



RE02 Network Interface

Class Library Reference Manual

Version 1.0.110
Mon May 30 2011

Ocular Robotics Pty. Ltd.
Level 3, 12-14 Ormonde Pde
Hurstville NSW 2220
Australia

www.ocularrobotics.com

Contents

1	RE02 Network Interface Class Library Reference Manual	1
2	RE02 Conventions	3
2.1	RE02 Coordinate System Definitions	3
2.2	RE02 Angle Conventions	5
3	RE02 Network Configuration Information	7
3.1	RE02 Network Setup	7
3.2	RE02 IP Address	7
3.2.1	RE02 IP Address Configuration	7
3.3	Broadcast vs. Unicast	8
3.4	The Primary Client	8
3.5	Preferred Network Configuration	8
3.6	Configuring Your Firewall	10
4	Installation	11
4.1	Windows	11
4.2	Linux	12
5	Using the RE02 Network Interface API	13
5.1	API Usage	13
5.1.1	The RE02 Network Interface Library Namespace	13
5.1.2	A Simple Example	13
5.1.3	Error Handling	14
5.1.4	A Realistic Example	15
5.2	Example Code	16
6	Building Applications	17
6.1	Setting Up Your Project	17
6.1.1	Using CMake	17
6.1.2	Manually Setting up Projects	19
6.1.2.1	Microsoft Visual Studio	19
7	Examples	21
7.1	The RE02 Console Interface	21
7.2	The RE02 Logger Utility	22
7.3	The RE02 Binary to CSV (Ascii) Converter Utility	22
7.4	Typical Usage	23
8	Namespace Documentation	25
8.1	ocular Namespace Reference	25
8.1.1	Detailed Description	25
9	Class Documentation	27
9.1	ocular::RE02NetworkInterface Class Reference	27
9.1.1	Detailed Description	28

9.1.2	Constructor & Destructor Documentation	28
9.1.2.1	RE02NetworkInterface	28
9.1.2.2	~RE02NetworkInterface	28
9.1.3	Member Function Documentation	28
9.1.3.1	ChangeToDHCP	28
9.1.3.2	GetLibraryVersion	29
9.1.3.3	GetRE02Addresses	29
9.1.3.4	SetBoundedElevationScanSettings	29
9.1.3.5	SetDestinationIPAddress	30
9.1.3.6	SetFullFieldScanSettings	30
9.1.3.7	SetRegionScanSettings	32
9.1.3.8	SetSampleFrequencySettings	32
9.1.3.9	SetStaticIPAddress	33
9.1.3.10	SetToUnicast	33
9.1.3.11	Start	33
9.1.3.12	Stop	33
9.2	ocular::RE02NetworkInterfaceCallback Class Reference	33
9.2.1	Detailed Description	34
9.2.2	Member Function Documentation	34
9.2.2.1	ReceiveInfoMessage	34
9.2.2.2	ReceiveRangeData	34
9.2.2.3	ReceiveStatus	35

Chapter 1

RE02 Network Interface Class Library Reference Manual



This document is the programmer's reference for Ocular Robotics' Pty. Ltd. RE02 network interface driver and its Application Programming Interface (API).

It is implemented as a cross-platform C++ class library with simple API.

Chapter 2

RE02 Conventions

2.1 RE02 Coordinate System Definitions

The RE02 device uses a right-handed (or *positive*) coordinate system fixed to the centre of the *head* of the RE02 with the aperture centred on the *y* axis when the aperture is at zero degrees azimuth and zero degrees elevation. The *x* axis is constrained to the horizontal plane, and the *z* axis *up* with respect to *x* and *y*. The aperture angles (usually specified as the *azimuthAngle* and *elevationAngle*) are angular offsets in azimuth and elevation from the *y* axis.

The following diagrams summarise the coordinate system used for the RE02 system.

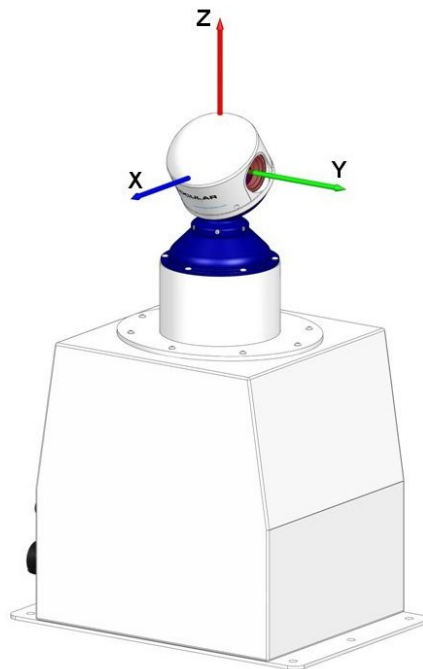


Figure 2.1: The RE02 Coordinate System

The diagram above shows the right-handed frame used for the RE02.

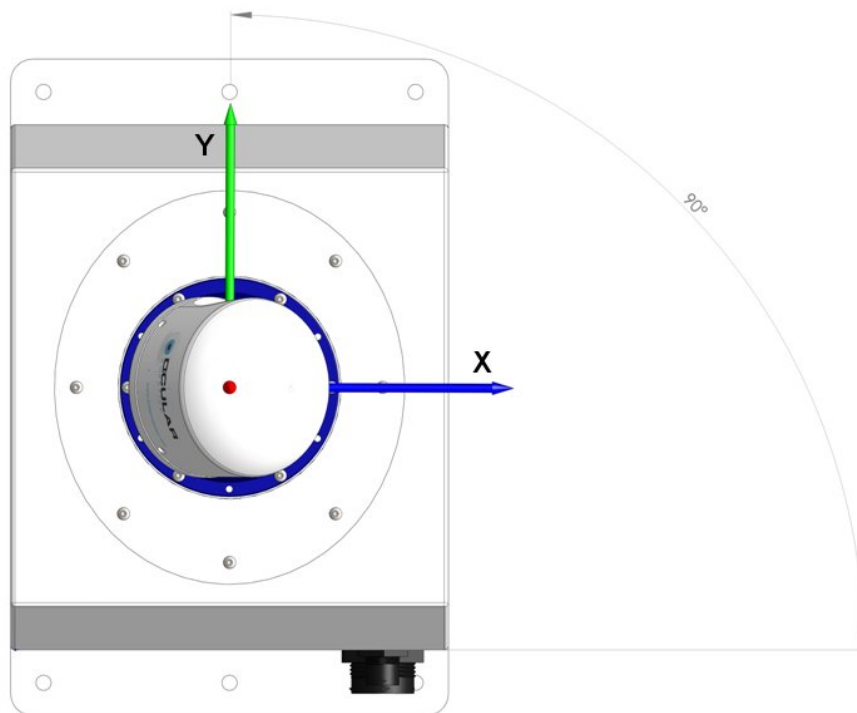


Figure 2.2: The RE02 Azimuth Reference Surface

The diagram above shows the reference used for the direction of the y axis. The y axis, which is the *azimuthAngle* zero reference is defined as being opposite to the vertical RE02 face with the power and ethernet connections.

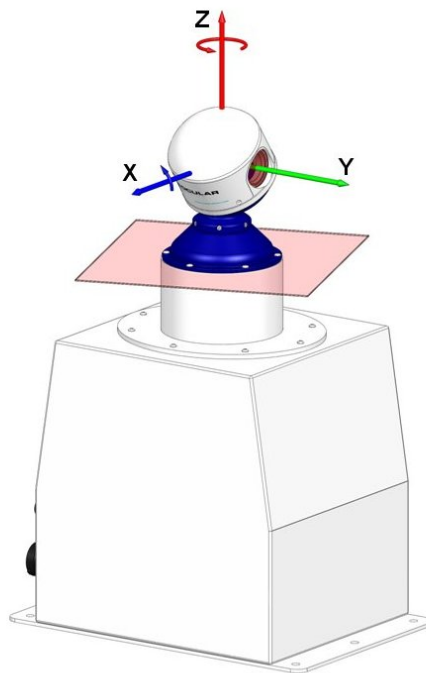


Figure 2.3: The RE02 Elevation Reference Surface and Azimuth and Elevation Rotation Sign

The final diagram shows two things;

- The aperture zero elevation angle lies in a plane parallel to the to the RE02 mounting flange.
- The sign of the aperture angles, and angular rates. AzimuthAngle is more positive anti-clockwise, and ElevationAngle is positive upwards.

