channel` - the ID of the channel to connect to [0 .. `getChannelCount()` -1]

Returns:

channel record of connected channel, or NULL on failure

`NRCException`

---

disconnectFromChannel

```
public boolean disconnectFromChannel()
                                throws NRCException
```

Disconnects from the currently connected channel.

Returns:

true if successful, false otherwise

`NRCException`

---

connectToChannel

```
public com.aegis.NRC.sdk.java.ChannelBean connectToChannel()
                                                    throws NRCException
```

Attempts to connect to the next available channel. If successful, returns the channel record.

Returns:

channel record of connected channel, or NULL on failure

`NRCException`

---

getConnectedChannel

```
public com.aegis.NRC.sdk.java.ChannelBean getConnectedChannel()
```

Returns the channel that the client is currently connected to, or null, if no current connection.

Returns:

current connected channel, or null, if no connection

---

getReceiverModel

```
public java.lang.String getReceiverModel(int channel)
                                    throws NRCException
```

Returns the manufacturer and model name of receiver attached to the channel with the given channel ID. A zero-length string is returned if no receiver is connected.

Parameters:

`channel` - the channel ID [0 .. `getChannelCount()` -1]

Returns:

the receiver model at the given channel

`NRCException`

getAntenna

```
public java.lang.String getAntenna(int channel)
                            throws NRCException
```

Returns the name of the antenna connected to the receiver attached to the channel with the given channel ID. A zero-length string is returned if no name is provided for the antenna.

Parameters:

`channel` - the channel ID [0 .. `getChannelCount()` -1]

Returns:

the name of the antenna connected to the receiver

`NRCException`

---

getClientLocation

```
public java.lang.String getClientLocation(int channel)
                                    throws NRCException
```

Returns the location of the client connected to the channel with the given ID. This is typically the IP address of the attached client. If no client is connected, "Not connected" is returned.

Parameters:

`channel` - the ID of the channel to connect to [0 .. `getChannelCount()` -1]

Returns:

the client location at the given channel

`NRCException`

---

getChannelSummary

```
public java.lang.String getChannelSummary()
                                    throws NRCException
```

Returns the client location and receiver model for each channel of the NRC. The string format is: | ch | model length (l1) | model | location length | location | ch ...

Returns:

encoded string

`NRCException`

---

getNRCConfiguration

```
public java.lang.String getNRCConfiguration()
                                      throws NRCException
```

Returns the NRC configuration. The string format is defined in the NRC Interface Control Document.

Returns:

encoded string

`NRCException`

---

rebootNRC

```
public boolean rebootNRC()
                  throws NRCException
```

Causes the NRC software to perform a warm restart. Any currently active socket connections will be terminated and all parameters will be restored to their initial power up condition. A reboot will take about one minute to complete. Any attempts to establish a socket connection with the Network Receiver Controller during the reboot period will fail.

Returns:

true if command is issued

`NRCException`

---

getVersionInfo

```
public ChannelDocumentHandler.FileInfo[] getVersionInfo()
                                                    throws NRCException
```

Requests version information from the NRC.

Returns:

array of file info

`NRCException`

---

getADCSettings

```
public int[] getADCSettings()
                     throws NRCException
```

Gets the current ADC (Audio/Digital Converter) settings from the NRC.

Returns:

the current settings. Constants beginning with "FLAG_ADC" from com.aegis.NRC.sdk.java.RPMConstants have been ORed together to get the reported setting.

`NRCException`

---

getMonitorData

```
public byte[] getMonitorData()
                      throws NRCException
```

Returns the monitor data fetched from the NRC.

`NRCException`

---

setADCSettings

```
public void setADCSettings(int flags,
                           int ch0Cfg,
                           int ch1Cfg,
                           int ch2Cfg,
                           int ch3Cfg,
                           int ch4Cfg,
                           int ch5Cfg,
```



```
                    int ch6Cfg,
                    int ch7Cfg)
             throws NRCException
```

Changes the ADC (Audio/Digital Converter) settings in the NRC. If any specified settings are different than the existing set, the NRC data acquisition process is "rebooted", which will disrupt data capture on all channels. Use with care.

Parameters:

`flags` - ADCSetting - class to access: Clock & Diff/Single Ended mode settings - Constants from com.aegis.NRC.sdk.java.RPMConstants should be ORed together to get the desired setting.

`ch0Cfg` - ch<0-7>Cfg the ADC channel settings. - bit 0-1: Gain: Gain=1: (00), Gain=2: (01), Gain=4: (10), Gain=8: (11) - bit 8: RESERVED - for Slow bit setting: Slow on (1), Slow off (0)

`NRCException`

---

notifyRPMProtocolMgrClosed

```
public void
notifyRPMProtocolMgrClosed(com.aegis.jcommon.protocol.RPMProtocolMgr mgr
)
```

Used internally by the protocol manager to notify NRCBean if the NRC connection is dropped. Not for general use.

Specified by:

`notifyRPMProtocolMgrClosed` in interface `com.aegis.jcommon.protocol.RPMProtocolMgrOwner`

---

toString

```
public java.lang.String toString()
```

Returns the IP address (dot-notation) or DNS-resolvable host name of the connected NRC; or null if no connection.

Overrides:

`toString` in class `java.lang.Object`

Returns:

the IP address (dot-notation) or DNS-resolvable host name of the connected NRC

Class NRCException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--com.aegis.NRC.sdk.java.NRCException
```

All Implemented Interfaces:

java.io.Serializable

---

public class **NRCException**
extends java.lang.Exception

This class represents exceptions in NRC-based operations.

See Also:

Serialized Form

Constructor Summary

NRCException() Default constructor with error code = 0.
NRCException(int errorCode) Create an NRCException with the given error code
NRCException(int errorCode, java.lang.String message) Create an NRCException with the given error code and message

Method Summary

int	**getErrorCode**() Returns the error code associated with this exception.
java.lang.String	**getMessage**() Returns the error message associated with this exception.
static java.lang.String	**getMessage**(int errorCode) Returns a user viewable message based on a given RPMConstants error code.
java.lang.String	**toString**()

Methods inherited from class java.lang.Throwable
fillInStackTrace, getCause, getLocalizedMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

NRCException

```
public NRCException()
```

Default constructor with error code = 0.

NRCException

```
public NRCException(int errorCode)
```

Create an NRCException with the given error code

Parameters:

`errorCode` - the error code

NRCException

```
public NRCException(int errorCode,
                    java.lang.String message)
```

Create an NRCException with the given error code and message

Parameters:

`errorCode` - the error code

`message` - the error message

Method Detail

toString

```
public java.lang.String toString()
```

Overrides:

`toString` in class `java.lang.Throwable`

---

getMessage

