

Codamotion RTNet SDK for C++ and Java

Table of Contents

1 Overview 1

2 Getting Started (C++) 3

3 Examples 5

4 Symbol Reference 6

4.1 codaRTNet C++ Classes 6

4.1.1 AutoDiscover 6

4.1.2 DataStream 7

4.1.2.1 DataStream::DataStream 7

4.1.2.2 DataStream::DataStream 7

4.1.2.3 DataStream::receivePacket 7

4.1.3 DeviceEnum 8

4.1.4 DeviceInfo 8

4.1.5 DeviceInfoAlignment 9

4.1.5.1 DeviceInfoAlignment::dev 9

4.1.5.2 DeviceInfoAlignment::DeviceInfoAlignment 9

4.1.6 DeviceInfoCodaPacketMode 9

4.1.6.1 DeviceInfoCodaPacketMode::dev 10

4.1.6.2 DeviceInfoCodaPacketMode::DeviceInfoCodaPacketMode 10

4.1.7 DeviceInfoGS16AIOInputs 10

4.1.7.1 DeviceInfoGS16AIOInputs::dev 11

4.1.7.2 DeviceInfoGS16AIOInputs::DeviceInfoGS16AIOInputs 11

4.1.8 DeviceInfoUnitCoordSystem 11

4.1.8.1 DeviceInfoUnitCoordSystem::dev 12

4.1.8.2 DeviceInfoUnitCoordSystem::DeviceInfoUnitCoordSystem 12

4.1.9 DeviceOptions 12

4.1.10 DeviceOptionsAlignment 12

4.1.10.1 DeviceOptionsAlignment::opt 13

4.1.10.2 DeviceOptionsAlignment::DeviceOptionsAlignment 13

4.1.11 DeviceOptionsCodaMode 13

4.1.11.1 DeviceOptionsCodaMode::opt 14

4.1.11.2 DeviceOptionsCodaMode::DeviceOptionsCodaMode 14

4.1.12 DeviceOptionsCodaPacketMode 15

4.1.12.1 DeviceOptionsCodaPacketMode::opt 15

4.1.12.2 DeviceOptionsCodaPacketMode::DeviceOptionsCodaPacketMode 16

4.1.13 DeviceOptionsGS16AIOInputs 16

4.1.13.1 DeviceOptionsGS16AIOInputs::opt 17

4.1.13.2 DeviceOptionsGS16AIOInputs::DeviceOptionsGS16AIOInputs 17

4.1.14 DeviceStatusArray 17

- 4.1.14.1 DeviceStatusArray::DeviceStatusArray 18
- 4.1.15 HWConfigEnum 18
- 4.1.16 NetworkException 18
 - 4.1.16.1 NetworkException::errorcode 19
 - 4.1.16.2 NetworkException::NetworkException 19
- 4.1.17 PacketDecode 19
 - 4.1.17.1 PacketDecode::canDecodePacket 20
 - 4.1.17.2 PacketDecode::clear 21
 - 4.1.17.3 PacketDecode::decode 21
 - 4.1.17.4 PacketDecode::getDeviceID 21
 - 4.1.17.5 PacketDecode::getPage 21
 - 4.1.17.6 PacketDecode::getTick 22
- 4.1.18 PacketDecodeADC16 22
 - 4.1.18.1 PacketDecodeADC16::getNumValues 23
 - 4.1.18.2 PacketDecodeADC16::getValues 23
 - 4.1.18.3 PacketDecodeADC16::PacketDecodeADC16 23
 - 4.1.18.4 PacketDecodeADC16::PacketDecodeADC16 23
- 4.1.19 RTNetClient 23
 - 4.1.19.1 RTNetClient::RTNetClient 24
 - 4.1.19.2 RTNetClient::RTNetClient 25
 - 4.1.19.3 RTNetClient::connect 25
 - 4.1.19.4 RTNetClient::createDataStream 25
 - 4.1.19.5 RTNetClient::disconnect 26
 - 4.1.19.6 RTNetClient::isConnected 26
 - 4.1.19.7 RTNetClient::doAutoDiscoverServer 26
 - 4.1.19.8 RTNetClient::enumerateDevices 26
 - 4.1.19.9 RTNetClient::enumerateHWConfig 26
 - 4.1.19.10 RTNetClient::startSystem 27
 - 4.1.19.11 RTNetClient::stopSystem 27
 - 4.1.19.12 RTNetClient::getDeviceEnable 27
 - 4.1.19.13 RTNetClient::setDeviceOptions 28
 - 4.1.19.14 RTNetClient::getDeviceInfo 28
 - 4.1.19.15 RTNetClient::closeDataStream 28
 - 4.1.19.16 RTNetClient::doSingleShotAcq 29
 - 4.1.19.17 RTNetClient::getAcqMaxTicks 29
 - 4.1.19.18 RTNetClient::prepareForAcq 29
 - 4.1.19.19 RTNetClient::getRunningHWConfig 30
 - 4.1.19.20 RTNetClient::startAcqContinuous 30
 - 4.1.19.21 RTNetClient::getServerVersion 30
 - 4.1.19.22 RTNetClient::startAcqContinuousBuffered 30
 - 4.1.19.23 RTNetClient::stopAcq 31
 - 4.1.19.24 RTNetClient::isAcqInProgress 31

4.1.19.25 RTNetClient::monitorAcqBuffer	31
4.1.19.26 RTNetClient::getAcqBufferNumPackets	32
4.1.19.27 RTNetClient::requestAcqBufferPacket	32
4.1.19.28 RTNetClient::setAcqMaxTicks	32
4.1.19.29 RTNetClient::getDeviceTickSeconds	33
4.1.20 RTNetworkPacket	33
4.1.20.1 RTNetworkPacket::RTNetworkPacket	34
4.1.20.2 RTNetworkPacket::verifyChecksum	34
4.1.21 PacketDecode3DResultExt	34
4.1.21.1 PacketDecode3DResultExt::getIntensity	35
4.1.21.2 PacketDecode3DResultExt::getNumCamerasPerMarker	35
4.1.21.3 PacketDecode3DResultExt::getNumMarkers	35
4.1.21.4 PacketDecode3DResultExt::getNumResidualsPerMarker	36
4.1.21.5 PacketDecode3DResultExt::getPosition	36
4.1.21.6 PacketDecode3DResultExt::getResiduals	36
4.1.21.7 PacketDecode3DResultExt::getValid	36
4.1.21.8 PacketDecode3DResultExt::PacketDecode3DResultExt	37
4.1.21.9 PacketDecode3DResultExt::PacketDecode3DResultExt	37
4.1.22 Version	37
4.1.22.1 Version::Version	37
4.2 Structs	38
4.2.1 CODANET_DEVICEINFO_CODAMODE	38
4.2.2 CODANET_DEVICEINFO_CODAPACKETMODE	38
4.2.3 CODANET_DEVICEINFO_GS16AIO_INPUTS	39
4.2.4 CODANET_DEVICEINFO_UNITCOORDSYSTEM	39
4.2.5 CODANET_DEVICEOPTIONS_ALIGNMENT_EXT	39
4.2.6 CODANET_DEVICEOPTIONS_STROBEOUTPUT	40
4.2.7 CODANET_DATASTREAM_DEST	40
4.2.8 CODANET_DEVICEENUM	40
4.2.9 CODANET_DEVICEINFO_ALIGNMENT	40
4.2.10 CODANET_VERSION	41
4.2.11 CODANET_AUTODISCOVER	41
4.2.12 CODANET_DEVICESTATUS	42
4.2.13 CODANET_DEVICESTATUS_ARRAY	42
4.2.14 CODANET_HWCONFIG_DEVICEENABLE	42
4.3 Data Types	43
4.3.1 BYTE	43
4.3.2 DWORD	43
4.3.3 LPBYTE	44
4.3.4 LPDWORD	44
4.3.5 LPWCHAR	44
4.3.6 LPWORD	44

4.3.7 WORD 44

4.3.8 CODANET_SOCKET 45

4.4 C SDK Functions 45

4.4.1 codanet_sleep 45

4.5 Constants 45

4.5.1 Device ID's 46

4.5.1.1 DEVICEID_CX1 46

4.5.1.2 DEVICEID_GS16AIO 46

4.5.2 Size Constants 46

4.5.2.1 CODANET_DEVICEENUM_MAXITEMS 46

4.5.2.2 CODANET_DEVICESTATUS_MAXITEMS 47

4.5.2.3 CODANET_HWCONFIG_MAXITEMS 47

4.5.2.4 CODANET_HWCONFIG_MAXNAMELENGTH 47

4.5.3 Symbolic Constants 47

4.5.3.1 CODANET_ACQ_UNLIMITED 47

4.5.3.2 CODANET_PACKET_WAIT_INDEFINITELY 48

5 Index 49

Codamotion RTNet SDK for C++ and Java

1 Overview

Introduction

Welcome to the Codamotion RTNet Software Development Kit (SDK) for C++ and Java. This software allows full real-time access to Codamotion 3D measurement systems and supported peripheral devices. To make use of the SDK you must have programming skills in either C++ or Java. For non-programmers, a variety of off-the-shelf software packages are available instead - please see the [Codamotion](#) website for more information.

Hardware System Design and Requirements

A Codamotion system comprises one or more Codamotion movement analysis devices connected to a Codamotion Active Hub or a standard PC (**Host PC**) connected to a Codamotion Mini Hub. The Codamotion RTNet Server must be running on the Active Hub or Host PC. The Codamotion RTNet SDK provides the tools to connect to this server over an ethernet connection. Programs developed using the RTNet SDK are therefore clients to the Codamotion RTNet Server. They can be run either on the Active Hub / Host PC itself, or on another computer connected by ethernet.

Codamotion RTNet Control and Data Flow

This SDK provides tools to control the RTNet server and retrieve data from it. This SDK achieves control of the Codamotion RTNet Server using the TCP/IP protocol over an ethernet connection. This is used to configure the server and to request realtime data. The part of the SDK required to do this is called the **RTNet Client**. Measurement data itself is received as a UDP stream over an ethernet connection. For this to happen the SDK instantiates a UDP socket on the client computer. The socket and its associated code is known as the **RTNet Data Stream**.

The Codamotion RTNet Server has five main operating states:

1. **Shut down** This is the state which occurs when the server first begins and no hardware configuration has been selected, nor any connected devices have been initialised.
2. **Started** This means that a hardware configuration has been chosen and the relevant devices initialised successfully. Measurement data cannot be generated until the system is in the prepared state.
3. **Prepared** The system has been started and necessary preparations made for data acquisition. In the case of Codamotion CX1 systems this means that diode offset calibration has been performed to provide correct input data. In this state data is not continuously streaming but single frames of measurement data can be requested at any time for monitoring purposes.
4. **Continuous Acquisition** The system has been started, prepared, and is streaming data continuously over one or more RTNet Data Streams. None of the measurement data persists on the server once it has been sent across the data streams - it is up to the receiving applications to store it.
5. **Continuous Buffered Acquisition** An alternative to continuous acquisition - the system has been started, prepared, and is acquiring data continuously. But in this mode the data is not automatically sent over any data stream. Instead data is held in buffers on the server for post-acquisition retrieval. The latest frame in the buffer can be requested by the client at any time to allow real-time monitoring.

The RTNet Client has the necessary control features to change from one state to another, and to request measurement data over an RTNet Data Stream as required.

Codamotion RTNet Devices

The Codamotion system supports a number of peripheral devices and accessories. To provide the flexibility to control and access these different physical devices, the RTNet server comprises one or more software 'devices'. Each type of software device has a unique numeric identifier, known as the **device ID**.

Changes between the different modes described above are applied to all devices simultaneously, but other RTNet client functions may control or request data from a particular specified device.

The device which is common to most configurations is the Codamotion CX1 system device for 3D movement data, though it is possible to have configurations which use only an analog interface such as the GS16AIO and no movement system.

Programming Languages

To access the RTNet Client and RTNet Data Stream features described above, this SDK provides applications programming interfaces (API's) in both C++ and Java. Where possible the same naming conventions have been used in both.

This guide provides a Getting Started (▣ see page 3) section for the C++ API, and a reference guide for all C++ SDK Classes (▣ see page 6). For the equivalent Java API, please see the HTML Java Documentation usually located in <docs/help/javadocs/index.html>.