2.2 RE02 Angle Conventions

The RE02 network interface library uses the following conventions when dealing with angles.

- Angles passed to the library must be expressed in degrees
- Functions that command the RE02 to an absolute position will always take the shortest (or acute angle) path from the current location to the new location. i.e. if the RE02 azimuth angle is currently 359 degrees, and it is commanded to 1 degree, it will only move 2 degrees in azimuth.
- Azimuth angles must be constrained to the domain 0 -> 360 degrees, where zero is defined as per the [RE02 Coordinate System Definitions](#).
- Elevation angles must be constrained to the domain -maxElevation -> maxElevation, where maxElevation is typically 35 degrees. This angle can be varied for custom RE02 devices. Zero degrees elevation is horizontal, as per the definitions given in the [RE02 Coordinate System Definitions](#).

Chapter 3

RE02 Network Configuration Information

3.1 RE02 Network Setup

The RE02 communicates over gigabit ethernet using broadcast or unicast UDP packets. Data sent from the client software (using the library described in this document) is sent as unicast UDP packets directly to the RE02.

UDP packets are not *guaranteed delivery*, however, on a well designed network it is very unlikely that data will be lost, even at the highest sensor sample rates (200KHz). See [Preferred Network Configuration](#) for the recommended network setup for the RE02.

It is possible to use the RE02 in a 100 or 10 Mbit network, however using the sensor on one of these lower bandwidth networks may result in packet loss. The advantage of UDP/IP over TCP/IP is the avoidance of possible TCP/IP transmission delays, which is essential for real-time applications of the RE02.

3.2 RE02 IP Address

The RE02 will come preconfigured with a static IP address on a private IPV4 subnet. See [IPv4 private addresses](#) for more information. The specific IP address will be provided with the RE02 packaging documentation. Please contact Ocular Robotics Pty Ltd prior to delivery if you require a specific IP address for your device.

3.2.1 RE02 IP Address Configuration

The RE02 can be configured to use a static IP address (the default), or to obtain an IP address from a DHCP server. These settings can be changed using the [ocular::RE02NetworkInterface::SetStaticIPAddress](#) and [ocular::RE02NetworkInterface::ChangeToDHCP](#) functions in the class library. To set a static IP address, a valid IP address, netMask and default gateway must be supplied. To use DHCP, no arguments are required.

The IP address information is non-volatile. i.e. the RE02 will *remember* its IP address settings across power cycles.

Note that in the event of a change of IP address, the RE02 device will reset to its default state of sending broadcast UDP packets on the local subnet given by the new IP address and netmask. See [Broadcast vs. Unicast](#) for a discussion of the relative merits of broadcast vs. unicast datagrams.

3.3 Broadcast vs. Unicast

By default, the RE02 uses broadcast UDP packets to transmit laser data. This allows multiple clients to be present in the network, each able to consume the data produced by the RE02. This mode however has the downside that all of the RE02 data will be sent to every network device on the RE02 subnet. This may be undesirable in certain network configurations.

The alternative configuration for the RE02 is to use unicast UDP packets. In this mode, the RE02 will only transmit data to the *Primary Client*. In a fully switched network, the RE02 data will therefore not arrive at any other network device.

The RE02 can be figured for broadcast or unicast by using the `ocular::RE02NetworkInterface::SetToBroadcast` or `ocular::RE02NetworkInterface::SetToUnicast` functions in the class library.

See [this Wikipedia article](#) for a brief discussion and diagrams of these routing policies.

Note that the RE02 will always power up in broadcast mode. This allows the device's IP address to be discovered by possible clients.

3.4 The Primary Client

The RE02 is configured to allow any number of clients to receive data when in *broadcast* mode. The RE02 will only accept commands from the *Primary Client* however. This is to ensure that multiple clients cannot simultaneously send conflicting command to the sensor.

The *Primary Client* is determined by the RE02 in a very simple manner: The *Primary Client* is the client that first sends data to the RE02 after it is powered up.

Note that the *Primary Client* is reset on a change of IP address, to allow for a change of address range that would effectively block the original *Primary Client*.

3.5 Preferred Network Configuration

It is strongly recommended that the local network for the RE02 device be set up as a *private network*, primarily to avoid large amounts of data potentially being sent (unwanted) to all the computers on the network. If a computer must be connected to another network (i.e. a corporate intranet, or the internet), it should be via a separate network adapter on a different subnet to the RE02 device.

It is possible to connect multiple RE02 devices in a single network. In this case, it is recommended that the network be fully switched to avoid potential data collisions on the network.

The following diagrams illustrate the preferred network setup for one or multiple RE02s.

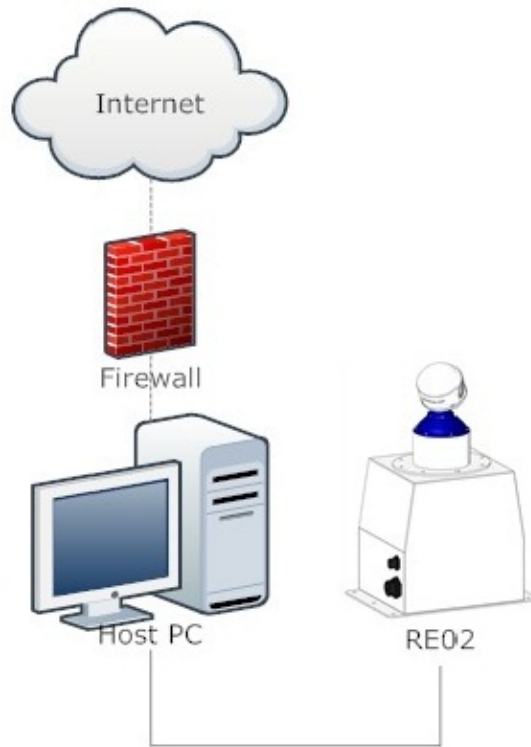


Figure 3.1: Network Schematic for Connecting a Single Host PC to a Single RE02

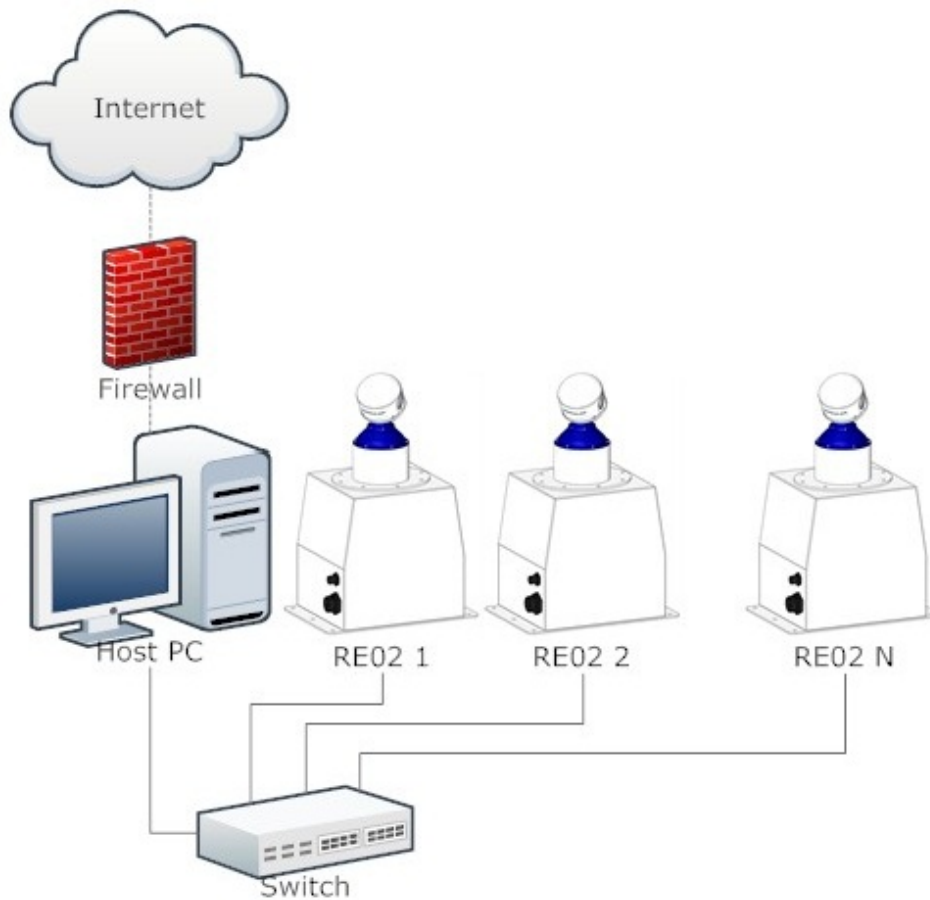


Figure 3.2: Network Schematic for Connecting a Host PC to Multiple RE02s

3.6 Configuring Your Firewall

In general, if the advice given in [Preferred Network Configuration](#) is followed, then it should be acceptable to disable any firewalls on the network adapter connected to the RE02 device, as it is on a private network not accessible to the outside world. This is the simplest option if you are having network related issues.

