LibTmsApi

1.2.3

Generated by Doxygen 1.5.6

Thu Dec 18 13:21:07 2008

Contents

Chapter 1

Main Page

Author:

Dr Terry Barnaby

Version:

1.2.3

Date:

2007-12-10

1.1 Introduction

This document covers the BEAM LibTms software API for the CERN trajectory measurement system. This API provides the ability to control and receive data from the TMS System. The API is an object orientated API implemented in 'C++' with a number of object classes. The API operates over a network type interface using an RPC type mechanism.

The LibTms API makes use of the BEAM standard class library. The BEAM standard class library provides a small set of low level 'C++' classes for strings, lists and system interface functions. There is some brief information on the BEAM class library later on in this page.

1.2 Overview

Generally users of the system are only concerned with the top level, System Controller API. This is the API that control and data gathering clients use to control and gather data from the system. The System Controller API (TmsApi) is implemented using a simple, object orientated, RPC mechanism. Two main objects, the **Tms::TmsControl** and **Tms::TmsProcess** objects, provide the full API.

The TmsApi has been developed using the BOAP (BEAM Object Access Protocol). This provides a simple but powerful Object Orientated RPC mechanism. The TmsApi is written in a high level interface definition language (IDL). The bidl tool generates the client and server side 'C++' interface and implementation files for the API. These are then provided as a set of 'C++' header files and a binary library file for the clients to link to. The BOAP system employs a simple BOAP name server process that provides a translation between object names and IPAddress/Socket numbers. The BOAP name server runs on the System Controller. More information on the BOAP system can be found in the libBeam documentation.

There are two main Objects that are used by clients of the TMS API. They are the **Tms::TmsControl** and the **Tms::TmsProcess** objects. The **Tms::TmsControl** object is used for system configuration, testing and diagnostics. The **Tms::TmsProcess** object is used for normal clients for Proton Synchrotron (PS) Cycle information configuration and data access. There is some example client code in the tmsExamples of the source code and displayed later on this page. These objects communicate through a network connection with the TmsServer process running on the TMS System Controller. The TMS System Controller operates as a multi-threaded process and can communicate with multiple clients simultaneously.

The TMS system takes most of its system timing signals from digital timing lines connected to the TMS rack hardware. The only timing information that external software needs to supply is the next cycle number and cycle type information. The cycle number is a 32bit unsigned number identifying the next Proton Synchrotron (PS) machine cycle. The cycle type is and ASCII string defining the type of BEAM present in the PS machine. The cycle type defines a set of state/phase tables to be loaded in order to measure the BEAM in the machine. The CERN client software needs to provide this information by calling the setNextCycle() function before the next PS cycle is initiated.

The TMS system keeps a library of state/phase tables indexed by the cycle type. These are loaded into the individual PUPE engines FPGA's during the CYCLE_STOP to CYCLE_START period. The API provides the **setControlInfo** and **delControlInfo** calls to maintain this database.

A client would generally use the **Tms::TmsProcess** object for its interface to the TMS system. It would use call getData() to fetch the required data from the system. There is also an event based data interface implemented using the requestData() call and dataEvent() event call.

Each of the TMS API calls return an error object. If there is an error, an appropriate error number will be given together with an ASCII string describing the error.

1.3 BEAM class library

The BEAM class library implements some basic low level classes and is used by the TMS API implementation itself. The main class functionaly includes:

- BString - A simple string class
- BList - A templated list class
- BArray - A templated array class
- BError - An error return class containg an integer and string
- BSocket - A Network socket access class
- BThread - A thread implementation class
- BPoll - A file descriptor event polling class
- BMutex - A mutex lock
- BRWLock - A read/write lock
- BSema - A semaphore
- BCondInt - An integer condition class
- BFile - A simple file access class
- BDir - A simple directory access class
- BEntry - A name/value pair list class

- BNameValue - A name/value pair class
- BRtc - A realtime clock
- BTimer - A simple timer class
- BUrl - URL access class

1.4 Examples

There are some examples of client applications using the TmsApi in the **tmsExamples** directory of the source code. A simple Data Access client example is listed bellow:

```
/*******************************************************************************
 *      TmsDataClient.cpp       TMS API example code for a Data Client
 *                      T.Barnaby,      BEAM Ltd,       2007-02-07
 *******************************************************************************
 *
 *      This is a very basic example of using the TmsApi from a clients perspective.
 *      It is designed to give an overview of using the API.
 */
#include <iostream>
#include <stdio.h>
#include <TmsD.h>
#include <TmsC.h>

using namespace Tms;
using namespace std;


// Function to reads some data
BError tmsTest(TmsProcess& tmsProcess){
        BError                  err;
        DataInfo                dataInfo;
        Data                    data;
        UInt32                  cn = 0;
        BString                 ct;

        // Find out the current cycle number and type
        if(err = tmsProcess.getCycleInfo(cn, ct)){
                return err.set(1, BString("Error: Getting Cycle Number: ") + err.getString());
        }

        printf("Getting data for cycles starting at cycle: %u\n", cn);

        for(; ; cn++){
                // Setup dataInfo
                printf("GetData: Cycle Number: %u\n", cn);
                dataInfo.cycleNumber    = cn;
                dataInfo.channel        = 1;
                dataInfo.cyclePeriod    = CyclePeriodEvent0;
                dataInfo.startTime      = 0;
                dataInfo.orbitNumber    = 0;
                dataInfo.bunchNumber    = 0;
                dataInfo.function       = DataFunctionRaw;
                dataInfo.argument       = 0;
                dataInfo.numValues      = 1024;
                dataInfo.beyondPeriod   = 1;

                if(err = tmsProcess.getData(dataInfo, data)){
                        return err.set(1, BString("Error: Getting Data: ") + err.getString());
                }
                printf("Data: NumValues: %d\n", data.numValues);
        }

        return err;
}

int main(int argc, char** argv){
        BError                  err;
        BString                 host = "localhost";
        TmsProcess              tmsProcess;

        if(argc == 2)
                host = argv[1];

        // Connect to the Process service
        if(err = tmsProcess.connectService(BString("//") + host + "/tmsProcess")){
                cerr << "Error: " << err.getString() << "\n";
                return 1;
        }

        // Run a normal data gathering cycle as a normal client would.
        if(err = tmsTest(tmsProcess)){
                cerr << "Error: " << err.getString() << "\n";
                return 1;
        }

        return 0;
}
```

A simple Control client example is listed below:

```
/*******************************************************************************
 *      TmsControlClient1.cpp   TMS API example code
 *                      T.Barnaby,      BEAM Ltd,       2007-02-07
 *******************************************************************************
 *
 *      This is a very basic example of using the TmsApi from a clients perspective.
 *      It is designed to give an overview of using the API.
 */
#include <iostream>
#include <stdio.h>
#include <TmsD.h>
#include <TmsC.h>

using namespace Tms;
using namespace std;

const UInt32    tmsStateNum = 16;
const UInt32    tmsPickupNum = 40;

// Initialise and test the TMS system
BError tmsInit(TmsControl& tmsControl){
        BError                  err;
        ConfigInfo              configInfo;
        BIter                   i;
        BList<BError>           errorList;
        BList<NameValue>        nvList;
        BString                 version;

        // Get Version
        if(err = tmsControl.getVersion(version)){
                return err.set(1, BString("Error: initialising TMS: ") + err.getString());
        }
        cout << "Version: " << version << "\n";

        // Initialise TMS system
        if(err = tmsControl.init()){
                return err.set(1, BString("Error: initialising TMS: ") + err.getString());
        }

        // Test TMS system
        if(err = tmsControl.test(errorList)){
                return err.set(1, BString("Error: testing TMS: ") + err.getString());
        }

        for(errorList.start(i); !errorList.isEnd(i); errorList.next(i)){
                cout << "Warning: " << errorList[i].getString() << "\n";
        }

        // Get Status of TMS system
        if(err = tmsControl.getStatus(nvList)){
                return err.set(1, BString("Error: getting status: ") + err.getString());
        }

        for(nvList.start(i); !nvList.isEnd(i); nvList.next(i)){
                cout << nvList[i].name << ":\t" << nvList[i].value << "\n";
        }

        return err;
}

int main(int argc, char** argv){
        BError                  err;
        BString                 host = "localhost";
        TmsControl              tmsControl;
        TmsProcess              tmsProcess;

        if(argc == 2)
                host = argv[1];

        // Connect to the Control service
        if(err = tmsControl.connectService(BString("//") + host + "/tmsControl")){
                cerr << "Error: " << err.getString() << "\n";
                return 1;
        }

        // Connect to the Process service
        if(err = tmsProcess.connectService(BString("//") + host + "/tmsProcess")){
                cerr << "Error: " << err.getString() << "\n";
                return 1;
        }

        // Initialise and test the TMS system. Normally carried out by a configuration
        // and test client program.
        if(err = tmsInit(tmsControl)){
                cerr << "Error: " << err.getString() << "\n";
                return 1;
        }

        return 0;
}
```

A simple Control client to set the next cycle information example is listed below:

```
/*******************************************************************************
 *      TmsControlClient2.cpp   TMS API example code
 *                      T.Barnaby,      BEAM Ltd,       2007-02-07
 *******************************************************************************
 *
 *      This is a very basic example of using the TmsApi to set the
 *      TMS's cycleNumber and cycleType.
 *      It is designed to give an overview of using the API.
 */
#include <iostream>
#include <stdio.h>
```

```
#include <unistd.h>
#include <TmsD.h>
#include <TmsC.h>

using namespace Tms;
using namespace std;

// Loop sending next cycle information
BError tmsControlLoop(TmsControl& tmsControl){
        BError                  err;
        UInt32                  cn = 0;
        BString                 ct = "Beam3";

        while(1){
                // Wait for next cycle information
                usleep(1200000);

                // Set next cycle information
                cn = cn + 1;
                ct = "Beam3";

                printf("SendNextCycle\n");
                // Send the next cycle information to the TMS server
                if(err = tmsControl.setNextCycle(cn, ct)){
                        cerr << "Error: " << err.getString() << "\n";
                }
        }

        return err;
}

int main(int argc, char** argv){
        BError                  err;
        BString                 host = "localhost";
        TmsControl              tmsControl;

        if(argc == 2)
                host = argv[1];

        // Connect to the Control service
        if(err = tmsControl.connectService(BString("//") + host + "/tmsControl")){
                cerr << "Error: " << err.getString() << "\n";
                return 1;
        }

        // Set the network priority high
        if(err = tmsControl.setPriority(BSocket::PriorityHigh)){
                cerr << "Error: " << err.getString() << "\n";
                return 1;
        }

        // Set the TmsServer thread priority high
        if(err = tmsControl.setProcessPriority(PriorityHigh)){
                cerr << "Error: " << err.getString() << "\n";
                return 1;
        }

        if(err = tmsControlLoop(tmsControl)){
                cerr << "Error: " << err.getString() << "\n";
                return 1;
        }

        return 0;
}
```

Chapter 2

Directory Hierarchy

2.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 4

Class Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Directory Documentation

7.1 /src/cern/tms/beam/ Directory Reference

Directories

- directory libBeam

7.2 /src/cern/tms/beam/libBeam/ Directory Reference

Files

- file BArray.h
- file BBuffer.cpp
- file BBuffer.h
- file BCond.cpp
- file BCond.h
- file BCondInt.cpp
- file BCondInt.h
- file BDir.cpp
- file BDir.h
- file BEntry.cpp
- file BEntry.h
- file BError.cpp
- file BError.h
- file BEvent.cpp
- file BEvent.h
- file BFile.cpp
- file BFile.h
- file BList.h
- file BList_func.h
- file BMutex.cpp
- file BMutex.h
- file BNameValue.h
- file Boap.cpp
- file Boap.h
- file BoapnsC.cc
- file BoapnsC.h
- file BoapnsD.cc
- file BoapnsD.h
- file BoapSimple.cc
- file BoapSimple.h
- file BObject.cc
- file BObject.h
- file BPoll-1.cpp
- file BPoll.cpp
- file BPoll.h
- file BRefData.cpp
- file BRefData.h
- file BRtc.cpp
- file BRtc.h
- file BRWLock.cpp
- file BRWLock.h
- file BSema.cpp
- file BSema.h
- file BSocket.cpp
- file BSocket.h
- file BString.cpp

- file BString.h
- file BThread.cpp
- file BThread.h
- file BTimer.cpp
- file BTimer.h
- file BTypes.h
- file BUrl.cpp
- file BUrl.h

Chapter 8

Namespace Documentation

8.1 Boapns Namespace Reference

Classes

- class Boapns
- class BoapEntry

Functions

- Boapns (BString name)
- BError getVersion (BString &version)
- BError getEntryList (BList< BoapEntry > &entryList)
- BError getEntry (BString name, BoapEntry &entry)
- BError addEntry (BoapEntry entry)
- BError delEntry (BString name)
- BError getNewName (BString &name)

Variables

- const BUInt32 apiVersion = 0

8.1.1 Function Documentation

8.1.1.1 BError Boapns::addEntry (BoapEntry *entry*)

8.1.1.2 Boapns::Boapns (BString *name*)

8.1.1.3 BError Boapns::delEntry (BString *name*)

8.1.1.4 BError Boapns::getEntry (BString *name*, BoapEntry & *entry*)

8.1.1.5 BError Boapns::getEntryList (BList< BoapEntry > & *entryList*)

8.1.1.6 BError Boapns::getNewName (BString & *name*)

8.1.1.7 BError Boapns::getVersion (BString & *version*)

8.1.2 Variable Documentation

8.1.2.1 const BUInt32 Boapns::apiVersion = 0

8.2 Tms Namespace Reference

Classes

- class PuControl

  This class defines the parameters for a test data capture.

- class PuProcess

  This interface provides functions to configure and capture data from individual pick-up.

- class TmsControl

  This interface provides functions to control, test and get statistics from the TMS as a whole.

- class TmsProcess

  This interface provides functions to capture data from the TMS as a whole.

- class TmsEvent

  This interface provides functions for events to be sent to clients from the TMS as a whole.

- class CycleParamState
- class CycleParamEdit

  Cycle Parameter management class.

- class NameValue
- class PuChannel

  This class stores a Physical Pick-Up channel id.

- class PuStatus

  This class stores the status of an individual Pick-Up.

- class ConfigInfo

  This class describes the configuration of the TMS.

- class DataInfo

  This class defines the data to be acquired and/or fetched.

- class DataValue

  This is the definition of a single data value.

- class Data

  This class stores the raw data.

- class PuStateTable

  This class defines the Pick-Up state table.

- class CycleParam

  This class defines the parameters for a PS processing cycle.

- class CycleParamItem
- class TestCaptureInfo

This class defines the parameters for a test data capture.

- class PupeConfig
- class CycleInformationPeriod

  Cycle information.

- class CycleInformation
- class CycleTypeInformationPeriod

  Cycle Type information.

- class CycleTypeInformation
- class Simulation
- class TmsEventServerList
- union TmsState

  The Tms State entry.

- union TmsPhase

  The Tms Phase Table Entry.

- class CycleParamDb

  Internal CycleParameter management class.

Enumerations

- enum Errors {
  ErrorOk, ErrorMisc, ErrorWarning, ErrorInit,
  ErrorConfig, ErrorParam, ErrorNotImplemented, ErrorComms,
  ErrorCommsTimeout, ErrorMC, ErrorFpga, ErrorStateTable,
  ErrorCycleNumber, ErrorDataNotAvailable, ErrorDataGone, ErrorDataFuture,
  ErrorTimeout }
- enum CyclePeriod {
  CyclePeriodAll, CyclePeriodCalibration, CyclePeriodEvent0, CyclePeriodEvent1,
  CyclePeriodEvent2, CyclePeriodEvent3, CyclePeriodEvent4, CyclePeriodEvent5,
  CyclePeriodEvent6, CyclePeriodEvent7, CyclePeriodEvent8, CyclePeriodEvent9 }
- enum DataType { DataTypeRaw }
- enum DataFunction {
  DataFunctionRaw, DataFunctionMean, DataFunctionMeanAll, DataFunctionMean0,
  DataFunctionMean1 }
- enum TestOutput { TestOutputFrefLocal, TestOutputPllL1, TestOutputPllL2 }
- enum Priority { PriorityLow, PriorityNormal, PriorityHigh }
- enum TimingSig {
  TimingSigClock = 0x01, TimingSigCycleStart = 0x02, TimingSigCycleStop = 0x04, TimingSigCal-Start = 0x08,
  TimingSigCalStop = 0x10, TimingSigInjection = 0x20, TimingSigHChange = 0x40, TimingSigFRef = 0x80 }

  The timing signal bits.

- enum CaptureClock {
  ClkAdcDiv_1 = 0x00, ClkAdcDiv_2 = 0x01, ClkAdcDiv_5 = 0x02, ClkAdcDiv_10 = 0x03,
  ClkAdcDiv_20 = 0x04, ClkAdcDiv_50 = 0x05, ClkAdcDiv_100 = 0x06, ClkAdcDiv_200 = 0x07,
  ClkAdcDiv_500 = 0x08, ClkAdcDiv_1000 = 0x09, ClkAdcDiv_2000 = 0x0A, ClkAdcDiv_5000 = 0x0B,
  ClkAdcDiv_10000 = 0x0C, ClkAdcDiv_20000 = 0x0D, ClkAdcDiv_50000 = 0x0E, ClkAdcDiv_-100000 = 0x0F,
  ClkMs = 0x10, ClkFref = 0x11 }

  The Diagnostics Capture Clock settings.

Variables

- const BUInt32 apiVersion = 0
- const unsigned int tmsNumPickups = 40

  The default number of pick ups.

- const unsigned int tmsPhaseTableSize = 512

  The size of the Phase Table.

8.2.1 Enumeration Type Documentation

8.2.1.1 enum Tms::CaptureClock

The Diagnostics Capture Clock settings.

Enumerator:

ClkAdcDiv_1	ADC Clock.
ClkAdcDiv_2	ADC Clock divided by 2.
ClkAdcDiv_5	ADC Clock divided by 5.
ClkAdcDiv_10	ADC Clock divided by 10.
ClkAdcDiv_20	ADC Clock divided by 20.
ClkAdcDiv_50	ADC Clock divided by 50.
ClkAdcDiv_100	ADC Clock divided by 100.
ClkAdcDiv_200	ADC Clock divided by 200.
ClkAdcDiv_500	ADC Clock divided by 500.
ClkAdcDiv_1000	ADC Clock divided by 1000.
ClkAdcDiv_2000	ADC Clock divided by 2000.
ClkAdcDiv_5000	ADC Clock divided by 5000.
ClkAdcDiv_10000	ADC Clock divided by 10000.
ClkAdcDiv_20000	ADC Clock divided by 20000.
ClkAdcDiv_50000	ADC Clock divided by 50000.
ClkAdcDiv_100000	ADC Clock divided by 100000.
ClkMs	Millisecond Clock.
ClkFref	FREF.

8.2.1.2 enum Tms::CyclePeriod

Enumerator:

CyclePeriodAll

CyclePeriodCalibration

CyclePeriodEvent0

CyclePeriodEvent1

CyclePeriodEvent2

CyclePeriodEvent3

CyclePeriodEvent4

CyclePeriodEvent5

CyclePeriodEvent6

CyclePeriodEvent7

CyclePeriodEvent8

CyclePeriodEvent9

8.2.1.3 enum Tms::DataFunction

Enumerator:

DataFunctionRaw

DataFunctionMean

DataFunctionMeanAll

DataFunctionMean0

DataFunctionMean1

8.2.1.4 enum Tms::DataType

Enumerator:

DataTypeRaw

8.2.1.5 enum Tms::Errors

Enumerator:

ErrorOk

ErrorMisc

ErrorWarning

ErrorInit

ErrorConfig

ErrorParam

ErrorNotImplemented

ErrorComms

ErrorCommsTimeout

ErrorMC

ErrorFpga

ErrorStateTable

ErrorCycleNumber

ErrorDataNotAvailable

ErrorDataGone

ErrorDataFuture

ErrorTimeout

8.2.1.6 enum Tms::Priority

Enumerator:

PriorityLow

PriorityNormal

PriorityHigh

8.2.1.7 enum Tms::TestOutput

Enumerator:

TestOutputFrefLocal

TestOutputPllL1

TestOutputPllL2

8.2.1.8 enum Tms::TimingSig

The timing signal bits.

Enumerator:

TimingSigClock	10MHz System Clock
TimingSigCycleStart	CYCLE_START event.
TimingSigCycleStop	CYCLE_STOP event.
TimingSigCalStart	CAL_START event.
TimingSigCalStop	CAL_STOP event.
TimingSigInjection	INJECTION event.
TimingSigHChange	HCHANGE event.
TimingSigFRef	FREF signal.

8.2.2 Variable Documentation

8.2.2.1 const BUInt32 Tms::apiVersion = 0

8.2.2.2 const unsigned int Tms::tmsNumPickups = 40

The default number of pick ups.

8.2.2.3 const unsigned int Tms::tmsPhaseTableSize = 512

The size of the Phase Table.

Chapter 9

Class Documentation

9.1 BArray< T > Class Template Reference

```
#include <BArray.h>
```

Public Member Functions

- BArray ()
- BArray (BSize size, T value=T())
- BArray (const BArray &array)

9.1.1 Detailed Description

template<class T> class BArray< T >

Template based Array class. This is based on the Standard C++ library vector class and has all of the functionality of that class.

9.1.2 Constructor & Destructor Documentation

9.1.2.1 template<class T> BArray< T >::BArray () `[inline]`

9.1.2.2 template<class T> BArray< T >::BArray (BSize *size*, T *value* = `T()`) `[inline]`

9.1.2.3 template<class T> BArray< T >::BArray (const BArray< T > & *array*) `[inline]`

The documentation for this class was generated from the following file:

- /src/cern/tms/beam/libBeam/BArray.h

9.2 BBuffer Class Reference

```
#include <BBuffer.h>
```

Public Member Functions

- BBuffer ()

  Create and manipulate a data buffer. On creation the buffer size defaults to 1024 bytes.

- ∼BBuffer ()
- int setSize (uint32_t size)

  Sets the bufer size.

- int setData (const void ∗data, uint32_t size)

  Sets buffer data resized to contain the data.

- int writeData (uint32_t pos, const void ∗data, uint32_t size)

  Writes data into buffer from offset pos.

- void ∗ data ()

  The data.

- uint32_t size ()

  Size of the buffer in bytes.

Private Attributes

- uint32_t osize
- uint32_t odatasize
- void ∗ odata

9.2.1 Constructor & Destructor Documentation

9.2.1.1 BBuffer::BBuffer ()

Create and manipulate a data buffer. On creation the buffer size defaults to 1024 bytes.

9.2.1.2 BBuffer::∼BBuffer ()

9.2.2 Member Function Documentation

9.2.2.1 int BBuffer::setSize (uint32_t *size*)

Sets the bufer size.

9.2.2.2 int BBuffer::setData (const void ∗ *data*, uint32_t *size*)

Sets buffer data resized to contain the data.

9.2.2.3 int BBuffer::writeData (uint32_t *pos*, const void ∗ *data*, uint32_t *size*)

Writes data into buffer from offset pos.

9.2.2.4 void ∗ BBuffer::data ()

The data.

9.2.2.5 uint32_t BBuffer::size ()

Size of the buffer in bytes.

9.2.3 Member Data Documentation

9.2.3.1 uint32_t BBuffer::osize `[private]`

9.2.3.2 uint32_t BBuffer::odatasize `[private]`

9.2.3.3 void∗ BBuffer::odata `[private]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BBuffer.h
- /src/cern/tms/beam/libBeam/BBuffer.cpp

9.3 BCond Class Reference

```
#include <BCond.h>
```

Public Member Functions

- BCond ()

  Thread conditional variable.

- ∼BCond ()
- int signal ()
- int wait ()
- int timedWait (int timeOutUs)

Private Attributes

- pthread_mutex_t omutex
- pthread_cond_t ocond

9.3.1 Constructor & Destructor Documentation

9.3.1.1 BCond::BCond ()

Thread conditional variable.

9.3.1.2 BCond::∼BCond ()

9.3.2 Member Function Documentation

9.3.2.1 int BCond::signal ()

9.3.2.2 int BCond::wait ()

9.3.2.3 int BCond::timedWait (int *timeOutUs*)

9.3.3 Member Data Documentation

9.3.3.1 pthread_mutex_t BCond::omutex `[private]`

9.3.3.2 pthread_cond_t BCond::ocond `[private]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BCond.h
- /src/cern/tms/beam/libBeam/BCond.cpp

9.4 BCondBool Class Reference

Thread conditional boolean.

`#include <BCondInt.h>`

Public Member Functions

- BCondBool ()
- ~BCondBool ()
- int set ()

    Set value. Wakes waiting.

- int clear ()

    Clear Value.

- int value ()

    Current value.

- int wait ()

    Wait until value is true.

- int timedWait (int timeOutUs)

    Wait until set, with timeout.

Private Attributes

- pthread_mutex_t omutex
- pthread_cond_t ocond
- int ovalue

9.4.1 Detailed Description

Thread conditional boolean.

9.4.2 Constructor & Destructor Documentation

9.4.2.1 BCondBool::BCondBool ()

9.4.2.2 BCondBool::~BCondBool ()

9.4.3 Member Function Documentation

9.4.3.1 int BCondBool::set ()

Set value. Wakes waiting.

9.4.3.2 int BCondBool::clear ()

Clear Value.

9.4.3.3 int BCondBool::value ()

Current value.

9.4.3.4 int BCondBool::wait ()

Wait until value is true.

9.4.3.5 int BCondBool::timedWait (int *timeOutUs*)

Wait until set, with timeout.

9.4.4 Member Data Documentation

9.4.4.1 pthread_mutex_t BCondBool::omutex `[private]`

9.4.4.2 pthread_cond_t BCondBool::ocond `[private]`

9.4.4.3 int BCondBool::ovalue `[private]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BCondInt.h
- /src/cern/tms/beam/libBeam/BCondInt.cpp

9.5 BCondInt Class Reference

Thread conditional integer.

```
#include <BCondInt.h>
```

Public Member Functions

- BCondInt ()
- ∼BCondInt ()
- void setValue (int value)

  Set value.
- int increment ()

  Increment.
- int decrement ()

  Decrement.
- int value ()

  Current value.
- int wait ()

  Wait until value is 0.
- int waitIncrement (int timeOutUs=0)

  Wait until value is 0 then increment.
- int waitNotZero ()

  Wait until value is not 0.
- int waitNotZeroDecrement ()

  Wait until value is not 0 and then decrement.
- int tryNotZeroDecrement ()

  Test if value is not 0, if not zero then decrement.
- int timedWait (int timeOutUs)

  Wait for the condition, with timeout.
- void operator++ (int)
- void operator– (int)

Private Attributes

- pthread_mutex_t omutex
- pthread_cond_t ocond
- int ovalue

9.5.1 Detailed Description

Thread conditional integer.

9.5.2 Constructor & Destructor Documentation

9.5.2.1 BCondInt::BCondInt ()

9.5.2.2 BCondInt::∼BCondInt ()

9.5.3 Member Function Documentation

9.5.3.1 void BCondInt::setValue (int *value*)

Set value.

9.5.3.2 int BCondInt::increment ()

Increment.

9.5.3.3 int BCondInt::decrement ()

Decrement.

9.5.3.4 int BCondInt::value ()

Current value.

9.5.3.5 int BCondInt::wait ()

Wait until value is 0.

9.5.3.6 int BCondInt::waitIncrement (int *timeOutUs* = `0`)

Wait until value is 0 then increment.

9.5.3.7 int BCondInt::waitNotZero ()

Wait until value is not 0.

9.5.3.8 int BCondInt::waitNotZeroDecrement ()

Wait until value is not 0 and then decrement.

9.5.3.9 int BCondInt::tryNotZeroDecrement ()

Test if value is not 0, if not zero then decrement.

9.5.3.10 int BCondInt::timedWait (int *timeOutUs*)

Wait for the condition, with timeout.

9.5.3.11 void BCondInt::operator++ (int) `[inline]`

9.5.3.12 void BCondInt::operator– (int) `[inline]`

9.5.4 Member Data Documentation

9.5.4.1 pthread_mutex_t BCondInt::omutex `[private]`

9.5.4.2 pthread_cond_t BCondInt::ocond `[private]`

9.5.4.3 int BCondInt::ovalue `[private]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BCondInt.h
- /src/cern/tms/beam/libBeam/BCondInt.cpp

9.6 BCondValue Class Reference

Thread conditional value.

```
#include <BCondInt.h>
```

Public Member Functions

- BCondValue ()
- ~BCondValue ()
- void setValue (int value)

  Set the value. Wakes waiting.

- int value ()

  Current value.

- int increment (int v=1)

  Increment. Wakes waiting.

- int decrement (int v=1)

  Decrement. Wakes waiting.

- int waitMoreThanOrEqual (int v, int decrement=0, int timeOutUs=0)

  Wait until value is at least the value given.

- int waitLessThanOrEqual (int v, int increment=0, int timeOutUs=0)

  Wait until value is equal to or below the value given.

- int waitLessThan (int v, int timeOutUs=0)

  Wait until value is equal to or below the value given.

- void operator+= (int v)

  Add to value. Wakes waiting.

- void operator-= (int v)

  Subtract from value. Wakes waiting.

- void operator++ (int)

  Increment value. Wakes waiting.

- void operator– (int)

  Decrement value. Wakes waiting.