Some of the underlying RTNet client code is written in C rather than C++, and some of these function calls are exposed in the header files. Please note, however, that Charnwood Dynamics will only respond to requests for technical assistance in relation to the documented C++ or Java API's and not the underlying C API.

2 Getting Started (C++)

Background

The SDK was designed for use with Microsoft Visual Studio 6 or Microsoft Visual C++ 7 (Microsoft Visual Studio .NET 2003). The core functions are included in a dynamic-link library, `bin\codARTNetProtocol.dll`. A static C++ library provides C++ wrapper classes to access this DLL.

For use with other compilers, see the section below on Rebuilding the C++ Library. For use with Java, see the Java documentation (javadoc format) in <docs/help/javadocs/index.html>

Creating a New Project

Use visual studio to create a new C++ project. For your first program it is usual to create a console application so that you can use the standard library functions (`printf` etc.) to print diagnostic output. Ensure that the Runtime Library is set to Multithreaded (or Multithreaded Debug), and that Runtime Type Information is enabled.

Ensure that the RTNet `lib` folder is on the library path and the `include` folder is on the include path.

Your project must link to the core protocol library `codARTNetProtocol.lib` which corresponds to the RTNet DLL file. It must also link to the correct version of the C++ static library. The different versions available in the `lib` directory are given below:

Filename	Visual Studio Version	Runtime Library
<code>codARTNetProtocolCPP_VC7SR.lib</code>	.NET 2003	Multi-threaded (/MT)
<code>codARTNetProtocolCPP_VC7SD.lib</code>	.NET 2003	Multi-threaded Debug (/MTd)
<code>codARTNetProtocolCPP_VS6SR.lib</code>	6	Multithreaded (/MT)
<code>codARTNetProtocolCPP_VS6SD.lib</code>	6	Debug Multithreaded (/MTd)

If you wish to use a different compiler or a different version of the C Runtime library, you will need to build your own version of the `codARTNetProtocolCPP` library. See the section on Rebuilding the C++ library below.

Writing a Program to Use the SDK

A typical program contains the following steps:

1. Use the `RTNetClient::doAutoDiscoverServer` (see page 26) feature to find available RTNet servers on the network. Usually there is just one server available, but if more than one you may wish to give the user a choice of which to connect to.
2. Use `RTNetClient::connect` (see page 25) to a server found in the first step.
3. (Optional) Use `RTNetClient::stopSystem` (see page 27) and `RTNetClient::startSystem` (see page 27) to stop and restart the system with a specific hardware configuration.
4. Prepare the system for acquisition using `RTNetClient::prepareForAcq` (see page 29)
5. Create one or more data streams using `RTNetClient::createDataStream` (see page 25) and a `DataStream` (see page 7) object.
6. Ask the server to acquire data using one or more of the following `RTNetClient` methods: `doSingleShotAcq` (see page 29), `startAcqContinuous` (see page 30), `startAcqContinuousBuffered` (see page 30), `monitorAcqBuffer` (see page 31).
7. Receive data on the `DataStream` (see page 7) object using `RTNetworkPacket` (see page 33) objects, and decode the data using a class derived from `PacketDecode` (see page 19) (such as `PacketDecode3DResultExt` (see page 34) or `PacketDecodeADC16` (see page 22)) ; then display or store the data in some way.

The example program <examples/runtext/runtext.cpp> shows each of these steps.

Rebuilding the C++ Library

You will need to perform your own build of the C++ library (codaRTNetProtocolCPP) if you wish to use an alternative compiler or a C Runtime Library which is not listed in the table above. Source code is included in the `source` directory.

To perform your own build, simply add all the C++ files (*.cpp) from `source/codaRTNetProtocolCPP` to your project or Makefile. Header (*.h) files from the `source/codaRTNetProtocol` and `source/codaRTNetProtocolCPP` folders are referenced in those C++ (*.cpp) files. Example project files for Visual Studio 6 and Visual Studio .NET 2003 are included.

3 Examples

C++ Examples

Example File	Description
examples/runtext/runtext.cpp	Connects to first available server and, if system is not already running, starts system using first available hardware configuration. Uses buffered acquisition mode and monitor function for low-latency realtime monitoring. Prints realtime 3D and analog data to the screen and continues acquiring until stopped by the user.
examples/rtnetdemo	Allows most common RT Net commands and parameters to be read from a command file and executed, and writes all received packet data in a structured text file format.

Java Examples

Example File	Description
examples/java/RTNetDataPrinter.java	Connects to first available server and, if system is not already running, starts system using first available hardware configuration. Uses buffered acquisition mode and monitor function for low-latency realtime monitoring. Prints realtime 3D and analog data to the screen at 20 x 100ms intervals, then stops acquisition. After acquisition has been stoppped, marker and analog buffered data is downloaded and printed to standard output as tab-delimited tables.

4 Symbol Reference

4.1 codaRTNet C++ Classes

Classes

Class	Description
AutoDiscover (see page 6)	An array of available Codamotion RTNet servers as found by RTNetClient::doAutoDiscoverServer (see page 26)
DataStream (see page 7)	Represents a data stream for receiving measurement data from RTNet Server (a UDP socket)
DeviceEnum (see page 8)	An array of available system devices as found by RTNetClient::enumerateDevices (see page 26).
DeviceInfo (see page 8)	Base class for retrieving runtime information about an enabled device.
DeviceInfoAlignment (see page 9)	Use with RTNetClient::getDeviceInfo (see page 28) to retrieve diagnostic information about most recent Codamotion CX1 system alignment.
DeviceInfoCodaPacketMode (see page 9)	Use with RTNetClient::getDeviceInfo (see page 28) to retrieve current packet settings for CODA data stream.
DeviceInfoGS16AIOInputs (see page 10)	Use with RTNetClient::getDeviceInfo (see page 28) to retrieve information about GS16AIO analog interface.
DeviceInfoUnitCoordSystem (see page 11)	Use with RTNetClient::getDeviceInfo (see page 28) to retrieve current CX1 unit coordinate systems.
DeviceOptions (see page 12)	Base class for setting runtime device parameters or performing runtime setup operation on an enabled device.
DeviceOptionsAlignment (see page 12)	Use as argument to RTNetClient::setDeviceOptions (see page 28) to perform a CX1 system alignment operation
DeviceOptionsCodaMode (see page 13)	Use as argument to RTNetClient::setDeviceOptions (see page 28) to set Coda mode information (CX1 sample rates and decimation).
DeviceOptionsCodaPacketMode (see page 15)	Use as argument to RTNetClient::setDeviceOptions (see page 28) to set options on which packets are transmitted from the CX1 system.
DeviceOptionsGS16AIOInputs (see page 16)	Use as argument to RTNetClient::setDeviceOptions (see page 28) to set GS16AIO analog input options (sample rates and decimation).
DeviceStatusArray (see page 17)	Provides error state information for multiple RTNet devices
HWConfigEnum (see page 18)	Array of all available hardware configurations as returned by RTNetClient::enumerateHWConfig (see page 26).
NetworkException (see page 18)	Contains error code for network connection errors
PacketDecode (see page 19)	Abstract base class for decoding packet information received on a data stream.
PacketDecodeADC16 (see page 22)	Decodes 16-bit measurement data received from a data stream.
RTNetClient (see page 23)	Provides the main programming interface for communication with Codamotion RTNet Server.
RTNetworkPacket (see page 33)	Holds the data from a single packet on a data stream
PacketDecode3DResultExt (see page 34)	Decodes 3D measurement packets received on a data stream.
Version (see page 37)	Version information structure retrieved from RTNet server

4.1.1 AutoDiscover

An array of available Codamotion RTNet servers as found by RTNetClient::doAutoDiscoverServer (see page 26)

Class Hierarchy

```
CODANET_AUTODISCOVER
  AutoDiscover
```

```
class AutoDiscover : public CODANET_AUTODISCOVER;
```

File

AutoDiscover.h

Description

For details see the underlying C-style structure, CODANET_AUTODISCOVER (see page 41).

4.1.2 DataStream

Represents a data stream for receiving measurement data from RTNet Server (a UDP socket)

Class Hierarchy

DataStream

```
class DataStream;
```

File




DataStream.h

Description

To create a data stream use `RTNetClient::createDataStream` (see page 25). Packets can then be requested using the `RTNetClient` (see page 23) methods (`doSingleShotAcq`, `monitorAcqBuffer`, `startAcqContinuous`, `requestAcqBufferPacket`). To receive requested packets on the data stream use the `receivePacket` (see page 7) method.

Members

Methods

Method	Description
 <code>DataStream</code> (see page 7)	Construct a data stream object in a disconnected state
 <code>DataStream</code> (see page 7)	Construct a data stream object from an existing UDP socket
 <code>receivePacket</code> (see page 7)	Receive network packets on this data stream

Legend

	Method
	protected

4.1.2.1 DataStream::DataStream

Construct a data stream object in a disconnected state

```
DataStream();
```

4.1.2.2 DataStream::DataStream

Construct a data stream object from an existing UDP socket

```
DataStream(CODANET_SOCKET nskt);
```

Description

This function is documented for completeness and is not usually called directly. To create a connected data stream first create an unconnected one using the `DataStream::DataStream` (see page 7)() constructor, and then connect it using `RTNetClient::createDataStream` (see page 25).

@params nskt existing UDP socket

4.1.2.3 DataStream::receivePacket

Receive network packets on this data stream

```
int receivePacket(RTNetworkPacket& packet, DWORD timeoutusec);
```

Parameters

Parameters	Description
RTNetworkPacket& packet	Reference to an RTNetworkPacket (see page 33) object which will be filled with new data on successful completion.
DWORD timeoutusec	Maximum time to block whilst waiting for a new data packet (microseconds), or CODANET_PACKET_WAIT_INDEFINITELY (see page 48) to wait indefinitely.

Return Values

Return Values	Description
CODANET_OK	Packet was received ok .
CODANET_STREAMTIMEOUT	timeoutusec was reached before any data packets were received.
CODANET_SOCKETERROR_BROKEN	Stream connection was broken.

Description

Packets will arrive in the data stream after they have been requested using one of the RTNetClient (see page 23) request methods (doSingleShotAcq, monitorAcqBuffer, startAcqContinuous, requestAcqBufferPacket).

Packets received on the stream and not yet retrieved using this function are buffered up to the system default UDP receive-buffer limit. On windows XP systems that default limit is 8192 bytes. Once this buffer is full, any additional packets received will be lost.

If there are no packets available when this function is called, it will block whilst waiting for new data, for up to the maximum number of microseconds specified in the timeoutusec parameter. To block indefinitely use the symbolic constant CODANET_PACKET_WAIT_INDEFINITELY (see page 48).

4.1.3 DeviceEnum

An array of available system devices as found by RTNetClient::enumerateDevices (see page 26).

Class Hierarchy

CODANET_DEVICEENUM
DeviceEnum

```
class DeviceEnum : public CODANET_DEVICEENUM;
```

File

DeviceEnum.h

Description

For details see the underlying C-style structure, CODANET_DEVICEENUM (see page 40).

4.1.4 DeviceInfo

Base class for retrieving runtime information about an enabled device.

Class Hierarchy

DeviceInfo
class DeviceInfo;

File

DeviceInfo.h

Description

Should not be used directly - see derived classes instead.

4.1.5 DeviceInfoAlignment

Use with `RTNetClient::getDeviceInfo` (see page 28) to retrieve diagnostic information about most recent Codamotion CX1 system alignment.

Class Hierarchy

`codaRTNet::DeviceInfo`
`DeviceInfoAlignment`

```
class DeviceInfoAlignment : public DeviceInfo;
```

File

`DeviceInfoAlignment.h`

Description


On successful completion, the `dev` (see page 9) member will be filled with information about the most recent system alignment. To perform system alignments, use `RTNetClient::setDeviceOptions` (see page 28) with a `DeviceOptionsAlignment` (see page 12) class.