```
public java.lang.String getMessage()
```

Returns the error message associated with this exception.

Overrides:

`getMessage` in class `java.lang.Throwable`

Returns:

the error message associated with this exception.

---

getErrorCode

`public int` **`getErrorCode`**`()`

Returns the error code associated with this exception.

Returns:

the error code associated with this exception.

---

getMessage

`public static java.lang.String` **`getMessage`**`(int errorCode)`

Returns a user viewable message based on a given RPMConstants error code. Acceptable error codes can be found in the RPMConstants class.

Parameters:

`errorCode` - the error code

Returns:

a user viewable message representing the error. If there is no message for the error, "Unknown error" is returned.

Interface SampleProcessor

public interface **SampleProcessor**

This interface is to be implemented in classes that want to process captured audio data. The derived class should be used in conjunction with a call to `ChannelBean.captureDataToProcessor()`.

Method Summary

`boolean`	**`processDone`**`()` Callback method is called when the sample capture stream has been closed
`boolean`	**`processSamples`**`(byte[] signal)` Method is called to process samples.

Method Detail

processSamples

`public boolean` **`processSamples`**`(byte[] signal)`

Method is called to process samples. Sample data is provided in bytes, so it is important to reconstruct multibyte samples before processing.

Parameters:

`signal` - byte samples array

Returns:

true on success, false otherwise

---

processDone

`public boolean` **`processDone`**`()`

Callback method is called when the sample capture stream has been closed

Returns:

true on success, false otherwise

Interface RPMConstants

All Known Implementing Classes:
ChannelDocumentHandler

---

public interface **RPMConstants**

This file includes various NRC constants.
Copyright (c) 2005 Aegis, Inc.

Field Summary

static int	**CMD_CHANNEL_COUNT_REQUEST** Request command: get channel count
static int	**CMD_CHANNEL_COUNT_RESPONSE** Response command: get channel count
static int	**CMD_CHANNEL_SUMMARY_REQUEST** Request command: get channel summary
static int	**CMD_CHANNEL_SUMMARY_RESPONSE** Response command: get channel summary
static int	**CMD_CLIENT_LOCATION_REQUEST** Request command: get channel client location
static int	**CMD_CLIENT_LOCATION_RESPONSE** Response command: get channel client location
static int	**CMD_CONNECT_REQUEST** Request command: connect to channel
static int	**CMD_CONNECT_RESPONSE** Response command: connect to channel
static int	**CMD_DATA_OPTIONS_REQUEST** Request command: get data options (sample rate, packet size)
static int	**CMD_DATA_OPTIONS_RESPONSE** Response command: get data options (sample rate, packet size)
static int	**CMD_DATA_PACKET** Data command: data packet
static int	**CMD_DETMODE_LIST_REQUEST** Request command: get valid detection modes
static int	**CMD_DETMODE_LIST_RESPONSE** Response command: get valid detection modes

static int	**CMD_DISCONNECT_REQUEST** Request command: disconnect from channel
static int	**CMD_DISCONNECT_RESPONSE** Response command: disconnect from channel
static int	**CMD_ERROR_RESPONSE** Error command
static int	**CMD_GET_ADC_SETTINGS_REQUEST** Request command: get A/D converter settings
static int	**CMD_GET_ADC_SETTINGS_RESPONSE** Response command: get A/D converter settings
static int	**CMD_GET_ANTENNA_NAME_REQUEST** Request command: get antenna name
static int	**CMD_GET_ANTENNA_NAME_RESPONSE** Response command: get antenna name
static int	**CMD_GET_ONLINE_POLL_INTERVAL_REQUEST** Request command: get receiver online poll interval
static int	**CMD_GET_ONLINE_POLL_INTERVAL_RESPONSE** Response command: get receiver online poll interval
static int	**CMD_GET_WRITABLE_REQUEST** Request command: is the channel/receiver writable by this device
static int	**CMD_GET_WRITABLE_RESPONSE** Response command: is the channel/receiver writable by this device
static int	**CMD_LOADMEM_REQUEST** Request command: Load receiver memory location
static int	**CMD_LOADMEM_RESPONSE** Response command: Load receiver memory location
static int	**CMD_MEMCAPACITY_REQUEST** Request command: Query receiver memory capacity
static int	**CMD_MEMCAPACITY_RESPONSE** Response command: Query receiver memory capacity
static int	**CMD_MEMFIELDS_REQUEST** Request command: Query receiver memory fields
static int	**CMD_MEMFIELDS_RESPONSE** Response command: Query receiver memory fields
static int	**CMD_MEMLOC_REQUEST** Request command: Query receiver memory location
static int	**CMD_MEMLOC_RESPONSE** Response command: Query receiver memory location
static int	**CMD_MODEL_NAME_REQUEST**

	Request command: get receiver model
static int	**CMD_MODEL_NAME_RESPONSE** Response command: get receiver model
static int	**CMD_MONITOR_REQUEST** Request command: read monitor information
static int	**CMD_MONITOR_RESPONSE** Response command: read monitor information
static int	**CMD_NOTIFY_WRITABLE** Notification command: client may now alter the channel/receiver configuration
static int	**CMD_NRC_CONFIGURATION_REQUEST** Request command: get receiver configuration
static int	**CMD_NRC_CONFIGURATION_RESPONSE** Response command: get NRC configuration
static int	**CMD_NRC_REBOOT_REQUEST** Request command: reboot NRC
static int	**CMD_PASSTHRU_RESPONSE_REQUEST** Request command: Send a passthrough command that expects a response from the receiver
static int	**CMD_PASSTHRU_RESPONSE_RESPONSE** Response command: Send a passthrough command that expects a response from the receiver
static int	**CMD_PING** Request command: ping
static int	**CMD_PONG** Response command: ping
static int	**CMD_READ_DATAOPTS_REQUEST** Request command: read data options for the current channel
static int	**CMD_READ_DATAOPTS_RESPONSE** Response command: read data options for the current channel
static int	**CMD_RECEIVER_BFO_REQUEST** Request command: set receiver BFO
static int	**CMD_RECEIVER_BFO_RESPONSE** Response command: set receiver BFO
static int	**CMD_RECEIVER_CONFIGURATION_REQUEST** **Deprecated.** *Use CMD_MONITOR_REQUEST instead*
static int	**CMD_RECEIVER_CONFIGURATION_RESPONSE** **Deprecated.** *Use CMD_MONITOR_REQUEST instead*
static int	**CMD_RECEIVER_DETMODE_REQUEST**

	Request command: set receiver detection mode
static int	**CMD_RECEIVER_DETMODE_RESPONSE** Response command: set receiver detection mode
static int	**CMD_RECEIVER_GETAGCMODE_REQUEST** Request command: get receiver agc mode
static int	**CMD_RECEIVER_GETAGCMODE_RESPONSE** Response command: get receiver agc mode
static int	**CMD_RECEIVER_GETBFO_REQUEST** Request command: get receiver BFO
static int	**CMD_RECEIVER_GETBFO_RESPONSE** Response command: get receiver BFO
static int	**CMD_RECEIVER_GETDETMODE_REQUEST** Request command: get receiver detection mode
static int	**CMD_RECEIVER_GETDETMODE_RESPONSE** Response command: get receiver detection mode
static int	**CMD_RECEIVER_GETFREQ_REQUEST** Request command: get receiver frequency
static int	**CMD_RECEIVER_GETFREQ_RESPONSE** Response command: get receiver frequency
static int	**CMD_RECEIVER_GETIFBANDWIDTH_REQUEST** Request command: get receiver IF bandwidth
static int	**CMD_RECEIVER_GETIFBANDWIDTH_RESPONSE** Response command: get receiver IF bandwidth
static int	**CMD_RECEIVER_GETSAMPLESSKIPPEDONTUNE_REQUEST** Request command: get number of samples skipped on tune
static int	**CMD_RECEIVER_GETSAMPLESSKIPPEDONTUNE_RESPONSE** Response command: get number of samples skipped on tune
static int	**CMD_RECEIVER_IFBANDWIDTH_REQUEST** Request command: set receiver IF bandwidth
static int	**CMD_RECEIVER_IFBANDWIDTH_RESPONSE** Response command: set receiver IF bandwidth
static int	**CMD_RECEIVER_MODEL_LIST_REQUEST** Request command: get valid receiver model list
static int	**CMD_RECEIVER_MODEL_LIST_RESPONSE** Response command: get valid receiver model list
static int	**CMD_RECEIVER_OFFLINE** Notification command: receiver is offline
static int	**CMD_RECEIVER_ONLINE** Notification command: receiver is online
static int	**CMD_RECEIVER_PASSTHRU_REQUEST**

	Request command: send receiver passthrough command
static int	**CMD_RECEIVER_PASSTHRU_RESPONSE** Response command: send receiver passthrough command
static int	**CMD_RECEIVER_REBOOT_REQUEST** Request command: reboot Receiver
static int	**CMD_RECEIVER_REBOOT_RESPONSE** Response command: reboot Receiver
static int	**CMD_RECEIVER_RESET_REQUEST** Request command: reset Receiver
static int	**CMD_RECEIVER_RESET_RESPONSE** Response command: reset Receiver
static int	**CMD_RECEIVER_SET_MODEL_REQUEST** Request command: set channel receiver model
static int	**CMD_RECEIVER_SET_MODEL_RESPONSE** Response command: set channel receiver model
static int	**CMD_RECEIVER_SETAGC_REQUEST** Request command: set AGC
static int	**CMD_RECEIVER_SETAGC_RESPONSE** Response command: set AGC
static int	**CMD_RECEIVER_SETMODE_REQUEST** Request command: set samples skipped on tune
static int	**CMD_RECEIVER_SETMODE_RESPONSE** Response command: set samples skipped on tune
static int	**CMD_RECEIVER_STATUS_REQUEST** Request command: receiver status
static int	**CMD_RECEIVER_STATUS_RESPONSE** Response command: receiver status
static int	**CMD_RECEIVER_TUNE_REQUEST** Request command: set receiver frequency
static int	**CMD_RECEIVER_TUNE_RESPONSE** Response command: set receiver frequency
static int	**CMD_SET_ADC_SETTINGS_REQUEST** Request command: set A/D converter settings
static int	**CMD_SET_ADC_SETTINGS_RESPONSE** Response command: set A/D converter settings
static int	**CMD_SET_ANTENNA_NAME_REQUEST** Request command: set antenna name
static int	**CMD_SET_ANTENNA_NAME_RESPONSE** Response command: set antenna name
static int	**CMD_SET_ONLINE_POLL_INTERVAL_REQUEST** Request command: set receiver online poll interval

static int	**CMD_SET_ONLINE_POLL_INTERVAL_RESPONSE** Response command: set receiver online poll interval
static int	**CMD_START_DATA_REQUEST** Request command: start data capture
static int	**CMD_STOP_DATA_REQUEST** Request command: stop data capture
static int	**CMD_STOP_DATA_RESPONSE** Response command: stop data capture
static int	**CMD_VERSION_REQUEST** Request command: Query NRC version information
static int	**CMD_VERSION_RESPONSE** Response command: Query NRC version information
static int	**CMD_WRITEMEM_REQUEST** Request command: Write receiver memory location
static int	**CMD_WRITEMEM_RESPONSE** Response command: Write receiver memory location
static int	**EALREADYCON** Error data: A receiver is already connected
static int	**EARGUMENT** Error data: The command argument is invalid
static int	**ENORESOURCE** Error data: No more receivers are available
static int	**ENOTCONNECTED** Error data: No receiver is connected
static int	**ERBADCMDTRANSLATION** Error data: The command could not be translated
static int	**ERCANNOTATTACHRECEIVER** Error data: Cannot attach the receiver
static int	**ERCANNOTREMOVERECEIVER** Error data: Cannot remove the receiver
static int	**ERCAPTURING** Error data: Error while capturing audio data
static int	**ERCHANALRDYCON** Error data: The channel requested was already connected to another socket
static int	**ERCHANSELFTESTPENDING** Error data: The channel requested has a self test pending
static int	**ERCHANSELFTESTRUNNING** Error data: The channel requested has a self-test running
static int	**ERINVALIDCHAN** Error data: An invalid channel value was used in the packet

	message
static int	**ERINVALIDCMD** Error data: An invalid command value was used in the packet message
static int	**ERINVALIDLEN** Error data: An invalid length value was used in the packet message
static int	**ERINVALIDSAMPLERATE** Error data: The requested sample rate is invalid
static int	**ERINVALIDSAMPLESPERPACKET** Error data: An invalid number of samples per packet was specified
static int	**ERINVALIDVGCFILTEROPTION** Error data: An invalid VGC Filter option for audio data
static int	**ERNODATAWRITTEN** Error data: Zero bytes of data written to the socket
static int	**ERNORECEIVERDEF** Error data: A receiver definition file could not be found
static int	**ERREADONLY** Error data: The requested command requires write access to the receiver.
static int	**ERSOCKALRDYCON** Error data: The socket already has a channel connected to it
static int	**ERSOCKNOTCON** Error data: The socket and a channel have not yet been connected
static int	**ERTIMEOUT** Error data: The command timed out
static int	**ERUNKNOWN** Error data: An unknown error occurred
static int	**ERWRITEERROR** Error data: An unspecified error has occurred
static int	**ESERIALOVLD** Error data: The serial port is overloaded
static int	**FLAG_ADC_USE_DIFFERENTIAL_MODE** Flag: Use differential input mode for A/D converter.
static int	**FLAG_ADC_USE_EXT_CLOCK** Flag: Use external clock signal for A/D converter.
static int	**SUCCESS** Error data: success (no error)

Field Detail

CMD_PING