Private Attributes

- pthread_mutex_t omutex
- pthread_cond_t ocond
- int ovalue

9.6.1 Detailed Description

Thread conditional value.

9.6.2 Constructor & Destructor Documentation

9.6.2.1 BCondValue::BCondValue ()

9.6.2.2 BCondValue::∼BCondValue ()

9.6.3 Member Function Documentation

9.6.3.1 void BCondValue::setValue (int *value*)

Set the value. Wakes waiting.

9.6.3.2 int BCondValue::value ()

Current value.

9.6.3.3 int BCondValue::increment (int *v* = `1`)

Increment. Wakes waiting.

9.6.3.4 int BCondValue::decrement (int *v* = `1`)

Decrement. Wakes waiting.

9.6.3.5 int BCondValue::waitMoreThanOrEqual (int *v*, int *decrement* = `0`, int *timeOutUs* = `0`)

Wait until value is at least the value given.

9.6.3.6 int BCondValue::waitLessThanOrEqual (int *v*, int *increment* = `0`, int *timeOutUs* = `0`)

Wait until value is equal to or below the value given.

9.6.3.7 int BCondValue::waitLessThan (int *v*, int *timeOutUs* = `0`)

Wait until value is equal to or below the value given.

9.6.3.8 void BCondValue::operator+= (int *v*) `[inline]`

Add to value. Wakes waiting.

9.6.3.9 void BCondValue::operator-= (int *v*) `[inline]`

Subtract from value. Wakes waiting.

9.6.3.10 void BCondValue::operator++ (int) `[inline]`

Increment value. Wakes waiting.

9.6.3.11 void BCondValue::operator-- (int) `[inline]`

Decrement value. Wakes waiting.

9.6.4 Member Data Documentation

9.6.4.1 pthread_mutex_t BCondValue::omutex `[private]`

9.6.4.2 pthread_cond_t BCondValue::ocond `[private]`

9.6.4.3 int BCondValue::ovalue `[private]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BCondInt.h
- /src/cern/tms/beam/libBeam/BCondInt.cpp

9.7 BCondWrap Class Reference

`#include <BCondInt.h>`

Public Member Functions

- BCondWrap ()
- ~BCondWrap ()
- void setValue (uint32_t value)

  Set the value. Wakes waiting.
- uint32_t value ()

  Current value.
- uint32_t increment (uint32_t v=1)

  Increment. Wakes waiting.
- uint32_t decrement (uint32_t v=1)

  Decrement. Wakes waiting.
- int waitMoreThanOrEqual (uint32_t v, uint32_t decrement=0, uint32_t timeOutUs=0)

  Wait until value is at least the value given.
- int waitLessThanOrEqual (uint32_t v, uint32_t increment=0, uint32_t timeOutUs=0)

  Wait until value is equal to or below the value given.
- int waitLessThan (uint32_t v, uint32_t timeOutUs=0)

  Wait until value is equal to or below the value given.
- void operator+= (int v)

  Add to value. Wakes waiting.
- void operator-= (int v)

  Subtract from value. Wakes waiting.
- void operator++ (int)

  Increment value. Wakes waiting.
- void operator– (int)

  Decrement value. Wakes waiting.

Private Member Functions

- int diff (uint32_t v)

Private Attributes

- pthread_mutex_t omutex
- pthread_cond_t ocond
- uint32_t ovalue

9.7.1 Constructor & Destructor Documentation

9.7.1.1 BCondWrap::BCondWrap ()

9.7.1.2 BCondWrap::∼BCondWrap ()

9.7.2 Member Function Documentation

9.7.2.1 void BCondWrap::setValue (uint32_t *value*)

Set the value. Wakes waiting.

9.7.2.2 uint32_t BCondWrap::value ()

Current value.

9.7.2.3 uint32_t BCondWrap::increment (uint32_t *v* = `1`)

Increment. Wakes waiting.

9.7.2.4 uint32_t BCondWrap::decrement (uint32_t *v* = `1`)

Decrement. Wakes waiting.

9.7.2.5 int BCondWrap::waitMoreThanOrEqual (uint32_t *v*, uint32_t *decrement* = `0`, uint32_t *timeOutUs* = `0`)

Wait until value is at least the value given.

9.7.2.6 int BCondWrap::waitLessThanOrEqual (uint32_t *v*, uint32_t *increment* = `0`, uint32_t *timeOutUs* = `0`)

Wait until value is equal to or below the value given.

9.7.2.7 int BCondWrap::waitLessThan (uint32_t *v*, uint32_t *timeOutUs* = `0`)

Wait until value is equal to or below the value given.

9.7.2.8 void BCondWrap::operator+= (int *v*) `[inline]`

Add to value. Wakes waiting.

9.7.2.9 void BCondWrap::operator-= (int *v*) `[inline]`

Subtract from value. Wakes waiting.

9.7.2.10 void BCondWrap::operator++ (int) `[inline]`

Increment value. Wakes waiting.

9.7.2.11 void BCondWrap::operator– (int) `[inline]`

Decrement value. Wakes waiting.

9.7.2.12 int BCondWrap::diff (uint32_t *v*) `[private]`

9.7.3 Member Data Documentation

9.7.3.1 pthread_mutex_t BCondWrap::omutex `[private]`

9.7.3.2 pthread_cond_t BCondWrap::ocond `[private]`

9.7.3.3 uint32_t BCondWrap::ovalue `[private]`

The documentation for this class was generated from the following files:

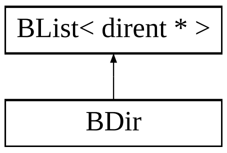
- /src/cern/tms/beam/libBeam/BCondInt.h
- /src/cern/tms/beam/libBeam/BCondInt.cpp

9.8 BDir Class Reference

File system directory class.

```
#include <BDir.h>
```

Inheritance diagram for BDir::



Public Member Functions

- BDir ()
- BDir (BString name)
- ~BDir ()
- BError open (BString name)

    Reads named directory.

- BError error ()

    Current value of error.

- BError read ()

    read/re-reads directory

- void clear ()

    Clears list.

- void setWild (BString wild)

    Set wildcard filter string used on read.

- void setSort (int on)

    Set alpha sort on/off.

- BString entryName (BIter i)

    Get filename.

- struct stat entryStat (BIter i)

    Get file stats.

- struct stat64 entryStat64 (BIter i)

    Get file stats 64.

Private Attributes

- BError oerror
- BString odirname
- BString owild
- int osort

9.8.1 Detailed Description

File system directory class.

9.8.2 Constructor & Destructor Documentation

9.8.2.1 BDir::BDir ()

9.8.2.2 BDir::BDir (BString *name*)

9.8.2.3 BDir::∼BDir ()

9.8.3 Member Function Documentation

9.8.3.1 BError BDir::open (BString *name*)

Reads named directory.

9.8.3.2 BError BDir::error ()

Current value of error.

9.8.3.3 BError BDir::read ()

read/re-reads directory

9.8.3.4 void BDir::clear () `[virtual]`

Clears list.

Reimplemented from BList< T >.

9.8.3.5 void BDir::setWild (BString *wild*)

Set wildcard filter string used on read.

9.8.3.6 void BDir::setSort (int *on*)

Set alpha sort on/off.

9.8.3.7 BString BDir::entryName (BIter *i*)

Get filename.

9.8.3.8 struct stat BDir::entryStat (BIter *i*) `[read]`

Get file stats.

9.8.3.9 struct stat64 BDir::entryStat64 (BIter *i*) `[read]`

Get file stats 64.

9.8.4 Member Data Documentation

9.8.4.1 BError BDir::oerror `[private]`

9.8.4.2 BString BDir::odirname `[private]`

9.8.4.3 BString BDir::owild `[private]`

9.8.4.4 int BDir::osort `[private]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BDir.h
- /src/cern/tms/beam/libBeam/BDir.cpp

9.9 BEntry Class Reference

Manipulate a name value pair.

`#include <BEntry.h>`

Public Member Functions

- BEntry ()
- BEntry (BString name, BString value)

  Set name and value.
- BEntry (BString line)

  Set name and value from white space deliminated string.
- BString getName ()

  Get the name.
- BString getValue ()

  Get the value.
- void setLine (BString line)

  Set name and value from white space deliminated string.
- void setName (BString name)

  Set the name.
- void setValue (BString value)

  Set the value.
- BString line ()

  Return name and value as padded single string.
- void print ()

  Print name and value.

Private Attributes

- BString oname
- BString ovalue

9.9.1 Detailed Description

Manipulate a name value pair.

9.9.2 Constructor & Destructor Documentation

9.9.2.1 BEntry::BEntry ()

9.9.2.2 BEntry::BEntry (BString *name*, BString *value*)

Set name and value.

9.9.2.3 BEntry::BEntry (BString *line*)

Set name and value from white space deliminated string.

9.9.3 Member Function Documentation

9.9.3.1 BString BEntry::getName ()

Get the name.

9.9.3.2 BString BEntry::getValue ()

Get the value.

9.9.3.3 void BEntry::setLine (BString *line*)

Set name and value from white space deliminated string.

9.9.3.4 void BEntry::setName (BString *name*)

Set the name.

9.9.3.5 void BEntry::setValue (BString *value*)

Set the value.

9.9.3.6 BString BEntry::line ()

Return name and value as padded single string.

9.9.3.7 void BEntry::print ()

Print name and value.

9.9.4 Member Data Documentation

9.9.4.1 BString BEntry::oname `[private]`

9.9.4.2 BString BEntry::ovalue `[private]`

The documentation for this class was generated from the following files:

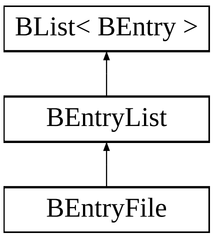
- /src/cern/tms/beam/libBeam/BEntry.h
- /src/cern/tms/beam/libBeam/BEntry.cpp

9.10 BEntryFile Class Reference

File of Entries.

```
#include <BEntry.h>
```

Inheritance diagram for BEntryFile::



Public Member Functions

- BEntryFile ()
- BEntryFile (BString filename)

  Opens entryfile.
- ∼BEntryFile ()
- int open (BString filename)

  Opens entryfile.
- int read ()

  Reads entry file and builds list.
- int write ()

  Writes list to entryfile.
- int writeList (BEntryList &l)

  Writes specified list to file.
- void clear ()

  Clears current list.

Private Attributes

- BString ofilename
- BString ocomments

9.10.1 Detailed Description

File of Entries.

9.10.2 Constructor & Destructor Documentation

9.10.2.1 BEntryFile::BEntryFile ()

9.10.2.2 BEntryFile::BEntryFile (BString *filename*)

Opens entryfile.

9.10.2.3 BEntryFile::∼BEntryFile ()

9.10.3 Member Function Documentation

9.10.3.1 int BEntryFile::open (BString *filename*)

Opens entryfile.

9.10.3.2 int BEntryFile::read ()

Reads entry file and builds list.

9.10.3.3 int BEntryFile::write ()

Writes list to entryfile.

9.10.3.4 int BEntryFile::writeList (BEntryList & *l*)

Writes specified list to file.

9.10.3.5 void BEntryFile::clear () `[virtual]`

Clears current list.

Reimplemented from BEntryList.

9.10.4 Member Data Documentation

9.10.4.1 BString BEntryFile::ofilename `[private]`

9.10.4.2 BString BEntryFile::ocomments `[private]`

The documentation for this class was generated from the following files:

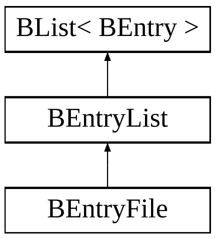
- /src/cern/tms/beam/libBeam/BEntry.h
- /src/cern/tms/beam/libBeam/BEntry.cpp

9.11 BEntryList Class Reference

List of Entries. Where an entry is a name value pair.

`#include <BEntry.h>`

Inheritance diagram for BEntryList::



Public Member Functions

- BEntryList ()
- int isSet (BString name)

    1 if name is in list and value is set

- BEntry ∗ find (BString name)

    Returns entry if name is found otherwise NULL.

- BString findValue (BString name)

    Returns value of name. Returns "" if name not found.

- int setValue (BString name, BString value)

    Set the value of name. Returns 0 if name not found.

- int setValueRaw (BString name, BString value)

    Raw setting of value without looking up existing entry.

- void deleteEntry (BString name)

    Deletes the entry.

- void print ()

    Print list.

- BString getString ()

    Return list as string. Each Entry padded and on a new line.

- void insert (BIter &i, const BEntry &item)

    Insert item before item.

- void del (BIter &i)

    Delete specified item.

- void clear ()

    Clear the list.

Private Attributes

- BIter olastPos

9.11.1 Detailed Description

List of Entries. Where an entry is a name value pair.

9.11.2 Constructor & Destructor Documentation

9.11.2.1 BEntryList::BEntryList ()

9.11.3 Member Function Documentation

9.11.3.1 int BEntryList::isSet (BString *name*)

1 if name is in list and value is set

9.11.3.2 BEntry ∗ BEntryList::find (BString *name*)

Returns entry if name is found otherwise NULL.

9.11.3.3 BString BEntryList::findValue (BString *name*)

Returns value of name. Returns "" if name not found.

9.11.3.4 int BEntryList::setValue (BString *name*, BString *value*)

Set the value of name. Returns 0 if name not found.

9.11.3.5 int BEntryList::setValueRaw (BString *name*, BString *value*)

Raw setting of value without looking up existing entry.

9.11.3.6 void BEntryList::deleteEntry (BString *name*)

Deletes the entry.

9.11.3.7 void BEntryList::print ()

Print list.

9.11.3.8 BString BEntryList::getString ()

Return list as string. Each Entry padded and on a new line.

9.11.3.9 void BEntryList::insert (BIter & *i*, const BEntry & *item*) `[virtual]`

Insert item before item.

Reimplemented from BList< BEntry >.

9.11.3.10 void BEntryList::del (BIter & *i*) `[virtual]`

Delete specified item.

Reimplemented from BList< BEntry >.

9.11.3.11 void BEntryList::clear () `[virtual]`

Clear the list.

Reimplemented from BList< BEntry >.

Reimplemented in BEntryFile.

9.11.4 Member Data Documentation

9.11.4.1 BIter BEntryList::olastPos `[private]`

The documentation for this class was generated from the following files:

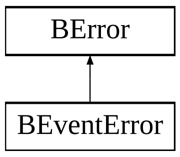
- /src/cern/tms/beam/libBeam/BEntry.h
- /src/cern/tms/beam/libBeam/BEntry.cpp

9.12 BError Class Reference

Error return class.

`#include <BError.h>`

Inheritance diagram for BError::



Public Types

- enum Type { NONE = 0, ERROR = 1 }

Public Member Functions

- BError (int errNo=NONE, BString errStr="")

    Create object.

- BError (BString errStr)

    Create with error set and error string.

- BError copy ()

    Return an independant copy.

- BError & set (int errNo, BString errStr="")

    Set error number and message.

- BError & setError (BString errStr="")

    Set error type ERROR with optional message.

- BString getString () const

    Get error message.

- int getErrorNo () const

    Get The error number.

- operator int () const

    Return error number.

Private Attributes

- int oerrNo
- BString oerrStr

9.12.1 Detailed Description

Error return class.

9.12.2 Member Enumeration Documentation

9.12.2.1 enum BError::Type

Enumerator:

NONE

ERROR

9.12.3 Constructor & Destructor Documentation

9.12.3.1 BError::BError (int *errNo* = `NONE`, BString *errStr* = "")

Create object.

9.12.3.2 BError::BError (BString *errStr*)

Create with error set and error string.

9.12.4 Member Function Documentation

9.12.4.1 BError BError::copy ()

Return an independant copy.

9.12.4.2 BError & BError::set (int *errNo*, BString *errStr* = "")

Set error number and message.

9.12.4.3 BError & BError::setError (BString *errStr* = "")

Set error type ERROR with optional message.

9.12.4.4 BString BError::getString () const

Get error message.

9.12.4.5 int BError::getErrorNo () const

Get The error number.

9.12.4.6 BError::operator int () const

Return error number.

9.12.5 Member Data Documentation

9.12.5.1 int BError::oerrNo `[private]`

9.12.5.2 BString BError::oerrStr `[private]`

The documentation for this class was generated from the following files:

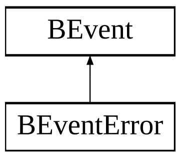
- /src/cern/tms/beam/libBeam/BError.h
- /src/cern/tms/beam/libBeam/BError.cpp

9.13 BEvent Class Reference

This class provides a base class for all event objects that can be sent over the events interface.

`#include <BEvent.h>`

Inheritance diagram for BEvent::



Public Member Functions

- BEvent (uint32_t type)
- virtual ∼BEvent ()
- uint32_t getType ()
- virtual BError getBinary (void ∗data, uint32_t &size)
- virtual BError setBinary (void ∗data, uint32_t &size)

Private Attributes

- uint32_t otype
    The event type.

9.13.1 Detailed Description

This class provides a base class for all event objects that can be sent over the events interface.

9.13.2 Constructor & Destructor Documentation

9.13.2.1 BEvent::BEvent (uint32_t *type*)

9.13.2.2 BEvent::∼BEvent () `[virtual]`

9.13.3 Member Function Documentation

9.13.3.1 uint32_t BEvent::getType ()

9.13.3.2 BError BEvent::getBinary (void ∗ *data*, uint32_t & *size*) `[virtual]`

Reimplemented in BEventError.

9.13.3.3 BError BEvent::setBinary (void ∗ *data*, uint32_t & *size*) `[virtual]`

Reimplemented in BEventError.

9.13.4 Member Data Documentation

9.13.4.1 uint32_t BEvent::otype `[private]`

The event type.

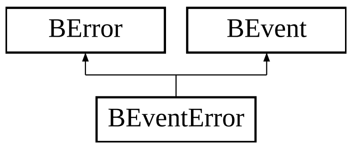
The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BEvent.h
- /src/cern/tms/beam/libBeam/BEvent.cpp

9.14 BEventError Class Reference

```
#include <BEvent.h>
```

Inheritance diagram for BEventError::



Public Member Functions

- BEventError (int errNo=NONE, BString errStr="")
- BError getBinary (void ∗data, uint32_t &size)
- BError setBinary (void ∗data, uint32_t &size)

9.14.1 Constructor & Destructor Documentation

9.14.1.1 BEventError::BEventError (int *errNo* = `NONE`, BString *errStr* = "")

9.14.2 Member Function Documentation

9.14.2.1 BError BEventError::getBinary (void ∗ *data*, uint32_t & *size*) `[virtual]`

Reimplemented from BEvent.

9.14.2.2 BError BEventError::setBinary (void ∗ *data*, uint32_t & *size*) `[virtual]`

Reimplemented from BEvent.

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BEvent.h
- /src/cern/tms/beam/libBeam/BEvent.cpp

9.15 BEventInt Class Reference

This class provides an interface for sending simple integer events via a file descriptor. This allows threads to send events that can be picked up by the poll system call.

`#include <BEvent.h>`

Public Member Functions

- BEventInt ()
- ~BEventInt ()
- BError sendEvent (int event)

  Send an event.
- BError getEvent (int &event, int timeOutUs=-1)

  Receive the event.
- int getFd ()

Private Attributes

- int ofds [2]

  File descriptors for pipe.

9.15.1 Detailed Description

This class provides an interface for sending simple integer events via a file descriptor. This allows threads to send events that can be picked up by the poll system call.

9.15.2 Constructor & Destructor Documentation

9.15.2.1 BEventInt::BEventInt ()

9.15.2.2 BEventInt::~BEventInt ()

9.15.3 Member Function Documentation

9.15.3.1 BError BEventInt::sendEvent (int *event*)

Send an event.

9.15.3.2 BError BEventInt::getEvent (int & *event*, int *timeOutUs* = `-1`)

Receive the event.

9.15.3.3 int BEventInt::getFd ()

9.15.4 Member Data Documentation

9.15.4.1 int BEventInt::ofds[2] `[private]`

File descriptors for pipe.

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BEvent.h
- /src/cern/tms/beam/libBeam/BEvent.cpp

9.16 BEventPipe Class Reference

This class provides a base interface for sending events via a pipe. This allows threads to send events that can be picked up by the poll system call.

```
#include <BEvent.h>
```

Public Member Functions

- BEventPipe ()
- ∼BEventPipe ()
- BError sendEvent (BEvent ∗event)

  Send an event.
- BError getEvent (BEvent ∗event, int timeOutUs=-1)

  Receive the event.
- int getReceiveFd ()

  returns the receive file descriptor for the poll system call

Private Attributes

- int ofds [2]

  File descriptors for pipe.

9.16.1 Detailed Description

This class provides a base interface for sending events via a pipe. This allows threads to send events that can be picked up by the poll system call.

9.16.2 Constructor & Destructor Documentation

9.16.2.1 BEventPipe::BEventPipe ()

9.16.2.2 BEventPipe::∼BEventPipe ()

9.16.3 Member Function Documentation

9.16.3.1 BError BEventPipe::sendEvent (BEvent ∗ *event*)

Send an event.

9.16.3.2 BError BEventPipe::getEvent (BEvent ∗ *event*, int *timeOutUs* = `-1`)

Receive the event.

9.16.3.3 int BEventPipe::getReceiveFd ()

returns the receive file descriptor for the poll system call

9.16.4 Member Data Documentation

9.16.4.1 int BEventPipe::ofds[2] `[private]`

File descriptors for pipe.

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BEvent.h
- /src/cern/tms/beam/libBeam/BEvent.cpp

9.17 BFile Class Reference

File operations class.

`#include <BFile.h>`

Public Member Functions

- BFile ()
- BFile (BString name, BString mode)

  Create opened specifed file.

- BFile (const BFile &file)

  Create opened specified file.

- ∼BFile ()
- BError open (BString name, BString mode)

  Open file.

- BError open (FILE ∗file)

  Assign object to opened file handle.

- BError close ()

  Close file.

- BError error ()

  Returns current error state.

- FILE ∗ getFd ()

  File descriptor.

- int length ()

  File size in bytes.

- int setVBuf (char ∗buf, int mode, size_t size)

  Set stream buffering options.

- int read (void ∗buf, int nbytes)

  Read from file.

- int readString (BString &str)

  Read string. (ref fgets).

- int write (const void ∗buf, int nbytes)

  Write to file.

- int writeString (const BString &str)

  Write string to file.

- int seek (int pos, int whence)

Set seek position.

- int printf (const char ∗fmt,...)

  Formated print into the file.

- BFile & operator= (const BFile &file)

Private Attributes

- FILE ∗ ofile
- BString ofileName
- BString omode
- BError oerror

9.17.1 Detailed Description

File operations class.

9.17.2 Constructor & Destructor Documentation

9.17.2.1 BFile::BFile ()

9.17.2.2 BFile::BFile (BString *name*, BString *mode*)

Create opened specifed file.

9.17.2.3 BFile::BFile (const BFile & *file*)

Create opened specified file.

9.17.2.4 BFile::∼BFile ()

9.17.3 Member Function Documentation

9.17.3.1 BError BFile::open (BString *name*, BString *mode*)

Open file.

9.17.3.2 BError BFile::open (FILE ∗ *file*)

Assign object to opened file handle.

9.17.3.3 BError BFile::close ()

Close file.

9.17.3.4 BError BFile::error ()

Returns current error state.

9.17.3.5 FILE ∗ BFile::getFd ()

File descriptor.

9.17.3.6 int BFile::length ()

File size in bytes.

9.17.3.7 int BFile::setVBuf (char ∗ *buf*, int *mode*, size_t *size*)

Set stream buffering options.

9.17.3.8 int BFile::read (void ∗ *buf*, int *nbytes*)

Read from file.

9.17.3.9 int BFile::readString (BString & *str*)

Read string. (ref fgets).

9.17.3.10 int BFile::write (const void ∗ *buf*, int *nbytes*)

Write to file.

9.17.3.11 int BFile::writeString (const BString & *str*)

Write string to file.

9.17.3.12 int BFile::seek (int *pos*, int *whence*)

Set seek position.

9.17.3.13 int BFile::printf (const char ∗ *fmt*, ...)

Formated print into the file.

9.17.3.14 BFile & BFile::operator= (const BFile & *file*)

9.17.4 Member Data Documentation

9.17.4.1 FILE* BFile::ofile `[private]`

9.17.4.2 BString BFile::ofileName `[private]`

9.17.4.3 BString BFile::omode `[private]`

9.17.4.4 BError BFile::oerror `[private]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BFile.h
- /src/cern/tms/beam/libBeam/BFile.cpp

9.18 BIter Class Reference

Iterator for BList.

`#include <BList.h>`

Public Member Functions

- BIter (void ∗i=0)
- operator void ∗ ()
- int operator== (const BIter &i)

Private Attributes

- void ∗ oi

9.18.1 Detailed Description

Iterator for BList.

9.18.2 Constructor & Destructor Documentation

9.18.2.1 BIter::BIter (void ∗ *i* = 0) `[inline]`

9.18.3 Member Function Documentation

9.18.3.1 BIter::operator void ∗ () `[inline]`

9.18.3.2 int BIter::operator== (const BIter & *i*) `[inline]`

9.18.4 Member Data Documentation

9.18.4.1 void∗ BIter::oi `[private]`

The documentation for this class was generated from the following file:

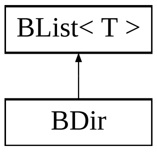
- /src/cern/tms/beam/libBeam/BList.h

9.19 BList< T > Class Template Reference

Template based list class.

`#include <BList.h>`

Inheritance diagram for BList< T >::



Public Types

- typedef int(∗ SortFunc)(T &a, T &b)

    Prototype for sorting function.

Public Member Functions

- BList ()
- BList (const BList< T > &l)
- virtual ∼BList ()
- void start (BIter &i) const

    Iterator to start of list.

- BIter begin () const

    Iterator for start of list.

- BIter end () const

    Iterator for end of list.

- BIter end (BIter &i) const

    Iterator for end of list.

- void next (BIter &i) const

    Iterator for next item in list.

- void prev (BIter &i)

    Iterator for previous item in list.

- BIter goTo (int pos)

    Iterator for pos item in list.

- int position (BIter i)

    Postition in list item with iterator i.

- unsigned int number ()

Number of items in list.

- int isEnd (BIter i) const

  True if iterator refers to last item.

- T & front ()

  Get first item in list.

- T & rear ()

  Get last item in list.

- T & get (BIter i)

  Get item specified by iterator in list.

- const T & get (BIter i) const

  Get item specified by iterator in list.

- void append (const T &item)

  Append item to list.

- virtual void insert (BIter &i, const T &item)

  Insert item before item.

- void insertAfter (BIter &i, const T &item)

  Insert item after item.

- virtual void clear ()

  Clear the list.

- virtual void del (BIter &i)

  Delete specified item.

- void deleteLast ()

  Delete last item.

- void deleteFirst ()

  Delete fisrt item.

- void push (const T &i)

  Push item onto list.

- T pop ()

  Pop item from list deleteing item.

- void queueAdd (const T &i)

  Add item to end of list.

- T queueGet ()

  Get item from front of list deleteing item.

- void append (const BList< T > &l)

  Append list to list.

- void swap (BIter i1, BIter i2)

  Swap two items in list.

- void sort ()

  Sort list based on get(i) values.

- void sort (SortFunc func)

  Sort list based on Sort func.

- BList< T > & operator= (const BList< T > &l)
- T & operator[ ] (int i)
- const T & operator[ ] (int i) const
- T & operator[ ] (BIter i)
- const T & operator[ ] (BIter i) const
- BList< T > operator+ (const BList< T > &l) const

Protected Member Functions

- virtual Node ∗ nodeGet (BIter i)
- virtual const Node ∗ nodeGet (BIter i) const
- virtual Node ∗ nodeCreate (const T &item)

Protected Attributes

- Node ∗ onodes
- unsigned int olength

Private Member Functions

- virtual Node ∗ nodeCreate ()

Classes

- class Node

9.19.1 Detailed Description

template<class T> class BList< T >

Template based list class.

9.19.2 Member Typedef Documentation

9.19.2.1 template<class T> typedef int(∗ BList< T >::SortFunc)(T &a, T &b)

Prototype for sorting function.

9.19.3 Constructor & Destructor Documentation

9.19.3.1 template<class T> BList< T >::BList () `[inline]`

9.19.3.2 template<class T> BList< T >::BList (const BList< T > & *l*) `[inline]`

9.19.3.3 template<class T> BList< T >::~BList () `[inline, virtual]`

9.19.4 Member Function Documentation

9.19.4.1 template<class T> void BList< T >::start (BIter & *i*) const `[inline]`

Iterator to start of list.

9.19.4.2 template<class T> BIter BList< T >::begin () const `[inline]`

Iterator for start of list.

9.19.4.3 template<class T> BIter BList< T >::end () const `[inline]`

Iterator for end of list.

9.19.4.4 template<class T> BIter BList< T >::end (BIter & *i*) const `[inline]`

Iterator for end of list.

9.19.4.5 template<class T> void BList< T >::next (BIter & *i*) const `[inline]`

Iterator for next item in list.

9.19.4.6 template<class T> void BList< T >::prev (BIter & *i*) `[inline]`

Iterator for previous item in list.

9.19.4.7 template<class T> BIter BList< T >::goTo (int *pos*) `[inline]`

Iterator for pos item in list.

9.19.4.8 template<class T> int BList< T >::position (BIter *i*) `[inline]`

Postition in list item with iterator i.

9.19.4.9 template<class T> unsigned int BList< T >::number () `[inline]`

Number of items in list.