Members

Data Members

Data Member	Description
 <code>dev</code> (see page 9)	Device information retrieved

Methods

Method	Description
 <code>DeviceInfoAlignment</code> (see page 9)	Construct alignment request ready to be filled by call to <code>RTNetClient::getDeviceInfo</code> (see page 28)

Legend

	Data Member
	Method

4.1.5.1 DeviceInfoAlignment::dev

```
CODANET_DEVICEINFO_ALIGNMENT dev;
```

Description

Device information retrieved

4.1.5.2 DeviceInfoAlignment::DeviceInfoAlignment

```
DeviceInfoAlignment();
```

Description

Construct alignment request ready to be filled by call to `RTNetClient::getDeviceInfo` (see page 28)

4.1.6 DeviceInfoCodaPacketMode

Use with `RTNetClient::getDeviceInfo` (see page 28) to retrieve current packet settings for CODA data stream.

Class Hierarchy

`codaRTNet::DeviceInfo`

DeviceInfoCodaPacketMode

```
class DeviceInfoCodaPacketMode : public DeviceInfo;
```

File

DeviceInfoCodaPacketMode.h


Description

On successful completion, the dev (see page 10) member will be filled with information about the current packet settings for CODA data stream. To set these, use RTNetClient::setDeviceOptions (see page 28) with a DeviceOptionsCodaPacketMode (see page 15) class.

Members**Data Members**

Data Member	Description
 dev (see page 10)	Device information retrieved

Methods

Method	Description
 DeviceInfoCodaPacketMode (see page 10)	Construct alignment request ready to be filled by call to RTNetClient::getDeviceInfo (see page 28)

Legend

	Data Member
	Method

4.1.6.1 DeviceInfoCodaPacketMode::dev

```
CODANET_DEVICEINFO_CODAPACKETMODE dev;
```

Description

Device information retrieved

4.1.6.2 DeviceInfoCodaPacketMode::DeviceInfoCodaPacketMode

```
DeviceInfoCodaPacketMode();
```

Description

Construct alignment request ready to be filled by call to RTNetClient::getDeviceInfo (see page 28)

4.1.7 DeviceInfoGS16AIOInputs

Use with RTNetClient::getDeviceInfo (see page 28) to retrieve information about GS16AIO analog interface.

Class Hierarchy

```
codaRTNet::DeviceInfo
  DeviceInfoGS16AIOInputs
```

```
class DeviceInfoGS16AIOInputs : public DeviceInfo;
```

File

DeviceInfoGS16AIOInputs.h


Description

On successful completion of RTNetClient::getDeviceInfo (see page 28), the dev (see page 11) member will be filled with information about the GS16AIO device.

Members**Data Members**

Data Member	Description
 dev (see page 11)	Device information retrieved

Methods

Method	Description
 DeviceInfoGS16AIOInputs (see page 11)	Construct GS16AIO information request ready to be filled by call to RTNetClient::getDeviceInfo (see page 28)

Legend

	Data Member
	Method

4.1.7.1 DeviceInfoGS16AIOInputs::dev

```
CODANET_DEVICEINFO_GS16AIO_INPUTS dev;
```

Description

Device information retrieved

4.1.7.2 DeviceInfoGS16AIOInputs::DeviceInfoGS16AIOInputs

Construct GS16AIO information request ready to be filled by call to RTNetClient::getDeviceInfo ([see page 28](#))

```
DeviceInfoGS16AIOInputs();
```

4.1.8 DeviceInfoUnitCoordSystem

Use with RTNetClient::getDeviceInfo ([see page 28](#)) to retrieve current CX1 unit coordinate systems.

Class Hierarchy

```
codaRTNet::DeviceInfo
  DeviceInfoUnitCoordSystem
```

```
class DeviceInfoUnitCoordSystem : public DeviceInfo;
```

File

DeviceInfoUnitCoordSystem.h


Description

On successful completion, the dev ([see page 12](#)) member will be filled with coordinate system information, representing the transform from local unit coordinates to global room coordinates. This is the information obtained during alignment.

Members**Data Members**

Data Member	Description
 dev (see page 12)	Device information retrieved

Methods

Method	Description
 DeviceInfoUnitCoordSystem (see page 12)	Construct alignment request ready to be filled by call to RTNetClient::getDeviceInfo (see page 28)

Legend

	Data Member
	Method

4.1.8.1 DeviceInfoUnitCoordSystem::dev

```
CODANET_DEVICEINFO_UNITCOORDSYSTEM dev;
```

Description

Device information retrieved

4.1.8.2 DeviceInfoUnitCoordSystem::DeviceInfoUnitCoordSystem

```
DeviceInfoUnitCoordSystem();
```

Description

Construct alignment request ready to be filled by call to RTNetClient::getDeviceInfo (see page 28)

4.1.9 DeviceOptions

Base class for setting runtime device parameters or performing runtime setup operation on an enabled device.

Class Hierarchy

DeviceOptions

```
class DeviceOptions;
```

File

DeviceOptions.h

Description

Should not be used directly - see derived classes instead.

4.1.10 DeviceOptionsAlignment

Use as argument to RTNetClient::setDeviceOptions (see page 28) to perform a CX1 system alignment operation

Class Hierarchy

codaRTNet::DeviceOptions
DeviceOptionsAlignment

```
class DeviceOptionsAlignment : public DeviceOptions;
```


File

DeviceOptionsAlignment.h


Description

The constructor allows a set of alignment marker identities to be specified. Once constructed, use RTNetClient::setDeviceOptions (see page 28) to perform the alignment operation. On completion, diagnostic information about the alignment can be retrieved using RTNetClient::getDeviceInfo (see page 28) with a DeviceInfoAlignment (see page 9) structure as the argument.

Members**Data Members**

Data Member	Description
 opt (see page 13)	Underlying alignment options request

Methods

Method	Description
 DeviceOptionsAlignment (see page 13)	Construct an alignment request with the specified marker identities

Legend

	Data Member
	Method

4.1.10.1 DeviceOptionsAlignment::opt

Underlying alignment options request

```
CODANET_DEVICEOPTIONS_ALIGNMENT opt;
```

Description

This is usually set via the DeviceOptionsAlignment::DeviceOptionsAlignment (see page 12) constructor rather than manipulated directly.

4.1.10.2 DeviceOptionsAlignment::DeviceOptionsAlignment

Construct an alignment request with the specified marker identities

```
DeviceOptionsAlignment(DWORD dwMarkerOrigin, DWORD dwMarkerX0, DWORD dwMarkerX1, DWORD dwMarkerXY0, DWORD dwMarkerXY1);
```

Parameters

Parameters	Description
DWORD dwMarkerOrigin	One-based identity of origin marker.
DWORD dwMarkerX0	One-based identity of marker defining start of X-axis vector.
DWORD dwMarkerX1	One-based identity of marker defining start of X-axis vector.
DWORD dwMarkerXY0	One-based identity of marker on axis in XY-plane.
DWORD dwMarkerXY1	One-based identity of marker on axis in XY-plane (can be same as origin but must differ from XY0)

Description

Marker identities are one-based. Following construction with an appropriate set of identities, use with RTNetClient::setDeviceOptions (see page 28) to perform the alignment procedure.

To perform an alignment, markers with know identities must be placed in an approximate right-angle configuration in the field of view. A minimum of three markers are required. One marker defines the origin, two markers define an X-axis, and another two markers define an XY-axis. The X-axis vector must be sufficiently perpendicular to the XY-plane vector (at least 60 degrees included angle), and the pairs of markers defining the vectors must be separated by sufficient distance (greater than 300mm).

To use the minimum number of three markers, the same marker can be used for the origin as for the X0 and XY0 points. One further marker is then required for the X1 point and another for the XY1 point (see parameters below).

4.1.11 DeviceOptionsCodaMode

Use as argument to RTNetClient::setDeviceOptions (see page 28) to set Coda mode information (CX1 sample rates and

decimation).

Class Hierarchy

codaRTNet::DeviceOptions
DeviceOptionsCodaMode

```
class DeviceOptionsCodaMode : public DeviceOptions;
```

File

DeviceOptionsCodaMode.h

Description

The constructor allows mode information to be specified. Once constructed, use RTNetClient::setDeviceOptions (see page 28) to set the new mode.

Members

Data Members

Data Member	Description
 opt (see page 14)	Underlying CX1 mode options request

Methods

Method	Description
 DeviceOptionsCodaMode (see page 14)	Construct specified mode options ready to be set using RTNetClient::setDeviceOptions (see page 28)

Legend

	Data Member
	Method

4.1.11.1 DeviceOptionsCodaMode::opt

Underlying CX1 mode options request

```
CODANET_DEVICEOPTIONS_CODAMODE opt;
```

Description

This is usually set via the DeviceOptionsCodaMode::DeviceOptionsCodaMode (see page 13) constructor rather than manipulated directly.

4.1.11.2 DeviceOptionsCodaMode::DeviceOptionsCodaMode

Construct specified mode options ready to be set using RTNetClient::setDeviceOptions (see page 28)

```
DeviceOptionsCodaMode(DWORD dwRateMode, DWORD dwDecimation, DWORD dwExternalSync);
```

Parameters

Parameters	Description
DWORD dwRateMode	The sample rate mode (see table above)
DWORD dwDecimation	Decimation factor to divide sample rate down and reduce system load.
DWORD dwExternalSync	Non-zero to enable external sync, zero otherwise.

Description

Set specified mode information. In Codamotion systems the mode determines not only the sample rate but also the maximum number of markers. Possible values for the dwRateMode parameter are as follows:

dwRateMode	Sample Rate	Number of Markers
CODANET_CODA_MODE_100	100Hz	56
CODANET_CODA_MODE_200	200Hz	28

CODANET_CODA_MODE_400	400Hz	12
CODANET_CODA_MODE_800	800Hz	6

The dwDecimation value reduces the bandwidth and system load by measuring only every *N*th sample period at the given rate. For example a dwRateMode of CODANET_CODA_MODE_200 with dwDecimation of 4 would give an actual sample rate of 200Hz / 4 = 50Hz. Most Codamotion systems use battery-powered drive boxes and using lower sample rates where possible will preserve battery life.

Systems can be configured to accept external synchronisation signals. In this case the dwRateMode should indicate a sample rate which is not less than the input signal rate.

4.1.12 DeviceOptionsCodaPacketMode

Use as argument to RTNetClient::setDeviceOptions (see page 28) to set options on which packets are transmitted from the CX1 system.

Class Hierarchy

```
codaRTNet::DeviceOptions
  DeviceOptionsCodaPacketMode
```

```
class DeviceOptionsCodaPacketMode : public DeviceOptions;
```

File

DeviceOptionsCodaPacketMode.h

Description


The constructor allows mode information to be specified. Once constructed, use RTNetClient::setDeviceOptions (see page 28) to set the new mode.

Members

Data Members

Data Member	Description
 opt (see page 15)	Underlying CX1 mode options request

Methods

Method	Description
 DeviceOptionsCodaPacketMode (see page 16)	Construct specified mode options ready to be set using RTNetClient::setDeviceOptions (see page 28)

Legend

	Data Member
	Method

4.1.12.1 DeviceOptionsCodaPacketMode::opt

Underlying CX1 mode options request

```
CODANET_DEVICEOPTIONS_CODAPACKETMODE opt;
```

Description

This is usually set via the DeviceOptionsCodaMode::DeviceOptionsCX1DataMode constructor rather than manipulated directly.

4.1.12.2

DeviceOptionsCodaPacketMode::DeviceOptionsCodaPacketMode

Construct specified mode options ready to be set using RTNetClient::setDeviceOptions (see page 28)

```
DeviceOptionsCodaPacketMode(DWORD dwPacketMode);
```

Parameters

Parameters	Description
DWORD dwPacketMode	The data mode (see table above)

Description

Set specified data streaming mode information. Possible values for the dwPacketMode parameter are as follows:

dwRateMode	Description
CODANET_CODAPACKETMODE_COMBINED_COORD	Single packet of 3D coords combined from all units
CODANET_CODAPACKETMODE_SEPARATE_COORD	Separate packets of 3D coords from each unit
CODANET_CODAPACKETMODE_SEPARATE_AND_COMBINED_COORD	One packet of combined 3D coords and separate packets of 3D coords from each unit

4.1.13 DeviceOptionsGS16AIOInputs

Use as argument to RTNetClient::setDeviceOptions (see page 28) to set GS16AIO analog input options (sample rates and decimation).

Class Hierarchy

```
codaRTNet::DeviceOptions
  DeviceOptionsGS16AIOInputs
```

```
class DeviceOptionsGS16AIOInputs : public DeviceOptions;
```

File


DeviceOptionsGS16AIOInputs.h

Description


The constructor allows sample rate and decimation information to be specified. Once constructed, use RTNetClient::setDeviceOptions (see page 28) to set the new rates.