```
public static final int CMD_PING
```

Request command: ping

See Also:

Constant Field Values

CMD_PONG

```
public static final int CMD_PONG
```

Response command: ping

See Also:

Constant Field Values

CMD_RECEIVER_STATUS_REQUEST

```
public static final int CMD_RECEIVER_STATUS_REQUEST
```

Request command: receiver status

See Also:

Constant Field Values

CMD_RECEIVER_STATUS_RESPONSE

```
public static final int CMD_RECEIVER_STATUS_RESPONSE
```

Response command: receiver status

See Also:

Constant Field Values

CMD_CONNECT_REQUEST

```
public static final int CMD_CONNECT_REQUEST
```

Request command: connect to channel

See Also:

Constant Field Values

CMD_CONNECT_RESPONSE

```
public static final int CMD_CONNECT_RESPONSE
```

Response command: connect to channel

See Also:

Constant Field Values

CMD_DISCONNECT_REQUEST

`public static final int` **`CMD_DISCONNECT_REQUEST`**

Request command: disconnect from channel

See Also:

Constant Field Values

CMD_DISCONNECT_RESPONSE

`public static final int` **`CMD_DISCONNECT_RESPONSE`**

Response command: disconnect from channel

See Also:

Constant Field Values

CMD_START_DATA_REQUEST

`public static final int` **`CMD_START_DATA_REQUEST`**

Request command: start data capture

See Also:

Constant Field Values

CMD_DATA_PACKET

`public static final int` **`CMD_DATA_PACKET`**

Data command: data packet

See Also:

Constant Field Values

CMD_STOP_DATA_REQUEST

`public static final int` **`CMD_STOP_DATA_REQUEST`**

Request command: stop data capture

See Also:

Constant Field Values

CMD_STOP_DATA_RESPONSE

`public static final int` **`CMD_STOP_DATA_RESPONSE`**

Response command: stop data capture

See Also:

Constant Field Values

CMD_RECEIVER_PASSTHRU_REQUEST

`public static final int` **`CMD_RECEIVER_PASSTHRU_REQUEST`**

Request command: send receiver passthrough command

See Also:

Constant Field Values

CMD_RECEIVER_PASSTHRU_RESPONSE

`public static final int` **`CMD_RECEIVER_PASSTHRU_RESPONSE`**

Response command: send receiver passthrough command

See Also:

Constant Field Values

CMD_RECEIVER_SETMODE_REQUEST

`public static final int` **`CMD_RECEIVER_SETMODE_REQUEST`**

Request command: set samples skipped on tune

See Also:

Constant Field Values

CMD_RECEIVER_SETMODE_RESPONSE

`public static final int` **`CMD_RECEIVER_SETMODE_RESPONSE`**

Response command: set samples skipped on tune

See Also:

Constant Field Values

CMD_RECEIVER_TUNE_REQUEST

`public static final int` **`CMD_RECEIVER_TUNE_REQUEST`**

Request command: set receiver frequency

See Also:

Constant Field Values

CMD_RECEIVER_TUNE_RESPONSE

`public static final int` **`CMD_RECEIVER_TUNE_RESPONSE`**

Response command: set receiver frequency

See Also:

Constant Field Values

CMD_RECEIVER_DETMODE_REQUEST

`public static final int` **`CMD_RECEIVER_DETMODE_REQUEST`**

Request command: set receiver detection mode

See Also:

Constant Field Values

CMD_RECEIVER_DETMODE_RESPONSE

`public static final int` **`CMD_RECEIVER_DETMODE_RESPONSE`**

Response command: set receiver detection mode

See Also:

Constant Field Values

CMD_RECEIVER_IFBANDWIDTH_REQUEST

`public static final int` **`CMD_RECEIVER_IFBANDWIDTH_REQUEST`**

Request command: set receiver IF bandwidth

See Also:

Constant Field Values

CMD_RECEIVER_IFBANDWIDTH_RESPONSE

`public static final int` **`CMD_RECEIVER_IFBANDWIDTH_RESPONSE`**

Response command: set receiver IF bandwidth

See Also:

Constant Field Values

CMD_RECEIVER_BFO_REQUEST

`public static final int` **`CMD_RECEIVER_BFO_REQUEST`**

Request command: set receiver BFO

See Also:

Constant Field Values

CMD_RECEIVER_BFO_RESPONSE

`public static final int` **`CMD_RECEIVER_BFO_RESPONSE`**

Response command: set receiver BFO

See Also:

Constant Field Values

CMD_NRC_CONFIGURATION_REQUEST

`public static final int` **`CMD_NRC_CONFIGURATION_REQUEST`**

Request command: get receiver configuration

See Also:

Constant Field Values

CMD_NRC_CONFIGURATION_RESPONSE

`public static final int` **`CMD_NRC_CONFIGURATION_RESPONSE`**

Response command: get NRC configuration

See Also:

Constant Field Values

CMD_RECEIVER_CONFIGURATION_REQUEST

`public static final int` **`CMD_RECEIVER_CONFIGURATION_REQUEST`**

Deprecated. *Use CMD_MONITOR_REQUEST instead*

Request command: get Receiver configuration

See Also:

Constant Field Values

CMD_RECEIVER_CONFIGURATION_RESPONSE

`public static final int` **`CMD_RECEIVER_CONFIGURATION_RESPONSE`**

Deprecated. *Use CMD_MONITOR_REQUEST instead*

Response command: get receiver configuration

See Also:

Constant Field Values

CMD_RECEIVER_RESET_REQUEST

`public static final int` **`CMD_RECEIVER_RESET_REQUEST`**

Request command: reset Receiver

See Also:

Constant Field Values

CMD_RECEIVER_RESET_RESPONSE

`public static final int` **`CMD_RECEIVER_RESET_RESPONSE`**

Response command: reset Receiver

See Also:

Constant Field Values

CMD_RECEIVER_SETAGC_REQUEST

`public static final int` **`CMD_RECEIVER_SETAGC_REQUEST`**

Request command: set AGC

See Also:

Constant Field Values

CMD_RECEIVER_SETAGC_RESPONSE

`public static final int` **`CMD_RECEIVER_SETAGC_RESPONSE`**

Response command: set AGC

See Also:

Constant Field Values

CMD_RECEIVER_REBOOT_REQUEST

`public static final int` **`CMD_RECEIVER_REBOOT_REQUEST`**

Request command: reboot Receiver
See Also:
Constant Field Values

CMD_RECEIVER_REBOOT_RESPONSE

`public static final int` **`CMD_RECEIVER_REBOOT_RESPONSE`**

Response command: reboot Receiver
See Also:
Constant Field Values

CMD_NRC_REBOOT_REQUEST

`public static final int` **`CMD_NRC_REBOOT_REQUEST`**

Request command: reboot NRC
See Also:
Constant Field Values

CMD_ERROR_RESPONSE

`public static final int` **`CMD_ERROR_RESPONSE`**

Error command
See Also:
Constant Field Values

CMD_RECEIVER_OFFLINE

`public static final int` **`CMD_RECEIVER_OFFLINE`**

Notification command: receiver is offline
See Also:
Constant Field Values

CMD_RECEIVER_ONLINE

`public static final int` **`CMD_RECEIVER_ONLINE`**

Notification command: receiver is online
See Also:
Constant Field Values

CMD_CHANNEL_COUNT_REQUEST

`public static final int` **`CMD_CHANNEL_COUNT_REQUEST`**

Request command: get channel count
See Also:
Constant Field Values

CMD_CHANNEL_COUNT_RESPONSE

`public static final int` **`CMD_CHANNEL_COUNT_RESPONSE`**

Response command: get channel count

See Also:

Constant Field Values

CMD_DATA_OPTIONS_REQUEST

`public static final int` **`CMD_DATA_OPTIONS_REQUEST`**

Request command: get data options (sample rate, packet size)

See Also:

Constant Field Values

CMD_DATA_OPTIONS_RESPONSE

`public static final int` **`CMD_DATA_OPTIONS_RESPONSE`**

Response command: get data options (sample rate, packet size)

See Also:

Constant Field Values

CMD_MODEL_NAME_REQUEST

`public static final int` **`CMD_MODEL_NAME_REQUEST`**

Request command: get receiver model

See Also:

Constant Field Values

CMD_MODEL_NAME_RESPONSE

`public static final int` **`CMD_MODEL_NAME_RESPONSE`**

Response command: get receiver model

See Also:

Constant Field Values

CMD_CLIENT_LOCATION_REQUEST

`public static final int` **`CMD_CLIENT_LOCATION_REQUEST`**

Request command: get channel client location

See Also:

Constant Field Values

CMD_CLIENT_LOCATION_RESPONSE

`public static final int` **`CMD_CLIENT_LOCATION_RESPONSE`**

Response command: get channel client location

See Also:

Constant Field Values

CMD_DETMODE_LIST_REQUEST

`public static final int` **`CMD_DETMODE_LIST_REQUEST`**

Request command: get valid detection modes

See Also:

Constant Field Values

CMD_DETMODE_LIST_RESPONSE

`public static final int` **`CMD_DETMODE_LIST_RESPONSE`**

Response command: get valid detection modes

See Also:

Constant Field Values

CMD_RECEIVER_SET_MODEL_REQUEST

`public static final int` **`CMD_RECEIVER_SET_MODEL_REQUEST`**

Request command: set channel receiver model

See Also:

Constant Field Values

CMD_RECEIVER_SET_MODEL_RESPONSE

`public static final int` **`CMD_RECEIVER_SET_MODEL_RESPONSE`**

Response command: set channel receiver model

See Also:

Constant Field Values

CMD_RECEIVER_MODEL_LIST_REQUEST

`public static final int` **`CMD_RECEIVER_MODEL_LIST_REQUEST`**

Request command: get valid receiver model list

See Also:

Constant Field Values

CMD_RECEIVER_MODEL_LIST_RESPONSE

`public static final int` **`CMD_RECEIVER_MODEL_LIST_RESPONSE`**

Response command: get valid receiver model list

See Also:

Constant Field Values

CMD_CHANNEL_SUMMARY_REQUEST

`public static final int` **`CMD_CHANNEL_SUMMARY_REQUEST`**

Request command: get channel summary
See Also:
Constant Field Values

CMD_CHANNEL_SUMMARY_RESPONSE

`public static final int` **`CMD_CHANNEL_SUMMARY_RESPONSE`**

Response command: get channel summary
See Also:
Constant Field Values

CMD_SET_ADC_SETTINGS_REQUEST

`public static final int` **`CMD_SET_ADC_SETTINGS_REQUEST`**

Request command: set A/D converter settings
See Also:
Constant Field Values

CMD_SET_ADC_SETTINGS_RESPONSE

`public static final int` **`CMD_SET_ADC_SETTINGS_RESPONSE`**

Response command: set A/D converter settings
See Also:
Constant Field Values

CMD_GET_ADC_SETTINGS_REQUEST

`public static final int` **`CMD_GET_ADC_SETTINGS_REQUEST`**