Alternatively, if you must retain a firewall on the given network adapter, then make sure that UDP ports 11000 and 11100 are given exceptions in the firewall program (i.e. allow bidirectional communication on those UDP ports).

Chapter 4

Installation

4.1 Windows

On Windows platforms, the driver is installed using an installer program. The installer filename will be of the form *RE02Lib-0.9.8-win32.exe*, where *0.9.8* indicates the driver version number, and *win32* indicates the target platform. By default, the driver is built as a 32-bit dll, however a 64-bit version may be requested if necessary.

To perform the installation under windows, simply double click the installer application, and follow the on-screen instructions. Note - you may need administrator privileges to run the installer.

The following figure shows the installer interface



Figure 4.1: The RE02 Network Interface Class Library Windows Installer

After accepting the Ocular Robotics license agreement, you will be prompted to enter an installation directory at the following screen.

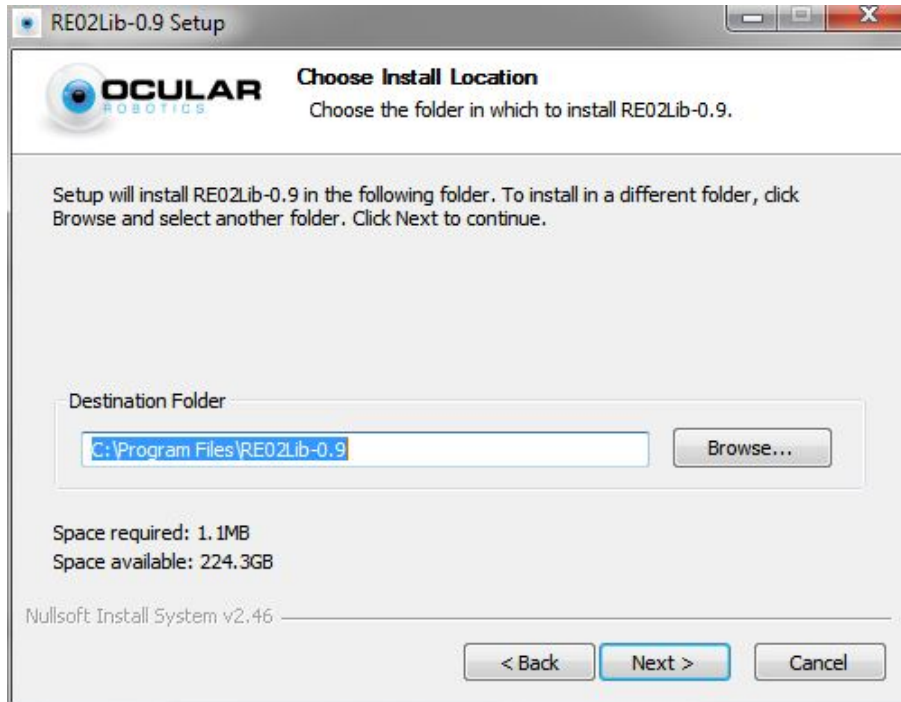


Figure 4.2: The Default RE02 Network Interface Class Library Windows Installation Directory

This directory will then contain all of the RE02 library files: the driver, documentation and examples. In a Windows installation, a start menu group with links to the driver documentation will also be created.

4.2 Linux

The RE02 network interface library is distributed under Linux in two forms:

- as a .deb Debian Package, and
- as a .rpm Red Hat Package

Under Linux, the library will be installed to `/opt/RE02Lib-version`, and is installed by invoking the relevant package manager.

Chapter 5

Using the RE02 Network Interface API

5.1 API Usage

Under Windows, the R02 network interface class library consists of the following components;

- A header file *RE02NetworkInterface.h* which contains the public interface of the library
- A header file *RE02NetworkInterfaceCallback.h* which contains the virtual prototype for receiving feedback from the RE02
- An import library *RE02.lib* built for the Microsoft Visual Studio compiler
- A shared library *RE02.dll*

In a Linux installation, the import library is not required, and the shared library will have a .so file extension, and have the *lib* prefix (i.e. *libRE02.so*).

5.1.1 The RE02 Network Interface Library Namespace

All C++ software provided by Ocular Robotics Pty. Ltd. is encapsulated within the namespace *ocular*. Therefore, all usage of this class library must either explicitly use the *ocular* namespace, or have the following line of code added before objects contained in the namespace can be used.

```
using namespace ocular;
```

In the examples that follow, the namespace is explicitly used.

5.1.2 A Simple Example

For the simplest possible working example (but not functional), create a source file (say *MyProject.cpp*), and include the *RE02NetworkInterface* and *RE02NetworkInterfaceCallback* header files. A callback must be derived from the prototype class provided in the *RE02NetworkInterfaceCallback* header file. Then, construct an *RE02NetworkInterface* object, passing your callback as the input argument.

This code would resemble the following;

```
#include <RE02NetworkInterface.h>
#include <RE02NetworkInterfaceCallback.h>

class myRE02NetworkInterfaceCallback:public ocular::RE02NetworkInterfaceCallback
```

```

{
public:
    myRE02NetworkInterfaceCallback() {}
    ~myRE02NetworkInterfaceCallback() {}

    void ReceiveRangeData( std::string & data )
    {
        // do something
    }

    void ReceiveInfoMessage( std::string & msg )
    {
        // do something
    }
};

int main( void )
{
    myRE02NetworkInterfaceCallback myCallback;
    ocular::RE02NetworkInterface re02( &myCallback );

    return 0;
}

```

The act of construction of the RE02NetworkInterface object will perform all necessary resource acquisition and initialisation. If the RE02NetworkInterface object is constructed successfully, it will be immediately ready to use.

5.1.3 Error Handling

The RE02 network interface library uses exceptions to indicate to a user when an error occurred, and what the source of the error is. It is therefore always recommended to use the C++ mechanism of try-catch blocks around the use of any RE02NetworkInterface member functions. This will help to ensure that any programmatic errors are identified and rectified in a timely manner.

All RE02NetworkInterface exceptions are derived from the C++ standard library *exception* class, so user code only has to catch these exceptions. The class reference indicates which functions throw exceptions, and what the likely causes of those exceptions may be.

An example showing the use of an appropriate try-catch block is shown below;

```

#include <RE02NetworkInterface.h>
#include <RE02NetworkInterfaceCallback.h>

#include <exception>
#include <iostream>

class myRE02NetworkInterfaceCallback:public ocular::RE02NetworkInterfaceCallback
{
public:
    myRE02NetworkInterfaceCallback() {}
    ~myRE02NetworkInterfaceCallback() {}

    void ReceiveRangeData( std::string & data )
    {
        // do something
    }

    void ReceiveInfoMessage( std::string & msg )
    {
        std::cout << std::endl;
        std::cout << "*****" << std::endl;
        std::cout << msg << std::endl;
        std::cout << "*****" << std::endl;
        std::cout << std::endl;
    }
}

```

```

};

int main( void )
{
    try
    {
        myRE02NetworkInterfaceCallback myCallback;
        ocular::RE02NetworkInterface re02( &myCallback );

    }
    catch( std::exception & e )
    {
        std::cout << "Caught Exception: " << e.what() << std::endl;
        return 1;
    }

    return 0;
}

```

5.1.4 A Realistic Example

This example is a complete example of the use of the RE02 network interface class library. The callback is configured to print to the screen the data from the first sample of each incoming data packet. Responses to any errors in RE02 configuration will be printed to the screen via the ReceiveInfoMessage function.