9.19.4.10 template<class T> int BList< T >::isEnd (BIter *i*) const `[inline]`

True if iterator refers to last item.

9.19.4.11 template<class T> T & BList< T >::front () `[inline]`

Get first item in list.

9.19.4.12 template<class T> T & BList< T >::rear () `[inline]`

Get last item in list.

9.19.4.13 template<class T> T & BList< T >::get (BIter *i*) `[inline]`

Get item specified by iterator in list.

9.19.4.14 template<class T> const T & BList< T >::get (BIter *i*) const `[inline]`

Get item specified by iterator in list.

9.19.4.15 template<class T> void BList< T >::append (const T & *item*) `[inline]`

Append item to list.

9.19.4.16 template<class T> void BList< T >::insert (BIter & *i*, const T & *item*) `[inline, virtual]`

Insert item before item.

Reimplemented in BEntryList.

9.19.4.17 template<class T> void BList< T >::insertAfter (BIter & *i*, const T & *item*) `[inline]`

Insert item after item.

9.19.4.18 template<class T> void BList< T >::clear () `[inline, virtual]`

Clear the list.

Reimplemented in BDir, BEntryList, and BEntryFile.

9.19.4.19 template<class T> void BList< T >::del (BIter & *i*) `[inline, virtual]`

Delete specified item.

Reimplemented in BEntryList.

9.19.4.20 template<class T> void BList< T >::deleteLast () `[inline]`

Delete last item.

9.19.4.21 template<class T> void BList< T >::deleteFirst () `[inline]`

Delete fisrt item.

9.19.4.22 template<class T> void BList< T >::push (const T & *i*) `[inline]`

Push item onto list.

9.19.4.23 template<class T> T BList< T >::pop () `[inline]`

Pop item from list deleteing item.

9.19.4.24 template<class T> void BList< T >::queueAdd (const T & *i*) `[inline]`

Add item to end of list.

9.19.4.25 template<class T> T BList< T >::queueGet () `[inline]`

Get item from front of list deleteing item.

9.19.4.26 template<class T> void BList< T >::append (const BList< T > & *l*) `[inline]`

Append list to list.

9.19.4.27 template<class T> void BList< T >::swap (BIter *i1*, BIter *i2*) `[inline]`

Swap two items in list.

9.19.4.28 template<class T> void BList< T >::sort () `[inline]`

Sort list based on get(i) values.

9.19.4.29 template<class T> void BList< T >::sort (SortFunc *func*) `[inline]`

Sort list based on Sort func.

9.19.4.30 template<class T> BList< T > & BList< T >::operator= (const BList< T > & *l*) `[inline]`

9.19.4.31]

template<class T> T & **BList**< T >::operator[ ] (int *i*) `[inline]`

9.19.4.32]

template<class T> const T & **BList**< T >::operator[ ] (int *i*) const `[inline]`

9.19.4.33]

template<class T> T & **BList**< T >::operator[ ] (**BIter** *i*) `[inline]`

9.19.4.34]

template<class T> const T & **BList**< T >::operator[ ] (**BIter** *i*) const `[inline]`

9.19.4.35 template<class T> BList< T > BList< T >::operator+ (const BList< T > & *l*) const `[inline]`

9.19.4.36 template<class T> BList< T >::Node ∗ BList< T >::nodeGet (BIter *i*) `[inline, protected, virtual]`

9.19.4.37 template<class T> const BList< T >::Node ∗ BList< T >::nodeGet (BIter *i*) const `[inline, protected, virtual]`

9.19.4.38 template<class T> BList< T >::Node ∗ BList< T >::nodeCreate (const T & *item*) `[inline, protected, virtual]`

9.19.4.39 template<class T> BList< T >::Node ∗ BList< T >::nodeCreate () `[inline, private, virtual]`

9.19.5 Member Data Documentation

9.19.5.1 template<class T> Node∗ BList< T >::onodes `[protected]`

9.19.5.2 template<class T> unsigned int BList< T >::olength `[protected]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BList.h
- /src/cern/tms/beam/libBeam/BList_func.h

9.20 BList< T >::Node Class Reference

```
#include <BList.h>
```

Public Member Functions

- Node (const T &i)

Public Attributes

- Node ∗ next
- Node ∗ prev
- T item

template<class T> class BList< T >::Node

9.20.1 Constructor & Destructor Documentation

9.20.1.1 template<class T> BList< T >::Node::Node (const T & *i*) `[inline]`

9.20.2 Member Data Documentation

9.20.2.1 template<class T> Node∗ BList< T >::Node::next

9.20.2.2 template<class T> Node∗ BList< T >::Node::prev

9.20.2.3 template<class T> T BList< T >::Node::item

The documentation for this class was generated from the following file:

- /src/cern/tms/beam/libBeam/BList.h

9.21 BMutex Class Reference

Mutex class.

```
#include <BMutex.h>
```

Public Member Functions

- BMutex ()
- BMutex (const BMutex &mutex)
- ~BMutex ()
- int lock ()

  Set lock, wait in necessary.
- int unlock ()

  Unlock the lock.
- int tryLock ()

  Test the lock.
- BMutex & operator= (const BMutex &mutex)

Private Attributes

- pthread_mutex_t omutex

9.21.1 Detailed Description

Mutex class.

9.21.2 Constructor & Destructor Documentation

9.21.2.1 BMutex::BMutex ()

9.21.2.2 BMutex::BMutex (const BMutex & *mutex*)

9.21.2.3 BMutex::~BMutex ()

9.21.3 Member Function Documentation

9.21.3.1 int BMutex::lock ()

Set lock, wait in necessary.

9.21.3.2 int BMutex::unlock ()

Unlock the lock.

9.21.3.3 int BMutex::tryLock ()

Test the lock.

9.21.3.4 BMutex & BMutex::operator= (const BMutex & *mutex*)

9.21.4 Member Data Documentation

9.21.4.1 pthread_mutex_t BMutex::omutex `[private]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BMutex.h
- /src/cern/tms/beam/libBeam/BMutex.cpp

9.22 BNameValue< T > Class Template Reference

```
#include <BNameValue.h>
```

Public Member Functions

- BNameValue ()
- BNameValue (BString name, const T &value)
- BString getName ()
- T & getValue ()

Private Attributes

- BString oname
- T ovalue

template<class T> class BNameValue< T >

9.22.1 Constructor & Destructor Documentation

9.22.1.1 template<class T> BNameValue< T >::BNameValue () `[inline]`

9.22.1.2 template<class T> BNameValue< T >::BNameValue (BString *name*, const T & *value*) `[inline]`

9.22.2 Member Function Documentation

9.22.2.1 template<class T> BString BNameValue< T >::getName () `[inline]`

9.22.2.2 template<class T> T& BNameValue< T >::getValue () `[inline]`

9.22.3 Member Data Documentation

9.22.3.1 template<class T> BString BNameValue< T >::oname `[private]`

9.22.3.2 template<class T> T BNameValue< T >::ovalue `[private]`

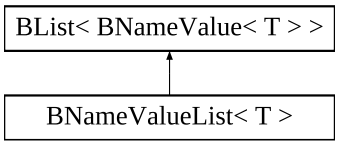
The documentation for this class was generated from the following file:

- /src/cern/tms/beam/libBeam/BNameValue.h

9.23 BNameValueList< T > Class Template Reference

`#include <BNameValue.h>`

Inheritance diagram for BNameValueList< T >::



Public Member Functions

- T ∗ find (BString name)

template<class T> class BNameValueList< T >

9.23.1 Member Function Documentation

9.23.1.1 template<class T> T∗ BNameValueList< T >::find (BString *name*) `[inline]`

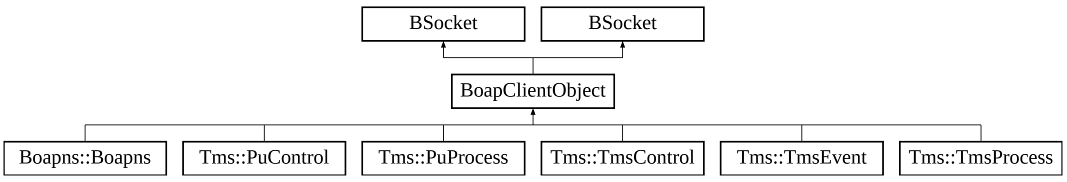
The documentation for this class was generated from the following file:

- /src/cern/tms/beam/libBeam/BNameValue.h

9.24 BoapClientObject Class Reference

```
#include <BoapSimple.h>
```

Inheritance diagram for BoapClientObject::



Public Member Functions

- BoapClientObject (BString name="")
- BError connectService (BString name)

    Connects to the named service.

- BError disconnectService ()

    Disconnects from the named service.

- BString getServiceName ()

    Get the name of the service.

- BError ping (BUInt32 &apiVersion)

    Pings the connection and finds the remotes version number.

- BError setConnectionPriority (BoapPriority priority)

    Sets the connection priority.

- void setMaxLength (BUInt32 maxLength)

    Sets the maximum packet length.

- void setTimeout (int timeout)

    Sets the timeout in micro seconds. -1 is wait indefinitely.

- BoapClientObject (BString name)
- BError connectService (BString name)

Protected Member Functions

- BError pingLocked (BUInt32 &apiVersion)
- BError checkApiVersion ()
- BError performCall (BoapPacket &tx, BoapPacket &rx)

    Performs a RPC call to the named service.

- BError performSend (BoapPacket &tx)

    Performs a send to the named service.

- BError performRecv (BoapPacket &rx)

  Performs a receive.

- BError performSend (BoapPacket &tx)
- BError performRecv (BoapPacket &rx)
- BError performCall (BoapPacket &tx, BoapPacket &rx)

Protected Attributes

- BString oname
- BUInt32 oapiVersion
- BoapPriority opriority
- BoapService oservice
- int oconnected
- BUInt32 omaxLength
- BoapPacket otx
- BoapPacket orx
- BMutex olock
- int otimeout
- int oreconnect

9.24.1 Constructor & Destructor Documentation

9.24.1.1 BoapClientObject::BoapClientObject (BString *name* = "")

9.24.1.2 BoapClientObject::BoapClientObject (BString *name*)

9.24.2 Member Function Documentation

9.24.2.1 BError BoapClientObject::connectService (BString *name*)

Connects to the named service.

9.24.2.2 BError BoapClientObject::disconnectService ()

Disconnects from the named service.

9.24.2.3 BString BoapClientObject::getServiceName ()

Get the name of the service.

9.24.2.4 BError BoapClientObject::ping (BUInt32 & *apiVersion*)

Pings the connection and finds the remotes version number.

9.24.2.5 BError BoapClientObject::setConnectionPriority (BoapPriority *priority*)

Sets the connection priority.

9.24.2.6 void BoapClientObject::setMaxLength (BUInt32 *maxLength*)

Sets the maximum packet length.

9.24.2.7 void BoapClientObject::setTimeout (int *timeout*)

Sets the timeout in micro seconds. -1 is wait indefinitely.

9.24.2.8 BError BoapClientObject::pingLocked (BUInt32 & *apiVersion*) `[protected]`

9.24.2.9 BError BoapClientObject::checkApiVersion () `[protected]`

9.24.2.10 BError BoapClientObject::performCall (BoapPacket & *tx*, BoapPacket & *rx*) `[protected]`

Performs a RPC call to the named service.

9.24.2.11 BError BoapClientObject::performSend (BoapPacket & *tx*) `[protected]`

Performs a send to the named service.

9.24.2.12 BError BoapClientObject::performRecv (BoapPacket & *rx*) `[protected]`

Performs a receive.

9.24.2.13 BError BoapClientObject::connectService (BString *name*)

9.24.2.14 BError BoapClientObject::performSend (BoapPacket & *tx*) `[protected]`

9.24.2.15 BError BoapClientObject::performRecv (BoapPacket & *rx*) `[protected]`

9.24.2.16 BError BoapClientObject::performCall (BoapPacket & *tx*, BoapPacket & *rx*) `[protected]`

9.24.3 Member Data Documentation

9.24.3.1 BString BoapClientObject::oname `[protected]`

9.24.3.2 BUInt32 BoapClientObject::oapiVersion `[protected]`

9.24.3.3 BoapPriority BoapClientObject::opriority `[protected]`

9.24.3.4 BoapService BoapClientObject::oservice `[protected]`

9.24.3.5 int BoapClientObject::oconnected `[protected]`

9.24.3.6 BUInt32 BoapClientObject::omaxLength `[protected]`

9.24.3.7 BoapPacket BoapClientObject::otx `[protected]`

9.24.3.8 BoapPacket BoapClientObject::orx `[protected]`

9.24.3.9 BMutex BoapClientObject::olock `[protected]`

9.24.3.10 int BoapClientObject::otimeout `[protected]`

9.24.3.11 int BoapClientObject::oreconnect `[protected]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/Boap.h
- /src/cern/tms/beam/libBeam/BoapSimple.h
- /src/cern/tms/beam/libBeam/Boap.cpp
- /src/cern/tms/beam/libBeam/BoapSimple.cc

9.25 Boapns::BoapEntry Class Reference

```
#include <BoapnsD.h>
```

Public Member Functions

- BoapEntry ()
- BoapEntry (BString pname, BString phostName, BList< BString > paddressList, UInt32 pport, UInt32 pservice)

Public Attributes

- BString name
- BString hostName
- BList< BString > addressList
- UInt32 port
- UInt32 service

9.25.1 Constructor & Destructor Documentation

9.25.1.1 Boapns::BoapEntry::BoapEntry ()

9.25.1.2 Boapns::BoapEntry::BoapEntry (BString *pname*, BString *phostName*, BList< BString > *paddressList*, UInt32 *pport*, UInt32 *pservice*)

9.25.2 Member Data Documentation

9.25.2.1 BString Boapns::BoapEntry::name

9.25.2.2 BString Boapns::BoapEntry::hostName

9.25.2.3 BList<BString> Boapns::BoapEntry::addressList

9.25.2.4 UInt32 Boapns::BoapEntry::port

9.25.2.5 UInt32 Boapns::BoapEntry::service

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BoapnsD.h
- /src/cern/tms/beam/libBeam/BoapnsD.cc

9.26 BoapFuncEntry Class Reference

`#include <BoapSimple.h>`

Public Member Functions

- BoapFuncEntry (int cmd, BoapFunc func)
- BoapFuncEntry (int cmd, BoapFunc func)

Public Attributes

- UInt32 ocmd
- BoapFunc ofunc

9.26.1 Constructor & Destructor Documentation

9.26.1.1 BoapFuncEntry::BoapFuncEntry (int *cmd*, BoapFunc *func*)

9.26.1.2 BoapFuncEntry::BoapFuncEntry (int *cmd*, BoapFunc *func*)

9.26.2 Member Data Documentation

9.26.2.1 UInt32 BoapFuncEntry::ocmd

9.26.2.2 BoapFunc BoapFuncEntry::ofunc

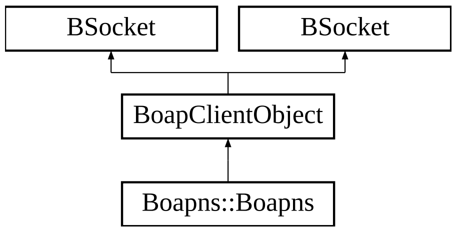
The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/Boap.h
- /src/cern/tms/beam/libBeam/BoapSimple.h
- /src/cern/tms/beam/libBeam/Boap.cpp
- /src/cern/tms/beam/libBeam/BoapSimple.cc

9.27 Boapns::Boapns Class Reference

```
#include <BoapnsC.h>
```

Inheritance diagram for Boapns::Boapns::



Public Member Functions

- Boapns (BString name="")
- BError getVersion (BString &version)
- BError getEntryList (BList< BoapEntry > &entryList)
- BError getEntry (BString name, BoapEntry &entry)
- BError addEntry (BoapEntry entry)
- BError delEntry (BString name)
- BError getNewName (BString &name)

9.27.1 Constructor & Destructor Documentation

9.27.1.1 Boapns::Boapns::Boapns (BString *name* = "")

9.27.2 Member Function Documentation

9.27.2.1 BError Boapns::Boapns::getVersion (BString & *version*)

9.27.2.2 BError Boapns::Boapns::getEntryList (BList< BoapEntry > & *entryList*)

9.27.2.3 BError Boapns::Boapns::getEntry (BString *name*, BoapEntry & *entry*)

9.27.2.4 BError Boapns::Boapns::addEntry (BoapEntry *entry*)

9.27.2.5 BError Boapns::Boapns::delEntry (BString *name*)

9.27.2.6 BError Boapns::Boapns::getNewName (BString & *name*)

The documentation for this class was generated from the following file:

- /src/cern/tms/beam/libBeam/BoapnsC.h

9.28 BoapPacket Class Reference

```
#include <BoapSimple.h>
```

Public Member Functions

- BoapPacket ()
- ∼BoapPacket ()
- int resize (int size)
- BError setData (void ∗data, int nbytes)
- int nbytes ()
- char ∗ data ()
- int peekHead (BoapPacketHead &head)
- UInt32 getCmd ()
- int pushHead (BoapPacketHead &head)
- int push (Int8 v)
- int push (UInt8 v)
- int push (Int16 v)
- int push (UInt16 v)
- int push (Int32 v)
- int push (UInt32 v)
- int push (Int64 v)
- int push (UInt64 v)
- int push (const BString &v)
- int push (Double v)
- int push (const BError &v)
- int push (UInt32 nBytes, const void ∗data, char ∗swapType="1")
- int popHead (BoapPacketHead &head)
- int pop (Int8 &v)
- int pop (UInt8 &v)
- int pop (Int16 &v)
- int pop (UInt16 &v)
- int pop (Int32 &v)
- int pop (UInt32 &v)
- int pop (Int64 &v)
- int pop (UInt64 &v)
- int pop (BString &v)
- int pop (Double &v)
- int pop (BError &v)
- int pop (UInt32 nBytes, void ∗data, char ∗swapType="1")
- BoapPacket ()
- ∼BoapPacket ()
- int resize (int size)
- BError setData (void ∗data, int nbytes)
- int nbytes ()
- char ∗ data ()
- int pushHead (BoapPacketHead &head)
- int push (Int8 v)
- int push (UInt8 v)

- int push (Int16 v)
- int push (UInt16 v)
- int push (Int32 v)
- int push (UInt32 v)
- int push (BString &v)
- int push (Double v)
- int push (BError &v)
- int push (UInt32 nBytes, const void ∗data)
- int popHead (BoapPacketHead &head)
- int pop (Int8 &v)
- int pop (UInt8 &v)
- int pop (Int16 &v)
- int pop (UInt16 &v)
- int pop (Int32 &v)
- int pop (UInt32 &v)
- int pop (BString &v)
- int pop (Double &v)
- int pop (BError &v)
- int pop (UInt32 nBytes, void ∗data)

Private Member Functions

- void copyWithSwap (void ∗dst, const void ∗src, UInt32 nBytes, char ∗swapType)
- void updateLen ()
- void updateLen ()

Private Attributes

- int osize
- int onbytes
- char ∗ odata
- int opos

9.28.1 Constructor & Destructor Documentation

9.28.1.1 BoapPacket::BoapPacket ()

9.28.1.2 BoapPacket::∼BoapPacket ()

9.28.1.3 BoapPacket::BoapPacket ()

9.28.1.4 BoapPacket::∼BoapPacket ()

9.28.2 Member Function Documentation

9.28.2.1 int BoapPacket::resize (int *size*)

9.28.2.2 BError BoapPacket::setData (void ∗ *data*, int *nbytes*)

9.28.2.3 int BoapPacket::nbytes ()

9.28.2.4 char ∗ BoapPacket::data ()

9.28.2.5 int BoapPacket::peekHead (BoapPacketHead & *head*)

9.28.2.6 UInt32 BoapPacket::getCmd ()

9.28.2.7 int BoapPacket::pushHead (BoapPacketHead & *head*)

9.28.2.8 int BoapPacket::push (Int8 *v*)

9.28.2.9 int BoapPacket::push (UInt8 *v*)

9.28.2.10 int BoapPacket::push (Int16 *v*)

9.28.2.11 int BoapPacket::push (UInt16 *v*)

9.28.2.12 int BoapPacket::push (Int32 *v*)

9.28.2.13 int BoapPacket::push (UInt32 *v*)

9.28.2.14 int BoapPacket::push (Int64 *v*)

9.28.2.15 int BoapPacket::push (UInt64 *v*)

9.28.2.16 int BoapPacket::push (const BString & *v*)

9.28.2.17 int BoapPacket::push (Double *v*)

9.28.2.18 int BoapPacket::push (const BError & *v*)

9.28.2.19 int BoapPacket::push (UInt32 *nBytes*, const void ∗ *data*, char ∗ *swapType* = `"1"`)

9.28.2.20 int BoapPacket::popHead (BoapPacketHead & *head*)

9.28.2.21 int BoapPacket::pop (Int8 & *v*)

9.28.2.22 int BoapPacket::pop (UInt8 & *v*)

9.28.2.23 int BoapPacket::pop (Int16 & *v*)

9.28.2.24 int BoapPacket::pop (UInt16 & *v*)

9.28.2.25 int BoapPacket::pop (Int32 & *v*)

- /src/cern/tms/beam/libBeam/Boap.h
- /src/cern/tms/beam/libBeam/BoapSimple.h
- /src/cern/tms/beam/libBeam/Boap.cpp
- /src/cern/tms/beam/libBeam/BoapSimple.cc

9.29 BoapPacketHead Struct Reference

```
#include <BoapSimple.h>
```

Public Attributes

- UInt32 type
- UInt32 length
- UInt32 service
- UInt32 cmd
- BoapType type
- BoapService service
- UInt32 reserved [12]

9.29.1 Member Data Documentation

9.29.1.1 UInt32 BoapPacketHead::type

9.29.1.2 UInt32 BoapPacketHead::length

9.29.1.3 UInt32 BoapPacketHead::service

9.29.1.4 UInt32 BoapPacketHead::cmd

9.29.1.5 BoapType BoapPacketHead::type

9.29.1.6 BoapService BoapPacketHead::service

9.29.1.7 UInt32 BoapPacketHead::reserved[12]

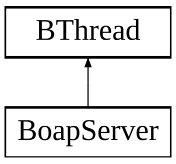
The documentation for this struct was generated from the following files:

- /src/cern/tms/beam/libBeam/Boap.h
- /src/cern/tms/beam/libBeam/BoapSimple.h

9.30 BoapServer Class Reference

```
#include <BoapSimple.h>
```

Inheritance diagram for BoapServer::



Public Types

- enum { NOTHREADS = 0, THREADED = 1 }

Public Member Functions

- BoapServer ()
- ∼BoapServer ()
- BError init (BString boapNsHost="", int threaded=0, int isBoapns=0)
- BError run (int inThread=0)
- BError processEvent (BoapPacket &rx)
- BError addObject (BoapServiceObject ∗object)
- BError process (BoapServerConnection ∗conn, BoapPacket &rx, BoapPacket &tx)
- BError sendEvent (BoapPacket &tx)
- BSocket & getSocket ()
- BSocket & getEventSocket ()
- BError processEvent (int fd)
- BString getHostName ()
- void clientGone (BoapServerConnection ∗client)
- int getConnectionsNumber ()
- BoapServer ()
- BError init (int boapNs=0)
- BError run ()
- BError processEvent (BoapPacket &rx)
- BError addObject (BoapServiceObject ∗object)
- BError process (int fd)
- BError sendEvent (BoapPacket &tx)
- BSocket & getSocket ()
- BSocket & getEventSocket ()
- BError processEvent (int fd)
- BString getHostName ()

Private Member Functions

- void ∗ function ()

Private Attributes

- int othreaded
- int oisBoapns
- Boapns::Boapns $*$ oboapns
- BList$<$ BoapServerConnection $*$ $>$ oclients
- BEventInt oclientGoneEvent
- BList$<$ BoapServiceEntry $>$ oservices
- BPoll opoll
- BSocket onet
- BSocket onetEvent
- BSocketAddressINET onetEventAddress
- BString ohostName
- int oboapNs
- BoapPacket orx
- BoapPacket otx

9.30.1 Member Enumeration Documentation

9.30.1.1 anonymous enum

Enumerator:

NOTHREADS

THREADED

9.30.2 Constructor & Destructor Documentation

9.30.2.1 BoapServer::BoapServer ()

9.30.2.2 BoapServer::∼BoapServer ()

9.30.2.3 BoapServer::BoapServer ()

9.30.3 Member Function Documentation

9.30.3.1 BError BoapServer::init (BString *boapNsHost* = `""`, int *threaded* = `0`, int *isBoapns* = `0`)

9.30.3.2 BError BoapServer::run (int *inThread* = `0`)

9.30.3.3 BError BoapServer::processEvent (BoapPacket & *rx*)

9.30.3.4 BError BoapServer::addObject (BoapServiceObject ∗ *object*)

9.30.3.5 BError BoapServer::process (BoapServerConnection ∗ *conn*, BoapPacket & *rx*, BoapPacket & *tx*)

9.30.3.6 BError BoapServer::sendEvent (BoapPacket & *tx*)

9.30.3.7 BSocket & BoapServer::getSocket ()

9.30.3.8 BSocket & BoapServer::getEventSocket ()

9.30.3.9 BError BoapServer::processEvent (int *fd*)

9.30.3.10 BString BoapServer::getHostName ()

9.30.3.11 void BoapServer::clientGone (BoapServerConnection ∗ *client*)

9.30.3.12 int BoapServer::getConnectionsNumber ()

9.30.3.13 void ∗ BoapServer::function () `[private, virtual]`

Reimplemented from BThread.

9.30.3.14 BError BoapServer::init (int *boapNs* = `0`)

9.30.3.15 BError BoapServer::run ()

9.30.3.16 BError BoapServer::processEvent (BoapPacket & *rx*)

9.30.3.17 BError BoapServer::addObject (BoapServiceObject ∗ *object*)

9.30.3.18 BError BoapServer::process (int *fd*)

9.30.3.19 BError BoapServer::sendEvent (BoapPacket & *tx*)

9.30.3.20 BSocket& BoapServer::getSocket ()

9.30.3.21 BSocket& BoapServer::getEventSocket ()

9.30.3.22 BError BoapServer::processEvent (int *fd*)

9.30.3.23 BString BoapServer::getHostName ()

9.30.4 Member Data Documentation

9.30.4.1 int BoapServer::othreaded `[private]`

9.30.4.2 int BoapServer::oisBoapns `[private]`

9.30.4.3 Boapns::Boapns∗ BoapServer::oboapns `[private]`

9.30.4.4 BList<BoapServerConnection∗> BoapServer::oclients `[private]`

9.30.4.5 BEventInt BoapServer::oclientGoneEvent `[private]`

9.30.4.6 BList< BoapServiceEntry > BoapServer::oservices `[private]`

9.30.4.7 BPoll BoapServer::opoll `[private]`

9.30.4.8 BSocket BoapServer::onet `[private]`

9.30.4.9 BSocket BoapServer::onetEvent `[private]`

9.30.4.10 BSocketAddressINET BoapServer::onetEventAddress `[private]`

9.30.4.11 BString BoapServer::ohostName `[private]`

9.30.4.12 int BoapServer::oboapNs `[private]`

9.30.4.13 BoapPacket BoapServer::orx `[private]`

9.30.4.14 BoapPacket BoapServer::otx `[private]`

The documentation for this class was generated from the following files:

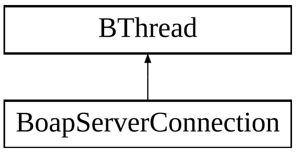
- /src/cern/tms/beam/libBeam/Boap.h

- /src/cern/tms/beam/libBeam/BoapSimple.h
- /src/cern/tms/beam/libBeam/Boap.cpp
- /src/cern/tms/beam/libBeam/BoapSimple.cc

9.31 BoapServerConnection Class Reference

`#include <Boap.h>`

Inheritance diagram for BoapServerConnection::



Public Member Functions

- BoapServerConnection (BoapServer &boapServer, int fd)
- BError process ()
- BSocket & getSocket ()
- void setMaxLength (BUInt32 maxLength)

Private Member Functions

- void $*$ function ()

Private Attributes

- BoapServer & oboapServer
- BSocket osocket
- BoapPacket orx
- BoapPacket otx
- BUInt32 omaxLength

9.31.1 Constructor & Destructor Documentation

9.31.1.1 BoapServerConnection::BoapServerConnection (BoapServer & *boapServer*, int *fd*)

9.31.2 Member Function Documentation

9.31.2.1 BError BoapServerConnection::process ()

9.31.2.2 BSocket & BoapServerConnection::getSocket ()

9.31.2.3 void BoapServerConnection::setMaxLength (BUInt32 *maxLength*)

9.31.2.4 void $*$ BoapServerConnection::function () `[private, virtual]`

Reimplemented from BThread.