Request command: get A/D converter settings
See Also:
Constant Field Values

CMD_GET_ADC_SETTINGS_RESPONSE

`public static final int` **`CMD_GET_ADC_SETTINGS_RESPONSE`**

Response command: get A/D converter settings
See Also:
Constant Field Values

CMD_RECEIVER_GETSAMPLESSKIPPEDONTUNE_REQUEST

`public static final int` **`CMD_RECEIVER_GETSAMPLESSKIPPEDONTUNE_REQUEST`**

Request command: get number of samples skipped on tune
See Also:
Constant Field Values

CMD_RECEIVER_GETSAMPLESSKIPPEDONTUNE_RESPONSE

`public static final int` **`CMD_RECEIVER_GETSAMPLESSKIPPEDONTUNE_RESPONSE`**

Response command: get number of samples skipped on tune

See Also:

Constant Field Values

CMD_RECEIVER_GETFREQ_REQUEST

`public static final int` **`CMD_RECEIVER_GETFREQ_REQUEST`**

Request command: get receiver frequency

See Also:

Constant Field Values

CMD_RECEIVER_GETFREQ_RESPONSE

`public static final int` **`CMD_RECEIVER_GETFREQ_RESPONSE`**

Response command: get receiver frequency

See Also:

Constant Field Values

CMD_RECEIVER_GETDETMODE_REQUEST

`public static final int` **`CMD_RECEIVER_GETDETMODE_REQUEST`**

Request command: get receiver detection mode

See Also:

Constant Field Values

CMD_RECEIVER_GETDETMODE_RESPONSE

`public static final int` **`CMD_RECEIVER_GETDETMODE_RESPONSE`**

Response command: get receiver detection mode

See Also:

Constant Field Values

CMD_RECEIVER_GETIFBANDWIDTH_REQUEST

`public static final int` **`CMD_RECEIVER_GETIFBANDWIDTH_REQUEST`**

Request command: get receiver IF bandwidth

See Also:

Constant Field Values

CMD_RECEIVER_GETIFBANDWIDTH_RESPONSE

`public static final int` **`CMD_RECEIVER_GETIFBANDWIDTH_RESPONSE`**

Response command: get receiver IF bandwidth

See Also:

Constant Field Values

CMD_RECEIVER_GETBFO_REQUEST

`public static final int` **`CMD_RECEIVER_GETBFO_REQUEST`**

Request command: get receiver BFO

See Also:

Constant Field Values

CMD_RECEIVER_GETBFO_RESPONSE

`public static final int` **`CMD_RECEIVER_GETBFO_RESPONSE`**

Response command: get receiver BFO

See Also:

Constant Field Values

CMD_SET_ONLINE_POLL_INTERVAL_REQUEST

`public static final int` **`CMD_SET_ONLINE_POLL_INTERVAL_REQUEST`**

Request command: set receiver online poll interval

See Also:

Constant Field Values

CMD_SET_ONLINE_POLL_INTERVAL_RESPONSE

`public static final int` **`CMD_SET_ONLINE_POLL_INTERVAL_RESPONSE`**

Response command: set receiver online poll interval

See Also:

Constant Field Values

CMD_GET_ONLINE_POLL_INTERVAL_REQUEST

`public static final int` **`CMD_GET_ONLINE_POLL_INTERVAL_REQUEST`**

Request command: get receiver online poll interval

See Also:

Constant Field Values

CMD_GET_ONLINE_POLL_INTERVAL_RESPONSE

`public static final int` **`CMD_GET_ONLINE_POLL_INTERVAL_RESPONSE`**

Response command: get receiver online poll interval

See Also:

Constant Field Values

CMD_GET_ANTENNA_NAME_REQUEST

`public static final int` **`CMD_GET_ANTENNA_NAME_REQUEST`**

Request command: get antenna name
See Also:
Constant Field Values

CMD_GET_ANTENNA_NAME_RESPONSE

`public static final int` **`CMD_GET_ANTENNA_NAME_RESPONSE`**

Response command: get antenna name
See Also:
Constant Field Values

CMD_SET_ANTENNA_NAME_REQUEST

`public static final int` **`CMD_SET_ANTENNA_NAME_REQUEST`**

Request command: set antenna name
See Also:
Constant Field Values

CMD_SET_ANTENNA_NAME_RESPONSE

`public static final int` **`CMD_SET_ANTENNA_NAME_RESPONSE`**

Response command: set antenna name
See Also:
Constant Field Values

CMD_RECEIVER_GETAGCMODE_REQUEST

`public static final int` **`CMD_RECEIVER_GETAGCMODE_REQUEST`**

Request command: get receiver agc mode
See Also:
Constant Field Values

CMD_RECEIVER_GETAGCMODE_RESPONSE

`public static final int` **`CMD_RECEIVER_GETAGCMODE_RESPONSE`**

Response command: get receiver agc mode
See Also:
Constant Field Values

CMD_NOTIFY_WRITABLE

`public static final int` **`CMD_NOTIFY_WRITABLE`**

Notification command: client may now alter the channel/receiver configuration
See Also:
Constant Field Values

CMD_READ_DATAOPTS_REQUEST

`public static final int` **`CMD_READ_DATAOPTS_REQUEST`**

Request command: read data options for the current channel

See Also:

Constant Field Values

CMD_READ_DATAOPTS_RESPONSE

`public static final int` **`CMD_READ_DATAOPTS_RESPONSE`**

Response command: read data options for the current channel

See Also:

Constant Field Values

CMD_GET_WRITABLE_REQUEST

`public static final int` **`CMD_GET_WRITABLE_REQUEST`**

Request command: is the channel/receiver writable by this device

See Also:

Constant Field Values

CMD_GET_WRITABLE_RESPONSE

`public static final int` **`CMD_GET_WRITABLE_RESPONSE`**

Response command: is the channel/receiver writable by this device

See Also:

Constant Field Values

CMD_MONITOR_REQUEST

`public static final int` **`CMD_MONITOR_REQUEST`**

Request command: read monitor information

See Also:

Constant Field Values

CMD_MONITOR_RESPONSE

`public static final int` **`CMD_MONITOR_RESPONSE`**

Response command: read monitor information

See Also:

Constant Field Values

CMD_PASSTHRU_RESPONSE_REQUEST

`public static final int` **`CMD_PASSTHRU_RESPONSE_REQUEST`**

Request command: Send a passthrough command that expects a response from the receiver

See Also:
Constant Field Values

CMD_PASSTHRU_RESPONSE_RESPONSE

`public static final int` **`CMD_PASSTHRU_RESPONSE_RESPONSE`**

Response command: Send a passthrough command that expects a response from the receiver

See Also:
Constant Field Values

CMD_WRITEMEM_REQUEST

`public static final int` **`CMD_WRITEMEM_REQUEST`**

Request command: Write receiver memory location

See Also:
Constant Field Values

CMD_WRITEMEM_RESPONSE

`public static final int` **`CMD_WRITEMEM_RESPONSE`**

Response command: Write receiver memory location

See Also:
Constant Field Values

CMD_LOADMEM_REQUEST

`public static final int` **`CMD_LOADMEM_REQUEST`**

Request command: Load receiver memory location

See Also:
Constant Field Values

CMD_LOADMEM_RESPONSE

`public static final int` **`CMD_LOADMEM_RESPONSE`**

Response command: Load receiver memory location

See Also:
Constant Field Values

CMD_MEMLOC_REQUEST

`public static final int` **`CMD_MEMLOC_REQUEST`**

Request command: Query receiver memory location

See Also:
Constant Field Values

CMD_MEMLOC_RESPONSE

`public static final int` **`CMD_MEMLOC_RESPONSE`**

Response command: Query receiver memory location

See Also:

Constant Field Values

CMD_MEMCAPACITY_REQUEST

`public static final int` **`CMD_MEMCAPACITY_REQUEST`**

Request command: Query receiver memory capacity

See Also:

Constant Field Values

CMD_MEMCAPACITY_RESPONSE

`public static final int` **`CMD_MEMCAPACITY_RESPONSE`**

Response command: Query receiver memory capacity

See Also:

Constant Field Values

CMD_MEMFIELDS_REQUEST

`public static final int` **`CMD_MEMFIELDS_REQUEST`**

Request command: Query receiver memory fields

See Also:

Constant Field Values

CMD_MEMFIELDS_RESPONSE

`public static final int` **`CMD_MEMFIELDS_RESPONSE`**

Response command: Query receiver memory fields

See Also:

Constant Field Values

CMD_VERSION_REQUEST

`public static final int` **`CMD_VERSION_REQUEST`**

Request command: Query NRC version information

See Also:

Constant Field Values

CMD_VERSION_RESPONSE

`public static final int` **`CMD_VERSION_RESPONSE`**

Response command: Query NRC version information

See Also:

Constant Field Values

SUCCESS

`public static final int` **`SUCCESS`**

Error data: success (no error)

See Also:

Constant Field Values

ERNODATAWRITTEN

`public static final int` **`ERNODATAWRITTEN`**

Error data: Zero bytes of data written to the socket

See Also:

Constant Field Values

ERWRITEERROR

`public static final int` **`ERWRITEERROR`**

Error data: An unspecified error has occurred

See Also:

Constant Field Values

ERINVALIDCMD

`public static final int` **`ERINVALIDCMD`**

Error data: An invalid command value was used in the packet message

See Also:

Constant Field Values

ERINVALIDLEN

`public static final int` **`ERINVALIDLEN`**

Error data: An invalid length value was used in the packet message

See Also:

Constant Field Values

ERINVALIDCHAN

`public static final int` **`ERINVALIDCHAN`**

Error data: An invalid channel value was used in the packet message

See Also:

Constant Field Values

ERCHANALRDYCON

`public static final int` **`ERCHANALRDYCON`**

Error data: The channel requested was already connected to another socket

See Also:

Constant Field Values

ERSOCKALRDYCON

`public static final int` **`ERSOCKALRDYCON`**

Error data: The socket already has a channel connected to it

See Also:

Constant Field Values

ERSOCKNOTCON

`public static final int` **`ERSOCKNOTCON`**

Error data: The socket and a channel have not yet been connected

See Also:

Constant Field Values

ENORESOURCE

`public static final int` **`ENORESOURCE`**

Error data: No more receivers are available

See Also:

Constant Field Values

ENOTCONNECTED

`public static final int` **`ENOTCONNECTED`**

Error data: No receiver is connected

See Also:

Constant Field Values

EALREADYCON

`public static final int` **`EALREADYCON`**

Error data: A receiver is already connected

See Also:

Constant Field Values