A call to the Start method must be issued before any other commands will have an effect. The callback will also not be executed until Start has been called. Once Start has been called, the main thread of execution is then free to perform any other tasks that may be necessary. In this example, it simply waits for user input on the keyboard. If the user presses the letter *q*, the program exits.

```

#include <RE02NetworkInterface.h>
#include <RE02NetworkInterfaceCallback.h>

#include <exception>
#include <iostream>
#include <vector>
#include <string>

class myRE02NetworkInterfaceCallback:public ocular::RE02NetworkInterfaceCallback
{
public:
    myRE02NetworkInterfaceCallback(){}
    ~myRE02NetworkInterfaceCallback(){}

    //note: this function is only printing the first sample from the data packet.

    void ReceiveRangeData( std::string & data )
    {
        float range = *(float *)(&data[0]);
        float azimuth = *(float *)(&data[4]);
        float elevation = *(float *)(&data[8]);
        unsigned char amplitude = *(char *)(&data[12]);
        std::cout << "Received range data: " << range << " " << azimuth
            << " " << elevation << " " << (unsigned int)amplitude << std::endl;
    }

    void ReceiveInfoMessage( std::string & msg )
    {
        std::cout << std::endl;
        std::cout << "*****" << std::endl;
        std::cout << msg << std::endl;
        std::cout << "*****" << std::endl;
        std::cout << std::endl;
    }
};

int main( void )

```

```

{
    try
    {
        myRE02NetworkInterfaceCallback myCallback;
        ocular::RE02NetworkInterface re02( &myCallback );

        // start up the service
        // the callback will only execute after Start has been called
        re02.Start();

        // get the addresses of all available RE02s and simply use the first one.

        std::vector<std::string> re02Addresses = re02.GetRE02Addresses( 500 );
        if( re02Addresses.size() != 0 )
        {
            re02.SetDestinationIPAddress( re02Addresses[0] );
        }
        else // if no RE02s, throw an error, and exit.
        {
            throw std::runtime_error( "Unable to find an RE02 on this subnet" );
        }

        //set azimuth and elevation aperture velocities
        re02.SetFullFieldScanSettings( 4.0, 0.1 );

        char c;

        do
        {
            std::cin >> c; // wait for the user to enter a 'q'

        }while( c != 'q' );

        // stop the service
        re02.Stop();

    }
    catch( std::exception & e )
    {
        std::cout << "Caught Exception: " << e.what() << std::endl;
        return 1;
    }

    return 0;
}

```

5.2 Example Code

Please see the *examples* subdirectory of the RE02 Library install directory. This directory contains examples showing the use of the different control modes of the RE02, a threaded logging system, and a simple data conversion utility.

For information on compilation and running of these examples, see [Building Applications](#) and [Examples](#).

Chapter 6

Building Applications

6.1 Setting Up Your Project

This section describes how to set up a C++ project to use the RE02 network interface class library.

In general, the project only needs to know where the header files for the library are, and the name and location of the import library. In linux the shared object file (.so) doubles as the import library.

Note that once your project has been built, it will not execute unless the shared library is in one of the library loader search paths. See [this Wikipedia article](#) for a guide to the search paths used for your operating system.

6.1.1 Using CMake

CMake is a build tool that aids in setting up a project in an operating system and compiler independent manner. It is used by Ocular Robotics Pty. Ltd. on its internal projects, and therefore its use can greatly simplify the process of setting up a new project for use with the RE02 library.

CMake can be downloaded from [the CMake website](#).

Linux users can use their favourite package manager to install CMake. It is recommended you also install CMake (a command line version of CMake which uses the curses library to emulate a simple GUI). There is also a QT based GUI for CMake which is identical to the Windows GUI.

CMake uses a script (CMakeLists.txt) to encode the rules about how a project should be built. These scripts will usually reside in the same directory as the source code for the project. For a simple application, it will look similar to the following;

```
# don't allow any version of CMake below this - or else it probably won't work
CMAKE_MINIMUM_REQUIRED( VERSION 2.6 )

# name for the project
PROJECT( RE02ConsoleInterface )

# this project requires the RE02 Lib, so find it
FIND_PACKAGE( RE02Lib REQUIRED )

# set the include directory for the RE02 lib
INCLUDE_DIRECTORIES( ${RE02Lib_INCLUDE_DIR} )

# name the executable, and add source to it
ADD_EXECUTABLE( RE02ConsoleInterface RE02ConsoleInterface.cpp )

# link to the RE02 library
TARGET_LINK_LIBRARIES( RE02ConsoleInterface ${RE02Lib_LIBRARIES} )
```

The script above simply gives a name to the project, says that the RE02 library is required, then adds the RE02Lib include directory to the project, specifies that an executable *RE02ConsoleInterface* should be built from *RE02ConsoleInterface.cpp* and linked to the RE02Lib import library. These CmakeLists.txt files are included with the examples supplied with the RE02 library and can be very simply modified for your own projects.

The following steps outline how projects are built using CMake (identical for Windows and the Linux QT GUI).

- First, download and install CMake, following the instructions at [the CMake website](#).
- Run the CMake GUI
- Now tell CMake where the *source code* is and where to build the *binaries* (ideally these will be different directories i.e. an out-of-source build)
- Click *Configure*
 - Answer the questions (create directory -> yes) (compiler -> Visual Studio 9 2008)
 - If the default RE02Lib install paths are used, CMake should automatically find the RE02 library resulting in the CMake GUI looking similar to the screen shot below

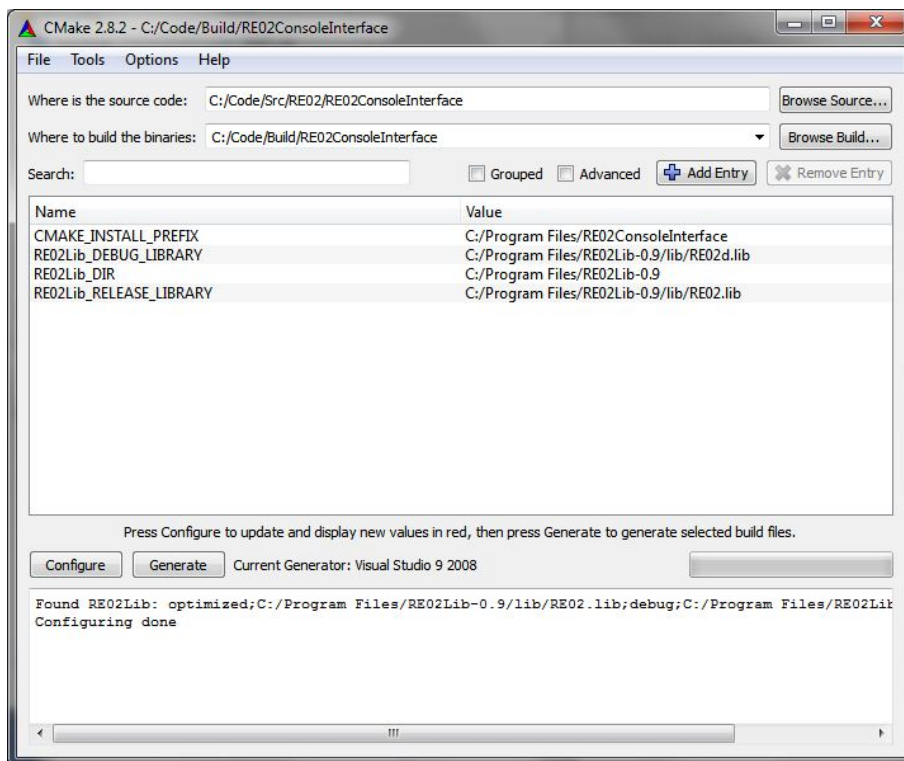


Figure 6.1: The CMake GUI for the *RE02ConsoleInterface* Project

- Now click *Configure* once more
- Now click *Generate*

Congratulations! - you should now have a Visual Studio solution ready to build and execute in the directory you gave CMake as the *binaries* directory (in the example "C:\Code\Build\RE02ConsoleInterface\RE02ConsoleInterface.sln")

- If you receive the CMake error "Error in configuration process...", it is likely to be because the RE02 library was not found.
- In this case, you will need to manually set the CMake variable `RE02Lib_DIR` to the top level directory of the RE02 library (for example "C:/Program Files/RE02Lib-0.9" - note the Unix style slashes)
- Then, as before, click *Configure*, followed by *Generate*.

6.1.2 Manually Setting up Projects

6.1.2.1 Microsoft Visual Studio

This section describes the steps necessary to manually configure Microsoft Visual Studio to build a project using the RE02 network interface class library

The following steps will allow compilation and execution of any of the examples given in [Using the RE02 Network Interface API](#).