9.31.3 Member Data Documentation

9.31.3.1 BoapServer& BoapServerConnection::oboapServer `[private]`

9.31.3.2 BSocket BoapServerConnection::osocket `[private]`

9.31.3.3 BoapPacket BoapServerConnection::orx `[private]`

9.31.3.4 BoapPacket BoapServerConnection::otx `[private]`

9.31.3.5 BUInt32 BoapServerConnection::omaxLength `[private]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/Boap.h
- /src/cern/tms/beam/libBeam/Boap.cpp

9.32 BoapServiceEntry Class Reference

`#include <BoapSimple.h>`

Public Member Functions

- BoapServiceEntry (BoapService service=0, BoapServiceObject ∗object=0)
- BoapServiceEntry (BoapService service=0, BoapServiceObject ∗object=0)

Public Attributes

- BoapService oservice
- BoapServiceObject ∗ oobject

9.32.1 Constructor & Destructor Documentation

9.32.1.1 BoapServiceEntry::BoapServiceEntry (BoapService *service* = `0`, BoapServiceObject ∗ *object* = `0`) `[inline]`

9.32.1.2 BoapServiceEntry::BoapServiceEntry (BoapService *service* = `0`, BoapServiceObject ∗ *object* = `0`) `[inline]`

9.32.2 Member Data Documentation

9.32.2.1 BoapService BoapServiceEntry::oservice

9.32.2.2 BoapServiceObject ∗ BoapServiceEntry::oobject

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/Boap.h
- /src/cern/tms/beam/libBeam/BoapSimple.h

9.33 BoapServiceObject Class Reference

```
#include <BoapSimple.h>
```

Public Member Functions

- BoapServiceObject (BoapServer &server, BString name="")
- virtual ∼BoapServiceObject ()
- BError setName (BString name)
- BError sendEvent (BString signalName, Int32 arg)
- virtual BError processEvent (BString objectName, BString name, Int32 arg)
- BString name ()
- BError doPing (BoapServerConnection ∗conn, BoapPacket &rx, BoapPacket &tx)
- BError doConnectionPriority (BoapServerConnection ∗conn, BoapPacket &rx, BoapPacket &tx)
- BError process (BoapServerConnection ∗conn, BoapPacket &rx, BoapPacket &tx)
- virtual BError processEvent (BoapPacket &rx)
- BoapServiceObject (BoapServer &server, BString name)
- virtual ∼BoapServiceObject ()
- BError sendEvent (BString signalName, Int32 arg)
- virtual BError processEvent (BString objectName, BString name, Int32 arg)
- BString name ()
- BError process (BoapPacket &rx, BoapPacket &tx)
- virtual BError processEvent (BoapPacket &rx)

Protected Member Functions

- BError sendEvent (BoapPacket &tx)
- BError sendEvent (BoapPacket &tx)

Protected Attributes

- BoapServer & oserver
- BString oname
- BUInt32 oapiVersion
- BList< BoapFuncEntry > ofuncList

9.33.1 Constructor & Destructor Documentation

9.33.1.1 BoapServiceObject::BoapServiceObject (BoapServer & *server*, BString *name* = "")

9.33.1.2 BoapServiceObject::∼BoapServiceObject () `[virtual]`

9.33.1.3 BoapServiceObject::BoapServiceObject (BoapServer & *server*, BString *name*)

9.33.1.4 virtual BoapServiceObject::∼BoapServiceObject () `[virtual]`

9.33.2 Member Function Documentation

9.33.2.1 BError BoapServiceObject::setName (BString *name*)

9.33.2.2 BError BoapServiceObject::sendEvent (BString *signalName*, Int32 *arg*)

9.33.2.3 BError BoapServiceObject::processEvent (BString *objectName*, BString *name*, Int32 *arg*) `[virtual]`

9.33.2.4 BString BoapServiceObject::name ()

9.33.2.5 BError BoapServiceObject::doPing (BoapServerConnection ∗ *conn*, BoapPacket & *rx*, BoapPacket & *tx*)

9.33.2.6 BError BoapServiceObject::doConnectionPriority (BoapServerConnection ∗ *conn*, BoapPacket & *rx*, BoapPacket & *tx*)

9.33.2.7 BError BoapServiceObject::process (BoapServerConnection ∗ *conn*, BoapPacket & *rx*, BoapPacket & *tx*)

9.33.2.8 BError BoapServiceObject::processEvent (BoapPacket & *rx*) `[virtual]`

9.33.2.9 BError BoapServiceObject::sendEvent (BoapPacket & *tx*) `[protected]`

9.33.2.10 BError BoapServiceObject::sendEvent (BString *signalName*, Int32 *arg*)

9.33.2.11 virtual BError BoapServiceObject::processEvent (BString *objectName*, BString *name*, Int32 *arg*) `[virtual]`

9.33.2.12 BString BoapServiceObject::name ()

9.33.2.13 BError BoapServiceObject::process (BoapPacket & *rx*, BoapPacket & *tx*)

9.33.2.14 virtual BError BoapServiceObject::processEvent (BoapPacket & *rx*) `[virtual]`

9.33.2.15 BError BoapServiceObject::sendEvent (BoapPacket & *tx*) `[protected]`

9.33.3 Member Data Documentation

9.33.3.1 BoapServer & BoapServiceObject::oserver `[protected]`

9.33.3.2 BString BoapServiceObject::oname `[protected]`

9.33.3.3 BUInt32 BoapServiceObject::oapiVersion `[protected]`

9.33.3.4 BList< BoapFuncEntry > BoapServiceObject::ofuncList `[protected]`

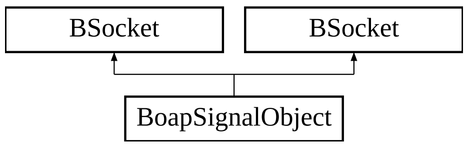
The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/Boap.h
- /src/cern/tms/beam/libBeam/BoapSimple.h
- /src/cern/tms/beam/libBeam/Boap.cpp
- /src/cern/tms/beam/libBeam/BoapSimple.cc

9.34 BoapSignalObject Class Reference

`#include <BoapSimple.h>`

Inheritance diagram for BoapSignalObject::



Public Member Functions

- BoapSignalObject ()
- BoapSignalObject ()

Protected Member Functions

- BError performSend (BoapPacket &tx)
- BError performSend (BoapPacket &tx)

Protected Attributes

- BoapPacket otx
- BoapPacket orx

9.34.1 Constructor & Destructor Documentation

9.34.1.1 BoapSignalObject::BoapSignalObject ()

9.34.1.2 BoapSignalObject::BoapSignalObject ()

9.34.2 Member Function Documentation

9.34.2.1 BError BoapSignalObject::performSend (BoapPacket & *tx*) `[protected]`

9.34.2.2 BError BoapSignalObject::performSend (BoapPacket & *tx*) `[protected]`

9.34.3 Member Data Documentation

9.34.3.1 BoapPacket BoapSignalObject::otx `[protected]`

9.34.3.2 BoapPacket BoapSignalObject::orx `[protected]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/Boap.h
- /src/cern/tms/beam/libBeam/BoapSimple.h

- /src/cern/tms/beam/libBeam/Boap.cpp
- /src/cern/tms/beam/libBeam/BoapSimple.cc

9.35 BObject Class Reference

```
#include <BObject.h>
```

Public Member Functions

- BObject ()
- virtual ∼BObject ()
- virtual BError getBinary (BDataBuf &buf)
- virtual BError setBinary (BDataBuf &buf)
- virtual BString getString ()
- virtual BError setString (BString str)
- virtual BMemberList getMemberList ()
- virtual BError addMember (BString name, BObject ∗object)
- virtual void printIt ()
- virtual BType & getType ()

Static Public Member Functions

- static BObject ∗ createObj ()

Static Public Attributes

- static BType otype = btypesList.appendType(BType("BObject", BTypeDomainBase, BTypeObject, createObj))

9.35.1 Constructor & Destructor Documentation

9.35.1.1 BObject::BObject ()

9.35.1.2 BObject::∼BObject () `[virtual]`

9.35.2 Member Function Documentation

9.35.2.1 BError BObject::getBinary (BDataBuf & *buf*) `[virtual]`

9.35.2.2 BError BObject::setBinary (BDataBuf & *buf*) `[virtual]`

9.35.2.3 BString BObject::getString () `[virtual]`

9.35.2.4 BError BObject::setString (BString *str*) `[virtual]`

9.35.2.5 BMemberList BObject::getMemberList () `[virtual]`

9.35.2.6 BError BObject::addMember (BString *name*, BObject ∗ *object*) `[virtual]`

9.35.2.7 void BObject::printIt () `[virtual]`

9.35.2.8 BType & BObject::getType () `[virtual]`

9.35.2.9 BObject ∗ BObject::createObj () `[static]`

9.35.3 Member Data Documentation

9.35.3.1 BType BObject::otype = btypesList.appendType(BType("BObject", BTypeDomainBase, BTypeObject, createObj)) `[static]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BObject.h
- /src/cern/tms/beam/libBeam/BObject.cc

9.36 BPoll Class Reference

This class provides an interface for polling a number of file descriptors. It uses round robin polling.

`#include <BPoll.h>`

Public Types

- typedef struct pollfd PollFd

Public Member Functions

- BPoll ()
- ~BPoll ()
- void append (int fd, int events=POLLIN|POLLERR|POLLHUP|POLLNVAL)

  Append a file descriptor to polling list.

- void delFd (int fd)

  Remove a file descriptor from polling list.

- BError doPoll (int &fd, int timeoutUs=-1)

  Perform polling operation.

- int getPollFdsNum ()
- PollFd * getPollFds ()
- void clear ()

Private Member Functions

- int nextFd (int i)

Private Attributes

- int ofdsNum

  The number of FD's in list.

- PollFd * ofds

  The list of poll fd's.

- int ofdsNext

  The next list entry for round robin polling.

9.36.1 Detailed Description

This class provides an interface for polling a number of file descriptors. It uses round robin polling.

9.36.2 Member Typedef Documentation

9.36.2.1 typedef struct pollfd BPoll::PollFd `[read]`

9.36.3 Constructor & Destructor Documentation

9.36.3.1 BPoll::BPoll ()

9.36.3.2 BPoll::∼BPoll ()

9.36.4 Member Function Documentation

9.36.4.1 void BPoll::append (int *fd*, int *events* = `POLLIN|POLLERR|POLLHUP|POLLNVAL`)

Append a file descriptor to polling list.

9.36.4.2 void BPoll::delFd (int *fd*)

Remove a file descriptor from polling list.

9.36.4.3 BError BPoll::doPoll (int & *fd*, int *timeoutUs* = `-1`)

Perform polling operation.

9.36.4.4 int BPoll::getPollFdsNum ()

9.36.4.5 BPoll::PollFd ∗ BPoll::getPollFds ()

9.36.4.6 void BPoll::clear ()

9.36.4.7 int BPoll::nextFd (int *i*) `[private]`

9.36.5 Member Data Documentation

9.36.5.1 int BPoll::ofdsNum `[private]`

The number of FD's in list.

9.36.5.2 PollFd∗ BPoll::ofds `[private]`

The list of poll fd's.

9.36.5.3 int BPoll::ofdsNext `[private]`

The next list entry for round robin polling.

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BPoll.h

- /src/cern/tms/beam/libBeam/BPoll-1.cpp
- /src/cern/tms/beam/libBeam/BPoll.cpp

9.37 BRefData Class Reference

Referenced data storage.

```
#include <BRefData.h>
```

Public Member Functions

- BRefData ()
- BRefData (int len)
- BRefData (const BRefData &refData)
- ∼BRefData ()
- BRefData ∗ copy ()
- BRefData ∗ addRef ()
- int deleteRef ()
- int refCount ()
- char ∗ data ()
- int len ()
- void setLen (int len)
- BRefData & operator= (BRefData &refData)

Private Attributes

- void ∗ oData
- int oLen
- int oSize
- int oRefCount

9.37.1 Detailed Description

Referenced data storage.

9.37.2 Constructor & Destructor Documentation

9.37.2.1 BRefData::BRefData ()

9.37.2.2 BRefData::BRefData (int *len*)

9.37.2.3 BRefData::BRefData (const BRefData & *refData*)

9.37.2.4 BRefData::∼BRefData ()

9.37.3 Member Function Documentation

9.37.3.1 BRefData ∗ BRefData::copy ()

9.37.3.2 BRefData ∗ BRefData::addRef ()

9.37.3.3 int BRefData::deleteRef ()

9.37.3.4 int BRefData::refCount () `[inline]`

9.37.3.5 char∗ BRefData::data () `[inline]`

9.37.3.6 int BRefData::len () `[inline]`

9.37.3.7 void BRefData::setLen (int *len*)

9.37.3.8 BRefData & BRefData::operator= (BRefData & *refData*)

9.37.4 Member Data Documentation

9.37.4.1 void∗ BRefData::oData `[private]`

9.37.4.2 int BRefData::oLen `[private]`

9.37.4.3 int BRefData::oSize `[private]`

9.37.4.4 int BRefData::oRefCount `[private]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BRefData.h
- /src/cern/tms/beam/libBeam/BRefData.cpp

9.38 BRtc Class Reference

Realtime clock.

`#include <BRtc.h>`

Public Member Functions

- BRtc ()
- ∼BRtc ()
- BError init (int rate)

  Setup interrupt rate.
- void wait (int delayUs)

  Wait specified uS.

Private Attributes

- int ofd
- int orate

9.38.1 Detailed Description

Realtime clock.

9.38.2 Constructor & Destructor Documentation

9.38.2.1 BRtc::BRtc ()

9.38.2.2 BRtc::∼BRtc ()

9.38.3 Member Function Documentation

9.38.3.1 BError BRtc::init (int *rate*)

Setup interrupt rate.

9.38.3.2 void BRtc::wait (int *delayUs*)

Wait specified uS.

9.38.4 Member Data Documentation

9.38.4.1 int BRtc::ofd `[private]`

9.38.4.2 int BRtc::orate `[private]`

The documentation for this class was generated from the following files:

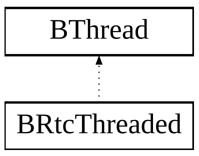
- /src/cern/tms/beam/libBeam/BRtc.h
- /src/cern/tms/beam/libBeam/BRtc.cpp

9.39 BRtcThreaded Class Reference

Threaded real time clock.

`#include <BRtc.h>`

Inheritance diagram for BRtcThreaded::



Public Member Functions

- BRtcThreaded ()
- ∼BRtcThreaded ()
- BError init (int rate)

  Setup interrupt rate.

- void wait (int delayUs)

  Wait specified uS.

Private Member Functions

- void ∗ function ()

Private Attributes

- BRtc ortc
- int orate
- BCond ocond

9.39.1 Detailed Description

Threaded real time clock.

9.39.2 Constructor & Destructor Documentation

9.39.2.1 BRtcThreaded::BRtcThreaded ()

9.39.2.2 BRtcThreaded::∼BRtcThreaded ()

9.39.3 Member Function Documentation

9.39.3.1 BError BRtcThreaded::init (int *rate*)

Setup interrupt rate.

9.39.3.2 void BRtcThreaded::wait (int *delayUs*)

Wait specified uS.

9.39.3.3 void ∗ BRtcThreaded::function () `[private, virtual]`

Reimplemented from BThread.

9.39.4 Member Data Documentation

9.39.4.1 BRtc BRtcThreaded::ortc `[private]`

9.39.4.2 int BRtcThreaded::orate `[private]`

9.39.4.3 BCond BRtcThreaded::ocond `[private]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BRtc.h
- /src/cern/tms/beam/libBeam/BRtc.cpp

9.40 BRWLock Class Reference

thread read-write locks

`#include <BRWLock.h>`

Public Member Functions

- BRWLock ()
- BRWLock (const BRWLock &rwlock)
- ~BRWLock ()
- int rdLock ()

  Set lock, wait if necessary.
- int tryRdLock ()

  Test the lock.
- int wrLock ()

  Set lock, wait if necessary.
- int tryWrLock ()

  Test the lock.
- int unlock ()

  Unlock the lock.
- BRWLock & operator= (const BRWLock &rwlock)

Private Attributes

- pthread_rwlock_t olock

9.40.1 Detailed Description

thread read-write locks

9.40.2 Constructor & Destructor Documentation

9.40.2.1 BRWLock::BRWLock ()

9.40.2.2 BRWLock::BRWLock (const BRWLock & *rwlock*)

9.40.2.3 BRWLock::~BRWLock ()

9.40.3 Member Function Documentation

9.40.3.1 int BRWLock::rdLock ()

Set lock, wait if necessary.

9.40.3.2 int BRWLock::tryRdLock ()

Test the lock.

9.40.3.3 int BRWLock::wrLock ()

Set lock, wait if necessary.

9.40.3.4 int BRWLock::tryWrLock ()

Test the lock.

9.40.3.5 int BRWLock::unlock ()

Unlock the lock.

9.40.3.6 BRWLock & BRWLock::operator= (const BRWLock & *rwlock*)

9.40.4 Member Data Documentation

9.40.4.1 pthread_rwlock_t BRWLock::olock `[private]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BRWLock.h
- /src/cern/tms/beam/libBeam/BRWLock.cpp

9.41 BSema Class Reference

Sempahore class.

`#include <BSema.h>`

Public Member Functions

- BSema (int value=0)
- BSema (const BSema &sema)
- ∼BSema ()
- int post ()

  Post condition.
- int wait ()

  Wait for contition.
- int timedWait (int timeUs)

  Wait for condition with timeout.
- int tryWait ()

  Test for the condition.
- int getValue () const
- BSema & operator= (const BSema &sema)

Private Attributes

- sem_t osema

9.41.1 Detailed Description

Sempahore class.

9.41.2 Constructor & Destructor Documentation

9.41.2.1 BSema::BSema (int *value* = `0`)

9.41.2.2 BSema::BSema (const BSema & *sema*)

9.41.2.3 BSema::∼BSema ()

9.41.3 Member Function Documentation

9.41.3.1 int BSema::post ()

Post condition.

9.41.3.2 int BSema::wait ()

Wait for contition.

9.41.3.3 int BSema::timedWait (int *timeUs*)

Wait for condition with timeout.

9.41.3.4 int BSema::tryWait ()

Test for the condition.

9.41.3.5 int BSema::getValue () const

9.41.3.6 BSema & BSema::operator= (const BSema & *sema*)

9.41.4 Member Data Documentation

9.41.4.1 sem_t BSema::osema `[private]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BSema.h
- /src/cern/tms/beam/libBeam/BSema.cpp

9.42 BSignal Class Reference

```
#include <SigGen.h>
```

Public Types

- enum { NumChannels = 9 }

Public Member Functions

- BSignal (int id=0, int numSamples=0, int numRepeat=0, int nextId=0)
- BSignal (const BSignal &sig)
- ∼BSignal ()
- BSignal & operator= (const BSignal &sig)

Public Attributes

- int id
- int numSamples
- int numRepeat
- int nextId
- Sample ∗ data [NumChannels]

9.42.1 Member Enumeration Documentation

9.42.1.1 anonymous enum

Enumerator:

NumChannels

9.42.2 Constructor & Destructor Documentation

9.42.2.1 BSignal::BSignal (int *id* = `0`, int *numSamples* = `0`, int *numRepeat* = `0`, int *nextId* = `0`)

9.42.2.2 BSignal::BSignal (const BSignal & *sig*)

9.42.2.3 BSignal::∼BSignal ()

9.42.3 Member Function Documentation

9.42.3.1 BSignal & BSignal::operator= (const BSignal & *sig*)

9.42.4 Member Data Documentation

9.42.4.1 int BSignal::id

9.42.4.2 int BSignal::numSamples

9.42.4.3 int BSignal::numRepeat

9.42.4.4 int BSignal::nextId

9.42.4.5 Sample∗ BSignal::data[NumChannels]

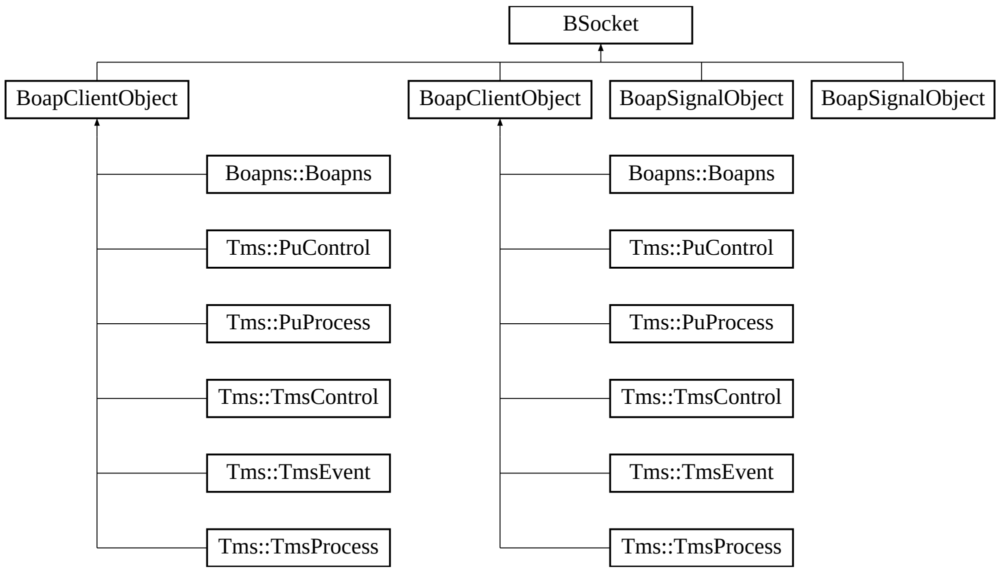
The documentation for this class was generated from the following files:

- SigGen.h
- SigGen.cpp

9.43 BSocket Class Reference

```
#include <BSocket.h>
```

Inheritance diagram for BSocket::



Public Types

- enum NType { STREAM, DGRAM }
- enum Priority { PriorityLow, PriorityNormal, PriorityHigh }

Public Member Functions

- BSocket ()
- BSocket (int fd)
- BSocket (NType type)
- ∼BSocket ()
- BError init (NType type)
- int getFd ()
- BError bind (const BSocketAddress &add)
- BError connect (const BSocketAddress &add)
- BError shutdown (int how)
- BError close ()
- BError listen (int backlog=5)
- BError accept (int &fd)
- BError accept (int &fd, BSocketAddress &address)
- BError send (const void ∗buf, BSize nbytes, BSize &nbytesSent, int flags=0)
- BError sendTo (const BSocketAddress &address, const void ∗buf, BSize nbytes, BSize &nbytesSent, int flags=0)
- BError recv (void ∗buf, BSize maxbytes, BSize &nbytesRecv, int flags=0)
- BError recvFrom (BSocketAddress &address, void ∗buf, BSize maxbytes, BSize &nbytesRecv, int flags=0)

- BError recvWithTimeout (void ∗buf, BSize maxbytes, BSize &nbytesRecv, int timeout, int flags=0)
- BError recvFromWithTimeout (BSocketAddress &address, void ∗buf, BSize maxbytes, BSize &nbytesRecv, int timeout, int flags=0)
- BError setSockOpt (int level, int optname, void ∗optval, unsigned int optlen)
- BError getSockOpt (int level, int optname, void ∗optval, unsigned int ∗optlen)
- BError setReuseAddress (int on)
- BError setBroadCast (int on)
- BError setPriority (Priority priority)
- BError getMTU (uint32_t &mtu)
- BError getAddress (BSocketAddress &address)

Private Attributes

- int osocket

9.43.1 Member Enumeration Documentation

9.43.1.1 enum BSocket::NType

Enumerator:

STREAM

DGRAM

9.43.1.2 enum BSocket::Priority

Enumerator:

PriorityLow

PriorityNormal

PriorityHigh

9.43.2 Constructor & Destructor Documentation

9.43.2.1 BSocket::BSocket ()

9.43.2.2 BSocket::BSocket (int *fd*)

9.43.2.3 BSocket::BSocket (NType *type*)

9.43.2.4 BSocket::∼BSocket ()

9.43.3 Member Function Documentation

9.43.3.1 BError BSocket::init (NType *type*)

9.43.3.2 int BSocket::getFd ()

9.43.3.3 BError BSocket::bind (const BSocketAddress & *add*)

9.43.3.4 BError BSocket::connect (const BSocketAddress & *add*)

9.43.3.5 BError BSocket::shutdown (int *how*)

9.43.3.6 BError BSocket::close ()

9.43.3.7 BError BSocket::listen (int *backlog* = 5)

9.43.3.8 BError BSocket::accept (int & *fd*)

9.43.3.9 BError BSocket::accept (int & *fd*, BSocketAddress & *address*)

9.43.3.10 BError BSocket::send (const void ∗ *buf*, BSize *nbytes*, BSize & *nbytesSent*, int *flags* = 0)

9.43.3.11 BError BSocket::sendTo (const BSocketAddress & *address*, const void ∗ *buf*, BSize *nbytes*, BSize & *nbytesSent*, int *flags* = 0)

9.43.3.12 BError BSocket::recv (void ∗ *buf*, BSize *maxbytes*, BSize & *nbytesRecv*, int *flags* = 0)

9.43.3.13 BError BSocket::recvFrom (BSocketAddress & *address*, void ∗ *buf*, BSize *maxbytes*, BSize & *nbytesRecv*, int *flags* = 0)

9.43.3.14 BError BSocket::recvWithTimeout (void ∗ *buf*, BSize *maxbytes*, BSize & *nbytesRecv*, int *timeout*, int *flags* = 0)

9.43.3.15 BError BSocket::recvFromWithTimeout (BSocketAddress & *address*, void ∗ *buf*, BSize *maxbytes*, BSize & *nbytesRecv*, int *timeout*, int *flags* = 0)

9.43.3.16 BError BSocket::setSockOpt (int *level*, int *optname*, void ∗ *optval*, unsigned int *optlen*)

9.43.3.17 BError BSocket::getSockOpt (int *level*, int *optname*, void ∗ *optval*, unsigned int ∗ *optlen*)

9.43.3.18 BError BSocket::setReuseAddress (int *on*)

9.43.3.19 BError BSocket::setBroadCast (int *on*)

9.43.3.20 BError BSocket::setPriority (Priority *priority*)

9.43.3.21 BError BSocket::getMTU (uint32_t & *mtu*)

9.43.3.22 BError BSocket::getAddress (BSocketAddress & *address*)

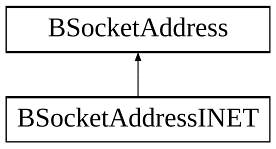
- /src/cern/tms/beam/libBeam/BSocket.h
- /src/cern/tms/beam/libBeam/BSocket.cpp

9.44 BSocketAddress Class Reference

Socket Address.

`#include <BSocket.h>`

Inheritance diagram for BSocketAddress::



Public Types

- typedef struct sockaddr SockAddr

Public Member Functions

- BSocketAddress ()
- BSocketAddress (const BSocketAddress &add)
- BSocketAddress (SockAddr ∗address, int len)
- ∼BSocketAddress ()
- BError set (SockAddr ∗address, int len)
- const SockAddr ∗ raw () const
- int len () const
- BSocketAddress & operator= (const BSocketAddress &add)
- operator const SockAddr ∗ () const
- int operator== (const BSocketAddress &add) const
- int operator!= (const BSocketAddress &add) const

Private Attributes

- int olen
- SockAddr ∗ oaddress

9.44.1 Detailed Description

Socket Address.

9.44.2 Member Typedef Documentation

9.44.2.1 typedef struct sockaddr BSocketAddress::SockAddr `[read]`

9.44.3 Constructor & Destructor Documentation

9.44.3.1 BSocketAddress::BSocketAddress ()

9.44.3.2 BSocketAddress::BSocketAddress (const BSocketAddress & *add*)

9.44.3.3 BSocketAddress::BSocketAddress (SockAddr ∗ *address*, int *len*)

9.44.3.4 BSocketAddress::∼BSocketAddress ()

9.44.4 Member Function Documentation

9.44.4.1 BError BSocketAddress::set (SockAddr ∗ *address*, int *len*)

9.44.4.2 const BSocketAddress::SockAddr ∗ BSocketAddress::raw () const

9.44.4.3 int BSocketAddress::len () const

9.44.4.4 BSocketAddress & BSocketAddress::operator= (const BSocketAddress & *add*)

9.44.4.5 BSocketAddress::operator const SockAddr ∗ () const `[inline]`

9.44.4.6 int BSocketAddress::operator== (const BSocketAddress & *add*) const

9.44.4.7 int BSocketAddress::operator!= (const BSocketAddress & *add*) const

9.44.5 Member Data Documentation

9.44.5.1 int BSocketAddress::olen `[private]`

9.44.5.2 SockAddr∗ BSocketAddress::oaddress `[private]`

The documentation for this class was generated from the following files:

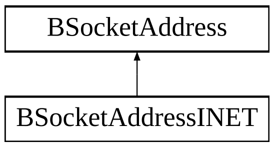
- /src/cern/tms/beam/libBeam/BSocket.h
- /src/cern/tms/beam/libBeam/BSocket.cpp

9.45 BSocketAddressINET Class Reference

IP aware socket address.

`#include <BSocket.h>`

Inheritance diagram for BSocketAddressINET::



Public Types

- typedef struct sockaddr_in SockAddrIP

Public Member Functions

- BError set (BString hostName, uint32_t port)
- BError set (uint32_t address, uint32_t port)
- BError set (BString hostName, BString service, BString type)
- void setPort (uint32_t port)
- uint32_t address ()

  Returns socket ip address.

- uint32_t port ()

  Returns socket port.

- BString getString ()

  Return string version of address <ip>:<port>.

Static Public Member Functions

- static BString getHostName ()

  Get this hosts network name.

- static BList< uint32_t > getIpAddresses ()

  Get a list of all the IP addresses of this host.

- static BList< BString > getIpAddressList ()

  Get a list of all the IP addresses of this host under hostname.

- static BList< BString > getIpAddressListAll ()

  Get a list of all the IP addresses of this host looking at physical interfaces.