Members

Data Members

Data Member	Description
 opt (see page 17)	Underlying device options structure

Methods

Method	Description
 DeviceOptionsGS16AIOInputs (see page 17)	Construct with the specified sample rate on all channels.

Legend

	Data Member
	Method

4.1.13.1 DeviceOptionsGS16AIOInputs::opt

```
CODANET_DEVICEOPTIONS_GS16AIO_INPUTS opt;
```

Description

Underlying device options structure

4.1.13.2

DeviceOptionsGS16AIOInputs::DeviceOptionsGS16AIOInputs

Construct with the specified sample rate on all channels.

```
DeviceOptionsGS16AIOInputs(float fBaseRateHz);
```

Description

Constructed like this, all GS16AIO channels will have the same sample rate once this object is passed to RTNetClient::setDeviceOptions (see page 28). To apply lower sample rates for certain channels, a decimation factor can be applied. Use the opt.channel_decimation member to set individual channel decimation factors relative to the base rate.

The actual base sample rate used will be the nearest whole divisor of the GS16AIO hardware clock rate which is 24MHz.

The GS16AIO device sends measurement data as ADC16 packets for receipt by the DataStream::receivePacket (see page 7) method and decoding by PacketDecodeADC16 (see page 22). When decimation is applied, a decimated channel will be absent from all packets whose tick value is not a whole multiple of the decimation value. The received packet tick number must be checked to determine which channels are present.

@params fBaseRateHz The base sample rate for all channels, in Hertz.

4.1.14 DeviceStatusArray

Provides error state information for multiple RTNet devices

Class Hierarchy

```
CODANET_DEVICESTATUS_ARRAY
DeviceStatusArray
```

```
class DeviceStatusArray : public CODANET_DEVICESTATUS_ARRAY;
```

File


DeviceStatusArray.h

Description

Objects of this class are thrown as exceptions from the RTNetClient (see page 23) class if a server-side error is generated during a function call. It is possible for a single interaction with the server to generate more than one error (for example the startSystem command may fail on multiple devices). This class therefore provides for an array of error information. See the underlying CODANET_DEVICESTATUS_ARRAY (see page 42) structure for details.

Members

Methods

Method	Description
 DeviceStatusArray (see page 18)	Construct an empty device status array (indicating no errors)

Legend

	Method
---	--------

4.1.14.1 DeviceStatusArray::DeviceStatusArray

Construct an empty device status array (indicating no errors)

```
DeviceStatusArray();
```

4.1.15 HWConfigEnum

Array of all available hardware configurations as returned by RTNetClient::enumerateHWConfig (see page 26).

Class Hierarchy

CODANET_HWCONFIGENUM
HWConfigEnum

```
class HWConfigEnum : public CODANET_HWCONFIGENUM;
```

File

HWConfigEnum.h

Description

See the underlying CODANET_HWCONFIGENUM structure for details.

4.1.16 NetworkException

Contains error code for network connection errors

Class Hierarchy

NetworkException

```
class NetworkException;
```

File

NetworkException.h

Description

Objects of this class are thrown by RTNetClient (see page 23) functions if network connection problems occur. The errorcode (see page 19) provides further information about the type of error

Members

Data Members

Data Member	Description
 errorcode (see page 19)	The error code

Methods

Method	Description
 NetworkException (see page 19)	Construct with specified error code

Legend

	Data Member
	Method

4.1.16.1 NetworkException::errorcode

The error code

```
int errorcode;
```

Description

This will be one of the following symbolic constants:

Symbolic Constant	Meaning
CODANET_OK	No error occurred
CODANET_SOCKETERROR_BROKEN	Network socket connection broken
CODANET_SOCKETERROR_WINDOWSDDL	Windows sockets DLL missing or unavailable
CODANET_SOCKETERROR_CREATE	Socket could not be created
CODANET_SOCKETERROR_HOSTNAME	Hostname could not be resolved
CODANET_SOCKETERROR_CONNECT	Could not connect to specified host
CODANET_SOCKETERROR_TCPOPTIONS	Error occurred whilst setting the required TCP options
CODANET_CLIENTPROTOCOLERROR_BADNUMSTATUS	Unexpected status results from server (possible data corruption or incorrect client version)
CODANET_SOCKETERROR_STREAMCREATE	Unable to create new stream
CODANET_SOCKETERROR_STREAMPORT	Unable to bind stream to the requested port - most commonly happens if there is an existing stream already created on that port
CODANET_CLIENTPROTOCOLERROR_TOOBIG	Unexpected return results from server (possible data corruption or incorrect client version)
CODANET_STREAMTIMEOUT	Timed out whilst waiting for data
CODANET_CLIENTPROTOCOLERROR_BADNUMCONFIG	Unexpected return results from server retrieving hardware configuration (possible data corruption or incorrect client version)
CODANET_CLIENTPROTOCOLERROR_BADSTRINGLENGTH	Unexpected string length from server (possible data corruption or incorrect client version)
CODANET_CLIENTPROTOCOLERROR_BADNUMDEVICES	Unexpected number of devices from server (possible data corruption or incorrect client version)
CODANET_CLIENTPROTOCOLERROR_BADSTRUCTSIZE	Unexpected structure size from server (possible data corruption or incorrect client version)

4.1.16.2 NetworkException::NetworkException

```
NetworkException(int ncode);
```

Description

Construct with specified error code

4.1.17 PacketDecode

Abstract base class for decoding packet information received on a data stream.

Class Hierarchy

PacketDecode

```
class PacketDecode;
```

File

PacketDecode.h

Description

Provides the base class for decoding of incoming data packets (RTNetworkPacket (see page 33) objects) received on a connected data stream (DataStream (see page 7) object).

Data packets may contain different data structures depending on the device used and the device options which have been set. Derived classes contain specific methods to decode (see page 21) the different types of packet data. For Codamotion CX1 data packets use the PacketDecode3DResultExt (see page 34) and for 16-bit analog data packets use PacketDecodeADC16 (see page 22).

All network packets received via a DataStream (see page 7) object will contain a numeric identifier for the device which created them, a device tick number indicating the time at which the measurements were taken, and an optional page number in the case that multiple packets (pages) are sent during the same tick.

The decode (see page 21) method will attempt to decode (see page 21) an incoming packet in a format determined by the derived class. To establish whether the packet is in a format which an instance of a derived class can decode (see page 21) (but without actually decoding it), use the canDecodePacket (see page 20) method.







Decoded data can be cleared using the clear (see page 21) method. This should be done if the RTNetworkPacket (see page 33) object used is destroyed before the PacketDecode object to avoid stale pointers being used.

See Also

PacketDecode3DResultExt, PacketDecodeADC16

Members

Methods

Method	Description
 canDecodePacket (see page 20)	Determine whether the specified packet is in a format which the instantiated decoder object can decode (see page 21).
 clear (see page 21)	Dissociates the decoder object from any incoming packet data and reset to an undecoded state
 decode (see page 21)	Attempt to decode the specified packet for retrieval via the methods of a derived class
 getDeviceID (see page 21)	Gets the numeric identifier for device from which the decoded data packet originates.
 getPage (see page 21)	Gets the page (packet per device clock tick) of this packet.
 getTick (see page 22)	Gets the clock tick time at which this packet was generated.

Legend

	Method
---	--------

4.1.17.1 PacketDecode::canDecodePacket

Determine whether the specified packet is in a format which the instantiated decoder object can decode (see page 21).

```
bool canDecodePacket(const RTNetworkPacket& packet) const;
```

Parameters

Parameters	Description
const RTNetworkPacket& packet	The packet to inspect.

Returns

true if the packet can be decoded by this object, false otherwise.

Description

This method does not actually perform any decoding operation but merely determines whether it is possible. Use decode (see page 21)

see page 21) to perform the decoding operation.

See Also

decode (see page 21)

4.1.17.2 PacketDecode::clear

```
void clear();
```

Description

Dissociates the decoder object from any incoming packet data and reset to an undecoded state

4.1.17.3 PacketDecode::decode

Attempt to decode the specified packet for retrieval via the methods of a derived class

```
bool decode(const RTNetworkPacket& packet);
```

Parameters

Parameters	Description
const RTNetworkPacket& packet	The network packet to decode.

Returns

true if packet was successfully decoded, false otherwise.

Description

The nature of the decoding which is done will depend upon the derived class. After decoding the derived class will have a variety of **get** methods to retrieve the decoded data. See for example PacketDecode3DResultExt::getNumMarkers (see page 35). If the packet is not in a form which can be decoded by the derived class, this method returns false.

4.1.17.4 PacketDecode::getDeviceID

```
WORD getDeviceID() const;
```

Returns

Numeric identifier for device or zero if no packet has been decoded.

Description

Gets the numeric identifier for device from which the decoded data packet originates.

4.1.17.5 PacketDecode::getPage

Gets the page (packet per device clock tick) of this packet.

```
DWORD getPage() const;
```

Returns

Zero-based page number. Will be zero if no packet has been decoded.

Description

This is only used for devices which send multiple packets per clock tick.

4.1.17.6 PacketDecode::getTick

Gets the clock tick time at which this packet was generated.

```
DWORD getTick() const;
```

Returns

Integer tick value. Will be zero if no packet has been decoded.

Description

The actual time interval corresponding to one tick will depend on the settings of the device from which this packet came. See `RTNetClient::getDeviceTickSeconds` (see page 33) to determine tick time in seconds.

4.1.18 PacketDecodeADC16

Decodes 16-bit measurement data received from a data stream.

Class Hierarchy

```
codaRTNet::PacketDecode
  PacketDecodeADC16
```

```
class PacketDecodeADC16 : public PacketDecode;
```

File

PacketDecodeADC16.h





Description

Use this class to decode network packets which are in the **ADC16** format. This is the format used to transmit measurements from 16-bit analog-to-digital converters such as the GS16AIO on a Codamotion Active Hub. They contain one 16-bit value per channel of analog data, for several channels. However, devices may have elected to decimate one or more channels, in which case the values corresponding to those channels are omitted for all packets whose device tick value is not a multiple of the decimation factor. When using the results from this decoder, it is therefore important to know the channel decimation factors which were applied in order to correctly identify which channel is which.







Note that decimation factors are only applied to packets received using `RTNetClient::startAcqContinuous` (see page 30) or `RTNetClient::requestAcqBufferFrame` and *not* when received using `RTNetClient::doAcqSingleShot` or `RTNetClient::monitorAcqBuffer` (see page 31).

Members

Methods

Method	Description
 <code>getNumValues</code> (see page 23)	Number of 16-bit values contained in the decoded packet
 <code>getValues</code> (see page 23)	Pointer to array of 16-bit values whose array length is given by <code>getNumValues</code> (see page 23)
 <code>PacketDecodeADC16</code> (see page 23)	Construct a decoder object with no associated data
 <code>PacketDecodeADC16</code> (see page 23)	Construct a decoder object and attempt to decode a specified network packet

PacketDecode

Method	Description
 <code>canDecodePacket</code> (see page 20)	Determine whether the specified packet is in a format which the instantiated decoder object can decode (see page 21).
 <code>clear</code> (see page 21)	Dissociates the decoder object from any incoming packet data and reset to an undecoded state
 <code>decode</code> (see page 21)	Attempt to decode the specified packet for retrieval via the methods of a derived class
 <code>getDeviceID</code> (see page 21)	Gets the numeric identifier for device from which the decoded data packet originates.
 <code>getPage</code> (see page 21)	Gets the page (packet per device clock tick) of this packet.
 <code>getTick</code> (see page 22)	Gets the clock tick time at which this packet was generated.

Legend

	Method
---	--------

4.1.18.1 PacketDecodeADC16::getNumValues

```
DWORD getNumValues();
```

Description

Number of 16-bit values contained in the decoded packet

4.1.18.2 PacketDecodeADC16::getValues

```
short* getValues();
```

Description

Pointer to array of 16-bit values whose array length is given by getNumValues (see page 23)

4.1.18.3 PacketDecodeADC16::PacketDecodeADC16

Construct a decoder object with no associated data

```
PacketDecodeADC16();
```

Description

When constructed in this way, use the PacketDecode::decode (see page 21) method in the base class to attempt to decode a network packet.