- In the Visual Studio IDE, create a new project using the following steps
 - File -> New -> Project
 - Visual C++ -> General -> Empty Project
 - Give a name and location for the project
 - Click OK
- In the Solution Explorer, right click the *Source Files* folder then Add -> Existing Item
 - Navigate to the cpp file you wish to use, select it, then click OK
- Now set the project properties with the following steps
 - Project -> Properties (or Alt+F7)
 - Configuration Properties -> C/C++ -> General
 - Add the full path to the directory containing [RE02NetworkInterface.h](#) to the *Additional Include Directories* box (e.g. "C:\Program Files\RE02Lib-0.9\include" (the quotation marks must be included))
 - Click Apply
 - Configuration Properties -> Linker -> Input
 - Add the full path to the RE02.lib file to the *Additional Dependencies* box (e.g. "C:\Program Files\RE02Lib-0.9\lib\RE02.lib" (the quotation marks must be included))
 - Click Apply
 - Configuration Properties -> C/C++ -> Code Generation
 - Set the *Runtime Library* to *Multi-threaded DLL (/MD)* if linking against the release version of the library (RE02.lib)
 - Set the *Runtime Library* to *Multi-threaded Debug DLL (/MDd)* if linking against the debug version of the library (RE02d.lib)
 - Click Apply
 - Click OK
- You can now build and run the project

Remember to put the RE02.dll somewhere in the library loader's search path.

Chapter 7

Examples

The RE02 Network interface class library described in this document provides a programmer with a convenient method for accessing the functionality of the RE02 device. The examples described in this section are intended to provide some simple examples of library use to guide a developer in creating their own implementations of high-level interface software. The applications may be built and run for both linux and Windows, and are invoked from the command-line.

The examples are;

- a Console Interface, to demonstrate how the library can be used to send commands to the RE02
- a Logger, to show how data may be received from the RE02 and incorporated in some useful task
- a Converter, which is used to convert the binary data recorded by the logger into a more flexible Comma Separated Value (CSV) Ascii format

These examples are fairly trivial in nature, and do not provide full error-checking of user input, for example. They are simply intended for reference. RE02 Tools, distributed in the installer, should be used in most instances.

7.1 The RE02 Console Interface

The console interface will display a *usage* screen on startup. Follow the instructions listed on the usage screen to access the functionality of the RE02 device.

For example, the command

```
RE02ConsoleInterface
```

will result in the following output

```
***** RE02 Console Interface *****
*
* type 'h' <enter> to show this message
* type 't' <enter> to toggle RE02 data display
*
* type 'd' <enter> to configure RE02 to use DHCP
* type 'i' <enter> to set static IP address for the RE02
* - usage 'i' <enter> 'y' <enter> 'ipaddress' 'netmask' 'gateway' <enter>
* type 'u' <enter> to set RE02 to Unicast mode (to this address)
* type 'b' <enter> to set RE02 to Broadcast mode (on by default)
*
* type 'f' <enter> to set full field scan settings
* - usage 'f' <enter> 'y' <enter> 'azimuthRate' 'elevationRate'
```

```

* type 'e' <enter> to set bounded elevation scan settings
* - usage 'e' <enter> 'y' <enter> 'azimuthRate' 'lowerElevation'
* 'upperElevation' 'lineSpacing' <enter>
* type 'r' <enter> to set Region Scan settings
* - usage 'r' <enter> 'y' <enter> 'azimuthRate' 'initialAzimuth'
* 'initialElevation' 'deltaAzimuth' 'deltaElevation' 'LineSpacing' <enter>
* type 's' <enter> to set Sample Frequency settings
* - usage 's' <enter> 'y' <enter> 'frequency' 'maxRange' <enter>
*
* type 'q' <enter> to quit this application
*****
Started RE02 Interface successfully

```

Typing *q* followed by <enter> will terminate this program.

7.2 The RE02 Logger Utility

The RE02 logger utility, records range data from the RE02 device to binary file. The data can be recorded in range, azimuth, elvation, intensity format, or alternatively it can be converted to x, y, z, intensity and recorded. It should be recorded as binary data to help speed disk-writes. Use of the ascii formats is unlikely to fast enough on standard hard disks.

The parameters passed to the logger are one or more switches to indicate the data type to log. The possible command line switches are *-f*, *-fxyz*, *-af* and *-stdOut*, where;

- *-f* saves as the Binary file *RE02Data.bin* in range, azimuth, elevation, intensity format
- *-fxyz* saves as the Binary file *RE02XYZData.bin* in x, y, z, intensity format
- *-af* saves as the ASCII file *RE02AsciiData.txt* in x, y, z, intensity format
- *-stdOut* logs to stdout in ASCII x, y, z, intensity format

Typical usage would appear similar to the following;

```
RE02Logger -f
```

Typing *q* followed by <enter> will terminate this program.

7.3 The RE02 Binary to CSV (Ascii) Converter Utility

The RE02 Binary to CSV converter is used to filter and convert binary files created by the logger into Comma Separated Value (CSV) Ascii files, to enable easy import to point-cloud manipulation software or data analysis tools such as Matlab or Excel.

The converter is typically invoked with two parameters, the first is the input file for conversion, while the second is the name for the output CSV file. There are two optional command line arguments, *-generate*, and *-convert*, where;

- *-generate* creates a default config file for this app
- *-convert* indicates that a binary file in range, azimuth, elevation format should be converted to x, y, z format

The config file (*RE02BinaryToCSV_Settings.xml*) is used to specify values for use in filtering the data on range and/or intensity. If the file does not exist, default settings (no filtering) will be used. The config file must be in the same directory as the application.

A typical invocation would be similar to;

```
RE02BinaryToCSV RE02Data.bin outputXYZ.csv -convert
```

This program will terminate once conversion and filtering is complete.

7.4 Typical Usage

Typical usage of the application described above would consist of the following steps;

1. Run *RE02ConsoleInterface* as described above. Configure the RE02 device as desired (usually scan mode and settings and laser sample rate)
2. Run *RE02Logger* as described above. Recording in range, azimuth, elevation, intensity format is the most flexible if post-processing of the data is desired
3. Run *RE02BinaryToCSV* as described above to create a CSV file (usually in XYZ format)
4. Import the resulting CSV file into your favourite point-cloud manipulation software (e.g. *Pointools* or *VRMesh*)

Chapter 8

Namespace Documentation

8.1 ocular Namespace Reference

the Ocular Robotics Pty Ltd default namespace.

Classes

- class [RE02NetworkInterface](#)
The [RE02NetworkInterface](#) class.
- class [RE02NetworkInterfaceCallback](#)
The [RE02NetworkInterfaceCallback](#) class.

8.1.1 Detailed Description

the Ocular Robotics Pty Ltd default namespace. All C++ software from ocular robotics is within the ocular namespace. In your implementation you may choose to add the

```
using namespace ocular;
```

command to access the [RE02NetworkInterface](#) class directly

Chapter 9

Class Documentation

9.1 ocular::RE02NetworkInterface Class Reference

The [RE02NetworkInterface](#) class.

```
#include <RE02NetworkInterface.h>
```

Public Member Functions

- [RE02NetworkInterface](#) ([RE02NetworkInterfaceCallback](#) *callback)
The default [RE02NetworkInterface](#) constructor.
- [~RE02NetworkInterface](#) ()
The default [RE02NetworkInterface](#) destructor.
- void [Start](#) (void)
This function starts the [RE02NetworkInterface](#) service.
- void [Stop](#) (void)
This function stops the [RE02NetworkInterface](#) service.
- void [SetDestinationIPAddress](#) (std::string &ipAddress)
This function sets the IP Address of the RE02 to be communicated with.
- std::vector< std::string > [GetRE02Addresses](#) (const unsigned int timeout)
This function returns a list of RE02 devices broadcasting on the current subnet.
- std::string [GetLibraryVersion](#) (void)
Get the current library version.
- void [ChangeToDHCP](#) (void)
Command the connected RE02 to obtain its IP address from a DHCP server.
- void [SetStaticIPAddress](#) (std::string &ipAddress, std::string &netMask, std::string &gateway)
Command the connected RE02 to set its IP address settings to those supplied with this function.
- void [SetToUnicast](#) (void)
Command the connected RE02 to exit the default broadcast mode and enter a unicast mode for all subsequent transmissions.
- void [SetToBroadcast](#) (void)
Command the connected RE02 to exit its unicast mode and enter the default broadcast mode for all subsequent transmissions.
- void [SetFullFieldScanSettings](#) (float azimuthRate, float elevationRate)
Method used to put the RE02 into its velocity mode.