9.45.1 Detailed Description

IP aware socket address.

9.45.2 Member Typedef Documentation

9.45.2.1 typedef struct sockaddr_in BSocketAddressINET::SockAddrIP `[read]`

9.45.3 Member Function Documentation

9.45.3.1 BError BSocketAddressINET::set (BString *hostName*, uint32_t *port*)

9.45.3.2 BError BSocketAddressINET::set (uint32_t *address*, uint32_t *port*)

9.45.3.3 BError BSocketAddressINET::set (BString *hostName*, BString *service*, BString *type*)

9.45.3.4 void BSocketAddressINET::setPort (uint32_t *port*)

9.45.3.5 uint32_t BSocketAddressINET::address ()

Returns socket ip address.

9.45.3.6 uint32_t BSocketAddressINET::port ()

Returns socket port.

9.45.3.7 BString BSocketAddressINET::getString ()

Return string version of address <ip>:<port>.

9.45.3.8 BString BSocketAddressINET::getHostName () `[static]`

Get this hosts network name.

9.45.3.9 BList< uint32_t > BSocketAddressINET::getIpAddresses () `[static]`

Get a list of all the IP addresses of this host.

9.45.3.10 BList< BString > BSocketAddressINET::getIpAddressList () `[static]`

Get a list of all the IP addresses of this host under hostname.

9.45.3.11 BList< BString > BSocketAddressINET::getIpAddressListAll () `[static]`

Get a list of all the IP addresses of this host looking at physical interfaces.

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BSocket.h

- /src/cern/tms/beam/libBeam/BSocket.cpp

9.46 BString Class Reference

```
#include <BString.h>
```

Public Member Functions

- BString ()
- BString (const BString &string)
- BString (const char ∗str)
- BString (char ch)
- BString (int v)
- BString (unsigned int v)
- BString (long v)
- BString (unsigned long long)
- BString (double v)
- virtual ∼BString ()
- BString copy ()

  Return an independant copy.

- virtual void strChanged ()
- int len () const

  Length of string.

- const char ∗ retStr () const

  Ptr to char∗ representation.

- char ∗ retStrDup () const

  Ptr to newly malloc'd char∗.

- int retInt () const

  Return string as a int.

- double retDouble () const

  Return string as a double.

- int compare (const BString &string) const

  Compare strings.

- int compareWild (const BString &string) const

  Compare string to string with wildcards.

- int compareWildExpression (const BString &string) const

  Compare string to space deliminated patterns.

- BString add (const BString &str) const

  Add two strings.

- BString & truncate (int len)

  Truncate to length len.

- BString & pad (int len)

    Pad to length len.

- BString & toUpper ()

    Convert to uppercase.

- BString & toLower ()

    Convert to lowercase.

- void removeNL ()

    Remove if present NL from last char.

- BString subString (int start, int len) const

    Returns substring.

- int del (int start, int len)

    Delete substring.

- int insert (int start, BString str)

    Insert substring.

- void printf (const char ∗fmt,...)

    Formated print into the string.

- int find (char ch) const

    Find ch in string searching forwards.

- int findReverse (char ch) const

    Find ch in string searching backwards.

- BList< BString > getTokenList (BString separators)

    Break string into tokens.

- BString removeSeparators (BString separators)

    Remove any char from sepatators from string.

- BString pullToken (BString terminators)

    Pull token from start of string.

- BString pullSeparators (BString separators)

    Pull separators from start of string.

- BString pullWord ()

    Pull a word out of the head of the string.

- BString pullLine ()

    Pull a line out of the head of the string.

- BString field (int field) const

- char ** fields ()
- BString & operator= (const BString &string)
- char & operator[ ] (int pos)
- int operator== (const BString &s) const
- int operator== (const char *s) const
- int operator> (const BString &s) const
- int operator> (const char *s) const
- int operator< (const BString &s) const
- int operator< (const char *s) const
- int operator>= (const BString &s) const
- int operator<= (const BString &s) const
- int operator!= (const BString &s) const
- int operator!= (const char *s) const
- BString operator+ (const BString &s) const
- BString operator+ (const char *s) const
- BString operator+= (const BString &s)
- BString operator+= (const char *s)
- BString operator+ (char ch) const
- BString operator+ (int i) const
- BString operator+ (unsigned int i) const
- BString operator+ (unsigned long long i) const
- operator const char * () const

Static Public Member Functions

- static BString convert (char ch)

  Converts char to string.

- static BString convert (int value)

  Converts int to string.

- static BString convert (unsigned int value)

  Converts uint to string.

- static BString convert (long value)

  Converts long to string.

- static BString convert (double value)

  Converts double to string.

- static BString convert (unsigned long long value)

  Converts u long long to string.

- static BString convertHex (int value)

  Converts int to string as hex value.

- static BString convertHex (unsigned int value)

  Converts uint to string as hex value.

Protected Attributes

- BRefData ∗ ostr

Private Member Functions

- void Init (const char ∗str)
- int inString (int pos) const
- int isSpace (char ch) const

9.46.1 Constructor & Destructor Documentation

9.46.1.1 BString::BString ()

9.46.1.2 BString::BString (const BString & *string*)

9.46.1.3 BString::BString (const char ∗ *str*)

9.46.1.4 BString::BString (char *ch*)

9.46.1.5 BString::BString (int *v*)

9.46.1.6 BString::BString (unsigned int *v*)

9.46.1.7 BString::BString (long *v*)

9.46.1.8 BString::BString (unsigned long long *value*)

9.46.1.9 BString::BString (double *v*)

9.46.1.10 BString::∼BString () `[virtual]`

9.46.2 Member Function Documentation

9.46.2.1 BString BString::convert (char *ch*) `[static]`

Converts char to string.

9.46.2.2 BString BString::convert (int *value*) `[static]`

Converts int to string.

9.46.2.3 BString BString::convert (unsigned int *value*) `[static]`

Converts uint to string.

9.46.2.4 BString BString::convert (long *value*) `[static]`

Converts long to string.

9.46.2.5 BString BString::convert (double *value*) `[static]`

Converts double to string.

9.46.2.6 BString BString::convert (unsigned long long *value*) `[static]`

Converts u long long to string.

9.46.2.7 BString BString::convertHex (int *value*) `[static]`

Converts int to string as hex value.

9.46.2.8 BString BString::convertHex (unsigned int *value*) `[static]`

Converts uint to string as hex value.

9.46.2.9 BString BString::copy ()

Return an independant copy.

9.46.2.10 void BString::strChanged () `[virtual]`

9.46.2.11 int BString::len () const

Length of string.

9.46.2.12 const char ∗ BString::retStr () const

Ptr to char∗ representation.

9.46.2.13 char ∗ BString::retStrDup () const

Ptr to newly malloc'd char∗.

9.46.2.14 int BString::retInt () const

Return string as a int.

9.46.2.15 double BString::retDouble () const

Return string as a double.

9.46.2.16 int BString::compare (const BString & *string*) const

Compare strings.

9.46.2.17 int BString::compareWild (const BString & *string*) const

Compare string to string with wildcards.

9.46.2.18 int BString::compareWildExpression (const BString & *string*) const

Compare string to space deliminated patterns.

9.46.2.19 BString BString::add (const BString & *str*) const

Add two strings.

9.46.2.20 BString & BString::truncate (int *len*)

Truncate to length len.

9.46.2.21 BString & BString::pad (int *len*)

Pad to length len.

9.46.2.22 BString & BString::toUpper ()

Convert to uppercase.

9.46.2.23 BString & BString::toLower ()

Convert to lowercase.

9.46.2.24 void BString::removeNL ()

Remove if present NL from last char.

9.46.2.25 BString BString::subString (int *start*, int *len*) const

Returns substring.

9.46.2.26 int BString::del (int *start*, int *len*)

Delete substring.

9.46.2.27 int BString::insert (int *start*, BString *str*)

Insert substring.

9.46.2.28 void BString::printf (const char $*$ *fmt*, ...)

Formated print into the string.

9.46.2.29 int BString::find (char *ch*) const

Find ch in string searching forwards.

9.46.2.30 int BString::findReverse (char *ch*) const

Find ch in string searching backwards.

9.46.2.31 BList< BString > BString::getTokenList (BString *separators*)

Break string into tokens.

9.46.2.32 BString BString::removeSeparators (BString *separators*)

Remove any char from sepatators from string.

9.46.2.33 BString BString::pullToken (BString *terminators*)

Pull token from start of string.

9.46.2.34 BString BString::pullSeparators (BString *separators*)

Pull separators from start of string.

9.46.2.35 BString BString::pullWord ()

Pull a word out of the head of the string.

9.46.2.36 BString BString::pullLine ()

Pull a line out of the head of the string.

9.46.2.37 BString BString::field (int *field*) const

9.46.2.38 char $**$ BString::fields ()

9.46.2.39 BString & BString::operator= (const BString & *string*)

9.46.2.40]

char & BString::operator[ ] (int *pos*)

9.46.2.41 int BString::operator== (const BString & *s*) const `[inline]`

9.46.2.42 int BString::operator== (const char $*$ *s*) const `[inline]`

9.46.2.43 int BString::operator> (const BString & *s*) const `[inline]`

9.46.2.44 int BString::operator> (const char $*$ *s*) const `[inline]`

9.46.2.45 int BString::operator< (const BString & *s*) const `[inline]`

9.46.2.46 int BString::operator< (const char $*$ *s*) const `[inline]`

9.46.2.47 int BString::operator>= (const BString & *s*) const `[inline]`

9.46.2.48 int BString::operator<= (const BString & *s*) const `[inline]`

9.46.2.49 int BString::operator!= (const BString & *s*) const `[inline]`

9.46.2.50 int BString::operator!= (const char $*$ *s*) const `[inline]`

9.46.2.51 BString BString::operator+ (const BString & *s*) const `[inline]`

9.46.2.52 BString BString::operator+ (const char $*$ *s*) const `[inline]`

9.46.2.53 BString BString::operator+= (const BString & *s*) `[inline]`

9.46.2.54 BString BString::operator+= (const char $*$ *s*) `[inline]`

9.46.2.55 BString BString::operator+ (char *ch*) const `[inline]`

9.46.2.56 BString BString::operator+ (int *i*) const `[inline]`

9.46.2.57 BString BString::operator+ (unsigned int *i*) const `[inline]`

9.46.2.58 BString BString::operator+ (unsigned long long *i*) const `[inline]`

9.46.2.59 BString::operator const char $*$ () const `[inline]`

9.46.2.60 void BString::Init (const char $*$ *str*) `[private]`

9.46.2.61 int BString::inString (int *pos*) const `[private]`

9.46.2.62 int BString::isSpace (char *ch*) const `[private]`

9.46.3 Member Data Documentation

9.46.3.1 BRefData$*$ BString::ostr `[protected]`

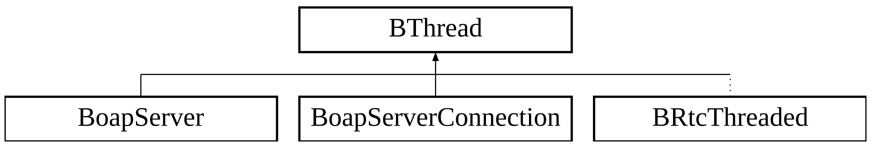
The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BString.h
- /src/cern/tms/beam/libBeam/BString.cpp

9.47 BThread Class Reference

`#include <BThread.h>`

Inheritance diagram for BThread::



Public Member Functions

- BThread ()
- virtual ∼BThread ()
- int setInitPriority (int policy, int priority)
- int setInitStackSize (size_t stackSize)
- int start ()
- void ∗ result ()
- int running ()
- int setPriority (int policy, int priority)
- int cancel ()
- void ∗ waitForCompletion ()
- pthread_t getThread ()
- virtual void ∗ function ()

Static Private Member Functions

- static void ∗ startFunc (void ∗)

Private Attributes

- pthread_t othread
- size_t ostackSize
- int opolicy
- int opriority
- int orunning
- void ∗ oresult

9.47.1 Constructor & Destructor Documentation

9.47.1.1 BThread::BThread ()

9.47.1.2 BThread::∼BThread () `[virtual]`

9.47.2 Member Function Documentation

9.47.2.1 int BThread::setInitPriority (int *policy*, int *priority*)

9.47.2.2 int BThread::setInitStackSize (size_t *stackSize*)

9.47.2.3 int BThread::start ()

9.47.2.4 void ∗ BThread::result ()

9.47.2.5 int BThread::running ()

9.47.2.6 int BThread::setPriority (int *policy*, int *priority*)

9.47.2.7 int BThread::cancel ()

9.47.2.8 void ∗ BThread::waitForCompletion ()

9.47.2.9 pthread_t BThread::getThread ()

9.47.2.10 void ∗ BThread::function () `[virtual]`

Reimplemented in BoapServerConnection, BoapServer, and BRtcThreaded.

9.47.2.11 void ∗ BThread::startFunc (void ∗ *arg*) `[static, private]`

9.47.3 Member Data Documentation

9.47.3.1 pthread_t BThread::othread `[private]`

9.47.3.2 size_t BThread::ostackSize `[private]`

9.47.3.3 int BThread::opolicy `[private]`

9.47.3.4 int BThread::opriority `[private]`

9.47.3.5 int BThread::orunning `[private]`

9.47.3.6 void∗ BThread::oresult `[private]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BThread.h
- /src/cern/tms/beam/libBeam/BThread.cpp

9.48 BTimer Class Reference

Stopwatch style timer.

`#include <BTimer.h>`

Public Member Functions

- BTimer ()
- ~BTimer ()
- void start ()

  Start timer.
- void stop ()

  Stop timer.
- void clear ()

  Clear timer.
- double getElapsedTime ()

  Returns the elapsed time from the last start.
- void add (BTimer &timer)

  Add two timers.
- double average ()

  Average time is duration between start() and stop() / number of stops.
- double peak ()

  Peak time.

Static Private Member Functions

- static double getTime ()

Private Attributes

- BMutex olock
- unsigned int onum
- double ostartTime
- double oendTime
- double oaverage
- double opeak

9.48.1 Detailed Description

Stopwatch style timer.

9.48.2 Constructor & Destructor Documentation

9.48.2.1 BTimer::BTimer ()

9.48.2.2 BTimer::∼BTimer ()

9.48.3 Member Function Documentation

9.48.3.1 void BTimer::start ()

Start timer.

9.48.3.2 void BTimer::stop ()

Stop timer.

9.48.3.3 void BTimer::clear ()

Clear timer.

9.48.3.4 double BTimer::getElapsedTime ()

Returns the elapsed time from the last start.

9.48.3.5 void BTimer::add (BTimer & *timer*)

Add two timers.

9.48.3.6 double BTimer::average ()

Average time is duration between start() and stop() / number of stops.

9.48.3.7 double BTimer::peak ()

Peak time.

9.48.3.8 double BTimer::getTime () `[static, private]`

9.48.4 Member Data Documentation

9.48.4.1 BMutex BTimer::olock `[private]`

9.48.4.2 unsigned int BTimer::onum `[private]`

9.48.4.3 double BTimer::ostartTime `[private]`

9.48.4.4 double BTimer::oendTime `[private]`

9.48.4.5 double BTimer::oaverage `[private]`

9.48.4.6 double BTimer::opeak `[private]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BTimer.h
- /src/cern/tms/beam/libBeam/BTimer.cpp

9.49 BUrl Class Reference

Basic access to a Url.

`#include <BUrl.h>`

Public Member Functions

- BUrl ()
- ∼BUrl ()
- BError readString (BString url, BString &str)

  Reads URL.

Static Private Member Functions

- static size_t writeData (void ∗data, size_t size, size_t elSize, void ∗stream)

Private Attributes

- BString ores

Static Private Attributes

- static int oinit

9.49.1 Detailed Description

Basic access to a Url.

9.49.2 Constructor & Destructor Documentation

9.49.2.1 BUrl::BUrl ()

9.49.2.2 BUrl::∼BUrl ()

9.49.3 Member Function Documentation

9.49.3.1 BError BUrl::readString (BString *url*, BString & *str*)

Reads URL.

9.49.3.2 size_t BUrl::writeData (void ∗ *data*, size_t *size*, size_t *elSize*, void ∗ *stream*) `[static, private]`

9.49.4 Member Data Documentation

9.49.4.1 int BUrl::oinit `[static, private]`

9.49.4.2 BString BUrl::ores `[private]`

The documentation for this class was generated from the following files:

- /src/cern/tms/beam/libBeam/BUrl.h
- /src/cern/tms/beam/libBeam/BUrl.cpp

9.50 Tms::ConfigInfo Class Reference

This class describes the configuration of the TMS.

`#include <TmsD.h>`

Public Member Functions

- ConfigInfo ()
- ConfigInfo (BArray< PuChannel > ppuReferences)

Public Attributes

- BArray< PuChannel > puReferences

  The logical to physical Pick-Up table. Each PuReference includes a Module Controller identifier, a Physical Pick-Up number and a Physical Channel.

9.50.1 Detailed Description

This class describes the configuration of the TMS.

9.50.2 Constructor & Destructor Documentation

9.50.2.1 Tms::ConfigInfo::ConfigInfo ()

9.50.2.2 Tms::ConfigInfo::ConfigInfo (BArray< PuChannel > *ppuReferences*)

9.50.3 Member Data Documentation

9.50.3.1 BArray<PuChannel> Tms::ConfigInfo::puReferences

The logical to physical Pick-Up table. Each PuReference includes a Module Controller identifier, a Physical Pick-Up number and a Physical Channel.

The documentation for this class was generated from the following files:

- TmsD.h
- TmsD.cc

9.51 Tms::CycleInformation Class Reference

```
#include <TmsD.h>
```

Public Member Functions

- CycleInformation ()
- CycleInformation (UInt32 pcycleNumber, BString pcycleType, BList< CycleInformationPeriod > pperiods)

Public Attributes

- UInt32 cycleNumber

  The PS Cycle number.

- BString cycleType

  The Cycle Type Name.

- BList< CycleInformationPeriod > periods

  The list of cycle periods.

9.51.1 Constructor & Destructor Documentation

9.51.1.1 Tms::CycleInformation::CycleInformation ()

9.51.1.2 Tms::CycleInformation::CycleInformation (UInt32 *pcycleNumber*, BString *pcycleType*, BList< CycleInformationPeriod > *pperiods*)

9.51.2 Member Data Documentation

9.51.2.1 UInt32 Tms::CycleInformation::cycleNumber

The PS Cycle number.

9.51.2.2 BString Tms::CycleInformation::cycleType

The Cycle Type Name.

9.51.2.3 BList<CycleInformationPeriod> Tms::CycleInformation::periods

The list of cycle periods.

The documentation for this class was generated from the following files:

- TmsD.h
- TmsD.cc

9.52 Tms::CycleInformationPeriod Class Reference

Cycle information.

```
#include <TmsD.h>
```

Public Member Functions

- CycleInformationPeriod ()
- CycleInformationPeriod (UInt32 pcyclePeriod, UInt32 pstartTime, UInt32 pendTime, UInt32 pharmonic, UInt32 pnumBunches, UInt32 pbunchMask, UInt32 pnumValues)

Public Attributes

- UInt32 cyclePeriod

  The Cycle Period.
- UInt32 startTime

  The start time in ms.
- UInt32 endTime

  The end time in ms.
- UInt32 harmonic

  The Machines harmonic number.
- UInt32 numBunches

  The number of bunches.
- UInt32 bunchMask

  Bitmask defining which buckets the bunches are captured from. Bit 0 is bucket 1, bit 1 is bucket 2 etc.
- UInt32 numValues

  The total number of raw data values available.

9.52.1 Detailed Description

Cycle information.

9.52.2 Constructor & Destructor Documentation

9.52.2.1 Tms::CycleInformationPeriod::CycleInformationPeriod ()

9.52.2.2 Tms::CycleInformationPeriod::CycleInformationPeriod (UInt32 *pcyclePeriod*, UInt32 *pstartTime*, UInt32 *pendTime*, UInt32 *pharmonic*, UInt32 *pnumBunches*, UInt32 *pbunchMask*, UInt32 *pnumValues*)

9.52.3 Member Data Documentation

9.52.3.1 UInt32 Tms::CycleInformationPeriod::cyclePeriod

The Cycle Period.

9.52.3.2 UInt32 Tms::CycleInformationPeriod::startTime

The start time in ms.

9.52.3.3 UInt32 Tms::CycleInformationPeriod::endTime

The end time in ms.

9.52.3.4 UInt32 Tms::CycleInformationPeriod::harmonic

The Machines harmonic number.

9.52.3.5 UInt32 Tms::CycleInformationPeriod::numBunches

The number of bunches.

9.52.3.6 UInt32 Tms::CycleInformationPeriod::bunchMask

Bitmask defining which buckets the bunches are captured from. Bit 0 is bucket 1, bit 1 is bucket 2 etc.

9.52.3.7 UInt32 Tms::CycleInformationPeriod::numValues

The total number of raw data values available.

The documentation for this class was generated from the following files:

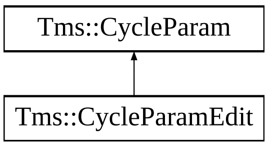
- TmsD.h
- TmsD.cc

9.53 Tms::CycleParam Class Reference

This class defines the parameters for a PS processing cycle.

`#include <TmsD.h>`

Inheritance diagram for Tms::CycleParam::



Public Member Functions

- CycleParam ()
- CycleParam (BString pcycleType, BString pinfo, UInt32 pchannel, UInt32 pllCycleStartFrequency, UInt32 ppllInitialFrequency, UInt32 ppllInitialFrequencyDelay, UInt32 ppllFrefGain, UInt32 ppllGain, UInt32 ppllDdsMinimum, UInt32 ppllDdsMaximum, BArray< Int32 > pfrefPhaseDelay, BArray< PuStateTable > pstateTable, BArray< BString > psettings)

Public Attributes

- BString cycleType

  The Cycle Type Name of this parameter set, normally the BEAM type the set of parameters is designed to measure.

- BString info

  Information on this parameter set.

- UInt32 channel

  The channel number this configuration is for, 0 defines all channels.

- UInt32 pllCycleStartFrequency

  This defines the initial PLL frequency. This is loaded on START_CYCLE.

- UInt32 pllInitialFrequency

  This defines the initial PLL frequency. This is loaded after the delay given in pllInitialFrequencyDelay.

- UInt32 pllInitialFrequencyDelay

  This defines the delay in milliseconds from START_CYCLE when the pllInitialFrequency is loaded.

- UInt32 pllFrefGain

  The gain the FREF signal. This is a value in the range +-8191. A normal value would be around 4096.

- UInt32 pllGain

  The gain of the PLL feedback system. This is the gain reduction of the PLL in terms of right bit shifts. A bit shift of 7 is about unity gain.

- UInt32 pllDdsMinimum

  PLL DDS minimum frequency. If this and pllDdsMaximum is set to 0, this feature is disabled.

- UInt32 pllDdsMaximum

  PLL DDS maximum frequency. If this and pllDdsMinimum is set to 0, this feature is disabled.

- BArray< Int32 > frefPhaseDelay

  The phase delay parameters for the Fref timing signal for each of the Pick-Up channels. This is set based on the position of the Pick-Up's in the PS ring. Its value is based of Fref / 512.

- BArray< PuStateTable > stateTable

  The array of State Table entries for the processing run.

- BArray< BString > settings

  A string array defining the settings for the states. Used for CycleParam editors.

9.53.1 Detailed Description

This class defines the parameters for a PS processing cycle.

9.53.2 Constructor & Destructor Documentation

9.53.2.1 Tms::CycleParam::CycleParam ()

9.53.2.2 Tms::CycleParam::CycleParam (BString *pcycleType*, BString *pinfo*, UInt32 *pchannel*, UInt32 *ppllCycleStartFrequency*, UInt32 *ppllInitialFrequency*, UInt32 *ppllInitialFrequencyDelay*, UInt32 *ppllFrefGain*, UInt32 *ppllGain*, UInt32 *ppllDdsMinimum*, UInt32 *ppllDdsMaximum*, BArray< Int32 > *pfrefPhaseDelay*, BArray< PuStateTable > *pstateTable*, BArray< BString > *psettings*)

9.53.3 Member Data Documentation

9.53.3.1 BString Tms::CycleParam::cycleType

The Cycle Type Name of this parameter set, normally the BEAM type the set of parameters is designed to measure.

9.53.3.2 BString Tms::CycleParam::info

Information on this parameter set.

9.53.3.3 UInt32 Tms::CycleParam::channel

The channel number this configuration is for, 0 defines all channels.

9.53.3.4 UInt32 Tms::CycleParam::pllCycleStartFrequency

This defines the initial PLL frequency. This is loaded on START_CYCLE.

9.53.3.5 UInt32 Tms::CycleParam::pllInitialFrequency

This defines the initial PLL frequency. This is loaded after the delay given in pllInitialFrequencyDelay.

9.53.3.6 UInt32 Tms::CycleParam::pllInitialFrequencyDelay

This defines the delay in milliseconds from START_CYCLE when the pllInitialFrequency is loaded.

9.53.3.7 UInt32 Tms::CycleParam::pllFrefGain

The gain the FREF signal. This is a value in the range +-8191. A normal value would be around 4096.

9.53.3.8 UInt32 Tms::CycleParam::pllGain

The gain of the PLL feedback system. This is the gain reduction of the PLL in terms of right bit shifts. A bit shift of 7 is about unity gain.

9.53.3.9 UInt32 Tms::CycleParam::pllDdsMinimum

PLL DDS minimum frequency. If this and pllDdsMaximum is set to 0, this feature is disabled.

9.53.3.10 UInt32 Tms::CycleParam::pllDdsMaximum

PLL DDS maximum frequency. If this and pllDdsMinimum is set to 0, this feature is disabled.

9.53.3.11 BArray<Int32> Tms::CycleParam::frefPhaseDelay

The phase delay parameters for the Fref timing signal for each of the Pick-Up channels. This is set based on the position of the Pick-Up's in the PS ring. Its value is based of Fref / 512.

9.53.3.12 BArray<PuStateTable> Tms::CycleParam::stateTable

The array of State Table entries for the processing run.

9.53.3.13 BArray<BString> Tms::CycleParam::settings

A string array defining the settings for the states. Used for CycleParam editors.

The documentation for this class was generated from the following files:

- TmsD.h
- TmsD.cc

9.54 Tms::CycleParamDb Class Reference

Internal CycleParameter management class.

```
#include <TmsLib.h>
```

Public Member Functions

- CycleParamDb (BString baseDir=".")
- BError getCycleTypes (BList< BString > &typeList)

    Get the list of CycleParameter types in the directory.

- BError getFileNames (BList< BString > &fileList)

    Get a list of all of the CycleParameter file names.

- BError getCycleParams (BString fileName, Tms::CycleParam &param)

    Get the CycleParameter from the given file name.

- BError setCycleParams (Tms::CycleParam param)

    Set the CycleParameters. Writes to the appropriate file name.

- BError deleteCycleParams (BString cycleType, UInt32 puChannel)

    Deletes a CycleParameter definition file.

- BError readCycleParams (BString fileName, Tms::CycleParam &param)

    Reads a set of CycleParameters from a file.

- BError writeCycleParams (BString fileName, Tms::CycleParam param)

    Writes a set of CycleParameters to a file.

Private Attributes

- BString obaseDir

9.54.1 Detailed Description

Internal CycleParameter management class.

9.54.2 Constructor & Destructor Documentation

9.54.2.1 Tms::CycleParamDb::CycleParamDb (BString *baseDir* = ".")

9.54.3 Member Function Documentation

9.54.3.1 BError Tms::CycleParamDb::getCycleTypes (BList< BString > & *typeList*)

Get the list of CycleParameter types in the directory.

9.54.3.2 BError Tms::CycleParamDb::getFileNames (BList< BString > & *fileList*)

Get a list of all of the CycleParameter file names.

9.54.3.3 BError Tms::CycleParamDb::getCycleParams (BString *fileName*, Tms::CycleParam & *param*)

Get the CycleParameter from the given file name.

9.54.3.4 BError Tms::CycleParamDb::setCycleParams (Tms::CycleParam *param*)

Set the CycleParameters. Writes to the appropriate file name.

9.54.3.5 BError Tms::CycleParamDb::deleteCycleParams (BString *cycleType*, UInt32 *puChannel*)

Deletes a CycleParameter definition file.

9.54.3.6 BError Tms::CycleParamDb::readCycleParams (BString *fileName*, Tms::CycleParam & *param*)

Reads a set of CycleParameters from a file.

9.54.3.7 BError Tms::CycleParamDb::writeCycleParams (BString *fileName*, Tms::CycleParam *param*)

Writes a set of CycleParameters to a file.

9.54.4 Member Data Documentation

9.54.4.1 BString Tms::CycleParamDb::obaseDir `[private]`

The documentation for this class was generated from the following files:

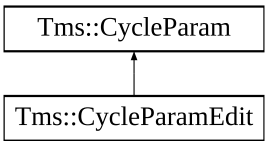
- TmsLib.h
- TmsLib.cc

9.55 Tms::CycleParamEdit Class Reference

Cycle Parameter management class.

`#include <TmsCycleParam.h>`

Inheritance diagram for Tms::CycleParamEdit::



Public Member Functions

- CycleParamEdit ()
- CycleParamEdit (const CycleParam &param)
- void clear ()
- BString getString ()

  Gets the Cycle Parameters in a string format for writing to a file or display.

- BError setString (BString str)

  Sets the Cycle Parameters from a string. For reading from a file.