EARGUMENT

`public static final int` **`EARGUMENT`**

Error data: The command argument is invalid

See Also:

Constant Field Values

ESERIALOVLD

`public static final int` **`ESERIALOVLD`**

Error data: The serial port is overloaded

See Also:

Constant Field Values

ERCHANSELFTESTPENDING

`public static final int` **`ERCHANSELFTESTPENDING`**

Error data: The channel requested has a self test pending

See Also:

Constant Field Values

ERCHANSELFTESTRUNNING

`public static final int` **`ERCHANSELFTESTRUNNING`**

Error data: The channel requested has a self-test running

See Also:

Constant Field Values

ERTIMEOUT

`public static final int` **`ERTIMEOUT`**

Error data: The command timed out

See Also:

Constant Field Values

ERUNKNOWN

`public static final int` **`ERUNKNOWN`**

Error data: An unknown error occurred

See Also:

Constant Field Values

ERINVALIDSAMPLERATE

`public static final int` **`ERINVALIDSAMPLERATE`**

Error data: The requested sample rate is invalid

See Also:

Constant Field Values

ERINVALIDSAMPLESPERPACKET

`public static final int` **`ERINVALIDSAMPLESPERPACKET`**

Error data: An invalid number of samples per packet was specified

See Also:

Constant Field Values

ERNORECEIVERDEF

`public static final int` **`ERNORECEIVERDEF`**

Error data: A receiver definition file could not be found

See Also:

Constant Field Values

ERBADCMDTRANSLATION

`public static final int` **`ERBADCMDTRANSLATION`**

Error data: The command could not be translated

See Also:

Constant Field Values

ERCANNOTREMOVERECEIVER

`public static final int` **`ERCANNOTREMOVERECEIVER`**

Error data: Cannot remove the receiver

See Also:

Constant Field Values

ERCANNOTATTACHRECEIVER

`public static final int` **`ERCANNOTATTACHRECEIVER`**

Error data: Cannot attach the receiver

See Also:

Constant Field Values

ERCAPTURING

`public static final int` **`ERCAPTURING`**

Error data: Error while capturing audio data

See Also:

Constant Field Values

ERINVALIDVGCFILTEROPTION

`public static final int` **`ERINVALIDVGCFILTEROPTION`**

Error data: An invalid VGC Filter option for audio data

See Also:

Constant Field Values

ERREADONLY

`public static final int` **`ERREADONLY`**

Error data: The requested command requires write access to the receiver.

See Also:

Constant Field Values

FLAG_ADC_USE_EXT_CLOCK

`public static final int` **`FLAG_ADC_USE_EXT_CLOCK`**

Flag: Use external clock signal for A/D converter. Omit this flag to specify internal clocking.

See Also:

Constant Field Values

FLAG_ADC_USE_DIFFERENTIAL_MODE

`public static final int` **`FLAG_ADC_USE_DIFFERENTIAL_MODE`**

Flag: Use differential input mode for A/D converter. Omit this flag to specify single-ended input mode.

See Also:

Constant Field Values

Appendix A- Interface Control Document

RPM Protocol

Aside from using the NRC Client SDK to communicate with the NRC, a client is left to communicate with the NRC at its lowest level, the RPM protocol. This protocol is initiated by connecting to the NRC server via TCP socket at port 30005. Once connected, the client can send commands to the NRC in the form request packets. Most commands issued to the NRC invoke a response whether it is a successful command completion response or an error response. The following describes the structure of an RPM packet:

Flag	Command		Tag		Length		Data
7E	C	C	T	T	L	L	…

(**Note**: numeric values contained in tables in this appendix are in hexadecimal notation.)

Each cell in the table represents a single byte. Multi-byte words are always formatted in network byte order (big endian). For example, commands are two-byte integers. The command 0x0401 is represented by 0x04 in the first byte and 0x01 in the second byte. Each field is described here in detail.

- `Flag`: each RPM packet must begin with the two-byte integer 0x7E.
- `Command`: this two-byte integer uniquely identifies the character of the packet
- `Tag`: this two-byte integer can be used to match a response packet with its corresponding request. The NRC server copies this field from the request into the tag field of its response.
- `Length`: this two-byte integer specifies the size of the following Data field.
- `Data`: this variable size field contains any data relevant to the specific packet.

When a command fails inside the NRC for any reason, an error packet is returned instead of the expected response packet. Error packets follow the standard structure, and are identified by the command value 0x07FF. If the Length field is 0x0002, the Data field contains a two-byte error code that can be decoded to determine the nature of the error. If the Length field contains another size field, the Data field contains an ASCII-encoded string error message. The following table lists the types of RPM packets:

	Flag	Command		Tag		Length		Data	
Request	7E	N	N	T	T	L	L	…	
Response	7E	N	N	T	T	L	L	…	
Error Response	7E	07	FF	T	T	00	02	N	N
Error Response	7E	07	FF	T	T	L	L	… (ASCII)	

Commands

Ping

The Ping request causes the NRC to echo back with a response.

	Flag	Command		Tag		Length		Data
Request	7E	06	01	T	T	00	00	
Response	7E	07	01	T	T	00	00	

Channel Status

When the NRC receives a channel status request, it queries the receiver attached to the channel. The receiver responds with an internal ASCII-encoded status string, which is passed to the client.
This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	06	02	T	T	00	00	
Response	7E	07	02	T	T	L	L	<Channel status>

Connect

The connect command causes the client to be assigned to a channel. Note that the client can only be connected to one channel at any time.
This command has two forms. In the first form, the Length field is zero, and the NRC attaches the client to the next available free receiver. If no free receiver is available, the NRC attaches the client in piggyback mode to the channel with the least number of clients already piggybacking. In the second form, the Length field is two, and the client specifies in the Data field which channel it wishes to be attached to in the form of a two-byte integer. In this case, the NRC always attaches the client to the specified channel. To determine if the client is attached in piggyback mode, see the "Channel Writable" notification and the "Get Writable" packet. For both forms of the connect packet, the NRC responds with a data field that contains the channel the client has been assigned.
Note: The client will be able to connect to receivers up to the maximum number of system receivers – 8, even if these receivers aren't correctly configured. To this end it is important to check the status of the receiver once a client connects to ascertain its current status.

	Flag	Command		Tag		Length		Data	
Request 1	7E	06	03	T	T	00	00		
Request 2	7E	06	03	T	T	00	02	N	N
Response	7E	07	03	T	T	00	02	N	N

Disconnect

The disconnect command causes the client to be disconnected from its assigned channel.
This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	06	04	T	T	00	00	
Response	7E	07	04	T	T	00	00	

Read Data Options

This command returns the data options which are currently set for this client. If the request is made before the client has set it's data options then the client will receive the default data options (16 KHz, 512 spp, Filter on, Timestamp off); if the client is piggybacking and the request is made before the client has set it's data options then the client will receive the data options set for the client which has write access.
The first two bytes indicate the sample rate, the second two bytes indicate the number of samples per packet, the third two bytes represent the filter option, and the last byte represents the time stamp option. See the description of the Set data options packet for more information about the values for each of these fields.
This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data						
Request	7E	A0	03	T	T	00	00							
Response	7E	A1	03	T	T	00	07	N	N	N	N	N	N	N

Set data options

This command allows the client to specify the sampling rate and packet size of data capture on the connected channel. This command can only be executed if data is not being captured, therefore, this command should be sent prior to the Start data command.
The request data is 6 or 7 bytes;
The first two bytes are the sample rate (unsigned two-byte integer). Valid sample rate values include 0x0001 (16 KHz sample rate) and 0x0002 (8 KHz sample rate). The default sample rate is 16 KHz.
The second set of two bytes is the packet size (unsigned two-byte integer), which is the number of samples to be sent in each Sampled Data Packet. The range of valid packet sizes is 256 - 4096. The default packet size is 640.
The third set of two bytes is the channel filter option (unsigned two-byte integer), which selects whether there is any channel pre-filtering of not. The valid values are 0 – FILTERING_OFF or 1 - LPF_4KHZ_ON. The default filter option is LPF_4KHZ_ON.
The client may choose to send the second version of this packet, in which case the seventh byte selects whether or not each data packet is prefixed with a 12 byte TAI64N timestamp representing the time that the last sample in the packet was captured. The valid values are 0 – TIMESTAMP_OFF or 1 – TIMESTAMP_ON. If the first version of the packet is sent then the time stamp option defaults to TIMESTAMP_OFF (0).