4.1.18.4 PacketDecodeADC16::PacketDecodeADC16

Construct a decoder object and attempt to decode a specified network packet

```
PacketDecodeADC16(const RTNetworkPacket& packet);
```

Parameters

Parameters	Description
const RTNetworkPacket& packet	A network packet to decode.

Description

This is the same as constructing using the default constructor and calling the decode method.

See Also

PacketDecode::decode

4.1.19 RTNetClient

Provides the main programming interface for communication with Codamotion RTNet Server.

Class Hierarchy

RTNetClient

```
class RTNetClient;
```






























File

RTNetClient.h


Description

Use `doAutoDiscoverServer` (see page 26) to discover available servers on the network and connect (see page 25) to establish a connection. `startSystem` (see page 27) initialises devices (start mode), and `prepareForAcq` (see page 29) prepares devices for acquisition (prepared mode). Attach data streams for receiving data by using `createDataStream` (see page 25). When in prepared mode, a single frame of data can be requested from all devices with `doSingleShotAcq` (see page 29). To continuously stream data use `startAcqContinuous` (see page 30). To continuously buffer data use `startAcqContinuousBuffered`. Whilst in buffered acquisition mode the most recently acquired frame can be monitored using `requestAcqBufferPacket` (see page 32)

Members**Methods**

Method	Description
 <code>RTNetClient</code> (see page 24)	Construct a disconnected RTNetClient object.
 <code>RTNetClient</code> (see page 25)	Construct an RTNetClient object and attempt to connect (see page 25) to server at specified IP address and port.
 <code>connect</code> (see page 25)	Connect to Codamotion RTNet Server using the specified IP address and port number.
 <code>createDataStream</code> (see page 25)	Create a data stream object for receiving measurement data from the server.
 <code>disconnect</code> (see page 26)	Disconnect from Codamotion RTNet Server.
 <code>isConnected</code> (see page 26)	Determine whether currently connected to server.
 <code>doAutoDiscoverServer</code> (see page 26)	Attempt to find Codamotion Realtime Network servers on the network.
 <code>enumerateDevices</code> (see page 26)	Provide array of all attached devices.
 <code>enumerateHWConfig</code> (see page 26)	Enumerate all available hardware configurations and their corresponding names
 <code>startSystem</code> (see page 27)	Attempt to start the system with a specified hardware configuration.
 <code>stopSystem</code> (see page 27)	If the system has been started, this will cause it to shut down.
 <code>getDeviceEnable</code> (see page 27)	Retrieve device enable/disable flags for the specified hardware configuration.
 <code>setDeviceOptions</code> (see page 28)	Set runtime device information or perform runtime device setup operation.
 <code>getDeviceInfo</code> (see page 28)	Get runtime device information.
 <code>closeDataStream</code> (see page 28)	This is <code>closeDataStream</code> , a member of class RTNetClient.
 <code>doSingleShotAcq</code> (see page 29)	Request that a single frame of measurements be gathered on all active devices and sent to all connected data streams.
 <code>getAcqMaxTicks</code> (see page 29)	Find acquisition time limit in terms of specified device ticks.
 <code>prepareForAcq</code> (see page 29)	Prepare for acquisition.
 <code>getRunningHWConfig</code> (see page 30)	Get the address handle of the currently running hardware configuration, or zero if system not started.
 <code>startAcqContinuous</code> (see page 30)	Begin continuous acquisition and continual streaming of measurements to all connected data stream sockets.
 <code>getServerVersion</code> (see page 30)	Get server version number
 <code>startAcqContinuousBuffered</code> (see page 30)	Begin continuous acquisition with server-side buffering of all incoming data.
 <code>stopAcq</code> (see page 31)	Stop a continuous or continuous buffered acquisition.
 <code>isAcqInProgress</code> (see page 31)	Establish whether system is currently in an acquisition mode
 <code>monitorAcqBuffer</code> (see page 31)	Request the most recent frame of data for monitoring during buffered acquisition.
 <code>getAcqBufferNumPackets</code> (see page 32)	Returns total data packets in the server-side buffer for the specified device during or following a buffered acquisition
 <code>requestAcqBufferPacket</code> (see page 32)	Request a specific packet of data from a buffered acquisition for the specified device.
 <code>setAcqMaxTicks</code> (see page 32)	Set limit on duration of continuous acquisitions
 <code>getDeviceTickSeconds</code> (see page 33)	Find the number of seconds corresponding to one device tick on the specified device

Legend

	Method
---	--------

4.1.19.1 RTNetClient::RTNetClient

Construct a disconnected RTNetClient object.

```
RTNetClient();
```

Description

Following construction, use `connect` (see page 25) to connect (see page 25) to a server.

4.1.19.2 RTNetClient::RTNetClient

Construct an RTNetClient object and attempt to connect (see page 25) to server at specified IP address and port.

```
RTNetClient(unsigned long ip, unsigned short port);
```

Parameters

Parameters	Description
unsigned long ip	IP address of server expressed as 32-bit value
unsigned short port	port number on which server is running

Description

Following construction, use `connect` (see page 25) to connect (see page 25) to a server.

4.1.19.3 RTNetClient::connect

Connect to Codamotion RTNet Server using the specified IP address and port number.

```
void connect(unsigned long ip, unsigned short port) throw(NetworkException);
```

Parameters

Parameters	Description
unsigned long ip	IP address of server expressed as 32-bit value
unsigned short port	port number on which server is running

Description

Suitable IP and port parameters can be retrieved using the `doAutoDiscoverServer` (see page 26) method, or can be specified explicitly. Throws a `NetworkException` (see page 18) on failure.

Exceptions

`NetworkException` (see page 18)

4.1.19.4 RTNetClient::createDataStream

Create a data stream object for receiving measurement data from the server.

```
void createDataStream(DataStream& stream, ::WORD port) throw(NetworkException, DeviceStatusArray);
```

Parameters

Parameters	Description
DataStream& stream	Reference to a DataStream (see page 7) object which holds this stream's details.

Description

Creates a UDP socket on the client and identifies this socket to the server for subsequent streaming of data. To request that packets of data are sent to the stream use `doSingleShotAcq` (see page 29), `acqBufferMonitor`, or `requestAcqBufferPacket` (see page 32).

Exceptions

`NetworkException` (see page 18), `DeviceStatusArray` (see page 17)

4.1.19.5 RTNetClient::disconnect

Disconnect from Codamotion RTNet Server.

```
void disconnect();
```

4.1.19.6 RTNetClient::isConnected

Determine whether currently connected to server.

```
bool isConnected() const;
```

Returns

true if connected, false otherwise

4.1.19.7 RTNetClient::doAutoDiscoverServer

Attempt to find Codamotion Realtime Network servers on the network.

```
static void doAutoDiscoverServer(AutoDiscover& discover);
```

Parameters

Parameters	Description
AutoDiscover& discover	structure to be filled with array of available servers

Description

This will use a UDP auto-discovery protocol to attempt to find all available Codamotion Realtime Network servers which are attached. The results are placed in an AutoDiscover (see page 6) class. If it failed to find any servers, the dwNumServers member of AutoDiscover (see page 6) will be zero. If this happens it may be because servers are down or not connected.

4.1.19.8 RTNetClient::enumerateDevices

Provide array of all attached devices.

```
void enumerateDevices(DeviceEnum& dev) throw(NetworkException, DeviceStatusArray);
```

Parameters

Parameters	Description
DeviceEnum& dev	Reference to a DeviceEnum (see page 8) object to be filled with device information.

Description

Provide array of all attached devices, identified by their device ID numbers. The DeviceEnum (see page 8) class will be filled with an array of ID's for the attached devices.

Exceptions

NetworkException (see page 18), DeviceStatusArray (see page 17)

4.1.19.9 RTNetClient::enumerateHWConfig

Enumerate all available hardware configurations and their corresponding names

```
void enumerateHWConfig(HWConfigEnum& config) throw(NetworkException, DeviceStatusArray);
```

Parameters

Parameters	Description
HWConfigEnum& config	Reference to a HWConfigEnum (see page 18) to be filled with configuration information.

Description

Available hardware configurations are returned in a HWConfigEnum (see page 18) class. Each configuration has an associated address handle and an associated name. The address handle can be used to select a specific configuration in a call to startSystem (see page 27), or to query the configuration using getDeviceEnable (see page 27).

Exceptions

NetworkException (see page 18), DeviceStatusArray (see page 17)

4.1.19.10 RTNetClient::startSystem

Attempt to start the system with a specified hardware configuration.

```
void startSystem(DWORD confighandle) throw(NetworkException, DeviceStatusArray);
```

Parameters

Parameters	Description
DWORD confighandle	Address handle of hardware configuration to use

Description

If the system is currently in a shut-down state, this will attempt to start the system by initialising all devices specified in the hardware config identified by the confighandle parameter. On successful completion the system will be in started mode. An array of all available config address handle values can be found using enumerateHWConfig (see page 26), and the devices available in any given config can be found using enumerateDevices (see page 26).

If the system has already been started, this function does nothing. If shutdown-restart is required, call stopSystem (see page 27) first and then call this function.

Exceptions

NetworkException (see page 18), DeviceStatusArray (see page 17)

4.1.19.11 RTNetClient::stopSystem

If the system has been started, this will cause it to shut down.

```
void stopSystem() throw(NetworkException, DeviceStatusArray);
```

Description

Shuts system down and enters shut-down state. This will generate an error only if the system was already shut down when it was called.

Exceptions

NetworkException (see page 18), DeviceStatusArray (see page 17)

4.1.19.12 RTNetClient::getDeviceEnable

Retrieve device enable/disable flags for the specified hardware configuration.

```
void getDeviceEnable(DWORD confighandle, CODANET_HWCONFIG_DEVICEENABLE& opt)
throw(NetworkException, DeviceStatusArray);
```


Parameters

Parameters	Description
DWORD confighandle	Address handle for an existing hardware configuration. A list of all such handles can be retrieved using enumerateDevices (see page 26).
CODANET_HWCONFIG_DEVICEENABLE& opt	Reference to a CODANET_HWCONFIG_DEVICEENABLE (see page 42) structure to be filled with enable/disable information

Exceptions

NetworkException ([see page 18](#)), DeviceStatusArray ([see page 17](#))

4.1.19.13 RTNetClient::setDeviceOptions

Set runtime device information or perform runtime device setup operation.

```
void setDeviceOptions(DeviceOptions& options) throw(NetworkException, DeviceStatusArray);
```

Parameters

Parameters	Description
info	Reference to an instance of a class derived from DeviceOptions (see page 12)

Description

Typically used to set those system parameters which can be changed after a device has been started, or to perform setup operations on a running device such as system alignment of a CX1 system.

The info parameter should be a reference to an instance of a class derived from DeviceOptions ([see page 12](#)). The overridden class will determine the type of data being set or setup operation being performed. For example DeviceOptionsAlignment ([see page 12](#)) will cause a CX1 alignment procedure to be executed.

Exceptions

NetworkException ([see page 18](#)), DeviceStatusArray ([see page 17](#))

4.1.19.14 RTNetClient::getDeviceInfo

Get runtime device information.

```
void getDeviceInfo(DeviceInfo& info) throw(NetworkException, DeviceStatusArray);
```

Parameters

Parameters	Description
DeviceInfo& info	Reference to an instance of a class derived from DeviceInfo (see page 8)

Description

Typically used to retrieve those system parameters which may be subject to change after a device has been started. Examples ([see page 5](#)) include sample rate or alignment data.

The info parameter should be a reference to an instance of a class derived from DeviceInfo ([see page 8](#)). The overridden class will determine the type of data being retrieved and will be filled with that data on successful completion. For example DeviceInfoAlignment ([see page 9](#)) retrieves information about the most recent alignment of a Codamotion CX1 system.

Exceptions

NetworkException ([see page 18](#)), DeviceStatusArray ([see page 17](#))

4.1.19.15 RTNetClient::closeDataStream

```
void closeDataStream(DataStream& stream);
```

Description

This is closeDataStream, a member of class RTNetClient.