- BError readFromFile (BString fileName)

  Reads the Cycle Parameters from a file.

- BError writeToFile (BString fileName)

  Writes the Cycle Parameters to a file.

- BError setStates (BList< CycleParamState > cycleStates)

  Sets the Cycle Parameter states given the state information list.

- BError getStates (BList< CycleParamState > &cycleStates)

  Returns the state information list describing the Cycle Parameter states. These may not be present.

Static Public Member Functions

- static void getDefaultState (CycleParamState &state)

  Get the default settings for a state.

- static void getdefaultPickupPositions (BArray< Int32 > &pos)

  Get the default pickup positions.

Private Member Functions

- double value (int numSamples, int harmonic, double phase, int sample)
- int bunch (int numSamples, int harmonic, double phase, int sample)
- BError generateState (int num, Tms::TmsState state, CycleParamState stateParam, int lo1Harmonic, double lo1Phase, int lo2Harmonic, double lo2Phase)

9.55.1 Detailed Description

Cycle Parameter management class.

9.55.2 Constructor & Destructor Documentation

9.55.2.1 Tms::CycleParamEdit::CycleParamEdit ()

9.55.2.2 Tms::CycleParamEdit::CycleParamEdit (const CycleParam & *param*)

9.55.3 Member Function Documentation

9.55.3.1 void Tms::CycleParamEdit::clear ()

9.55.3.2 BString Tms::CycleParamEdit::getString ()

Gets the Cycle Parameters in a string format for writing to a file or display.

9.55.3.3 BError Tms::CycleParamEdit::setString (BString *str*)

Sets the Cycle Parameters from a string. For reading from a file.

9.55.3.4 BError Tms::CycleParamEdit::readFromFile (BString *fileName*)

Reads the Cycle Parameters from a file.

9.55.3.5 BError Tms::CycleParamEdit::writeToFile (BString *fileName*)

Writes the Cycle Parameters to a file.

9.55.3.6 BError Tms::CycleParamEdit::setStates (BList< CycleParamState > *cycleStates*)

Sets the Cycle Parameter states given the state information list.

9.55.3.7 BError Tms::CycleParamEdit::getStates (BList< CycleParamState > & *cycleStates*)

Returns the state information list describing the Cycle Parameter states. These may not be present.

9.55.3.8 void Tms::CycleParamEdit::getDefaultState (CycleParamState & *state*) `[static]`

Get the default settings for a state.

9.55.3.9 void Tms::CycleParamEdit::getdefaultPickupPositions (BArray< Int32 > & *pos*) `[static]`

Get the default pickup positions.

Calculates the base pickup phase values for the PS ring.

9.55.3.10 double Tms::CycleParamEdit::value (int *numSamples*, int *harmonic*, double *phase*, int *sample*) `[private]`

9.55.3.11 int Tms::CycleParamEdit::bunch (int *numSamples*, int *harmonic*, double *phase*, int *sample*) `[private]`

9.55.3.12 BError Tms::CycleParamEdit::generateState (int *num*, Tms::TmsState *state*, CycleParamState *stateParam*, int *lo1Harmonic*, double *lo1Phase*, int *lo2Harmonic*, double *lo2Phase*) `[private]`

This function will generate the phase tables for a given state. It is passed the parameters for the LO1 reference and the LO2 reference. If lo?Harmonic is 1, then FREF is generated.

The documentation for this class was generated from the following files:

- TmsCycleParam.h
- TmsCycleParam-1.cc
- TmsCycleParam.cc

9.56 Tms::CycleParamItem Class Reference

```
#include <TmsD.h>
```

Public Member Functions

- CycleParamItem ()
- CycleParamItem (BString pcycleType, UInt32 pchannel)

Public Attributes

- BString cycleType

  The Cycle Type Name of this parameter set, normally the BEAM type the set of parameters is designed to measure.

- UInt32 channel

  The channel number this configuration is for, 0 defines all channels.

9.56.1 Constructor & Destructor Documentation

9.56.1.1 Tms::CycleParamItem::CycleParamItem ()

9.56.1.2 Tms::CycleParamItem::CycleParamItem (BString *pcycleType*, UInt32 *pchannel*)

9.56.2 Member Data Documentation

9.56.2.1 BString Tms::CycleParamItem::cycleType

The Cycle Type Name of this parameter set, normally the BEAM type the set of parameters is designed to measure.

9.56.2.2 UInt32 Tms::CycleParamItem::channel

The channel number this configuration is for, 0 defines all channels.

The documentation for this class was generated from the following files:

- TmsD.h
- TmsD.cc

9.57 Tms::CycleParamState Class Reference

```
#include <TmsCycleParam.h>
```

Public Member Functions

- CycleParamState ()
- void clear ()

  Clear the entry.

- BString getString ()

  Returns the CycleParamState in string form.

- BError setString (BString str)

  Sets the CycleParamState from a string.

Public Attributes

- UInt32 period

  The cycle period.

- UInt32 bunchMask

  The set of bunches to capture bit mask.

- UInt32 mean1Mask

  The set of bunches to pass through meanFilter1.

- UInt32 mean2Mask

  The set of bunches to pass through meanFilter2.

- UInt32 loHarmonic

  The LO harmonic number used in this state.

- double loPhase

  The phase offset of the LO as a fraction of FREF (+-1.0).

- int useLoFref

  Flag setting system to use LO as FREF rather than phase table address MSB.

- int acquireData

  Flag to acquire data during this state.

- double gateWidth

  The gate pulse width as a fraction of LO (0 - 1.0).

- double gatePhase

  The gate phase offset as a fraction of LO (0 - 1.0).

- double blrWidth

  The gate pulse width as a fraction of LO (0 - 1.0).

- double blrPhase

  The gate phase offset as a fraction of LO (0 - 1.0).

9.57.1 Constructor & Destructor Documentation

9.57.1.1 Tms::CycleParamState::CycleParamState ()

9.57.2 Member Function Documentation

9.57.2.1 void Tms::CycleParamState::clear ()

Clear the entry.

9.57.2.2 BString Tms::CycleParamState::getString ()

Returns the CycleParamState in string form.

9.57.2.3 BError Tms::CycleParamState::setString (BString *str*)

Sets the CycleParamState from a string.

9.57.3 Member Data Documentation

9.57.3.1 UInt32 Tms::CycleParamState::period

The cycle period.

9.57.3.2 UInt32 Tms::CycleParamState::bunchMask

The set of bunches to capture bit mask.

9.57.3.3 UInt32 Tms::CycleParamState::mean1Mask

The set of bunches to pass through meanFilter1.

9.57.3.4 UInt32 Tms::CycleParamState::mean2Mask

The set of bunches to pass through meanFilter2.

9.57.3.5 UInt32 Tms::CycleParamState::loHarmonic

The LO harmonic number used in this state.

9.57.3.6 double Tms::CycleParamState::loPhase

The phase offset of the LO as a fraction of FREF (+-1.0).

9.57.3.7 int Tms::CycleParamState::useLoFref

Flag setting system to use LO as FREF rather than phase table address MSB.

9.57.3.8 int Tms::CycleParamState::acquireData

Flag to acquire data during this state.

9.57.3.9 double Tms::CycleParamState::gateWidth

The gate pulse width as a fraction of LO (0 - 1.0).

9.57.3.10 double Tms::CycleParamState::gatePhase

The gate phase offset as a fraction of LO (0 - 1.0).

9.57.3.11 double Tms::CycleParamState::blrWidth

The gate pulse width as a fraction of LO (0 - 1.0).

9.57.3.12 double Tms::CycleParamState::blrPhase

The gate phase offset as a fraction of LO (0 - 1.0).

The documentation for this class was generated from the following files:

- TmsCycleParam.h
- TmsCycleParam-1.cc
- TmsCycleParam.cc

9.58 Tms::CycleTypeInformation Class Reference

```
#include <TmsD.h>
```

Public Member Functions

- CycleTypeInformation ()
- CycleTypeInformation (BString pcycleType, BString pinfo, BList< CycleTypeInformationPeriod > pperiods)

Public Attributes

- BString cycleType

  The Cycle Type Name.

- BString info

  Information string on this cycle type.

- BList< CycleTypeInformationPeriod > periods

  The list of cycle periods.

9.58.1 Constructor & Destructor Documentation

9.58.1.1 Tms::CycleTypeInformation::CycleTypeInformation ()

9.58.1.2 Tms::CycleTypeInformation::CycleTypeInformation (BString *pcycleType*, BString *pinfo*, BList< CycleTypeInformationPeriod > *pperiods*)

9.58.2 Member Data Documentation

9.58.2.1 BString Tms::CycleTypeInformation::cycleType

The Cycle Type Name.

9.58.2.2 BString Tms::CycleTypeInformation::info

Information string on this cycle type.

9.58.2.3 BList<CycleTypeInformationPeriod> Tms::CycleTypeInformation::periods

The list of cycle periods.

The documentation for this class was generated from the following files:

- TmsD.h
- TmsD.cc

9.59 Tms::CycleTypeInformationPeriod Class Reference

Cycle Type information.

```
#include <TmsD.h>
```

Public Member Functions

- CycleTypeInformationPeriod ()
- CycleTypeInformationPeriod (UInt32 pcyclePeriod, UInt32 pharmonic, UInt32 pnumBunches, UInt32 pbunchMask)

Public Attributes

- UInt32 cyclePeriod

  The Cycle Period.

- UInt32 harmonic

  The Machines harmonic number.

- UInt32 numBunches

  The number of bunches.

- UInt32 bunchMask

  Bitmask defining which buckets the bunches are captured from. Bit 0 is bucket 1, bit 1 is bucket 2 etc.

9.59.1 Detailed Description

Cycle Type information.

9.59.2 Constructor & Destructor Documentation

9.59.2.1 Tms::CycleTypeInformationPeriod::CycleTypeInformationPeriod ()

9.59.2.2 Tms::CycleTypeInformationPeriod::CycleTypeInformationPeriod (UInt32 *pcyclePeriod*, UInt32 *pharmonic*, UInt32 *pnumBunches*, UInt32 *pbunchMask*)

9.59.3 Member Data Documentation

9.59.3.1 UInt32 Tms::CycleTypeInformationPeriod::cyclePeriod

The Cycle Period.

9.59.3.2 UInt32 Tms::CycleTypeInformationPeriod::harmonic

The Machines harmonic number.

9.59.3.3 UInt32 Tms::CycleTypeInformationPeriod::numBunches

The number of bunches.

9.59.3.4 UInt32 Tms::CycleTypeInformationPeriod::bunchMask

Bitmask defining which buckets the bunches are captured from. Bit 0 is bucket 1, bit 1 is bucket 2 etc.

The documentation for this class was generated from the following files:

- TmsD.h
- TmsD.cc

9.60 Tms::Data Class Reference

This class stores the raw data.

`#include <TmsD.h>`

Public Member Functions

- Data ()
- Data (UInt32 pnumValues, UInt32 pdataType, UInt32 pnumBunches, UInt32 pnumChannels, BArray< DataValue > pdataValues, BArray< BError > perrors)

Public Attributes

- UInt32 numValues

  The total number of data samples.

- UInt32 dataType

  The type of data in the data block.

- UInt32 numBunches

  The number of bunches.

- UInt32 numChannels

  The number of channels.

- BArray< DataValue > dataValues

  The data.

- BArray< BError > errors

  Individual errors for each channel within dataValues.

9.60.1 Detailed Description

This class stores the raw data.

9.60.2 Constructor & Destructor Documentation

9.60.2.1 Tms::Data::Data ()

9.60.2.2 Tms::Data::Data (UInt32 *pnumValues*, UInt32 *pdataType*, UInt32 *pnumBunches*, UInt32 *pnumChannels*, BArray< DataValue > *pdataValues*, BArray< BError > *perrors*)

9.60.3 Member Data Documentation

9.60.3.1 UInt32 Tms::Data::numValues

The total number of data samples.

9.60.3.2 UInt32 Tms::Data::dataType

The type of data in the data block.

9.60.3.3 UInt32 Tms::Data::numBunches

The number of bunches.

9.60.3.4 UInt32 Tms::Data::numChannels

The number of channels.

9.60.3.5 BArray<DataValue> Tms::Data::dataValues

The data.

9.60.3.6 BArray<BError> Tms::Data::errors

Individual errors for each channel within dataValues.

The documentation for this class was generated from the following files:

- TmsD.h
- TmsD.cc

9.61 Tms::DataInfo Class Reference

This class defines the data to be acquired and/or fetched.

```
#include <TmsD.h>
```

Public Member Functions

- DataInfo ()
- DataInfo (UInt32 pcycleNumber, UInt32 pchannel, UInt32 pcyclePeriod, UInt32 pstartTime, UInt32 porbitNumber, UInt32 pbunchNumber, UInt32 pfunction, UInt32 pargument, UInt32 pnumValues, Int32 pbeyondPeriod)

Public Attributes

- UInt32 cycleNumber

  The PS Cycle number.
- UInt32 channel

  The pick-up channel number.
- UInt32 cyclePeriod

  The cycle period the data is from.
- UInt32 startTime

  The start time in milli-seconds in the cycle period (starting from 0).
- UInt32 orbitNumber

  The starting orbit number (starting from 0).
- UInt32 bunchNumber

  The Bunch number (starting from 1 (0 is all bunches)).
- UInt32 function

  The data processing function to perform or performed. (0 normal data).
- UInt32 argument

  The Argument to the data processing function.
- UInt32 numValues

  The total number of data points to return.
- Int32 beyondPeriod

  If set allows reads of data beyond the end of the period.

9.61.1 Detailed Description

This class defines the data to be acquired and/or fetched.

9.61.2 Constructor & Destructor Documentation

9.61.2.1 Tms::DataInfo::DataInfo ()

9.61.2.2 Tms::DataInfo::DataInfo (UInt32 *pcycleNumber*, UInt32 *pchannel*, UInt32 *pcyclePeriod*, UInt32 *pstartTime*, UInt32 *porbitNumber*, UInt32 *pbunchNumber*, UInt32 *pfunction*, UInt32 *pargument*, UInt32 *pnumValues*, Int32 *pbeyondPeriod*)

9.61.3 Member Data Documentation

9.61.3.1 UInt32 Tms::DataInfo::cycleNumber

The PS Cycle number.

9.61.3.2 UInt32 Tms::DataInfo::channel

The pick-up channel number.

9.61.3.3 UInt32 Tms::DataInfo::cyclePeriod

The cycle period the data is from.

9.61.3.4 UInt32 Tms::DataInfo::startTime

The start time in milli-seconds in the cycle period (starting from 0).

9.61.3.5 UInt32 Tms::DataInfo::orbitNumber

The starting orbit number (starting from 0).

9.61.3.6 UInt32 Tms::DataInfo::bunchNumber

The Bunch number (starting from 1 (0 is all bunches)).

9.61.3.7 UInt32 Tms::DataInfo::function

The data processing function to perform or performed. (0 normal data).

9.61.3.8 UInt32 Tms::DataInfo::argument

The Argument to the data processing function.

9.61.3.9 UInt32 Tms::DataInfo::numValues

The total number of data points to return.

9.61.3.10 Int32 Tms::DataInfo::beyondPeriod

If set allows reads of data beyond the end of the period.

The documentation for this class was generated from the following files:

- TmsD.h
- TmsD.cc

9.62 Tms::DataValue Class Reference

This is the definition of a single data value.

```
#include <TmsD.h>
```

Public Member Functions

- DataValue ()
- DataValue (Int16 psigma, Int16 pdeltaX, Int16 pdeltaY, Int16 ptime)

Public Attributes

- Int16 sigma

  The Sigma value.
- Int16 deltaX

  The DeltaX value.
- Int16 deltaY

  The DeltaY value.
- Int16 time

  The Time in ms this sample was processed.

9.62.1 Detailed Description

This is the definition of a single data value.

9.62.2 Constructor & Destructor Documentation

9.62.2.1 Tms::DataValue::DataValue ()

9.62.2.2 Tms::DataValue::DataValue (Int16 *psigma*, Int16 *pdeltaX*, Int16 *pdeltaY*, Int16 *ptime*)

9.62.3 Member Data Documentation

9.62.3.1 Int16 Tms::DataValue::sigma

The Sigma value.

9.62.3.2 Int16 Tms::DataValue::deltaX

The DeltaX value.

9.62.3.3 Int16 Tms::DataValue::deltaY

The DeltaY value.

9.62.3.4 Int16 Tms::DataValue::time

The Time in ms this sample was processed.

The documentation for this class was generated from the following files:

- TmsD.h
- TmsD.cc

9.63 Tms::NameValue Class Reference

```
#include <TmsD.h>
```

Public Member Functions

- NameValue ()
- NameValue (BString pname, BString pvalue)

Public Attributes

- BString name

  The Name of the value.

- BString value

  The actual value in string form.

9.63.1 Constructor & Destructor Documentation

9.63.1.1 Tms::NameValue::NameValue ()

9.63.1.2 Tms::NameValue::NameValue (BString *pname*, BString *pvalue*)

9.63.2 Member Data Documentation

9.63.2.1 BString Tms::NameValue::name

The Name of the value.

9.63.2.2 BString Tms::NameValue::value

The actual value in string form.

The documentation for this class was generated from the following files:

- TmsD.h
- TmsD.cc

9.64 Tms::PuChannel Class Reference

This class stores a Physical Pick-Up channel id.

`#include <TmsD.h>`

Public Member Functions

- PuChannel ()
- PuChannel (UInt8 pmoduleNum, UInt8 ppupeNum, UInt8 ppupeChan)

Public Attributes

- UInt8 moduleNum

  The Module number.
- UInt8 pupeNum

  The PUPE number.
- UInt8 pupeChan

  The PUPE channel.

9.64.1 Detailed Description

This class stores a Physical Pick-Up channel id.

9.64.2 Constructor & Destructor Documentation

9.64.2.1 Tms::PuChannel::PuChannel ()

9.64.2.2 Tms::PuChannel::PuChannel (UInt8 *pmoduleNum*, UInt8 *ppupeNum*, UInt8 *ppupeChan*)

9.64.3 Member Data Documentation

9.64.3.1 UInt8 Tms::PuChannel::moduleNum

The Module number.

9.64.3.2 UInt8 Tms::PuChannel::pupeNum

The PUPE number.

9.64.3.3 UInt8 Tms::PuChannel::pupeChan

The PUPE channel.

The documentation for this class was generated from the following files:

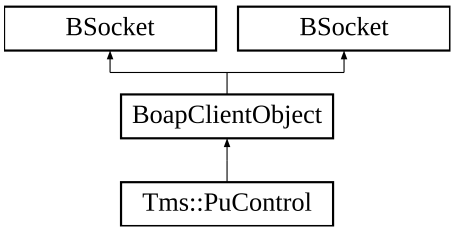
- TmsD.h
- TmsD.cc

9.65 Tms::PuControl Class Reference

This class defines the parameters for a test data capture.

`#include <TmsC.h>`

Inheritance diagram for Tms::PuControl::



Public Member Functions

- PuControl (BString name="")
- BError getVersion (BString &version)

  Gets the software version.

- BError init ()

  Initialises the system including loading all of the PUPE engines firmware. The call will return an error object indicating success or an error condition as appropriate.

- BError setProcessPriority (UInt32 priority)

  Sets the priority of the process servicing this service.

- BError configure (ConfigInfo configInfo)

  Configure the system for use. This includes mapping the individual physical PickUp channels to logical pickup channels.

- BError setControlInfo (CycleParam params)

  Sets the control information for the cycle type given and subsequent cycles. The parameters for the processing cycle are passed, this includes the Phase and State table information. The call will return an error object indicating success or an error.

- BError setNextCycle (UInt32 cycleNumber, BString cycleType)

  Sets the cycle number and type for the next processing cycle. The call will return an error object indicating success or an error condition as appropriate. This should be called at least 10ms before the next CYCLE_-START event.

- BError test (BList< BError > &errors)

  Performs a basic test of the system returning a list of errors. The call will return an error object indicating success or an error condition as appropriate.

- BError getStatus (BList< NameValue > &statusList)

  Returns the current status of the system. This information includes the number of Pick-Up's present and their individual status.

- BError getStatistics (BList< NameValue > &statsList)

  Returns a list of the statistic values as name/value pairs. The call will return an error object indicating success or an error condition as appropriate.

- BError getMasterPuChannel (PuChannel &puChannel)

  Returns the master PU Channel for timing.

- BError setTestMode (PuChannel puChannel, UInt32 testOutput, UInt32 timingDisableMask)

  The signal source for the digital test output connector. 0: None, 1: FrefLocal. The timingDisableMask bit mask defines which of the timing inputs should be disabled. If a timing input is disabled it can be still operated by software command.

- BError setTimingSignals (PuChannel puChannel, UInt32 timingSignals)

  This function sets the given timing signals to the values as defined in the timingSignals bit array.

- BError captureDiagnostics (PuChannel puChannel, TestCaptureInfo captureInfo, BArray< UInt64 > &data)

  This function will capture test data.

- BError setTestData (PuChannel puChannel, Int32 on, BArray< UInt32 > data)

  This function will set a PU channel to sample data from memory rather than the ADC's.

- BError setPupeConfig (PuChannel puPhysChannel, PupeConfig pupeConfig)

  Sets special PUPE configuration for test purposes.

- BError getPupeConfig (PuChannel puPhysChannel, PupeConfig &pupeConfig)

  Gets special PUPE configuration for test purposes.

9.65.1 Detailed Description

This class defines the parameters for a test data capture.

This class stores a Physical Pick-Up channel id This class stores the status of an individual Pick-Up This class describes the configuration of the TMS This class defines the data to be acquired and/or fetched This is the definition of a single data value This class stores the raw data This class defines the Pick-Up state table This class defines the parameters for a PS processing cycle Cycle information Cycle Type information This interface provides functions to control, test and get statistics from an individual pick-up

9.65.2 Constructor & Destructor Documentation

9.65.2.1 Tms::PuControl::PuControl (BString *name* = "")

9.65.3 Member Function Documentation

9.65.3.1 BError Tms::PuControl::getVersion (BString & *version*)

Gets the software version.

Parameters:

version A string variable filled in with the version number string.

9.65.3.2 BError Tms::PuControl::init ()

Initialises the system including loading all of the PUPE engines firmware. The call will return an error object indicating success or an error condition as appropriate.

9.65.3.3 BError Tms::PuControl::setProcessPriority (UInt32 *priority*)

Sets the priority of the process servicing this service.

9.65.3.4 BError Tms::PuControl::configure (ConfigInfo *configInfo*)

Configure the system for use. This includes mapping the individual physical PickUp channels to logical pickup channels.

9.65.3.5 BError Tms::PuControl::setControlInfo (CycleParam *params*)

Sets the control information for the cycle type given and subsequent cycles. The parameters for the processing cycle are passed, this includes the Phase and State table information. The call will return an error object indicating success or an error.

9.65.3.6 BError Tms::PuControl::setNextCycle (UInt32 *cycleNumber*, BString *cycleType*)

Sets the cycle number and type for the next processing cycle. The call will return an error object indicating success or an error condition as appropriate. This should be called at least 10ms before the next CYCLE_-START event.

9.65.3.7 BError Tms::PuControl::test (BList< BError > & *errors*)

Performs a basic test of the system returning a list of errors. The call will return an error object indicating success or an error condition as appropriate.

9.65.3.8 BError Tms::PuControl::getStatus (BList< NameValue > & *statusList*)

Returns the current status of the system. This information includes the number of Pick-Up's present and their individual status.

9.65.3.9 BError Tms::PuControl::getStatistics (BList< NameValue > & *statsList*)

Returns a list of the statistic values as name/value pairs. The call will return an error object indicating success or an error condition as appropriate.

9.65.3.10 BError Tms::PuControl::getMasterPuChannel (PuChannel & *puChannel*)

Returns the master PU Channel for timing.

9.65.3.11 BError Tms::PuControl::setTestMode (PuChannel *puChannel*, UInt32 *testOutput*, UInt32 *timingDisableMask*)

The signal source for the digital test output connector. 0: None, 1: FrefLocal. The timingDisableMask bit mask defines which of the timing inputs should be disabled. If a timing input is disabled it can be still operated by software command.

9.65.3.12 BError Tms::PuControl::setTimingSignals (PuChannel *puChannel*, UInt32 *timingSignals*)

This function sets the given timing signals to the values as defined in the timingSignals bit array.

9.65.3.13 BError Tms::PuControl::captureDiagnostics (PuChannel *puChannel*, TestCaptureInfo *captureInfo*, BArray< UInt64 > & *data*)

This function will capture test data.

9.65.3.14 BError Tms::PuControl::setTestData (PuChannel *puChannel*, Int32 *on*, BArray< UInt32 > *data*)

This function will set a PU channel to sample data from memory rather than the ADC's.

9.65.3.15 BError Tms::PuControl::setPupeConfig (PuChannel *puPhysChannel*, PupeConfig *pupeConfig*)

Sets special PUPE configuration for test purposes.

9.65.3.16 BError Tms::PuControl::getPupeConfig (PuChannel *puPhysChannel*, PupeConfig & *pupeConfig*)

Gets special PUPE configuration for test purposes.

The documentation for this class was generated from the following files:

- TmsC.h
- TmsC.cc
- tmsFunctions.dox

9.66 Tms::PupeConfig Class Reference

```
#include <TmsD.h>
```

Public Member Functions

- PupeConfig ()
- PupeConfig (UInt32 pinternalTimingMask, Int32 padcSysclkSync, Int32 pdisableBlr)

Public Attributes

- UInt32 internalTimingMask

  Use internal, software/hardware generated, timing signals for the given signals.
- Int32 adcSysclkSync

  Sets the ADC clock to be synchronised with the SYSCLK timing clock.
- Int32 disableBlr

  Disable the BLR algorithm.

9.66.1 Constructor & Destructor Documentation

9.66.1.1 Tms::PupeConfig::PupeConfig ()

9.66.1.2 Tms::PupeConfig::PupeConfig (UInt32 *pinternalTimingMask*, Int32 *padcSysclkSync*, Int32 *pdisableBlr*)

9.66.2 Member Data Documentation

9.66.2.1 UInt32 Tms::PupeConfig::internalTimingMask

Use internal, software/hardware generated, timing signals for the given signals.

9.66.2.2 Int32 Tms::PupeConfig::adcSysclkSync

Sets the ADC clock to be synchronised with the SYSCLK timing clock.

9.66.2.3 Int32 Tms::PupeConfig::disableBlr

Disable the BLR algorithm.

The documentation for this class was generated from the following files:

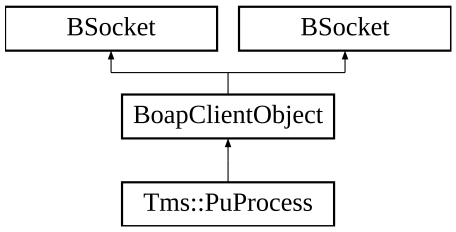
- TmsD.h
- TmsD.cc

9.67 Tms::PuProcess Class Reference

This interface provides functions to configure and capture data from individual pick-up.

`#include <TmsC.h>`

Inheritance diagram for Tms::PuProcess::



Public Member Functions

- PuProcess (BString name="")
- BError getVersion (BString &version)

  Gets the software version.

- BError getCycleInformation (UInt32 cycleNumber, CycleInformation &cycleInformation)

  Gets information on given cycle number.

- BError getStatus (PuChannel puChannel, PuStatus &puStatus)
- BError getData (PuChannel puChannel, DataInfo dataInfo, Data &data)

  This function returns a set of data from the data present in the data cache or directly from the Pick-Up processing engines. The DataInfo object describes the data required. The call will return the required data along with an error object indicating success or an error condition as appropriate. The call will block until data is ready.

- BError addEventServer (BString name)

  Adds an event server.

- BError requestData (PuChannel puChannel, DataInfo dataInfo)

  This adds a request for some data. The DataInfo object defines the data required. This request can be made at any time. If the data is present in cache the data will be available immediately, if not the system will await the data from a subsequent processing cycle. When the data is available a "data" event will be sent to the client. Not that it is not necessary to use requestData. The client can call getData() directly although this call will block until the data is actually ready.

9.67.1 Detailed Description

This interface provides functions to configure and capture data from individual pick-up.

9.67.2 Constructor & Destructor Documentation

9.67.2.1 Tms::PuProcess::PuProcess (BString *name* = "")

9.67.3 Member Function Documentation

9.67.3.1 BError Tms::PuProcess::getVersion (BString & *version*)

Gets the software version.

9.67.3.2 BError Tms::PuProcess::getCycleInformation (UInt32 *cycleNumber*, CycleInformation & *cycleInformation*)

Gets information on given cycle number.

9.67.3.3 BError Tms::PuProcess::getStatus (PuChannel *puChannel*, PuStatus & *puStatus*)

9.67.3.4 BError Tms::PuProcess::getData (PuChannel *puChannel*, DataInfo *dataInfo*, Data & *data*)

This function returns a set of data from the data present in the data cache or directly from the Pick-Up processing engines. The DataInfo object describes the data required. The call will return the required data along with an error object indicating success or an error condition as appropriate. The call will block until data is ready.

9.67.3.5 BError Tms::PuProcess::addEventServer (BString *name*)

Adds an event server.

9.67.3.6 BError Tms::PuProcess::requestData (PuChannel *puChannel*, DataInfo *dataInfo*)

This adds a request for some data. The DataInfo object defines the data required. This request can be made at any time. If the data is present in cache the data will be available immediately, if not the system will await the data from a subsequent processing cycle. When the data is available a "data" event will be sent to the client. Not that it is not necessary to use requestData. The client can call getData() directly although this call will block until the data is actually ready.