	Flag	Command		Tag		Length		Data						
Request 1	7E	A0	04	T	T	00	06	N	N	N	N	N	N	
Request 2	7E	A0	04	T	T	00	07	N	N	N	N	N	N	N
Response	7E	A1	04	T	T	00	00							

Start data

The start data command causes the NRC to start streaming digitally sampled data from the receiver attached to the connected channel to the client. There is no response to this command. Instead, Sampled Data Packets will begin to arrive at the client.
This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	06	05	T	T	00	00	

Sampled Data Packet

This packet is sent by the NRC server to deliver sampled audio data to the client. These packets will be delivered after successfully receiving a Start data request and will continue until receiving a Stop data request. The Length field will be the twice the packet size specified in a Set data options request or the default, if not specified. Each sample is a two-byte integer in network byte order (big endian).
If the time stamp option is set to TIMESTAMP_ON (1) in the Set Data Options request, then the first 12 bytes of this packet will be a TAI64N time stamp representing the time that the last sample in the packet was captured. The length of the packet will be twice the packet size specified in the Set data options request plus 12 bytes to hold the time stamp.
This packet will only be sent if the client is connected to a channel (see Connect).
The tag on each sampled data packet matches the tag given in the Start data request.

	Flag	Command		Tag		Length		Data
Response	7E	07	05	T	T	LL	LL	…

Stop Data

This command causes the NRC to stop streaming data to the client.
This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	06	06	T	T	00	00	
Response	7E	07	06	T	T	00	00	

Send receiver pass-through command

The receiver pass-through command sends a native command to the receiver attached to the connected channel. The receiver command must be an ASCII-encoded string. Please refer to the receiver operating manual for the proper command set. The first version of the packet should be sent when a response from the receiver is not required or when the command executes provides no response. The second version of the packet should only be sent if the receiver is expected to provide a response to the command. The NRC does not perform any parsing, processing, or validation of pass-through commands so it is up to the sender of the packet to ensure the proper packet is chosen and the command is properly formatted.
If the first version of the packet is sent, a response from the NRC signals that the

command was successfully dispatched to the receiver. Any response sent from the receiver is not forwarded to the client.
If the second version of the packet is sent, a response from the NRC will not be sent until the receiver responds to the command. The response is sent back to the client in the data field of the packet without any modification. If the receiver does not respond within a time period (specified in the RCPConfig.xml file) then the receiver will go offline, discard any pending requests, and then come back online.
This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	06	07	T	T	L	L	*receiver command*
Response	7E	07	07	T	T	00	00	

	Flag	Command		Tag		Length		Data
Request	7E	A6	14	T	T	L	L	*receiver command*
Response	7E	A7	14	T	T	L	L	*Receiver response*

Set drop samples

This command is used to set the number of samples the NRC should drop when the receiver attached to the channel is tuned. The request Data field is an unsigned two-byte integer. The default setting is 300 samples.

	Flag	Command		Tag		Length		Data	
Request	7E	06	08	T	T	00	02	N	N
Response	7E	07	08	T	T	00	00		

Get drop samples

This command is used to get the number of samples the NRC will drop when the receiver attached to the channel is tuned. The response Data field is an unsigned two-byte integer. The default setting, active without taking any action, is 300 samples.

	Flag	Command		Tag		Length		Data	
Request	7E	A6	08	T	T	00	00		
Response	7E	A7	08	T	T	00	02	N	N

Set receiver frequency

This command is used to tune the receiver attached to the connected channel. The request Data field should contain an ASCII-encoded string of the desired frequency in MHz. The valid range is dependent upon the capability of the receiver.
When the receiver is tuned, the current data buffer is flushed and incoming data samples are dropped until the number of samples specified by the channel have been skipped (see Drop samples). Data capture is then resumed.
This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	06	09	T	T	L	L	*ASCII frequency*
Response	7E	07	09	T	T	00	00	

Get receiver frequency

This command is used to get the frequency of the receiver attached to the connected channel. The response Data field should contain an ASCII-encoded string of the desired frequency in MHz.
This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	A6	09	T	T	00	00	
Response	7E	A7	09	T	T	L	L	*ASCII frequency*

Set Detection Mode

Sets the detection mode of the receiver attached to the connected channel. The request Data field should contain an ASCII-encoded string of the desired detection mode. The set of valid values is dependent upon the capability of the receiver.
When the detection mode of the receiver is changed, the current data buffer is flushed and incoming data samples are dropped until the number of samples specified by the channel have been skipped (see Drop samples). Data capture is then resumed.
This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	06	0A	T	T	L	L	*ASCII detection mode*
Response	7E	07	0A	T	T	00	00	

Get Detection Mode

Gets the detection mode of the receiver attached to the connected channel. The response Data field will contain an ASCII-encoded string of the desired detection mode.
This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	A6	0A	T	T	00	00	
Response	7E	A7	0A	T	T	L	L	*ASCII detection mode*

Set IF Bandwidth

Sets the IF bandwidth of the receiver attached to the connected channel. The request Data field should contain an ASCII-encoded string of the desired IF bandwidth in Hz. The range of values is dependent upon the capability of the receiver.
When the IF bandwidth of the receiver is changed, the current data buffer is flushed and incoming data samples are dropped until the number specified by the channel have been

skipped (see Drop samples). Data capture is then resumed.
This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	06	0B	T	T	L	L	*ASCII IF bandwidth*
Response	7E	07	0B	T	T	00	00	

Get IF Bandwidth

Gets the IF bandwidth of the receiver attached to the connected channel. The response Data field should contain an ASCII-encoded string of the desired IF bandwidth in Hz.
This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	A6	0B	T	T	00	00	
Response	7E	A7	0B	T	T	L	L	*ASCII IF bandwidth*

Set BFO

Sets the BFO of the receiver attached to the connected channel. The request Data field should contain an ASCII-encoded string of the desired BFO in Hz. The range of values is dependent upon the capability of the receiver.
When the BFO of the receiver is changed, the current data buffer is flushed and incoming data samples are dropped until the number of samples specified by the channel have been skipped (see Drop samples). Data capture is then resumed.
This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	06	0C	T	T	L	L	*ASCII BFO*
Response	7E	07	0C	T	T	00	00	

Get BFO

Gets the BFO of the receiver attached to the connected channel. The response Data field will contain an ASCII-encoded string of the desired BFO in Hz.
This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	A6	0C	T	T	00	00	
Response	7E	A7	0C	T	T	L	L	*ASCII BFO*

Get channel count

Gets the number of channels contained in the NRC server. The response packet contains an unsigned two-byte integer that represents this number.

	Flag	Command		Tag		Length		Data	
Request	7E	A0	01	T	T	00	00		
Response	7E	A1	01	T	T	00	02	N	N

Get model name

Gets the model of the receiver attached to the specified channel. The channel of interest should be specified as an unsigned two-byte integer in the request Data field. The model is returned as an ASCII-encoded string in the response Data field. If no receiver is attached, response Length `0x0000` is returned.

	Flag	Command		Tag		Length		Data	
Request	7E	A0	05	T	T	00	02	N	N
Response	7E	A1	05	T	T	L	L	*ASCII model name*	

Get client location

Gets the location of the client which currently has "write" access that is attached to the specified channel. This is typically an IP address or hostname. The channel of interest should be specified as an unsigned two-byte integer in the request Data field. The location is returned as an ASCII-encoded string in the response Data field. If no client is connected to the channel, the string "Not connected" is returned in the response Data field.

	Flag	Command		Tag		Length		Data	
Request	7E	A0	06	T	T	00	02	N	N
Response	7E	A1	06	T	T	L	L	*ASCII client location*	

Get detection mode list

Returns a comma-separated list of valid detection modes for the receiver attached to the connected channel.