- void [SetBoundedElevationScanSettings](#) (float azimuthRate, float lowerElevation, float upperElevation, float lineSpacing)
Method used to set up spiral trajectories in aperture space.
- void [SetRegionScanSettings](#) (float azimuthRate, float initialAzimuth, float initialElevation, float deltaAzimuth, float deltaElevation, float lineSpacing)
Method used to set up and execute raster-like trajectories in aperture space.
- void [SetSampleFrequencySettings](#) (unsigned int sampleFrequency, float maxRange=16.51)
Method used to set the desired sample period and maximum range for the laser sensor.

9.1.1 Detailed Description

The [RE02NetworkInterface](#) class.

This class contains all of the public programmatic interfaces intended to allow a software developer to both monitor and control an RE02 device over a network. It is implemented using a 'service' type model - once the class has been successfully instantiated, work does not begin until the [Start\(\)](#) function is called. After that point, the associated [RE02NetworkInterfaceCallback](#) is executed asynchronously to the main thread of execution. Commands sent to the RE02 can be executed at any time.

9.1.2 Constructor & Destructor Documentation

9.1.2.1 [RE02NetworkInterface::RE02NetworkInterface](#) ([RE02NetworkInterfaceCallback](#) * *callback*)

The default [RE02NetworkInterface](#) constructor.

All resource acquisition is done during construction.

Parameters

<i>callback</i>	- a user defined subclass of RE02NetworkInterfaceCallback , used to receive range data, and informational messages from the RE02
-----------------	--

Exceptions

<i>std::exception</i>	- likely causes; resources could not be allocated, RE02IpAddress is not in a valid format.
-----------------------	--

9.1.2.2 [RE02NetworkInterface::~RE02NetworkInterface](#) ()

The default [RE02NetworkInterface](#) destructor.

Issues [Stop\(\)](#) command, if not executed previously and releases allocated resources.

9.1.3 Member Function Documentation

9.1.3.1 void [RE02NetworkInterface::ChangeToDHCP](#) (void)

Command the connected RE02 to obtain its IP address from a DHCP server.

Note

After this command has executed, the RE02 will reset to its default power-on state of sending broadcast UDP packets.

9.1.3.2 `std::string RE02NetworkInterface::GetLibraryVersion (void)`

Get the current library version.

Useful to check the dll or so version currently in use, particularly if there are multiple copies on a system, and the user is not sure which one is being loaded.

Returns

a string containing the current RE02 network interface library version

9.1.3.3 `std::vector< std::string > RE02NetworkInterface::GetRE02Addresses (const unsigned int timeout)`

This function returns a list of RE02 devices broadcasting on the current subnet.

The timeout parameter is used to determine the amount of time spent *listening* for RE02 data.

Note

This function will only operate correctly after [Start\(\)](#) has been called.

Parameters

<i>timeout</i>	- How long should this function listen for RE02 devices? In milliseconds.
----------------	---

Returns

a vector of strings containing current RE02 IP addresses (on the current subnet). Each string is an IP address in IPv4 *dot* notation i.e. "5.6.7.8"

9.1.3.4 `void RE02NetworkInterface::SetBoundedElevationScanSettings (float azimuthRate, float lowerElevation, float upperElevation, float lineSpacing)`

Method used to set up spiral trajectories in aperture space.

When viewed in azimuth-elevation space, this function will create a spiral from 'lowerElevation' to 'upperElevation' and back again continuously at the speed 'azimuthRate'. The density of the spiral is set by the 'lineSpacing' parameter. Note that it takes approximately 3 revolutions in azimuth of the RE02 to transition into this mode. A subsequent call to this, or another motion function will be responded to within 1/4 of a revolution in azimuth.

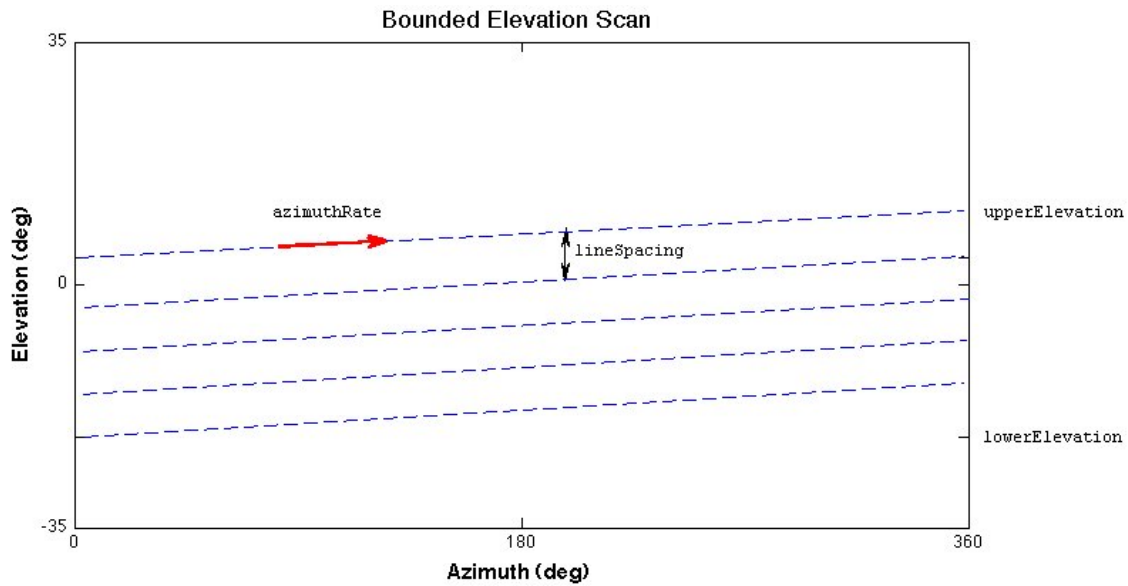


Figure 9.1: The Bounded Elevation Scan trajectory of the RobotEye in Aperture (Azimuth-Elevation) space, showing the relationship to the function's input parameters. Scan direction is bottom to top, and back again.

Parameters

<i>azimuthRate</i>	- in Hz (rotations per second)
<i>lowerElevation</i>	- in degrees: the lower elevation bound of the spiral scan.
<i>upperElevation</i>	- in degrees: the upper elevation bound of the spiral scan.
<i>lineSpacing</i>	- the vertical spacing, in degrees, between the (roughly) horizontal lines in the scan.

Note

Note that any out-of-bounds errors in the settings will result in a message being received on the attached callback.

9.1.3.5 void RE02NetworkInterface::SetDestinationIPAddress (std::string & ipAddress)

This function sets the IP Address of the RE02 to be communicated with.

The address can be changed at any time, before or after [Start\(\)](#) has been called. In most instances, the address will be determined by first calling [GetRE02Addresses\(\)](#), which will list any RE02 devices currently communicating on the current subnet.

Parameters

<i>ipAddress</i>	- the RE02 IP Address in IPv4 <i>dot</i> notation i.e. "1.2.3.4"
------------------	--

9.1.3.6 void RE02NetworkInterface::SetFullFieldScanSettings (float azimuthRate, float elevationRate)

Method used to put the RE02 into its velocity mode.

In this mode, the aperture velocities are specified (in Hz). i.e. a 1.0 azimuthRate will result in 1 rotation per second. An elevationRate of 1.0 will result in the aperture sweeping from its lowest to highest extent in 1 second. This mode is a useful and simple method for creating sinusoidal or spiral sweeps of the aperture across its full extents.