The documentation for this class was generated from the following files:

- TmsC.h
- TmsC.cc

9.68 Tms::PuStateTable Class Reference

This class defines the Pick-Up state table.

```
#include <TmsD.h>
```

Public Member Functions

- PuStateTable ()
- PuStateTable (UInt32 pperiod, UInt32 pstate, UInt32 pharmonic, UInt32 pnumBunches, UInt32 pbunchMask, BArray< UInt8 > pphaseTable)

Public Attributes

- UInt32 period

  The Cycle period this state is used for.

- UInt32 state

  The State table entry.

- UInt32 harmonic

  The harmonic number for this state.

- UInt32 numBunches

  The number of bunches to capture.

- UInt32 bunchMask

  Bitmask defining which buckets the bunches are captured from. Bit 0 is bucket 1, bit 1 is bucket 2 etc.

- BArray< UInt8 > phaseTable

  The Phase table for this state.

9.68.1 Detailed Description

This class defines the Pick-Up state table.

9.68.2 Constructor & Destructor Documentation

9.68.2.1 Tms::PuStateTable::PuStateTable ()

9.68.2.2 Tms::PuStateTable::PuStateTable (UInt32 *pperiod*, UInt32 *pstate*, UInt32 *pharmonic*, UInt32 *pnumBunches*, UInt32 *pbunchMask*, BArray< UInt8 > *pphaseTable*)

9.68.3 Member Data Documentation

9.68.3.1 UInt32 Tms::PuStateTable::period

The Cycle period this state is used for.

9.68.3.2 UInt32 Tms::PuStateTable::state

The State table entry.

9.68.3.3 UInt32 Tms::PuStateTable::harmonic

The harmonic number for this state.

9.68.3.4 UInt32 Tms::PuStateTable::numBunches

The number of bunches to capture.

9.68.3.5 UInt32 Tms::PuStateTable::bunchMask

Bitmask defining which buckets the bunches are captured from. Bit 0 is bucket 1, bit 1 is bucket 2 etc.

9.68.3.6 BArray<UInt8> Tms::PuStateTable::phaseTable

The Phase table for this state.

The documentation for this class was generated from the following files:

- TmsD.h
- TmsD.cc

9.69 Tms::PuStatus Class Reference

This class stores the status of an individual Pick-Up.

```
#include <TmsD.h>
```

Public Member Functions

- PuStatus ()
- PuStatus (Int32 prunning, BError perror)

Public Attributes

- Int32 running

  The Pick-Up is currently running.

- BError error

  The Pick-Up's current error status.

9.69.1 Detailed Description

This class stores the status of an individual Pick-Up.

9.69.2 Constructor & Destructor Documentation

9.69.2.1 Tms::PuStatus::PuStatus ()

9.69.2.2 Tms::PuStatus::PuStatus (Int32 *prunning*, BError *perror*)

9.69.3 Member Data Documentation

9.69.3.1 Int32 Tms::PuStatus::running

The Pick-Up is currently running.

9.69.3.2 BError Tms::PuStatus::error

The Pick-Up's current error status.

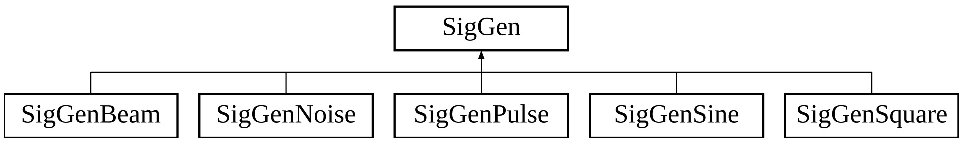
The documentation for this class was generated from the following files:

- TmsD.h
- TmsD.cc

9.70 SigGen Class Reference

```
#include <SigGen.h>
```

Inheritance diagram for SigGen::



Public Member Functions

- SigGen ()
- virtual ∼SigGen ()
- BError config (double sampleRate)
- virtual BError generate (Sample ∗data, int numSamples)

Protected Attributes

- double osampleRate
- unsigned long long ox

9.70.1 Constructor & Destructor Documentation

9.70.1.1 SigGen::SigGen ()

9.70.1.2 SigGen::∼SigGen () `[virtual]`

9.70.2 Member Function Documentation

9.70.2.1 BError SigGen::config (double *sampleRate*)

9.70.2.2 BError SigGen::generate (Sample ∗ *data*, int *numSamples*) `[virtual]`

Reimplemented in SigGenSine, SigGenSquare, SigGenNoise, SigGenPulse, and SigGenBeam.

9.70.3 Member Data Documentation

9.70.3.1 double SigGen::osampleRate `[protected]`

9.70.3.2 unsigned long long SigGen::ox `[protected]`

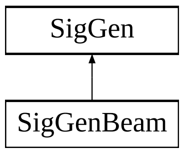
The documentation for this class was generated from the following files:

- SigGen.h
- SigGen.cpp

9.71 SigGenBeam Class Reference

`#include <SigGen.h>`

Inheritance diagram for SigGenBeam::



Public Member Functions

- SigGenBeam ()
- virtual ∼SigGenBeam ()
- BError config (double sampleRate, double fref, int harmonic, int bunchSet, double reduce, int blr, double amplitude)
- BError generate (Sample ∗data, int numSamples)
- BError generateIntegrated (Sample ∗data, int numSamples)

Public Attributes

- int oharmonic
- int obunchSet
- double oreduce
- int oblr
- double oamplitude
- double ofref

9.71.1 Constructor & Destructor Documentation

9.71.1.1 SigGenBeam::SigGenBeam ()

9.71.1.2 SigGenBeam::∼SigGenBeam () `[virtual]`

9.71.2 Member Function Documentation

9.71.2.1 BError SigGenBeam::config (double *sampleRate*, double *fref*, int *harmonic*, int *bunchSet*, double *reduce*, int *blr*, double *amplitude*)

9.71.2.2 BError SigGenBeam::generate (Sample ∗ *data*, int *numSamples*) `[virtual]`

Reimplemented from SigGen.

9.71.2.3 BError SigGenBeam::generateIntegrated (Sample ∗ *data*, int *numSamples*)

9.71.3 Member Data Documentation

9.71.3.1 int SigGenBeam::oharmonic

9.71.3.2 int SigGenBeam::obunchSet

9.71.3.3 double SigGenBeam::oreduce

9.71.3.4 int SigGenBeam::oblr

9.71.3.5 double SigGenBeam::oamplitude

9.71.3.6 double SigGenBeam::ofref

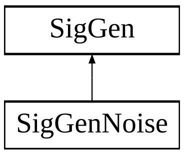
The documentation for this class was generated from the following files:

- SigGen.h
- SigGen.cpp

9.72 SigGenNoise Class Reference

```
#include <SigGen.h>
```

Inheritance diagram for SigGenNoise::



Public Member Functions

- SigGenNoise ()
- virtual ∼SigGenNoise ()
- BError config (double sampleRate, double amplitude=1.0)
- BError generate (Sample ∗data, int numSamples)

Public Attributes

- double oamplitude

9.72.1 Constructor & Destructor Documentation

9.72.1.1 SigGenNoise::SigGenNoise ()

9.72.1.2 SigGenNoise::∼SigGenNoise () `[virtual]`

9.72.2 Member Function Documentation

9.72.2.1 BError SigGenNoise::config (double *sampleRate*, double *amplitude* = `1.0`)

9.72.2.2 BError SigGenNoise::generate (Sample ∗ *data*, int *numSamples*) `[virtual]`

Reimplemented from SigGen.

9.72.3 Member Data Documentation

9.72.3.1 double SigGenNoise::oamplitude

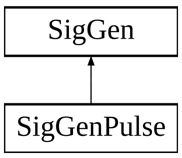
The documentation for this class was generated from the following files:

- SigGen.h
- SigGen.cpp

9.73 SigGenPulse Class Reference

`#include <SigGen.h>`

Inheritance diagram for SigGenPulse::



Public Member Functions

- SigGenPulse ()
- virtual ∼SigGenPulse ()
- BError config (double sampleRate, double freq, double amplitude, double onTime, double start-Time=0.0)
- BError generate (Sample ∗data, int numSamples)

Public Attributes

- double ofreq
- double oamplitude
- double oonTime
- double ostartTime

9.73.1 Constructor & Destructor Documentation

9.73.1.1 SigGenPulse::SigGenPulse ()

9.73.1.2 SigGenPulse::∼SigGenPulse () `[virtual]`

9.73.2 Member Function Documentation

9.73.2.1 BError SigGenPulse::config (double *sampleRate*, double *freq*, double *amplitude*, double *onTime*, double *startTime* = `0.0`)

9.73.2.2 BError SigGenPulse::generate (Sample ∗ *data*, int *numSamples*) `[virtual]`

Reimplemented from SigGen.

9.73.3 Member Data Documentation

9.73.3.1 double SigGenPulse::ofreq

9.73.3.2 double SigGenPulse::oamplitude

9.73.3.3 double SigGenPulse::oonTime

9.73.3.4 double SigGenPulse::ostartTime

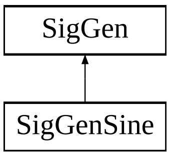
The documentation for this class was generated from the following files:

- SigGen.h
- SigGen.cpp

9.74 SigGenSine Class Reference

`#include <SigGen.h>`

Inheritance diagram for SigGenSine::



Public Member Functions

- SigGenSine ()
- virtual ∼SigGenSine ()
- BError config (double sampleRate, double freq, double amplitude=1.0)
- BError generate (Sample ∗data, int numSamples)

Public Attributes

- double ofreq
- double oamplitude

9.74.1 Constructor & Destructor Documentation

9.74.1.1 SigGenSine::SigGenSine ()

9.74.1.2 SigGenSine::∼SigGenSine () `[virtual]`

9.74.2 Member Function Documentation

9.74.2.1 BError SigGenSine::config (double *sampleRate*, double *freq*, double *amplitude* = `1.0`)

9.74.2.2 BError SigGenSine::generate (Sample ∗ *data*, int *numSamples*) `[virtual]`

Reimplemented from SigGen.

9.74.3 Member Data Documentation

9.74.3.1 double SigGenSine::ofreq

9.74.3.2 double SigGenSine::oamplitude

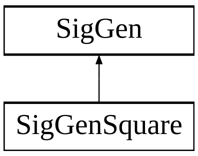
The documentation for this class was generated from the following files:

- SigGen.h
- SigGen.cpp

9.75 SigGenSquare Class Reference

`#include <SigGen.h>`

Inheritance diagram for SigGenSquare::



Public Member Functions

- SigGenSquare ()
- virtual ∼SigGenSquare ()
- BError config (double sampleRate, double freq, double amplitude=1.0, double offset=0.0)
- BError generate (Sample ∗data, int numSamples)

Public Attributes

- double ofreq
- double oamplitude
- double ooffset

9.75.1 Constructor & Destructor Documentation

9.75.1.1 SigGenSquare::SigGenSquare ()

9.75.1.2 SigGenSquare::∼SigGenSquare () `[virtual]`

9.75.2 Member Function Documentation

9.75.2.1 BError SigGenSquare::config (double *sampleRate*, double *freq*, double *amplitude* = `1.0`, double *offset* = `0.0`)

9.75.2.2 BError SigGenSquare::generate (Sample ∗ *data*, int *numSamples*) `[virtual]`

Reimplemented from SigGen.

9.75.3 Member Data Documentation

9.75.3.1 double SigGenSquare::ofreq

9.75.3.2 double SigGenSquare::oamplitude

9.75.3.3 double SigGenSquare::ooffset

The documentation for this class was generated from the following files:

- SigGen.h
- SigGen.cpp

9.76 Tms::Simulation Class Reference

```
#include <TmsD.h>
```

Public Member Functions

- Simulation ()
- Simulation (Int32 ptiming, Int32 pdata, Int32 psetNextCycle, BString pcycleType)

Public Attributes

- Int32 timing

    Simulate timing signals.

- Int32 data

    Simulate FREF and Analogue Sigma, DeltaX and DeltaY data.

- Int32 setNextCycle

    Simulate setNextCycle.

- BString cycleType

    The cycle type to set.

9.76.1 Constructor & Destructor Documentation

9.76.1.1 Tms::Simulation::Simulation ()

9.76.1.2 Tms::Simulation::Simulation (Int32 *ptiming*, Int32 *pdata*, Int32 *psetNextCycle*, BString *pcycleType*)

9.76.2 Member Data Documentation

9.76.2.1 Int32 Tms::Simulation::timing

Simulate timing signals.

9.76.2.2 Int32 Tms::Simulation::data

Simulate FREF and Analogue Sigma, DeltaX and DeltaY data.

9.76.2.3 Int32 Tms::Simulation::setNextCycle

Simulate setNextCycle.

9.76.2.4 BString Tms::Simulation::cycleType

The cycle type to set.

The documentation for this class was generated from the following files:

- TmsD.h
- TmsD.cc

9.77 vector Class Reference

Inherited by BArray< BError >, BArray< BString >, BArray< int32_t >, BArray< Tms::DataValue >, BArray< Tms::PuChannel >, BArray< Tms::PuStateTable >, and BArray< uint8_t >.

The documentation for this class was generated from the following file:

- /src/cern/tms/beam/libBeam/BArray.h

9.78 Tms::TestCaptureInfo Class Reference

This class defines the parameters for a test data capture.

`#include <TmsD.h>`

Public Member Functions

- TestCaptureInfo ()
- TestCaptureInfo (UInt32 psource, UInt32 pclock, UInt32 pstartTime, UInt32 ppostTriggerDelay, UInt32 ptriggerMask, Int32 ptriggerAnd, Int32 ptriggerStore, Int32 ptriggerSourceData)

Public Attributes

- UInt32 source

  The source data (0 - 3).

- UInt32 clock

  The Clock source.

- UInt32 startTime

  The start time in ms from CYCLE_START before trigger is activated.

- UInt32 postTriggerDelay

  The delay, in clock cycles, after the trigger before capture starts.

- UInt32 triggerMask

  The Trigger bit mask. This is the bit mask of the 8 timing signals.

- Int32 triggerAnd

  The Trigger function is an AND rather than an OR.

- Int32 triggerStore

  Store the trigger in the upper 8 data bits.

- Int32 triggerSourceData

  Use lower 32bits of data as trigger source rather than timing signals.

9.78.1 Detailed Description

This class defines the parameters for a test data capture.

9.78.2 Constructor & Destructor Documentation

9.78.2.1 Tms::TestCaptureInfo::TestCaptureInfo ()

9.78.2.2 Tms::TestCaptureInfo::TestCaptureInfo (UInt32 *psource*, UInt32 *pclock*, UInt32 *pstartTime*, UInt32 *ppostTriggerDelay*, UInt32 *ptriggerMask*, Int32 *ptriggerAnd*, Int32 *ptriggerStore*, Int32 *ptriggerSourceData*)

9.78.3 Member Data Documentation

9.78.3.1 UInt32 Tms::TestCaptureInfo::source

The source data (0 - 3).

9.78.3.2 UInt32 Tms::TestCaptureInfo::clock

The Clock source.

9.78.3.3 UInt32 Tms::TestCaptureInfo::startTime

The start time in ms from CYCLE_START before trigger is activated.

9.78.3.4 UInt32 Tms::TestCaptureInfo::postTriggerDelay

The delay, in clock cycles, after the trigger before capture starts.

9.78.3.5 UInt32 Tms::TestCaptureInfo::triggerMask

The Trigger bit mask. This is the bit mask of the 8 timing signals.

9.78.3.6 Int32 Tms::TestCaptureInfo::triggerAnd

The Trigger function is an AND rather than an OR.

9.78.3.7 Int32 Tms::TestCaptureInfo::triggerStore

Store the trigger in the upper 8 data bits.

9.78.3.8 Int32 Tms::TestCaptureInfo::triggerSourceData

Use lower 32bits of data as trigger source rather than timing signals.

The documentation for this class was generated from the following files:

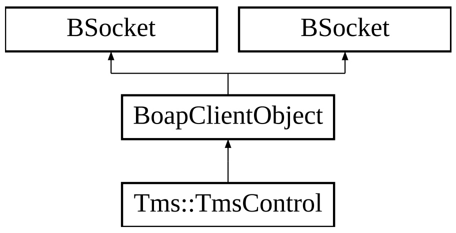
- TmsD.h
- TmsD.cc

9.79 Tms::TmsControl Class Reference

This interface provides functions to control, test and get statistics from the TMS as a whole.

```
#include <TmsC.h>
```

Inheritance diagram for Tms::TmsControl::



Public Member Functions

- TmsControl (BString name="")
- BError getVersion (BString &version)

  Gets the software version.

- BError setProcessPriority (UInt32 priority)

  Sets the priority of the process servicing this service.

- BError init ()

  Initialises the system including resetting all of the PUPE engines firmware. The call will return an error object indicating success or an error condition as appropriate.

- BError configure (ConfigInfo configInfo)

  Configure the system for use. This includes mapping the individual physical PickUp channels to logical pickup channels.

- BError getConfiguration (ConfigInfo &configInfo)

  Get the current configuration.

- BError setControlInfo (CycleParam params)

  Sets the control information for the cycle type given. The parameters for the processing cycle are passed, this includes the Phase and State table information. The call will return an error object indicating success or an error.

- BError getControlInfo (BString cycleType, UInt32 puChannel, CycleParam &params)

  Gets the control information for the cycle type and puChannel number given. The call will return an error object indicating success or an error.

- BError delControlInfo (BString cycleType, UInt32 puChannel)

  Deletes the control information for the cycle type and puChannel number given. The call will return an error object indicating success or an error.

- BError getControlList (BList< CycleParamItem > &itemList)

  Gets the list of Cycle Parameters present in the system.

- BError setNextCycle (UInt32 cycleNumber, BString cycleType)

  Sets the cycle number and type for the next processing cycle. The call will return an error object indicating success or an error condition as appropriate. This should be called at least 10ms before the next CYCLE_-START event.

- BError test (BList< BError > &errors)

  Performs a basic test of the system returning a list of errors. The call will return an error object indicating success or an error condition as appropriate.

- BError getStatus (BList< NameValue > &statusList)

  Returns the current status of the system. This information includes the number of Pick-Up's present and their individual status.

- BError getStatistics (BList< NameValue > &statsList)

  Returns a list of the statistic values as name/value pairs. The call will return an error object indicating success or an error condition as appropriate.

- BError getPuChannel (UInt32 puChannel, PuChannel &puPhysChannel)

  Returns the physical module/Pupe/Channel number given a logical PickUp id. This can be used so that the individual PickUps test functions can be accessed etc.

- BError setSimulation (Simulation simulation)

  Sets overall simulation modes.

- BError getSimulation (Simulation &simulation)

  Gets current simulation modes.

- BError setTestMode (PuChannel puPhysChannel, UInt32 testOutput, UInt32 timingDisableMask)

  The signal source for the digital test output connector. 0: None, 1: FrefLocal. The timingDisableMask bit mask defines which of the timing inputs should be disabled. If a timing input is disabled it can be still operated by software command.

- BError setTimingSignals (PuChannel puPhysChannel, UInt32 timingSignals)

  This function sets the given timing signals to the values as defined in the timingSignals bit array.

- BError captureDiagnostics (PuChannel puPhysChannel, TestCaptureInfo captureInfo, BArray< UInt64 > &data)

  This function will capture the diagnostics.

- BError setTestData (PuChannel puPhysChannel, Int32 on, BArray< UInt32 > data)

  This function will set a PU channel to sample data from memory rather than the ADC's.

- BError setPupeConfig (PuChannel puPhysChannel, PupeConfig pupeConfig)

  Sets special PUPE configuration for test purposes.

- BError getPupeConfig (PuChannel puPhysChannel, PupeConfig &pupeConfig)

  Gets special PUPE configuration for test purposes.

- BError puServerStarted (UInt32 number)

  A TmsPuServer has started.

9.79.1 Detailed Description

This interface provides functions to control, test and get statistics from the TMS as a whole.

9.79.2 Constructor & Destructor Documentation

9.79.2.1 Tms::TmsControl::TmsControl (BString *name* = "")

9.79.3 Member Function Documentation

9.79.3.1 BError Tms::TmsControl::getVersion (BString & *version*)

Gets the software version.

Parameters:

version A string variable filled in with the version number string.

9.79.3.2 BError Tms::TmsControl::setProcessPriority (UInt32 *priority*)

Sets the priority of the process servicing this service.

Parameters:

priority This is the priority of the process. It can be set to one of: PriorityLow, PriorityNormal, PriorityHigh.

9.79.3.3 BError Tms::TmsControl::init ()

Initialises the system including resetting all of the PUPE engines firmware. The call will return an error object indicating success or an error condition as appropriate.

This function restarts the TMS system. It re-initialises each of the TmsPuServer processes running on the Module Controllers and reboots each of the PUPE boards from scratch loading the current FPGA firmware. All errors and statistics values are reset.

9.79.3.4 BError Tms::TmsControl::configure (ConfigInfo *configInfo*)

Configure the system for use. This includes mapping the individual physical PickUp channels to logical pickup channels.

Parameters:

configInfo The channel mapping table.

This function configures the logical to physical channel mapping table.

9.79.3.5 BError Tms::TmsControl::getConfiguration (ConfigInfo & *configInfo*)

Get the current configuration.

Parameters:

configInfo The channel mapping table that is filled in with the current curent channel mapping.

This function reads the current logical to physical channel mapping table.

9.79.3.6 BError Tms::TmsControl::setControlInfo (CycleParam *params*)

Sets the control information for the cycle type given. The parameters for the processing cycle are passed, this includes the Phase and State table information. The call will return an error object indicating success or an error.

Parameters:

params Cycle information parameters (state/phase table information).

This function over-writes or adds an entry in the Cycle Parameter database. The Cycle Parameters define the setting for each processing cycle including the state and phase tables for the PUPE FPGA engines.

9.79.3.7 BError Tms::TmsControl::getControlInfo (BString *cycleType*, UInt32 *puChannel*, CycleParam & *params*)

Gets the control information for the cycle type and puChannel number given. The call will return an error object indicating success or an error.

Parameters:

cycleType This string defines the cycle type for which to get the information.

puChannel This defines the channel to get the information for. 0 means all channels.

params The resuting cycle parameters are placed in this object.

This function reads back the set of Cycle parameters for the given cycle type and channel number. Normall the same cycle parameters are used for all PUPE engines. In this case setting the puChannel to 0 reads the Cycle Parameters that are being used on all channels. If a specific channel has other parameters the puChannel variable can be set to the appropriate channel number to get its particular settings.

9.79.3.8 BError Tms::TmsControl::delControlInfo (BString *cycleType*, UInt32 *puChannel*)

Deletes the control information for the cycle type and puChannel number given. The call will return an error object indicating success or an error.

Parameters:

cycleType This string defines the cycle type to delete from the database.

puChannel This defines the specific channel to delete the information for. 0 means all channels.

This function will delete a set of Cycle parameters from the TMS's Cycle parameter database.

9.79.3.9 BError Tms::TmsControl::getControlList (BList< CycleParamItem > & *itemList*)

Gets the list of Cycle Parameters present in the system.

Parameters:

itemList The list of CycleType information is returned.

This function will return a list of entries describing the Cycle Paramter sets present in the TMS database.

9.79.3.10 BError Tms::TmsControl::setNextCycle (UInt32 *cycleNumber*, BString *cycleType*)

Sets the cycle number and type for the next processing cycle. The call will return an error object indicating success or an error condition as appropriate. This should be called at least 10ms before the next CYCLE_-START event.

Parameters:

cycleNumber This is the next cycle number. This should be an incrementing 32bit unsigned value.

cycleType This is a string defining the cycle type for the next cycle.

This call configures the TMS system for the next processing cycle. It defines the cycle number that will be used to tag data captured during the cycle and it defines the type of machine cycle. The cycleType is used to lookup the appropriate state/phase table information to use in the FPGA's. The call should be made at least 10ms before the CYCLE_START event for the cycle it refers to. This gives time for the FPGA's to be loaded with the appropriate state/phase table information. As the function is time critical, the communications channel should be set to a high priority using the setPriority() call and the processing threads priority should be set to high using the setProcessPriority() call. The call will return the error: "ErrorCycleNumber", "The next cycle has already started" if the call has not completed before the CYCLE_START event. All client data reads, for this cycle, will also return this error message.

9.79.3.11 BError Tms::TmsControl::test (BList< BError > & *errors*)

Performs a basic test of the system returning a list of errors. The call will return an error object indicating success or an error condition as appropriate.

Parameters:

errors The list of errors is placed in this list object.

This function will perform a test of the TMS system. It will report each test performed and the status of the test in the BError object. A status value of 0 indicates all was Ok, any other value is an error where the number indicates the error. A string gives the test name and the Ok or error condition as a string.

9.79.3.12 BError Tms::TmsControl::getStatus (BList< NameValue > & *statusList*)

Returns the current status of the system. This information includes the number of Pick-Up's present and their individual status.

Parameters:

statusList The list of status items is placed in this list object.

This function gets the status of the TMS system. It returns a list of name/value pairs.

9.79.3.13 BError Tms::TmsControl::getStatistics (BList< NameValue > & *statsList*)

Returns a list of the statistic values as name/value pairs. The call will return an error object indicating success or an error condition as appropriate.

Parameters:

statsList The statistics list is placed in this list object.

This function gets the statistics values from the TMS system. It returns a list of name/value pairs.

9.79.3.14 BError Tms::TmsControl::getPuChannel (UInt32 *puChannel*, PuChannel & *puPhysChannel*)

Returns the physical module/Pupe/Channel number given a logical PickUp id. This can be used so that the individual PickUps test functions can be accessed etc.

Parameters:

puChannel The logical channel number.

puPhysChannel The physical channel identifier is returned in this variable.

This function is given a logical pick-up channel number. It will return the physical module, pupe number and pupe channle that has been allocated to this channel.

9.79.3.15 BError Tms::TmsControl::setSimulation (Simulation *simulation*)

Sets overall simulation modes.

9.79.3.16 BError Tms::TmsControl::getSimulation (Simulation & *simulation*)

Gets current simulation modes.

9.79.3.17 BError Tms::TmsControl::setTestMode (PuChannel *puPhysChannel*, UInt32 *testOutput*, UInt32 *timingDisableMask*)

The signal source for the digital test output connector. 0: None, 1: FrefLocal. The timingDisableMask bit mask defines which of the timing inputs should be disabled. If a timing input is disabled it can be still operated by software command.

Parameters:

puPhysChannel The physical channel identifier.

testOutput The signal to output on the test output. 0 is FREF any other value is undefined at the moment.

timingDisableMask This 8 bit mask defines which of the timing input signals are disabled.

This function sets up a particular pick-up channel's digital test output source and allows the channels input timing signals to be set to a software driven mode rather than taken from the hardware timing inputs. The timing mask bits are: 7 - FREF, 6 - HCHANGE, 5 - INJECTION, 4 - CAL_STOP, 3 - CAL_START, 2 - CYCLE_STOP, 1 - CYCLE_START, 0 - SYSCLOCK

9.79.3.18 BError Tms::TmsControl::setTimingSignals (PuChannel *puPhysChannel*, UInt32 *timingSignals*)

This function sets the given timing signals to the values as defined in the timingSignals bit array.

Parameters:

puPhysChannel The physical channel identifier.

timingSignals The 8 bit mask defining the state of the software driven timing signals.

If the setTestMode() function had been used to "enable" particular timing signals to be driven by software, then this function can be used to set/reset particular timing signals for the pick-up channel given. The timing signals bits are: 7 - FREF, 6 - HCHANGE, 5 - INJECTION, 4 - CAL_STOP, 3 - CAL_START, 2 - CYCLE_STOP, 1 - CYCLE_START, 0 - SYSCLOCK

9.79.3.19 BError Tms::TmsControl::captureDiagnostics (PuChannel *puPhysChannel*, TestCaptureInfo *captureInfo*, BArray< UInt64 > & *data*)

This function will capture the diagnostics.

9.79.3.20 BError Tms::TmsControl::setTestData (PuChannel *puPhysChannel*, Int32 *on*, BArray< UInt32 > *data*)

This function will set a PU channel to sample data from memory rather than the ADC's.

Parameters:

puPhysChannel The physical channel identifier.

on Boolean to enable the internal data source. 0 is off, 1 is on.

data The array of 32bit data values to use as the FREF,Sigma,DeltaX and DeltaY test signal.

This call loads the PUPE systems test data SDRAM with the data passed in the data array. It then sets up the individual channel to sources its FREF, Sigma, DeltaX and DelatY signals from the test SDRAM. The data source should have a multiple of 2 samples. The "on" parameter is used to enable or disable the individual channels inputs from this test data SDRAM.

9.79.3.21 BError Tms::TmsControl::setPupeConfig (PuChannel *puPhysChannel*, PupeConfig *pupeConfig*)

Sets special PUPE configuration for test purposes.

Parameters:

puPhysChannel The physical channel identifier.

pupeConfig The configuration parameters to use.

This functions sets up some special configuration parameters for the PUPE channel. It is used mainly for diagnostics and test purposes. The main settings it can affect are: The ADC Clock sources PLL synchronisation, internal timing for the digtital timing signals and the enabling/dissabling of the BLR algorithem.