4.1.19.16 RTNetClient::doSingleShotAcq

Request that a single frame of measurements be gathered on all active devices and sent to all connected data streams.

```
void doSingleShotAcq() throw(NetworkException, DeviceStatusArray);
```

Description

This will only succeed if the system has been started (using startSystem (see page 27)) and prepared for acquisition (using prepareForAcq (see page 29)). A single frame of data will be requested and, provided there are no network errors, will arrive on all connected data stream sockets.

Exceptions

NetworkException (see page 18), DeviceStatusArray (see page 17)

4.1.19.17 RTNetClient::getAcqMaxTicks

Find acquisition time limit in terms of specified device ticks.

```
DWORD getAcqMaxTicks(WORD device) throw(NetworkException, DeviceStatusArray);
```

Parameters

Parameters	Description
WORD device	Device whose tick length to use as the time scale.

Returns

time Time limit in device ticks or CODANET_ACQ_UNLIMITED (see page 47) if there is no time limit.

Description

This time limit is the one which would have been set by setAcqMaxTicks (see page 32).

See Also

setAcqMaxTicks (see page 32)

4.1.19.18 RTNetClient::prepareForAcq

Prepare for acquisition.

```
void prepareForAcq() throw(NetworkException, DeviceStatusArray);
```

Description

After the system has been started, this function should be called to enter a prepared state, ready for acquisition. This performs any pre-acquisition configuration such as taking diode offset levels in a CX1 system, or resetting force platforms. It may take several seconds to execute.

Once the system is prepared for acquisition, the following functions become available for use: doSingleShotAcq (see page 29), startAcqContinuous (see page 30), startAcqContinuousBuffered.

Exceptions

NetworkException (see page 18), DeviceStatusArray (see page 17)

4.1.19.19 RTNetClient::getRunningHWConfig

Get the address handle of the currently running hardware configuration, or zero if system not started.

```
DWORD getRunningHWConfig() throw(NetworkException, DeviceStatusArray);
```

Returns

Address handle of the currently running hardware configuration, or zero if system not started.

Description

This can be used to determine whether the system has been started (using `startSystem` (see page 27)), and if so which hardware configuration was used. See `enumerateHWConfig` (see page 26) for more information about hardware configurations.

Exceptions

`NetworkException` (see page 18), `DeviceStatusArray` (see page 17)

4.1.19.20 RTNetClient::startAcqContinuous

Begin continuous acquisition and continual streaming of measurements to all connected data stream sockets.

```
void startAcqContinuous() throw(NetworkException, DeviceStatusArray);
```

Description

Puts the system into continuous (non-buffered) acquisition mode. This will only succeed if the system has been started (using `startSystem` (see page 27)) and prepared for acquisition (using `prepareForAcq` (see page 29)). Data will be acquired continuously until either `stopAcq` (see page 31) is called or the limit set by `setAcqMaxTicks` (see page 32) is reached. All acquired data from all enabled devices is continually sent over the network to those data streams which were connected at the time this function was called.

Exceptions

`NetworkException` (see page 18), `DeviceStatusArray` (see page 17)

4.1.19.21 RTNetClient::getServerVersion

Get server version number

```
void getServerVersion(Version& v) throw(NetworkException, DeviceStatusArray);
```

Parameters

Parameters	Description
Version& v	Object to hold the retrieved version number

Description

Retrieve the current server version number.

Exceptions

`NetworkException` (see page 18), `DeviceStatusArray` (see page 17)

4.1.19.22 RTNetClient::startAcqContinuousBuffered

Begin continuous acquisition with server-side buffering of all incoming data.

```
void startAcqContinuousBuffered() throw(NetworkException, DeviceStatusArray);
```

Description

Puts the system into continuous buffered acquisition mode. This will only succeed if the system has been started (using `startSystem` (see page 27)) and prepared for acquisition (using `prepareForAcq` (see page 29)). Data will be acquired continuously until either `stopAcq` (see page 31) is called or the limit set by `setAcqMaxTicks` (see page 32) is reached. All acquired data from all enabled devices is stored on the server (or stored in on-board memory on connected devices) for later retrieval using `requestAcqBufferPacket` (see page 32). To determine the total number of packets available, use `getAcqBufferNumPackets` (see page 32). During continuous buffered acquisition, the most recent available frame of data can be retrieved by calling `monitorAcqBuffer` (see page 31). This allows a buffered acquisition to be monitored in realtime.

Following end of acquisition (using `stopAcq` (see page 31) or when max frames has been reached), buffered data persists until the next acquisition function is called (one of `prepareForAcq` (see page 29), `doSingleShotAcq` (see page 29), `startAcqContinuous` (see page 30), or `startAcqContinuousBuffered`), or until the system is stopped using `stopSystem` (see page 27).

Exceptions

`NetworkException` (see page 18), `DeviceStatusArray` (see page 17)

4.1.19.23 RTNetClient::stopAcq

Stop a continuous or continuous buffered acquisition.

```
void stopAcq() throw(NetworkException, DeviceStatusArray);
```

Description

If the system is currently acquiring data (following a call to `startAcqContinuous` (see page 30) or `startAcqContinuousBuffered` (see page 30)), this will put it back into a prepared but non-acquiring state.

Exceptions

`NetworkException` (see page 18), `DeviceStatusArray` (see page 17)

4.1.19.24 RTNetClient::isAcqInProgress

Establish whether system is currently in an acquisition mode

```
bool isAcqInProgress() throw(NetworkException, DeviceStatusArray);
```

Description

The system will be in an acquisition mode if `acqContinuous` or `acqContinuousBuffered` have successfully been called and the duration has not exceeded any limit set by `setAcqMaxTicks` (see page 32)

4.1.19.25 RTNetClient::monitorAcqBuffer

Request the most recent frame of data for monitoring during buffered acquisition.

```
void monitorAcqBuffer() throw(NetworkException, DeviceStatusArray);
```

Description

This method works only during buffered acquisition mode (acquisitions started using `startAcqContinuousBuffered` (see page 30)). It causes the most recent frame of data from all enabled devices to be sent to all data streams which were connected at the time at acquisition start.

Note that the monitoring process runs asynchronously from any acquisition process. As such it is not guaranteed that successive calls to `monitorAcqBuffer` give successive frames. For example, two consecutive calls may cause the same frame to be transmitted twice, or may contain different frames separated by several frame sampling periods.

Exceptions

NetworkException (see page 18), DeviceStatusArray (see page 17)

4.1.19.26 RTNetClient::getAcqBufferNumPackets

Returns total data packets in the server-side buffer for the specified device during or following a buffered acquisition

```
DWORD getAcqBufferNumPackets(WORD device) throw(NetworkException, DeviceStatusArray);
```

Description

Used in conjunction with requestAcqBufferPacket (see page 32) to retrieve data after or during a buffered acquisition.

@params device the device of interest @returns the number of packets available in the buffer

Exceptions

NetworkException (see page 18), DeviceStatusArray (see page 17)

4.1.19.27 RTNetClient::requestAcqBufferPacket

Request a specific packet of data from a buffered acquisition for the specified device.

```
void requestAcqBufferPacket(WORD device, DWORD packetindex) throw(NetworkException, DeviceStatusArray);
```

Parameters

Parameters	Description
the	device for which to get data @packetindex index of packet to get in range 0 to (getAcqBufferNumPackets (see page 32))-1

Description

This method works during or immediately after buffered acquisition mode (acquisitions started using startAcqContinuousBuffered (see page 30)). If used after acquisition mode, it must be called before any of the following acquisition and system functions: doSingleShotAcq (see page 29), prepareForAcq (see page 29), startAcqContinuous (see page 30), startAcqContinuousBuffered (see page 30), stopSystem (see page 27).

It causes the specified packet of data to be sent to all currently connected data streams. To determine which devices are available, use getDeviceEnable (see page 27). To determine the total number of packets available for a specific device, use getAcqBufferNumPackets (see page 32).

Exceptions

NetworkException (see page 18), DeviceStatusArray (see page 17)

4.1.19.28 RTNetClient::setAcqMaxTicks

Set limit on duration of continuous acquisitions

```
void setAcqMaxTicks(WORD tickdevice, DWORD maxticks) throw(NetworkException, DeviceStatusArray);
```

Parameters

Parameters	Description
WORD tickdevice	the device whose tick length to use
DWORD maxticks	the number of ticks after which acquisition will expire, or CODANET_ACQ_UNLIMITED (see page 47) to continue until stopAcq (see page 31) is called

Description

Following a call to startAcqContinuous (see page 30) or startAcqContinuousBuffered (see page 30), Acquisition will expire for all devices once the specified number of device ticks have elapsed. To continue acquisition indefinitely (until

stopAcq (see page 31) is called), use the symbolic constant CODANET_ACQ_UNLIMITED (see page 47).

The length of a device tick is usually (though not always) equal to the sample period on the device, for example the marker rate on Codamotion CX1 systems. To determine the length of a device tick based on current device settings for a given device, use `getDeviceTickSeconds` (see page 33). To get an array of all enabled devices, use `getDeviceEnable` (see page 27).

Note that the timing accuracy with which acquisition is stopped is not guaranteed. All devices will acquire for *at least* the amount of time specified. Most devices will stop within 0.1s after this, but please contact Charnwood Dynamics for more details if the length of this delay is critical for your application.

Exceptions

`networkException`, `DeviceStatusArray` (see page 17)

4.1.19.29 RTNetClient::getDeviceTickSeconds

Find the number of seconds corresponding to one device tick on the specified device

```
float getDeviceTickSeconds(WORD device) throw(NetworkException, DeviceStatusArray);
```

Description

A device tick is a native unit of measure used by hardware devices and usually (but not always) corresponds to one sample period. For example a Codamotion CX1 device sampling at 200Hz with decimation factor of 1 (no decimation) will have a device tick of 0.005 seconds.

Device ticks are used when specifying the maximum acquisition length, and are embedded in measurement packets received on a data stream using classes derived from `PacketDecode` (see page 19).

@params device the device whose tick length to get @returns tick length in seconds (floating point)

Exceptions

`NetworkException` (see page 18), `DeviceStatusArray` (see page 17)

4.1.20 RTNetworkPacket

Holds the data from a single packet on a data stream

Class Hierarchy

```
CODANET_PACKET
RTNetworkPacket
```

```
class RTNetworkPacket : protected CODANET_PACKET;
```

File



`RTNetworkPacket.h`

Description

Construct an instance of this object for use with `DataStream::receivePacket` (see page 7). Once received, packets can be decoded into measurement information using a class derived from `PacketDecode` (see page 19) (see `PacketDecode3DResultExt` (see page 34) and `PacketDecodeADC16` (see page 22)).

On busy networks it is possible for packets to become split or corrupted. Use `verifyChecksum` (see page 34) prior to decoding to ensure that no corrupt data is used.

Members**Methods**

Method	Description
 RTNetworkPacket (see page 34)	Construct an un-initialised packet object
 verifyChecksum (see page 34)	Provides a consistency check on incoming packets by comparing packet data with a checksum value which was originally transmitted with the data.

Legend

	Method
---	--------

4.1.20.1 RTNetworkPacket::RTNetworkPacket

```
RTNetworkPacket();
```

Description

Construct an un-initialised packet object

4.1.20.2 RTNetworkPacket::verifyChecksum

```
bool verifyChecksum() const;
```

Returns

true if packet data is valid, false if corrupt.

Description

Provides a consistency check on incoming packets by comparing packet data with a checksum value which was originally transmitted with the data.

4.1.21 PacketDecode3DResultExt

Decodes 3D measurement packets received on a data stream.

Class Hierarchy

```
codaRTNet::PacketDecode
  PacketDecode3DResultExt
```

```
class PacketDecode3DResultExt : public PacketDecode;
```



File

PacketDecode3DResultExt.h

Description

Use this class to decode 3D measurement packets which are in the **3DRESULT_EXT** format. This is the format used to transmit measurements from Codamotion CX1 systems. Such packets are received using a `DataStream` (see page 7) object as `RTNetworkPacket` (see page 33) objects. The **3DRESULT_EXT** format includes 3-dimensional position measurements for one or more markers and visibility flag (visible/occluded) for each marker. It may also include intensity data for each marker (three or more cameras per marker), and residuals (one or more residuals per marker).