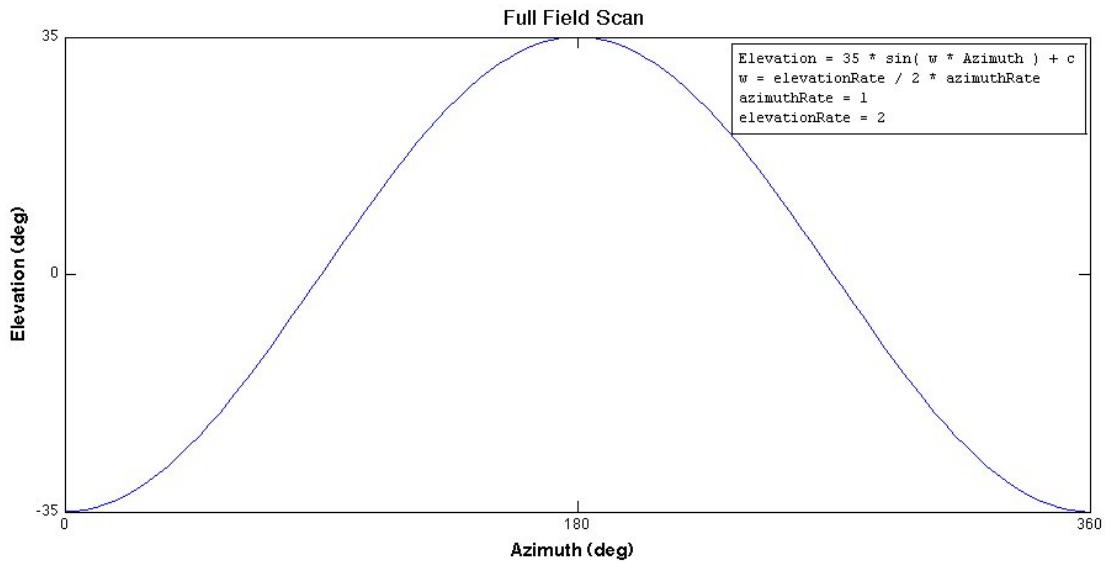


Figure 9.2: The trajectory of the RobotEye in Aperture (Azimuth-Elevation) space resulting from a call to SetFullFieldScanSettings. The trajectory shown is for a 1 second period with the given input parameters. Note that the trajectory could start at any location.

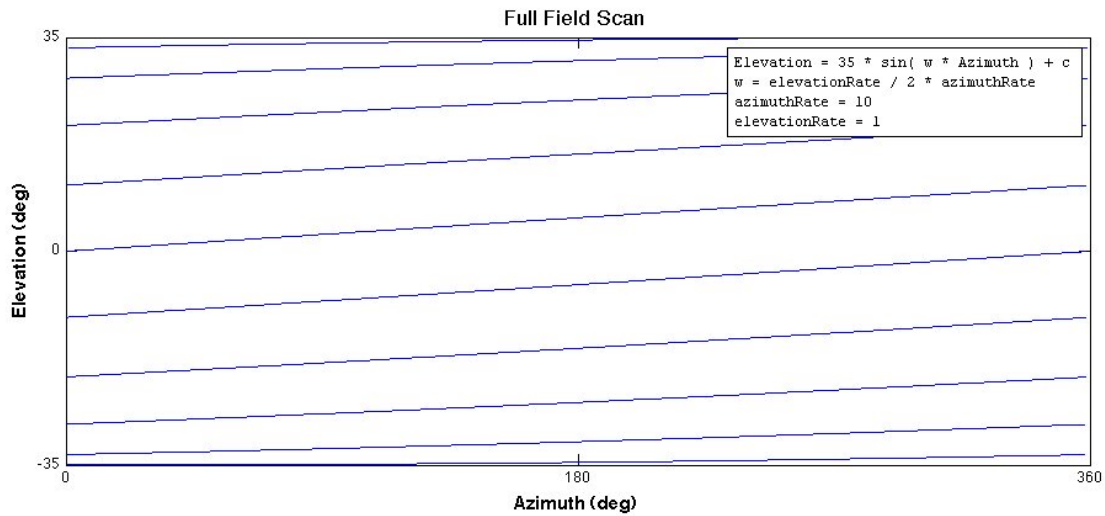


Figure 9.3: Another trajectory of the RobotEye in Aperture (Azimuth-Elevation) space resulting from a call to SetFullFieldScanSettings. The trajectory shown is for a 1 second period with the given input parameters. Note that the trajectory could start at any location.

Parameters

<i>azimuthRate</i>	- in Hz (rotations per second)
<i>elevationRate</i>	- in Hz (vertical sweeps per second)

Note

Note that any out-of-bounds errors in the settings will result in a message being received on the attached callback.

9.1.3.7 void RE02NetworkInterface::SetRegionScanSettings (float *azimuthRate*, float *initialAzimuth*, float *initialElevation*, float *deltaAzimuth*, float *deltaElevation*, float *lineSpacing*)

Method used to set up and execute raster-like trajectories in aperture space.

When viewed in azimuth-elevation space, this function will scan from the top-left corner of a rectangle defined by the initial Azimuth and Elevation (top-left corner) and the delta Azimuth and Elevation parameters (the width and height respectively). The density of the raster is given by the 'numberOfLines' parameter which defines the number of horizontal lines to scan within 'deltaElevation'. A subsequent call to this, or another motion function will be responded to within 3 scan lines.

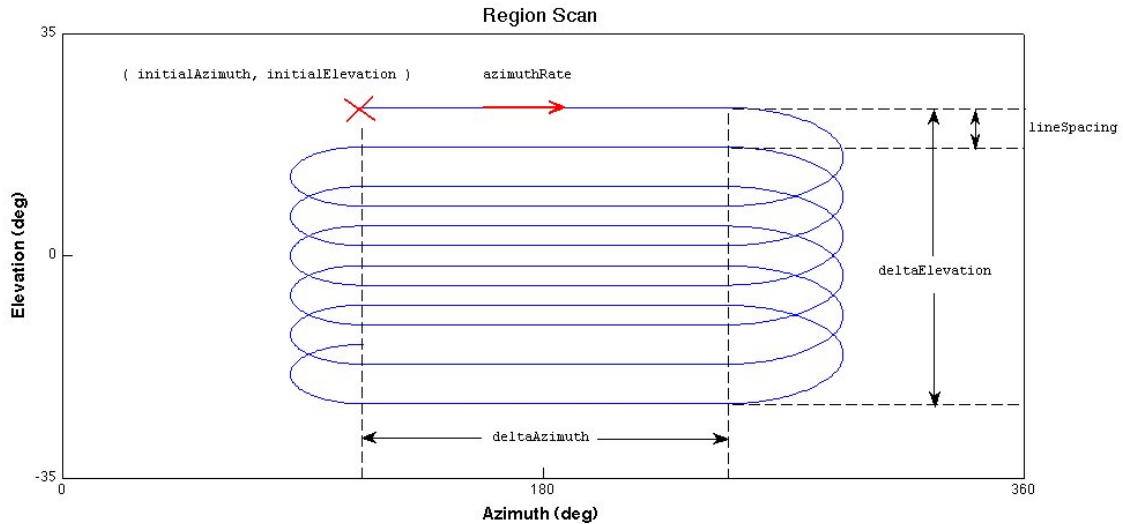


Figure 9.4: The partial trajectory of the RobotEye in Aperture (Azimuth-Elevation) space resulting in a call to SetRegionScanSettings, showing the relationship to the function's input parameters.

Parameters

<i>azimuthRate</i>	- speed in Hz (rotations per second)
<i>initialAzimuth</i>	- in degrees: the Azimuth coordinate of the top-left corner of the scan
<i>initialElevation</i>	- in degrees: the Elevation coordinate of the top-left corner of the scan
<i>deltaAzimuth</i>	- in degrees: the 'width' of the raster-like scan
<i>deltaElevation</i>	- in degrees: the 'height' of the raster-like scan
<i>lineSpacing</i>	- the vertical spacing, in degrees, between the horizontal lines in the scan.

Note

Note that any out-of-bounds errors in the settings will result in a message being received on the attached callback.

9.1.3.8 void RE02NetworkInterface::SetSampleFrequencySettings (unsigned int *sampleFrequency*, float *maxRange* = 16.51)

Method used to set the desired sample period and maximum range for the laser sensor.

The RE02 will attempt to send data at approximately 50Hz regardless of the sample period.

Parameters

<i>sampleFrequency</i>	- in Hz: the desired sample frequency in Hertz (min 50Hz, max 200000Hz)
<i>maxRange</i>	- in meters: the maximum range for the laser sensor (defaults to 16.51m (650in))

Note

Note that any out-of-bounds errors in the settings will result in a message being received on the attached callback.

9.1.3.9 void RE02NetworkInterface::SetStaticIPAddress (std::string & *ipAddress*, std::string & *netMask*, std::string & *gateway*)

Command the connected RE02 to set its IP address settings to those supplied with this function.