This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	A0	07	T	T	00	00	
Response	7E	A1	07	T	T	L	L	*ASCII detection mode list*

Set receiver model

Changes the model of receiver attached to the connected channel. A list of valid receiver models can be obtained through the Get supported receiver models command. The request Data field should contain the ASCII-encoded model name to set. To attach no receiver to this channel, set Length to 0x0000 and leave the Data field empty. The response Data field contains the model actually set. This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	A0	08	T	T	L	L	*ASCII receiver model*
Response	7E	A1	08	T	T	L	L	*ASCII receiver model*

Get supported receiver models

Returns a comma-separated list of receiver models supported by the NRC server.

	Flag	Command		Tag		Length		Data
Request	7E	A0	09	T	T	00	00	
Response	7E	A1	09	T	T	L	L	*ASCII receiver model list*

Get channel summary

Returns the receiver model and the client location for each channel of the NRC.

	Flag	Command		Tag		Length		Data
Request	7E	A0	0A	T	T	00	00	
Response	7E	A1	0A	T	T	L	L	*ch / model length / model name / location length / location / ch …*

Set ADC Settings

Sets the current ADC (Audio/Digital converter) settings for the NRC. There are two different types of settings which are settable; global ADC settings and individual ADC channel settings (currently only Gain).
Each data element is sent in the request Data field as a two-byte unsigned integer.
The global ADC settings are sent in the first two bytes, with the individual channel settings (Ch0Cfg-Ch7Cfg) following.

The global ADC settings are sent in an integer representing individual flags that are ORed together

- `FLAG_ADC_USE_EXT_CLOCK (0x0001)`: If set, the external clock signal will drive the ADC conversion. Otherwise, the internal clock signal will operate.
- `FLAG_ADC_USE_DIFFERENTIAL_MODE (0x0002)`: If set, the audio input mode will be differential. Otherwise, single-ended input mode will be put place.

The individual channel settings are sent in separate integers, with the gains mapped to the following values:

- `Gain=1… 0, Gain=2… 1, Gain=4… 2, Gain=8… 3`

Note: Bit 8 of each channel's configuration integer is reserved.

Important note: If any settings are different from the existing set, the NRC data acquisition process will be "reboot", which will disrupt data capture on all channels.

	Flag	Command		Tag		Length		Data
Request	7E	A0	0B	T	T	00	18	Global ADC Cfg /Ch0Cfg..Ch7Cfg
Response	7E	A1	0B	T	T	00	00	

Get ADC Settings

Gets the current ADC (Audio/Digital converter) settings from the NRC. These settings are returned in the response Data field in groups of two-byte unsigned integers.
The first integer represents the global ADC setting flags that have been ORed together. To decode each flag, AND it with the combined flags. If the result is not equal to zero, the flag has been set.

- `FLAG_ADC_USE_EXT_CLOCK (0x0001)`: If set, the external clock signal is driving the ADC conversion. Otherwise, the internal clock signal is operating.
- `FLAG_ADC_USE_DIFFERENTIAL_MODE (0x0002)`: If set, the audio input mode is differential. Otherwise, single-ended input mode is in place.

The individual ADC channel configuration integers follow, with channel 0 first, channel 1 next, with channel 7 settings being in the last integer. The integers are coded with a mapping value which sets channel's pre-ADC gain, with the mappings being:

- `value=0… Gain=1, value=1… Gain=2, value=2… Gain=4, value=3… Gain=8`

Note: Bit 8 of each channel's configuration integer is reserved.

	Flag	Command		Tag		Length		Data
Request	7E	A0	0C	T	T	00	00	
Response	7E	A1	0C	T	T	00	18	Global ADC Cfg /Ch0Cfg..Ch7Cfg

Receiver Off-line

This packet is sent by the NRC server to notify the client that the receiver attached to the connected channel is off-line. This means that for some reason, the NRC cannot communicate with the receiver. The receiver should be considered on-line unless receiving this notice. A Receiver On-line packet will be sent when the NRC is again able to communicate with the receiver.
This notification will only be sent if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Response	7E	A0	00	00	00	00	00	

Receiver On-line

This packet is sent by the NRC server to notify the client that the receiver attached to the connected channel is back online.
The receiver should be considered online unless receiving a Receiver Offline notice.
This notification will only be sent if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Response	7E	A1	00	00	00	00	00	

Channel Writable

This packet is sent by the NRC server to notify the client that the connected channel has become writable (usually this is because the client which had write access disconnected and this client is next in line). If the channel is empty when the client connects, a Channel Writable notification will be sent immediately following the connect response. The client should be considered piggybacking (read only) unless receiving a channel writable notice.
When a channel is writable, the client may change the receiver configuration. When a channel is not writable, the client is connected in "piggyback" mode, and may only read (not write) configuration options and capture data.
This notification will only be sent if the client is connected to a channel (see Connect). In most situations, this notification can be safely discarded (older clients should still function correctly).

	Flag	Command		Tag		Length		Data
Response	7E	A6	03	00	00	00	00	

NRC Configuration

The NRC configuration message contains general information about the NRC and the availability of each receiver. This packet has been superseded by the "Monitor Information" packet. The response is a comma-delimited ASCII string that contains the following fields:

- The name of the NRC.
- The type of the NRC
- An empty field
- A description of each receiver that will include:
    - The receiver number
    - The receiver status (READY, BUSY, FAILED)
    - The antenna feed for the receiver
    - The receiver make and model
    - An empty field

	Flag	Command		Tag		Length		Data
Request	7E	06	0d	T	T	00	00	
Response	7E	07	0d	T	T	L	L	*Config response*

Receiver Configuration

The receiver configuration message contains general information about the capabilities of the receiver attached to the channel. The response is a comma-delimited ASCII string that contains the following fields:

- A comma-separated list of data representations (16T,32Fr,32Fz)
- An empty field
- A comma-separated list of detection modes available

- An empty field
- Sample rate
- Block Size
- Minimum frequency (Hz)
- Maximum frequency (Hz)
- An empty field

	Flag	Command		Tag		Length		Data
Request	7E	06	0e	T	T	00	00	
Response	7E	07	0e	T	T	L	L	Config response

Receiver Reset

The receiver reset message is used to reset the attached receiver to a known state. When the NRC receives this message, the NRC sends a receiver specific string to the receiver to initiate a receiver reset. The result of receiver reset will be different for different receiver types. For example, when a WJ-8723 receiver is reset, it will be configured to a known set of parameters, while when a TenTec Rx331 receiver is reset it maintains its most recent set of configuration parameters.

	Flag	Command		Tag		Length		Data
Request	7E	06	0f	T	T	00	00	
Response	7E	07	0f	T	T	00	00	

Receiver Reboot

The receiver reboot message is used to perform a software reboot of the attached receiver. When the NRC receives this message, the NRC sends a receiver specific string to the receiver to initiate a reboot. This is usually used as a last resort to try and overcome a receiver lock-up problem, similar to the "Tone-of-death" condition associated with the WJ-8723 receiver. After the reboot the receiver is reset.

	Flag	Command		Tag		Length		Data
Request	7E	06	11	T	T	00	00	
Response	7E	07	11	T	T	00	00	

Set AGC Mode

Sets the AGC mode of the receiver attached to the connected channel. The request Data field should contain an ASCII-encoded string of the desired AGC mode. Valid values are "SLOW", "MEDIUM", and "FAST".
When the agc mode of the receiver is changed, the current data buffer is flushed and incoming data samples are dropped until the number of samples specified by the channel have been skipped (see Drop samples). Data capture is then resumed.
This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	06	10	T	T	L	L	*ASCII AGC mode*
Response	7E	07	10	T	T	00	00	

Get AGC Mode

Gets the AGC mode of the receiver attached to the connected channel. The response Data field will contain an ASCII-encoded string of the AGC mode.
This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	A6	11	T	T	00	00	
Response	7E	A7	11	T	T	L	L	*ASCII detection mode*

Get Antenna

Gets the name of the antenna connected to the receiver attached to the specified channel. The channel of interest should be specified as an unsigned two-byte integer in the request Data field. The antenna is returned as an ASCII-encoded string in the response Data field. If no antenna is attached, response Length `0x0000` is returned.

	Flag	Command		Tag		Length		Data	