Members**Methods**

Method	Description
 getIntensity (see page 35)	Retrieves infra-red light intensity information for decoded point.
 getNumCamerasPerMarker (see page 35)	Number of cameras per marker (indicates the number of intensity values present per marker)

◆ <code>getNumMarkers</code> (see page 35)	Number of markers
◆ <code>getNumResidualsPerMarker</code> (see page 36)	Number of residual values per marker
◆ <code>getPosition</code> (see page 36)	Retrieves 3D vector position of decoded point, in millimetres
◆ <code>getResiduals</code> (see page 36)	Retrieves marker residual values in millimetres.
◆ <code>getValid</code> (see page 36)	Retrieves validity flag for decoded marker
◆ <code>PacketDecode3DResultExt</code> (see page 37)	Construct a decoder object with no associated data
◆ <code>PacketDecode3DResultExt</code> (see page 37)	Construct a decoder object and attempt to decode a specified network packet

PacketDecode

Method	Description
◆ <code>canDecodePacket</code> (see page 20)	Determine whether the specified packet is in a format which the instantiated decoder object can decode (see page 21).
◆ <code>clear</code> (see page 21)	Dissociates the decoder object from any incoming packet data and reset to an undecoded state
◆ <code>decode</code> (see page 21)	Attempt to decode the specified packet for retrieval via the methods of a derived class
◆ <code>getDeviceID</code> (see page 21)	Gets the numeric identifier for device from which the decoded data packet originates.
◆ <code>getPage</code> (see page 21)	Gets the page (packet per device clock tick) of this packet.
◆ <code>getTick</code> (see page 22)	Gets the clock tick time at which this packet was generated.

Legend

◆	Method
---	--------

4.1.21.1 PacketDecode3DResultExt::getIntensity

Retrieves infra-red light intensity information for decoded point.

```
BYTE* getIntensity(DWORD ipoint);
```

Parameters

Parameters	Description
<code>DWORD ipoint</code>	Zero-based index of the marker whose intensity to get, in range zero to (<code>getNumMarkers</code> (see page 35))-1).

Returns

Pointer to array of bytes

Description

Light intensity is measured on a scale 0 to 255 although values below 10 are error codes and therefore should not be used. This function returns a pointer to an array of bytes, one per intensity value. The number of elements in the returned array is determined by `getNumCamerasPerMarker` (see page 35).

4.1.21.2 PacketDecode3DResultExt::getNumCamerasPerMarker

```
BYTE getNumCamerasPerMarker();
```

Description

Number of cameras per marker (indicates the number of intensity values present per marker)

4.1.21.3 PacketDecode3DResultExt::getNumMarkers

```
DWORD getNumMarkers();
```

Description

Number of markers

4.1.21.4 PacketDecode3DResultExt::getNumResidualsPerMarker

```
BYTE getNumResidualsPerMarker();
```

Description

Number of residual values per marker

4.1.21.5 PacketDecode3DResultExt::getPosition

```
float* getPosition(DWORD ipoint);
```

Parameters

Parameters	Description
DWORD ipoint	Zero-based index of the marker to get, in range zero to (getNumMarkers (see page 35))-1).

Returns

Pointer to three-element array with X,Y and Z coordinate values for the specified marker, measured in millimetres

Description

Retrieves 3D vector position of decoded point, in millimetres

4.1.21.6 PacketDecode3DResultExt::getResiduals

Retrieves marker residual values in millimetres.

```
WORD* getResiduals(DWORD ipoint);
```

Parameters

Parameters	Description
DWORD ipoint	Zero-based index of the marker whose residual to get, in range zero to (getNumMarkers (see page 35))-1).

Returns

Pointer to array of 16-bit unsigned values. Each element represents a residual in microns.

Description

getResiduals

The residual is the unweighted euclidean distance between the mean marker location reported by getPosition (see page 36) and that reported by an individual measurement unit, where two or more units are used. It is represented here as an integer value of microns. If only one unit is used, no residual information is transmitted since it would always be zero. Use getNumResidualsPerMarker (see page 36) to determine how many residual values are available per marker.

4.1.21.7 PacketDecode3DResultExt::getValid

```
BYTE getValid(DWORD ipoint);
```

Parameters

Parameters	Description
DWORD ipoint	Zero-based index of the flag to get, in range zero to (getNumMarkers (see page 35))-1).

Returns

Non-zero if marker was visible to at least one measurement unit, zero otherwise.

Description

Retrieves validity flag for decoded marker

4.1.21.8 PacketDecode3DResultExt::PacketDecode3DResultExt

Construct a decoder object with no associated data

```
PacketDecode3DResultExt();
```

Description

When constructed in this way, use the `PacketDecode::decode` (see page 21) method in the base class to attempt to decode a network packet.

4.1.21.9 PacketDecode3DResultExt::PacketDecode3DResultExt

Construct a decoder object and attempt to decode a specified network packet

```
PacketDecode3DResultExt(const RTNetworkPacket& packet);
```

Parameters

Parameters	Description
<code>const RTNetworkPacket& packet</code>	A network packet to decode.

Description

This is the same as constructing using the default constructor and calling the `decode` method.

See Also

`PacketDecode::decode`

4.1.22 Version

Version information structure retrieved from RTNet server

Class Hierarchy

```
CODANET_VERSION  
Version
```


```
class Version : public CODANET_VERSION;
```

File

Version.h

Members

Methods

Method	Description
 <code>Version</code> (see page 37)	Construct version structure and initialise version numbers to zero

Legend

	Method
---	--------

4.1.22.1 Version::Version

Construct version structure and initialise version numbers to zero

```
Version();
```

4.2 Structs

Structs

Struct	Description
CODANET_DEVICEINFO_CODAMODE (see page 38)	This is record CODANET_DEVICEINFO_CODAMODE.
CODANET_DEVICEINFO_CODAPACKETMODE (see page 38)	This is record CODANET_DEVICEINFO_CODAPACKETMODE.
CODANET_DEVICEINFO_GS16AIO_INPUTS (see page 39)	This is record CODANET_DEVICEINFO_GS16AIO_INPUTS.
CODANET_DEVICEINFO_UNITCOORDSYSTEM (see page 39)	This is record CODANET_DEVICEINFO_UNITCOORDSYSTEM.
CODANET_DEVICEOPTIONS_ALIGNMENT_EXT (see page 39)	This is record CODANET_DEVICEOPTIONS_ALIGNMENT_EXT.
CODANET_DEVICEOPTIONS_STROBEOUTPUT (see page 40)	This is record CODANET_DEVICEOPTIONS_STROBEOUTPUT.
CODANET_DATASTREAM_DEST (see page 40)	This is record CODANET_DATASTREAM_DEST.
CODANET_DEVICEENUM (see page 40)	This is record CODANET_DEVICEENUM.
CODANET_DEVICEINFO_ALIGNMENT (see page 40)	This is record CODANET_DEVICEINFO_ALIGNMENT.
CODANET_VERSION (see page 41)	Version information structure retrieved from RTNet server.
CODANET_AUTODISCOVER (see page 41)	This is record CODANET_AUTODISCOVER.
CODANET_DEVICESTATUS (see page 42)	Represents RTNet server-side error state on a particular device or device subsystem
CODANET_DEVICESTATUS_ARRAY (see page 42)	Represents RTNet server-side error state on one or more devices.
CODANET_HWCONFIG_DEVICEENABLE (see page 42)	This is record CODANET_HWCONFIG_DEVICEENABLE.

4.2.1 CODANET_DEVICEINFO_CODAMODE

```

struct CODANET_DEVICEINFO_CODAMODE {
    struct CODANET_DEVICEINFO_BASE base;
    DWORD dwRateMode;
    DWORD dwDecimation;
    DWORD dwExternalSync;
};

```

File

codartprotocol_cx1.h

Description

This is record CODANET_DEVICEINFO_CODAMODE.

4.2.2 CODANET_DEVICEINFO_CODAPACKETMODE

```

struct CODANET_DEVICEINFO_CODAPACKETMODE {
    struct CODANET_DEVICEINFO_BASE base;
    DWORD dwPacketMode;
};

```

File

codartprotocol_cx1.h

Description

This is record CODANET_DEVICEINFO_CODAPACKETMODE.

4.2.3 CODANET_DEVICEINFO_GS16AIO_INPUTS

```
struct CODANET_DEVICEINFO_GS16AIO_INPUTS {
    struct CODANET_DEVICEINFO_BASE base;
    DWORD dwNumBoards;
    DWORD dwNumChannels;
    float fBaseRateHz;
    float board_voltspers15bits[CODANET_GS16AIO_MAX_BOARDS];
    DWORD channel_decimation[CODANET_GS16AIO_MAX_CHANNELS];
    DWORD channel_time_offset[CODANET_GS16AIO_MAX_CHANNELS];
};
```

File

codartprotocol_gs16aio.h

Description

This is record CODANET_DEVICEINFO_GS16AIO_INPUTS.

4.2.4 CODANET_DEVICEINFO_UNITCOORDSYSTEM

```
struct CODANET_DEVICEINFO_UNITCOORDSYSTEM {
    struct CODANET_DEVICEINFO_BASE base;
    DWORD dwNumUnits;
    struct {
        double R[9];
        double t[3];
    } Rt;
};
```

File

codartprotocol_cx1.h

Description

This is record CODANET_DEVICEINFO_UNITCOORDSYSTEM.

4.2.5 CODANET_DEVICEOPTIONS_ALIGNMENT_EXT

```
struct CODANET_DEVICEOPTIONS_ALIGNMENT_EXT {
    struct CODANET_DEVICEOPTIONS_ALIGNMENT basic;
    BYTE enable[CODANET_CODA_MAXUNITS];
    double fOriginOffset[3];
};
```

File

codartprotocol_cx1.h

Description

This is record CODANET_DEVICEOPTIONS_ALIGNMENT_EXT.

4.2.6 CODANET_DEVICEOPTIONS_STROBEOUTPUT

```
struct CODANET_DEVICEOPTIONS_STROBEOUTPUT {  
    struct CODANET_DEVICEOPTIONS_BASE base;  
    DWORD dwStrobeOutput;  
};
```

File

codartprotocol_cx1.h

Description

This is record CODANET_DEVICEOPTIONS_STROBEOUTPUT.

4.2.7 CODANET_DATASTREAM_DEST

```
struct CODANET_DATASTREAM_DEST {  
    DWORD dwSize;  
    unsigned long dwCommand;  
    unsigned long dwIP;  
    unsigned short wPort;  
};
```

File

codartprotocol.h

Description

This is record CODANET_DATASTREAM_DEST.

4.2.8 CODANET_DEVICEENUM

```
struct CODANET_DEVICEENUM {  
    DWORD dwNumDevices;  
    WORD wDeviceID[CODANET_DEVICEENUM_MAXITEMS];  
};
```

File

codartprotocol.h

Description

This is record CODANET_DEVICEENUM.

4.2.9 CODANET_DEVICEINFO_ALIGNMENT

```
struct CODANET_DEVICEINFO_ALIGNMENT {  
    struct CODANET_DEVICEINFO_BASE base;  
    DWORD dwStatus;  
    DWORD dwUTC_MS;  
    DWORD dwUTC_LS;  
    WORD wMilliseconds;  
    WORD wReserved;  
    DWORD dwNumUnits;
```

```
    DWORD dwNumFrames;  
    BYTE camera_flag[3*5*CODANET_CODA_MAXUNITS*CODANET_DEVICEINFO_ALIGNMENT_MAXFRAMES];  
};
```

File

codartprotocol_cx1.h

Description

This is record CODANET_DEVICEINFO_ALIGNMENT.

4.2.10 CODANET_VERSION

Version information structure retrieved from RTNet server.

```
struct CODANET_VERSION {  
    WORD wMajor;  
    WORD wMinor;  
    WORD wBuild;  
    WORD wRevision;  
};
```

File

codartprotocol.h

Members

Members	Description
WORD wMajor;	Major version
WORD wMinor;	Minor version
WORD wBuild;	Build number
WORD wRevision;	Revision number

Description

The four members wMajor, wMinor, wBuild and wRevision constitute a value in the standard versioning system used throughout Codamotion software. They are usually presented to the user as a multi-digit version number of the form: [wMajor].[wMinor] Build [wBuild] Revision [wRevision], for example: Version 1.01 Build 3 Revision 1.