Parameters

<i>ipAddress</i>	- the desired IP address in dot notation. i.e. "1.2.3.4"
<i>netMask</i>	- the desired netMask in dot notation. i.e. "255.255.255.0"
<i>gateway</i>	- the default gateway in dot notation. i.e. "1.2.3.1"

Note

After this command has executed, the RE02 will reset to its default power-on state of sending broadcast UDP packets. Also note that any formatting errors in the settings strings will result in a message being received on the attached callback.

9.1.3.10 void RE02NetworkInterface::SetToUnicast (void)

Command the connected RE02 to exit the default broadcast mode and enter a unicast mode for all subsequent transmissions.

Note

The RE02 will send unicast UDP packets to the client that first issues this command

9.1.3.11 void RE02NetworkInterface::Start (void)

This function starts the [RE02NetworkInterface](#) service.

Prior to issuing a call to start, the callback will not receive data. After the call to start, the callback will be executed whenever new data is received on a socket.

9.1.3.12 void RE02NetworkInterface::Stop (void)

This function stops the [RE02NetworkInterface](#) service.

The service can be restarted again with subsequent calls to [Start\(\)](#) if desired.

9.2 ocular::RE02NetworkInterfaceCallback Class Reference

The [RE02NetworkInterfaceCallback](#) class.

```
#include <RE02NetworkInterfaceCallback.h>
```

Public Member Functions

- [RE02NetworkInterfaceCallback](#) ()
The default [RE02NetworkInterfaceCallback](#) constructor.
- virtual [~RE02NetworkInterfaceCallback](#) ()
The default [RE02NetworkInterfaceCallback](#) destructor.
- virtual void [ReceiveRangeData](#) (std::string &data)=0
This virtual method must be overridden by derived user class(es).
- virtual void [ReceiveInfoMessage](#) (std::string &msg)
This virtual method should be overridden by derived user class(es).
- virtual void [ReceiveStatus](#) (std::string &data)
This virtual method should be overridden by derived user class(es).

9.2.1 Detailed Description

The [RE02NetworkInterfaceCallback](#) class.

This class is a pure virtual class that defines the interfaces used to pass data from the RE02 network interface to user code. A user should derive their own classes from this one, and override the pure virtual members.

Note

- this callback is executed in the threadspace of the RE02 library. If user code is multi-threaded, it is the responsibility of the user to ensure their callback is threadsafe. See the logger for an example of this using boost::threads.

9.2.2 Member Function Documentation

9.2.2.1 virtual void `ocular::RE02NetworkInterfaceCallback::ReceiveInfoMessage (std::string & msg)` [`inline, virtual`]

This virtual method should be overridden by derived user class(es).

It is called by the [RE02NetworkInterface](#) object whenever an informational message is received from the RE02 device.

Parameters

<i>msg</i>	a string containing the human readable informational message.
------------	---

Note

The messages received using this callback method will usually be the result of having sent improperly formatted commands to the RE02.

It is not necessary to provide an implementation for this function, however it is *highly* recommended. This information message can provide very useful feedback when developing an application.

9.2.2.2 virtual void `ocular::RE02NetworkInterfaceCallback::ReceiveRangeData (std::string & data)` [`pure virtual`]

This virtual method must be overridden by derived user class(es).

It is called by the [RE02NetworkInterface](#) object whenever new range data is available.

Parameters

<i>data</i>	<p>- raw data bytes of current packet of range data (using std::string as the container). The number of samples in this packet is given by data.size()/13 - where 13 is the number of bytes per sample. The format for a single sample is 'float, float, float, unsigned char' representing 'range (in m), Azimuth (in deg), Elevation (in deg), reflectance (0-255)'. One method to parse the data is as follows</p> <pre> for(unsigned int i=0; i<(data.size()/13); i++) { long index = i*13; float distance = *(float *)(&data[index]); float azimuth = *(float *)(&data[index+4]); float elevation = *(float *)(&data[index+8]); unsigned char amplitude = *(char *)(&data[index+12]); } </pre> <p>The above code assumes that your compiler/OS treats 'float' as 4 bytes and 'char' as 1 byte.</p>
-------------	--

9.2.2.3 virtual void ocular::RE02NetworkInterfaceCallback::ReceiveStatus (std::string & data) [inline, virtual]

This virtual method should be overridden by derived user class(es).

It is called by the [RE02NetworkInterface](#) object whenever an status message is received from the RE02 device.

Parameters

<i>data</i>	<p>An ASCII formatted string containing the current RE02 status. The format of the string is variable, but will be similar to the following: "RS{1.2,3.4,5.6,7.8,9.0,1.2}FR{100000,14.8}DS{ON}SN{12345.678}FV{1.3.78}RV{0.9.88}" where;</p> <ul style="list-style-type: none"> • <i>FF{var1,var2}</i> indicates the RE02 is in Full Field Scan mode with <i>var1</i> AzimuthRate and <i>var2</i> ElevationRate • <i>BE{var1,var2,var3,var4}</i> indicates the RE02 is in Bounded Elevation Scan mode with <i>var1</i> AzimuthRate, <i>var2</i> LineSpacing <i>var3</i> LowerElevation and <i>var4</i> UpperElevation • <i>RS{var1,var2,var3,var4,var5,var6}</i> indicates the RE02 is in Region Scan mode with <i>var1</i> AzimuthRate, <i>var2</i> LineSpacing <i>var3</i> InitialAzimuth, <i>var4</i> InitialElevation, <i>var5</i> DeltaAzimuth and <i>var6</i> DeltaElevation • <i>FR{var1,var2}</i> indicates the current RE02 sample frequency (<i>var1</i>) and maximum range (<i>var2</i>) • <i>DS{var1}</i> indicates the RE02 laser diode status. <i>var1</i> will either be <i>ON</i> or <i>OFF</i> • <i>SN{var1}</i> indicates the RE02 serial number • <i>FV{var1}</i> indicates the RE02 firmware version • <i>RV{var1}</i> indicates the RobotEye library version used by the RE02
-------------	---

Note

This message is sent by the RE02 at 5Hz

The two letter codes are unique - it is therefore relatively straightforward to write a parser for these messages.

It is not necessary to provide an implementation for this function, however it is *highly* recommended. This information message can provide very useful feedback when developing an application.

Index

- ~RE02NetworkInterface
 - ocular::RE02NetworkInterface, 28
- ChangeToDHCP
 - ocular::RE02NetworkInterface, 28
- GetLibraryVersion
 - ocular::RE02NetworkInterface, 28
- GetRE02Addresses
 - ocular::RE02NetworkInterface, 29
- ocular, 25
- ocular::RE02NetworkInterface, 27
 - ~RE02NetworkInterface, 28
 - ChangeToDHCP, 28
 - GetLibraryVersion, 28
 - GetRE02Addresses, 29
 - RE02NetworkInterface, 28
 - SetBoundedElevationScanSettings, 29
 - SetDestinationIPAddress, 30
 - SetFullFieldScanSettings, 30
 - SetRegionScanSettings, 31
 - SetSampleFrequencySettings, 32
 - SetStaticIPAddress, 33
 - SetToUnicast, 33
 - Start, 33
 - Stop, 33
- ocular::RE02NetworkInterfaceCallback, 33
 - ReceiveInfoMessage, 34
 - ReceiveRangeData, 34
 - ReceiveStatus, 35
- RE02NetworkInterface
 - ocular::RE02NetworkInterface, 28
- ReceiveInfoMessage
 - ocular::RE02NetworkInterfaceCallback, 34
- ReceiveRangeData
 - ocular::RE02NetworkInterfaceCallback, 34
- ReceiveStatus
 - ocular::RE02NetworkInterfaceCallback, 35
- SetBoundedElevationScanSettings
 - ocular::RE02NetworkInterface, 29
- SetDestinationIPAddress
 - ocular::RE02NetworkInterface, 30
- SetFullFieldScanSettings
 - ocular::RE02NetworkInterface, 30
- SetRegionScanSettings
 - ocular::RE02NetworkInterface, 31
- SetSampleFrequencySettings
 - ocular::RE02NetworkInterface, 32
- SetStaticIPAddress
 - ocular::RE02NetworkInterface, 33
- SetToUnicast
 - ocular::RE02NetworkInterface, 33
- Start
 - ocular::RE02NetworkInterface, 33
- Stop
 - ocular::RE02NetworkInterface, 33