Request	7E	A6	0f	T	T	00	02	N	N
Response	7E	A7	0f	T	T	L	L	*ASCII antenna name*	

Set Antenna

Changes the name of the antenna connected to the receiver of the attached channel. The request Data field should contain the ASCII-encoded antenna name to set. To attach no antenna name to this channel, set Length to 0x0000 and leave the Data field empty. The response Data field contains the name of the antenna actually set.
This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	A6	10	T	T	L	L	*ASCII antenna name*
Response	7E	A7	10	T	T	L	L	*ASCII antenna name*

Get Writable

Returns a value indicating whether or not the client can write to the specified channel and hence whether the client can reconfigure the channel's attached receiver or not. When a channel is writable, the client establishes a primary connection to the channel and may change all receiver's configuration. When a channel is not writable, the client is connected in "piggyback" mode, and may only read (not write) receiver configuration options and capture data. The channel of interest is specified as an unsigned two-byte integer in the request Data field. The response is returned as a one byte value indicating whether the client can write to the channel (1) or not (0). If the client is not connected to the specified channel, the value indicates that the channel is currently open and if the client were to connect to it, it would receive write access.

	Flag	Command		Tag		Length		Data	
Request	7E	A6	12	T	T	00	02	N	N
Response	7E	A7	12	T	T	00	01	N	

NRC Reboot

The NRC reboot message causes the NRC software to perform a warm restart. Any currently active socket connections will be terminated and all parameters will be restored to their initial power up conditions. A restart of the NRC will take about one minute to complete. Any attempts to establish a socket connection with the NRC during the reboot period will fail. This command will not send an acknowledgment response to the client.

	Flag	Command		Tag		Length		Data
Request	7E	06	ff	T	T	00	00	

Monitor Information

This command causes the NRC to reply with detailed information about the system status and configuration for the NRC and all channels and connected clients. This command extends the functionality of the NRC Configuration packet.
The response is formatted as follows: (Strings are prefixed with a 2 byte length)

- 2 bytes - NRC flags (see the Get ADC Settings packet)
- 2 bytes – Number of gain entries
- For each gain entry:
    - 2 bytes – gain for this channel (See Get ADC Settings for more info)
- String – NRC Name
- String – NRC Type (usually “AEGIS_NRC”)
- 4 bytes – Up time of the NRC server (in seconds)
- 1 byte – Server CPU usage over the last 5 seconds (as a percentage)
- 1 byte – Server memory usage (as a percentage)
- 2 bytes – Incoming server network usage in Kbytes/second
- 2 bytes – Outgoing server network usage in Kbytes/second
- 2 bytes – Number of channels
- For each channel:
    - String – Receiver Model
    - String – Antenna
    - 1 byte – Is receiver on-line (1=yes, 0=no)
    - 2 bytes – Number of samples to skip on tune
    - String – Frequency (See Get Frequency for format information)
    - String – Detection Mode (See Get Detection Mode for format information)
    - String – AGC Mode (See Get AGC Mode for format information)
    - String – BFO (See Get BFO for format information)
    - String – IF bandwidth (See Get IF-BW for format information)
    - 2 bytes – number of device entries
    - For each device entry:
        - String – Device Location. If this string’s length is 0, then there is no device connected and the next 16 bytes are not present in the

packet for this device.
    - 4 bytes – The time the client has been connected to the NRC (secs)
    - 1 byte – Is the client currently capturing data (1=yes, 0=no) (see the "Set Data Options" packet for more explanations of the following data fields)
    - 4 bytes – Length of time the client has been capturing data (in seconds)
    - 2 bytes – Sample rate (See Set Data Options for more information)
    - 2 bytes – Number of samples per packet (See Set Data Options for more information)
    - 2 bytes – Filter option (See Set Data Options for more information)
    - 1 byte – Time stamp option (See Set Data Options for more information)
- 1 byte – Channel status (ASCII character that would be displayed on the NRC's front panel LCD)

	Flag	Command		Tag		Length		Data
Request	7E	A6	13	T	T	0	0	
Response	7E	A7	13	T	T	L	L	...

Query Receiver Memory Capacity

This command returns the number of internal memory locations present in the receiver. This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	A6	15	T	T	L	L	
Response	7E	A7	15	T	T	L	L	*ASCII number of locations*

Query Receiver Memory Location

This command returns a string representing the settings stored in the receiver's internal memory at the specified memory location. If the receiver does not support this command, the string will contain an error message instead of the contents of the requested memory location. This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	A6	16	T	T	L	L	*ASCII memory location*
Response	7E	A7	16	T	T	L	L	*ASCII settings*

Load Memory Location

This command causes the receiver to load the contents of the specified internal memory location into the receiver's current configuration. If nothing is stored at the specified location the behavior of this command is receiver dependant; it may do nothing, it may load a default set of values, or some other behavior. Consult the receiver documentation

for more information. This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	A6	17	T	T	L	L	*ASCII memory location*
Response	7E	A7	17	T	T	00	00	

Write Memory Location

This command causes the receiver to save the current receiver configuration to the specified internal receiver memory location. The volatility of this memory is receiver dependant; consult the receiver documentation for more information. This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	A6	18	T	T	L	L	*ASCII memory location*
Response	7E	A7	18	T	T	00	00	

Query Receiver Memory Fields

This command returns a string containing the names of receiver fields which are known to be stored in the receiver's internal memory. These fields can vary between receiver models, and possibly even different firmware versions of the same receiver model. This command can only be successfully executed if the client is connected to a channel (see Connect).

	Flag	Command		Tag		Length		Data
Request	7E	A6	19	T	T	00	00	
Response	7E	A7	19	T	T	L	L	*ASCII memory fields*

NRC File Information Request

This command returns file name, size, and modification date for several files defined by the NRC. The data contained in this packet is useful for determining the approximate version of the NRC and when various modules were last updated.
The returned data is in the following format:

- 2 bytes – Number of files in the list
- For each file:
    - 2 bytes – File name length
    - String – File name
    - 4 bytes – Modification time of the file (Represented as the number of seconds since 00:00:00 1/1/1970)
    - 4 bytes – File size (in bytes)

	Flag	Command		Tag		Length		Data
Request	7E	A6	1A	T	T	00	00	
Response	7E	A7	1A	T	T	L	L	*File information*

Errors

These error constants are defined in the NRC C++ Library in NRCRPMConstants.h and in com.aegis.NRC.sdk.java.RPMConstants in the NRC Java Library.

ERNODATAWRITTEN

(0x0000) Zero bytes of data were written to the socket.

ERWRITEERROR

(0x0001) An unspecified error has occurred.

ERINVALIDCMD

(0x0002) An invalid command value was used in the packet message.

ERINVALIDLEN

(0x0003) An invalid length value was used in the packet message.

ERINVALIDCHAN

(0x0004) An invalid channel value was used in the packet message.

ERCHANALRDYCON

(0x0005) The channel requested is already connected to another client.

ERSOCKALRDYCON

(0x0006) The client is already connected to a channel.

ERSOCKNOTCON

(0x0007) The client is not connected to a channel.

ENORESOURCE

(0x0008) No receivers are available.

ENOTCONNECTED

(0x0009) No receiver is connected.

EALREADYCON

(0x000A) A receiver is already connected.

EARGUMENT

(0x000B) The command argument is invalid.

ESERIALOVLD

(0x000C) The serial port is overloaded.

ERTIMEOUT
(0xA001) The command timed out.

ERUNKNOWN
(0xA002) An unspecified error has occurred.

ERINVALIDSAMPLERATE
(0xA003) An invalid sample rate was specified.

ERINVALIDSAMPLESPERPACKET
(0xA004) An invalid number of samples power packet was specified.

ERNORECEIVERDEF
(0xA005) A receiver definition file could not be found for this receiver.

ERBADCMDTRANSLATION
(0xA006) The command could not be translated for this receiver.

ERCANNOTREMOVERECEIVER
(0xA007) The receiver could not be removed from the channel.

ERCANNOTATTACHRECEIVER
(0xA008) The receiver could not be attached to the channel.

ERCAPTURING
(0xA009) The command cannot be executed because the channel is currently capturing data.

ERINVALIDVGCFILTEROPTION
(0xA00A) An invalid channel filter option was specified.

ERREADONLY
(0xA00B) The requested command requires write access to the receiver.