4.2.11 CODANET_AUTODISCOVER

```
struct CODANET_AUTODISCOVER {  
    DWORD dwNumServers;  
    struct {  
        char hostname[CODANET_AUTODISCOVER_HOSTNAME_MAXLENGTH+1];  
        DWORD ip;  
        WORD port;  
    } server;  
};
```

File

codartclient.h

Description

This is record CODANET_AUTODISCOVER.

4.2.12 CODANET_DEVICESTATUS

Represents RTNet server-side error state on a particular device or device subsystem

```
struct CODANET_DEVICESTATUS {
    WORD deviceID;
    WORD subsystemID;
    DWORD error;
};
```

File

codartprotocol.h

Members

Members	Description
WORD deviceID;	The device on which the error occurred, or zero to indicate an error which applies to all devices.
WORD subsystemID;	The subsystem of the device on which the error occurred if known, or zero otherwise. The meaning of the subsystemID number is device-dependent and state-dependent. Prior to being started, Codamotion CX1 systems give a subsystem ID which refers to the serial port on which the error occurred. Once started the subsystem ID can be used to indicate the serial number of a CX1 unit.
DWORD error;	Error code produced - see device error codes

Description

If an error occurs on any device or subsystem during an interaction with the RTNet server, a structure of this type will be created and returned to the client as part of a CODANET_DEVICESTATUS_ARRAY (see page 42) structure.

4.2.13 CODANET_DEVICESTATUS_ARRAY

Represents RTNet server-side error state on one or more devices.

```
struct CODANET_DEVICESTATUS_ARRAY {
    DWORD numstatus;
    struct CODANET_DEVICESTATUS status[CODANET_DEVICESTATUS_MAXITEMS];
};
```

File

codartprotocol.h

Members

Members	Description
DWORD numstatus;	The number of error codes contained in this object, or zero if no error codes were generated
struct CODANET_DEVICESTATUS status[CODANET_DEVICESTATUS_MAXITEMS];	Array of CODANET_DEVICESTATUS (see page 42) objects. The number of these which contain any valid information is determined by the numstatus value.

Description

A structure of this type is retrieved from the server after every interaction between server and client. It is possible for a single interaction to generate more than one error - for example an attempted system start may fail on more than one device. This structure therefore provides for an array of error information in the form of multiple CODANET_DEVICESTATUS (see page 42) structures.

4.2.14 CODANET_HWCONFIG_DEVICEENABLE

```
struct CODANET_HWCONFIG_DEVICEENABLE {
    DWORD dwSize;
```

```
DWORD dwCommand;
DWORD dwConfig;
DWORD dwNumOptions;
struct {
    WORD wDevice;
    WORD wEnable;
} option;
};
```

File
codartprotocol.h

Description
This is record CODANET_HWCONFIG_DEVICEENABLE.

4.3 Data Types

Types

Type	Description
BYTE (see page 43)	8-bit unsigned value
DWORD (see page 43)	32-bit unsigned value
LPBYTE (see page 44)	Pointer to 8-bit unsigned value
LPDWORD (see page 44)	Pointer to 32-bit unsigned value
LPWCHAR (see page 44)	Pointer to wide character string
LPWORD (see page 44)	Pointer to 16-bit unsigned value
WORD (see page 44)	16-bit unsigned value
CODANET_SOCKET (see page 45)	This is type CODANET_SOCKET.

4.3.1 BYTE

```
typedef unsigned char BYTE;
```

File
codartapi.h

Description
8-bit unsigned value

4.3.2 DWORD

```
typedef unsigned long DWORD;
```

File
codartapi.h

Description
32-bit unsigned value

4.3.3 LPBYTE

```
typedef BYTE* LPBYTE;
```

File

codartapi.h

Description

Pointer to 8-bit unsigned value

4.3.4 LPDWORD

```
typedef DWORD* LPDWORD;
```

File

codartapi.h

Description

Pointer to 32-bit unsigned value

4.3.5 LPWCHAR

```
typedef wchar_t* LPWCHAR;
```

File

codartapi.h

Description

Pointer to wide character string

4.3.6 LPWORD

```
typedef WORD* LPWORD;
```

File

codartapi.h

Description

Pointer to 16-bit unsigned value

4.3.7 WORD

```
typedef unsigned short WORD;
```

File

codartapi.h

Description16-bit unsigned value

4.3.8 CODANET_SOCKET

```
typedef int CODANET_SOCKET;
```

File

codartapi.h

DescriptionThis is type CODANET_SOCKET.

4.4 C SDK Functions

Functions

Function	Description
codanet_sleep (see page 45)	Wait for a period of time

4.4.1 codanet_sleep

Wait for a period of time

```
void CODANET_API codanet_sleep(DWORD milliseconds);
```

File

codartclient.h

Parameters

Parameters	Description
milliseconds	The number of milliseconds to wait

Description

Cause the calling thread to wait (pause execution) for at least the specified number of milliseconds. In MS Windows this maps to the Win32 function Sleep().

4.5 Constants

Numeric Constants

4.5.1 Device ID's

Known Device ID's - numeric identifiers for RTNet devices.

Macros

Macro	Description
DEVICEID_CX1 (see page 46)	Device ID for Codamotion CX1 System device which represents one or more CX1 measurement units
DEVICEID_GS16AIO (see page 46)	Device ID for GS16AIO analog-to-digital converter system, comprising one or more sets of 32 analog input channels.

4.5.1.1 DEVICEID_CX1

```
#define DEVICEID_CX1 1
```

File

codartprotocol_cx1.h

Description

Device ID for Codamotion CX1 System device which represents one or more CX1 measurement units

4.5.1.2 DEVICEID_GS16AIO

```
#define DEVICEID_GS16AIO 3
```

File

codartprotocol_gs16aio.h

Description

Device ID for GS16AIO analog-to-digital converter system, comprising one or more sets of 32 analog input channels.

4.5.2 Size Constants

Size constants used in data structure sizes

Macros

Macro	Description
CODANET_DEVICEENUM_MAXITEMS (see page 46)	Maximum number of devices.
CODANET_DEVICESTATUS_MAXITEMS (see page 47)	Maximum device status return codes after request to RTNet server.
CODANET_HWCONFIG_MAXITEMS (see page 47)	Maximum number of hardware configurations.
CODANET_HWCONFIG_MAXNAMELENGTH (see page 47)	Maximum characters in name of hardware configuration, excluding null terminator.

4.5.2.1 CODANET_DEVICEENUM_MAXITEMS

```
#define CODANET_DEVICEENUM_MAXITEMS 256
```

File

codartprotocol.h

Description

Maximum number of devices.

4.5.2.2 CODANET_DEVICESTATUS_MAXITEMS

```
#define CODANET_DEVICESTATUS_MAXITEMS 256
```

File

codartprotocol.h

Description

Maximum device status return codes after request to RTNet server.

4.5.2.3 CODANET_HWCONFIG_MAXITEMS

```
#define CODANET_HWCONFIG_MAXITEMS 256
```

File

codartprotocol.h

Description

Maximum number of hardware configurations.

4.5.2.4 CODANET_HWCONFIG_MAXNAMELENGTH

```
#define CODANET_HWCONFIG_MAXNAMELENGTH 255
```

File

codartprotocol.h

Description

Maximum characters in name of hardware configuration, excluding null terminator.

4.5.3 Symbolic Constants

Symbolic constants used as flags or function arguments.

Macros

Macro	Description
CODANET_ACQ_UNLIMITED (see page 47)	Constant used to indicate unlimited acquisition time
CODANET_PACKET_WAIT_INDEFINITELY (see page 48)	Indicates that function will not timeout but will block indefinitely waiting for data

4.5.3.1 CODANET_ACQ_UNLIMITED

```
#define CODANET_ACQ_UNLIMITED 0xFFFFFFFFFUL
```

File

codartprotocol.h

Description

Constant used to indicate unlimited acquisition time

4.5.3.2 CODANET_PACKET_WAIT_INDEFINITELY

```
#define CODANET_PACKET_WAIT_INDEFINITELY 0xFFFFFFFFFUL
```

File

codartdatastream.h

Description

Indicates that function will not timeout but will block indefinitely waiting for data

Index

A

AutoDiscover 6

B

BYTE 43

C

C SDK Functions 45

CODANET_ACQ_UNLIMITED 47

CODANET_AUTODISCOVER 41

CODANET_DATASTREAM_DEST 40

CODANET_DEVICEENUM 40

CODANET_DEVICEENUM_MAXITEMS 46

CODANET_DEVICEINFO_ALIGNMENT 40

CODANET_DEVICEINFO_CODAMODE 38

CODANET_DEVICEINFO_CODAPACKETMODE 38

CODANET_DEVICEINFO_GS16AIO_INPUTS 39

CODANET_DEVICEINFO_UNITCOORDSYSTEM 39

CODANET_DEVICEOPTIONS_ALIGNMENT_EXT 39

CODANET_DEVICEOPTIONS_STROBEOUTPUT 40

CODANET_DEVICESTATUS 42

CODANET_DEVICESTATUS_ARRAY 42

CODANET_DEVICESTATUS_MAXITEMS 47

CODANET_HWCONFIG_DEVICEENABLE 42

CODANET_HWCONFIG_MAXITEMS 47

CODANET_HWCONFIG_MAXNAMELENGTH 47

CODANET_PACKET_WAIT_INDEFINITELY 48

codanet_sleep 45

CODANET_SOCKET 45

CODANET_VERSION 41

codaRTNet C++ Classes 6

Constants 45

D

Data Types 43

DataStream 7

DataStream 7

receivePacket 7

Device ID's 46

DeviceEnum 8

DEVICEID_CX1 46

DEVICEID_GS16AIO 46

DeviceInfo 8

DeviceInfoAlignment 9

dev 9

DeviceInfoAlignment 9

DeviceInfoCodaPacketMode 9

dev 10

DeviceInfoCodaPacketMode 10

DeviceInfoGS16AIOInputs 10

dev 11

DeviceInfoGS16AIOInputs 11

DeviceInfoUnitCoordSystem 11

dev 12

DeviceInfoUnitCoordSystem 12

DeviceOptions 12

DeviceOptionsAlignment 12

DeviceOptionsAlignment 13

opt 13

DeviceOptionsCodaMode 13

DeviceOptionsCodaMode 14

opt 14

DeviceOptionsCodaPacketMode 15

DeviceOptionsCodaPacketMode 16

opt 15

DeviceOptionsGS16AIOInputs 16

DeviceOptionsGS16AIOInputs 17

opt 17

DeviceStatusArray 17

DeviceStatusArray 18

DWORD 43

E

Examples 5

G

Getting Started (C++) 3

H

HWConfigEnum 18

L

LPBYTE 44

LPDWORD 44

LPWCHAR 44

LPWORD 44

N

NetworkException 18

 errorCode 19

 NetworkException 19

O

Overview 1

P

PacketDecode 19

 canDecodePacket 20

 clear 21

 decode 21

 getDeviceID 21

 getPage 21

 getTick 22

PacketDecode3DResultExt 34

 getIntensity 35

 getNumCamerasPerMarker 35

 getNumMarkers 35

 getNumResidualsPerMarker 36

 getPosition 36

 getResiduals 36

 getValid 36

 PacketDecode3DResultExt 37

PacketDecodeADC16 22

 getNumValues 23

 getValues 23

 PacketDecodeADC16 23

R

RTNetClient 23

 closeDataStream 28

 connect 25

 createDataStream 25

 disconnect 26

 doAutoDiscoverServer 26

 doSingleShotAcq 29

 enumerateDevices 26

 enumerateHWConfig 26

 getAcqBufferNumPackets 32

 getAcqMaxTicks 29

 getDeviceEnable 27

 getDeviceInfo 28

 getDeviceTickSeconds 33

 getRunningHWConfig 30

 getServerVersion 30

 isAcqInProgress 31

 isConnected 26

 monitorAcqBuffer 31

 prepareForAcq 29

 requestAcqBufferPacket 32

 RTNetClient 24, 25

 setAcqMaxTicks 32

 setDeviceOptions 28

 startAcqContinuous 30

 startAcqContinuousBuffered 30

 startSystem 27

 stopAcq 31

 stopSystem 27

RTNetworkPacket 33

 RTNetworkPacket 34

 verifyChecksum 34

S

Size Constants 46

Structs 38

Symbolic Constants 47

V

Version 37

 Version 37

W

WORD 44