9.79.3.22 BError Tms::TmsControl::getPupeConfig (PuChannel *puPhysChannel*, PupeConfig & *pupeConfig*)

Gets special PUPE configuration for test purposes.

Parameters:

puPhysChannel The physical channel identifier.

pupeConfig The returned configuration parameters.

This function returns the current configuration of the given channel.

9.79.3.23 BError Tms::TmsControl::puServerStarted (UInt32 *number*)

A TmsPuServer has started.

Parameters:

number The number of the PuServer started.

This is an internal function called by the TmsPuServer processes to indicate to the TmsServer that they have just started running and are present in the system. The TmsServer will initialise the appropriate tmsPuServer program and its individual PUPE engines on receipt of this call.

The documentation for this class was generated from the following files:

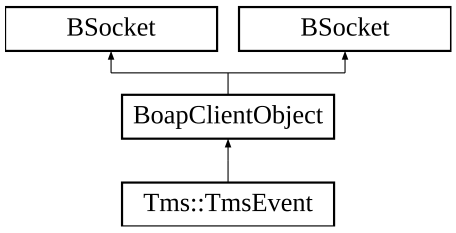
- TmsC.h
- TmsC.cc
- tmsFunctions.dox

9.80 Tms::TmsEvent Class Reference

This interface provides functions for events to be sent to clients from the TMS as a whole.

```
#include <TmsC.h>
```

Inheritance diagram for Tms::TmsEvent::



Public Member Functions

- TmsEvent (BString name="")
- BError errorEvent (UInt32 cycleNumber, BError error)

  This event function gets called on a system error. The errorEvent object contains and error number and string describing the error. The getStatus() call can be used to fetch further information.

- BError cycleStartEvent (UInt32 cycleNumber)

  This event function gets called on the CYCLE_START event with the cycle number about to be processed.

- BError cycleStopEvent (UInt32 cycleNumber)

  This event function gets called on the CYCLE_END event with the cycle number completed.

- BError dataEvent (DataInfo dataInfo)

  This event function gets called when some requested data becomes available. The DataInfo object contains information on the data. The getData() call can be used to fetch the actual data.

9.80.1 Detailed Description

This interface provides functions for events to be sent to clients from the TMS as a whole.

9.80.2 Constructor & Destructor Documentation

9.80.2.1 Tms::TmsEvent::TmsEvent (BString *name* = "")

9.80.3 Member Function Documentation

9.80.3.1 BError Tms::TmsEvent::errorEvent (UInt32 *cycleNumber*, BError *error*)

This event function gets called on a system error. The errorEvent object contains and error number and string describing the error. The getStatus() call can be used to fetch further information.

9.80.3.2 BError Tms::TmsEvent::cycleStartEvent (UInt32 *cycleNumber*)

This event function gets called on the CYCLE_START event with the cycle number about to be processed.

9.80.3.3 BError Tms::TmsEvent::cycleStopEvent (UInt32 *cycleNumber*)

This event function gets called on the CYCLE_END event with the cycle number completed.

9.80.3.4 BError Tms::TmsEvent::dataEvent (DataInfo *dataInfo*)

This event function gets called when some requested data becomes available. The DataInfo object contains information on the data. The getData() call can be used to fetch the actual data.

The documentation for this class was generated from the following files:

- TmsC.h
- TmsC.cc

9.81 Tms::TmsEventServerList Class Reference

```
#include <TmsEventServerList.h>
```

Public Member Functions

- TmsEventServerList ()
- ∼TmsEventServerList ()
- BError append (BString name)
- BError del (BString name)
- BError errorEvent (UInt32 cycleNumber, BError error)

  This event function gets called on a system error. The errorEvent object contains and error number and string describing the error. The getStatus() call can be used to fetch further information.
- BError cycleStartEvent (UInt32 cycleNumber)

  This event function gets called on the CYCLE_START event with the cycle number about to be processed.
- BError cycleStopEvent (UInt32 cycleNumber)

  This event function gets called on the CYCLE_END event with the cycle number completed.
- BError dataEvent (DataInfo dataInfo)

  This event function gets called when some requested data becomes available. The DataInfo object contains information on the data. The getData() call can be used to fetch the actual data.

Private Attributes

- BMutex olock
- BList< TmsEvent ∗ > oeventServers

9.81.1 Constructor & Destructor Documentation

9.81.1.1 Tms::TmsEventServerList::TmsEventServerList ()

9.81.1.2 Tms::TmsEventServerList::∼TmsEventServerList ()

9.81.2 Member Function Documentation

9.81.2.1 BError Tms::TmsEventServerList::append (BString *name*)

9.81.2.2 BError Tms::TmsEventServerList::del (BString *name*)

9.81.2.3 BError Tms::TmsEventServerList::errorEvent (UInt32 *cycleNumber*, BError *error*)

This event function gets called on a system error. The errorEvent object contains and error number and string describing the error. The getStatus() call can be used to fetch further information.

9.81.2.4 BError Tms::TmsEventServerList::cycleStartEvent (UInt32 *cycleNumber*)

This event function gets called on the CYCLE_START event with the cycle number about to be processed.

9.81.2.5 BError Tms::TmsEventServerList::cycleStopEvent (UInt32 *cycleNumber*)

This event function gets called on the CYCLE_END event with the cycle number completed.

9.81.2.6 BError Tms::TmsEventServerList::dataEvent (DataInfo *dataInfo*)

This event function gets called when some requested data becomes available. The DataInfo object contains information on the data. The getData() call can be used to fetch the actual data.

9.81.3 Member Data Documentation

9.81.3.1 BMutex Tms::TmsEventServerList::olock `[private]`

9.81.3.2 BList<TmsEvent*> Tms::TmsEventServerList::oeventServers `[private]`

The documentation for this class was generated from the following files:

- TmsEventServerList.h
- TmsEventServerList.cc

9.82 Tms::TmsPhase Union Reference

The Tms Phase Table Entry.

`#include <TmsLib.h>`

Public Attributes

- struct {
  unsigned int lo1:1
  unsigned int blr:1
  unsigned int gate:1
  unsigned int lo2:1
  unsigned int spare:2
  unsigned int meanFilter1:1
  unsigned int meanFilter2:1
  };
- unsigned char value

9.82.1 Detailed Description

The Tms Phase Table Entry.

9.82.2 Member Data Documentation

9.82.2.1 unsigned int Tms::TmsPhase::lo1

9.82.2.2 unsigned int Tms::TmsPhase::blr

9.82.2.3 unsigned int Tms::TmsPhase::gate

9.82.2.4 unsigned int Tms::TmsPhase::lo2

9.82.2.5 unsigned int Tms::TmsPhase::spare

9.82.2.6 unsigned int Tms::TmsPhase::meanFilter1

9.82.2.7 unsigned int Tms::TmsPhase::meanFilter2

9.82.2.8 struct { ... }

9.82.2.9 unsigned char Tms::TmsPhase::value

The documentation for this union was generated from the following file:

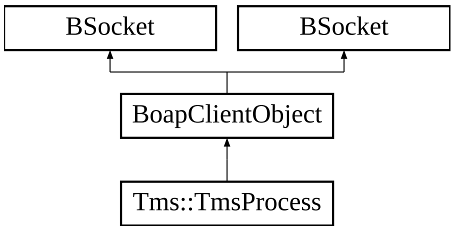
- TmsLib.h

9.83 Tms::TmsProcess Class Reference

This interface provides functions to capture data from the TMS as a whole.

```
#include <TmsC.h>
```

Inheritance diagram for Tms::TmsProcess::



Public Member Functions

- TmsProcess (BString name="")
- BError getVersion (BString &version)

  Gets the software version.

- BError getCycleInfo (UInt32 &cycleNumber, BString &cycleType)

  Gets the current cycle number and type.

- BError getCycleInformation (UInt32 cycleNumber, CycleInformation &cycleInformation)

  Gets information on given cycle number.

- BError getCycleTypeInformation (BString cycleType, CycleTypeInformation &cycleTypeInformation)

  Gets information on given cycle Type.

- BError getData (DataInfo dataInfo, Data &data)

  This function returns a set of data from the data present in the data cache or directly from the Pick-Up processing engines. The DataInfo object describes the data required. The call will return the required data along with an error object indicating success or an error condition as appropriate. The call will block until data is ready.

- BError addEventServer (BString name)

  This call adds an event server to call on events such as the "dataEvent" generated by the requestData(0 call as well as error events. The Client will use this to notify the TmsServer of its local TmsEvent object.

- BError requestData (DataInfo dataInfo)

  This adds a request for some data. The DataInfo object defines the data required. This request can be made at any time. If the data is present in cache the data will be available immediately, if not the system will await the data from a subsequent processing cycle. When the data is available a "data" event will be sent to the client. Not that it is not necessary to use requestData. The client can call getData() directly although this call will block until the data is actually ready.

9.83.1 Detailed Description

This interface provides functions to capture data from the TMS as a whole.

9.83.2 Constructor & Destructor Documentation

9.83.2.1 Tms::TmsProcess::TmsProcess (BString *name* = "")

Parameters:

name The name of the TmsProcess BOAP object to connect to.

The BOAP object name has the general form: "//HostName/ObjectName"

9.83.3 Member Function Documentation

9.83.3.1 BError Tms::TmsProcess::getVersion (BString & *version*)

Gets the software version.

Parameters:

version A string variable filled in with the version number string.

9.83.3.2 BError Tms::TmsProcess::getCycleInfo (UInt32 & *cycleNumber*, BString & *cycleType*)

Gets the current cycle number and type.

Parameters:

cycleNumber The current cycle number is returned here

cycleType The current cycle type is returned here.

This function returns the current TMS cycle number being processed and the type of the cycle.

9.83.3.3 BError Tms::TmsProcess::getCycleInformation (UInt32 *cycleNumber*, CycleInformation & *cycleInformation*)

Gets information on given cycle number.

Parameters:

cycleNumber The current cycle number to get information on

cycleInformation The returned cycle information

This function provides information on the given cycle. It interrogates the first PUPE channel and returns the Cycle information based on the internal state of that channel. The main information returned includes the cycle type and a list of all of the Cycle Periods captured and the times and ammount of data in each.

9.83.3.4 BError Tms::TmsProcess::getCycleTypeInformation (BString *cycleType*, CycleTypeInformation & *cycleTypeInformation*)

Gets information on given cycle Type.

9.83.3.5 BError Tms::TmsProcess::getData (DataInfo *dataInfo*, Data & *data*)

This function returns a set of data from the data present in the data cache or directly from the Pick-Up processing engines. The DataInfo object describes the data required. The call will return the required data along with an error object indicating success or an error condition as appropriate. The call will block until data is ready.

Parameters:

dataInfo Information on the type of data required.

data The raw data is returned in this object.

This is the main user function used by clients of the TMS system. It is used to return portions of the acquired data. The fields of the dataInfo parameter define which data is required and are defined in the DataInfo class documentation.

The call will check to see if the data for the cycle number requested is still present in the PUPE memory. The PUPE memory has enough storage for about 3 seconds worth of data (3 processing cycles). If the data has gone the call will return the error "ErrorDataGone". If the system has not processed the requested cycle, but will do so within 256 seconds, the call will block awaiting the data.

If the channel number is given as 0 the call will interrogate each of the Pick-Up channels and return the combined data from all of them. Note that this could take significant time and may not be possible if the parameter numValues is large. Within the Data structure returned there is an array of error values, one per channel. If an error occurs on any set of the channels the call will return the first error that occurred and the complete list of errors in the errors array. The actual data will be returned for all channels that did not have an error. Those channels that had anb error will have data values of 0 returned.

If the bunch number is given as 0, then the system will return the data for all of the bunches.

The data will be returned in the following order, where B - Bunch, C - Channel:

[C1.B1, C1.B2, C1.B3, C1.B4], [C1.B1, C1.B2, C1.B3, C1.B4], ... [C2.B1, C2.B2, C2.B3, C2.B4], [C2.B1, C2.B2, C2.B3, C2.B4], ...

That is the data is ordered by bunch, then sample, then channel. See the TMS Software documentation manual for more details of this functions operation.

9.83.3.6 BError Tms::TmsProcess::addEventServer (BString *name*)

This call adds an event server to call on events such as the "dataEvent" generated by the requestData(0 call as well as error events. The Client will use this to notify the TmsServer of its local TmsEvent object.

Parameters:

name The BOAP object name to add.

Adds an event server that gets called on certain TmsServer events such as data ready, CYCLE_START, CYCLE_STOP and errors.

9.83.3.7 BError Tms::TmsProcess::requestData (DataInfo *dataInfo*)

This adds a request for some data. The DataInfo object defines the data required. This request can be made at any time. If the data is present in cache the data will be available immediately, if not the system will await the data from a subsequent processing cycle. When the data is available a "data" event will be sent to the client. Not that it is not necessary to use requestData. The client can call getData() directly although this call will block until the data is actually ready.

Parameters:

dataInfo Information on the type of data required.

This calls sets up a request for data. The dataInfo parameter works in the same manner as the "getData" call, defining the portion of data required. This call will return immediatly. Assuming the client has informed the TMS system of an event server object using the "addEventServer" call, then the client will receive the "dataEvent" event when the data become available. The client can then fetch the data using the conventional "getData" call. In the current version of the software the "requestData" call simply sends a message when the data for the requested cycle is ready. In future implementations the TMS system could actually fetch the data automatically from the PUPE boards and store it in memory ready for later retrieval by the getData call.

The documentation for this class was generated from the following files:

- TmsC.h
- TmsC.cc
- tmsFunctions.dox

9.84 Tms::TmsState Union Reference

The Tms State entry.

`#include <TmsLib.h>`

Public Attributes

- struct {
  unsigned int aquireData:1
  unsigned int pllReference1:1
  unsigned int pllReference2:1
  unsigned int pllFeedbackSelect:1
  unsigned int pllLO1FromAddress:1
  unsigned int pllLO2FromAddress:1
  unsigned int spare0:2
  unsigned int cycleStop:4
  unsigned int calStop:4
  unsigned int calStart:4
  unsigned int injection:4
  unsigned int hchange:4
  unsigned int delay:4
  };
- unsigned int value

9.84.1 Detailed Description

The Tms State entry.

9.84.2 Member Data Documentation

9.84.2.1 unsigned int Tms::TmsState::aquireData

9.84.2.2 unsigned int Tms::TmsState::pllReference1

9.84.2.3 unsigned int Tms::TmsState::pllReference2

9.84.2.4 unsigned int Tms::TmsState::pllFeedbackSelect

9.84.2.5 unsigned int Tms::TmsState::pllLO1FromAddress

9.84.2.6 unsigned int Tms::TmsState::pllLO2FromAddress

9.84.2.7 unsigned int Tms::TmsState::spare0

9.84.2.8 unsigned int Tms::TmsState::cycleStop

9.84.2.9 unsigned int Tms::TmsState::calStop

9.84.2.10 unsigned int Tms::TmsState::calStart

9.84.2.11 unsigned int Tms::TmsState::injection

9.84.2.12 unsigned int Tms::TmsState::hchange

9.84.2.13 unsigned int Tms::TmsState::delay

9.84.2.14 struct { ... }

9.84.2.15 unsigned int Tms::TmsState::value

The documentation for this union was generated from the following file:

- TmsLib.h

Chapter 10

File Documentation

10.1 /src/cern/tms/beam/libBeam/BArray.h File Reference

```
#include <BTypes.h>
#include <vector>
```

Classes

- class BArray< T >

Defines

- #define BArray_H 1

10.1.1 Define Documentation

10.1.1.1 #define BArray_H 1

10.2 /src/cern/tms/beam/libBeam/BBuffer.cpp File Reference

```
#include <stdlib.h>
#include <memory.h>
#include <BBuffer.h>
```

Defines

- #define SIZE 1024

10.2.1 Define Documentation

10.2.1.1 #define SIZE 1024

10.3 /src/cern/tms/beam/libBeam/BBuffer.h File Reference

```
#include <stdint.h>
```

Classes

- class BBuffer

Defines

- #define BBUFFER_H 1

10.3.1 Define Documentation

10.3.1.1 #define BBUFFER_H 1

10.4 /src/cern/tms/beam/libBeam/BCond.cpp File Reference

```
#include <BCond.h>
#include <sys/time.h>
#include <stdio.h>
```

10.5 /src/cern/tms/beam/libBeam/BCond.h File Reference

```
#include <pthread.h>
```

Classes

- class BCond

Defines

- #define BCOND_H 1

10.5.1 Define Documentation

10.5.1.1 #define BCOND_H 1

10.6 /src/cern/tms/beam/libBeam/BCondInt.cpp File Reference

```
#include <BCondInt.h>
#include <sys/time.h>
#include <stdio.h>
#include <errno.h>
```

10.7 /src/cern/tms/beam/libBeam/BCondInt.h File Reference

```
#include <BTypes.h>
#include <pthread.h>
```

Classes

- class BCondValue

  Thread conditional value.

- class BCondInt

  Thread conditional integer.

- class BCondBool

  Thread conditional boolean.

- class BCondWrap

Defines

- #define BCONDINT_H 1

10.7.1 Define Documentation

10.7.1.1 #define BCONDINT_H 1

10.8 /src/cern/tms/beam/libBeam/BDir.cpp File Reference

```
#include <BDir.h>
#include <dirent.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
```

Functions

- static int wild (const dirent *e)

Variables

- static BString wildString

10.8.1 Function Documentation

10.8.1.1 static int wild (const dirent * *e*) `[static]`

10.8.2 Variable Documentation

10.8.2.1 BString wildString `[static]`

10.9 /src/cern/tms/beam/libBeam/BDir.h File Reference

```
#include <BList.h>
#include <BString.h>
#include <BError.h>
#include <sys/stat.h>
```

Classes

- class BDir

  File system directory class.

Defines

- #define BDIR_H 1

10.9.1 Define Documentation

10.9.1.1 #define BDIR_H 1

10.10 /src/cern/tms/beam/libBeam/BEntry.cpp File Reference

```
#include <ctype.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <BEntry.h>
```

10.11 /src/cern/tms/beam/libBeam/BEntry.h File Reference

```
#include <BList.h>
#include <BString.h>
```

Classes

- class BEntry

  Manipulate a name value pair.

- class BEntryList

  List of Entries. Where an entry is a name value pair.

- class BEntryFile

  File of Entries.

10.12 /src/cern/tms/beam/libBeam/BError.cpp File Reference

```
#include <BError.h>
```

10.13 /src/cern/tms/beam/libBeam/BError.h File Reference

```
#include <BString.h>
```

Classes

- class BError
  Error return class.

Defines

- #define BERROR_H 1

10.13.1 Define Documentation

10.13.1.1 #define BERROR_H 1

10.14 /src/cern/tms/beam/libBeam/BEvent.cpp File Reference

```
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <BEvent.h>
#include <BPoll.h>
```

10.15 /src/cern/tms/beam/libBeam/BEvent.h File Reference

```
#include <stdint.h>
#include <BError.h>
```

Classes

- class BEvent

  This class provides a base class for all event objects that can be sent over the events interface.

- class BEventError
- class BEventPipe

  This class provides a base interface for sending events via a pipe. This allows threads to send events that can be picked up by the poll system call.

- class BEventInt

  This class provides an interface for sending simple integer events via a file descriptor. This allows threads to send events that can be picked up by the poll system call.

Defines

- #define BEvent_H 1

Enumerations

- enum BEventType { BEventTypeNone, BEventTypeInt, BEventTypeError }

10.15.1 Define Documentation

10.15.1.1 #define BEvent_H 1

10.15.2 Enumeration Type Documentation

10.15.2.1 enum BEventType

Enumerator:

BEventTypeNone

BEventTypeInt

BEventTypeError

10.16 /src/cern/tms/beam/libBeam/BFile.cpp File Reference

```
#include <BFile.h>
#include <sys/stat.h>
#include <string.h>
#include <stdarg.h>
#include <errno.h>
```

Defines

- #define STRBUF 10240

10.16.1 Define Documentation

10.16.1.1 #define STRBUF 10240

10.17 /src/cern/tms/beam/libBeam/BFile.h File Reference

```
#include <stdio.h>
#include <BString.h>
#include <BError.h>
```

Classes

- class BFile

  File operations class.

Defines

- #define BFILE_H 1

10.17.1 Define Documentation

10.17.1.1 #define BFILE_H 1

10.18 /src/cern/tms/beam/libBeam/BList.h File Reference

```
#include <BList_func.h>
```

Classes

- class BIter

    Iterator for BList.

- class BList< T >

    Template based list class.

- class BList< T >::Node

Defines

- #define BLIST_H 1

10.18.1 Define Documentation

10.18.1.1 #define BLIST_H 1

10.19 /src/cern/tms/beam/libBeam/BList_func.h File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <memory.h>
```

10.20 /src/cern/tms/beam/libBeam/BMutex.cpp File Reference

```
#include <BMutex.h>
```

Defines

- #define MDEBUG 0

10.20.1 Define Documentation

10.20.1.1 #define MDEBUG 0

10.21 /src/cern/tms/beam/libBeam/BMutex.h File Reference

```
#include <pthread.h>
```

Classes

- class BMutex
  Mutex class.

Defines

- #define BMUTEX_H 1

10.21.1 Define Documentation

10.21.1.1 #define BMUTEX_H 1

10.22 /src/cern/tms/beam/libBeam/BNameValue.h File Reference

```
#include <BList.h>
#include <BString.h>
```

Classes

- class BNameValue< T >
- class BNameValueList< T >

Defines

- #define BNAMEVALUE_H 1
- #define TEMPLATE_NEW 1

10.22.1 Define Documentation

10.22.1.1 #define BNAMEVALUE_H 1

10.22.1.2 #define TEMPLATE_NEW 1

10.23 /src/cern/tms/beam/libBeam/Boap.cpp File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <Boap.h>
#include <byteswap.h>
#include <BoapnsD.h>
#include <BoapnsC.h>
```

Defines

- #define DEBUG 0
- #define APIVERSION_TEST 1
- #define dprintf(fmt, a...)
- #define IS_BIG_ENDIAN 1

Functions

- static void swap8 (char ∗d, char ∗s)
- static void swap16 (char ∗d, char ∗s)
- static void swap32 (char ∗d, char ∗s)
- static void swap64 (char ∗d, char ∗s)

Variables

- const int boapPort = 12000

  The default BOAP connection port.

- const int roundSize = 256

10.23.1 Define Documentation

10.23.1.1 #define APIVERSION_TEST 1

10.23.1.2 #define DEBUG 0

10.23.1.3 #define dprintf(fmt, a...)

10.23.1.4 #define IS_BIG_ENDIAN 1

10.23.2 Function Documentation

10.23.2.1 static void swap16 (char ∗ *d*, char ∗ *s*) `[inline, static]`

10.23.2.2 static void swap32 (char ∗ *d*, char ∗ *s*) `[inline, static]`

10.23.2.3 static void swap64 (char ∗ *d*, char ∗ *s*) `[inline, static]`

10.23.2.4 static void swap8 (char ∗ *d*, char ∗ *s*) `[inline, static]`

10.23.3 Variable Documentation

10.23.3.1 const int boapPort = 12000

The default BOAP connection port.

10.23.3.2 const int roundSize = 256

10.24 /src/cern/tms/beam/libBeam/Boap.h File Reference

```
#include <stdint.h>
#include <BPoll.h>
#include <BSocket.h>
#include <BThread.h>
#include <BError.h>
#include <BEvent.h>
#include <BMutex.h>
#include <BTypes.h>
```

Namespaces

- namespace Boapns

Classes

- struct BoapPacketHead
- class BoapPacket
- class BoapClientObject
- class BoapSignalObject
- class BoapServiceEntry
- class BoapServerConnection
- class BoapServer
- class BoapFuncEntry
- class BoapServiceObject

Typedefs

- typedef UInt32 BoapService
- typedef BError(BoapServiceObject::∗ BoapFunc)(BoapServerConnection ∗conn, BoapPacket &rx, BoapPacket &tx)

Enumerations

- enum BoapType {
  BoapTypeRpc, BoapTypeRpcReply, BoapTypeSignal, BoapTypeRpc,
  BoapTypeSignal }
- enum BoapPriority { BoapPriorityLow, BoapPriorityNormal, BoapPriorityHigh }

Variables

- const UInt32 BoapMagic = 0x424F4100

10.24.1 Typedef Documentation

10.24.1.1 typedef BError(BoapServiceObject::$*$ BoapFunc)(BoapServerConnection $*$conn, BoapPacket &rx, BoapPacket &tx)

10.24.1.2 typedef UInt32 BoapService

10.24.2 Enumeration Type Documentation

10.24.2.1 enum BoapPriority

Enumerator:

BoapPriorityLow

BoapPriorityNormal

BoapPriorityHigh

10.24.2.2 enum BoapType

Enumerator:

BoapTypeRpc

BoapTypeRpcReply

BoapTypeSignal

BoapTypeRpc

BoapTypeSignal

10.24.3 Variable Documentation

10.24.3.1 const UInt32 BoapMagic = 0x424F4100

10.25 /src/cern/tms/beam/libBeam/BoapnsC.cc File Reference

```
#include <BoapnsC.h>
```

Namespaces

- namespace Boapns

Functions

- Boapns::Boapns (BString name)
- BError Boapns::getVersion (BString &version)
- BError Boapns::getEntryList (BList< BoapEntry > &entryList)
- BError Boapns::getEntry (BString name, BoapEntry &entry)
- BError Boapns::addEntry (BoapEntry entry)
- BError Boapns::delEntry (BString name)
- BError Boapns::getNewName (BString &name)

10.26 /src/cern/tms/beam/libBeam/BoapnsC.h File Reference

```
#include <stdlib.h>
#include <stdint.h>
#include <Boap.h>
#include <BString.h>
#include <BList.h>
#include <BArray.h>
#include <BoapnsD.h>
```

Namespaces

- namespace Boapns

Classes

- class Boapns::Boapns

Defines

- #define BOAPNSC_H 1

Variables

- const BUInt32 Boapns::apiVersion = 0

10.26.1 Define Documentation

10.26.1.1 #define BOAPNSC_H 1

10.27 /src/cern/tms/beam/libBeam/BoapnsD.cc File Reference

```
#include <BoapnsD.h>
```

Namespaces

- namespace Boapns

10.28 /src/cern/tms/beam/libBeam/BoapnsD.h File Reference

```
#include <Boap.h>
#include <BList.h>
#include <BArray.h>
```

Namespaces

- namespace Boapns

Classes

- class Boapns::BoapEntry

Defines

- #define BOAPNSD_H 1

10.28.1 Define Documentation

10.28.1.1 #define BOAPNSD_H 1

10.29 /src/cern/tms/beam/libBeam/BoapSimple.cc File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <Boap.h>
#include <BoapnsD.h>
#include <BoapnsC.h>
```

Defines

- #define DEBUG 0
- #define dprintf(fmt, a...)

Variables

- const int roundSize = 256

10.29.1 Define Documentation

10.29.1.1 #define DEBUG 0

10.29.1.2 #define dprintf(fmt, a...)

10.29.2 Variable Documentation

10.29.2.1 const int roundSize = 256

10.30 /src/cern/tms/beam/libBeam/BoapSimple.h File Reference

```
#include <stdint.h>
#include <BPoll.h>
#include <BSocket.h>
#include <BError.h>
```

Classes

- struct BoapPacketHead
- class BoapPacket
- class BoapClientObject
- class BoapSignalObject
- class BoapServiceEntry
- class BoapServer
- class BoapFuncEntry
- class BoapServiceObject

Typedefs

- typedef int8_t Int8
- typedef uint8_t UInt8
- typedef int16_t Int16
- typedef uint16_t UInt16
- typedef int32_t Int32
- typedef uint32_t UInt32
- typedef double Double
- typedef uint32_t BoapService
- typedef BError(BoapServiceObject::∗ BoapFunc)(BoapPacket &rx, BoapPacket &tx)

Enumerations

- enum BoapType {

  BoapTypeRpc, BoapTypeRpcReply, BoapTypeSignal, BoapTypeRpc,

  BoapTypeSignal }

10.30.1 Typedef Documentation

10.30.1.1 typedef BError(BoapServiceObject::∗ BoapFunc)(BoapPacket &rx, BoapPacket &tx)

10.30.1.2 typedef uint32_t BoapService

10.30.1.3 typedef double Double

10.30.1.4 typedef int16_t Int16

10.30.1.5 typedef int32_t Int32

10.30.1.6 typedef int8_t Int8

10.30.1.7 typedef uint16_t UInt16

10.30.1.8 typedef uint32_t UInt32

10.30.1.9 typedef uint8_t UInt8

10.30.2 Enumeration Type Documentation

10.30.2.1 enum BoapType

Enumerator:

BoapTypeRpc
BoapTypeRpcReply
BoapTypeSignal
BoapTypeRpc
BoapTypeSignal

10.31 /src/cern/tms/beam/libBeam/BObject.cc File Reference

```
#include <stdio.h>
#include <ctype.h>
#include <memory.h>
#include <string.h>
#include <BObject.h>
#include <iostream>
```

Defines

- #define DEBUG 0

10.31.1 Define Documentation

10.31.1.1 #define DEBUG 0

10.32 /src/cern/tms/beam/libBeam/BObject.h File Reference

```
#include <BType.h>
#include <BDataBuf.h>
#include <BString.h>
#include <BNameValue.h>
#include <BList.h>
#include <BError.h>
```

Classes

- class BObject

Defines

- #define BOBJECT_H 1

Typedefs

- typedef BNameValue< BObject ∗ > BMember
- typedef BNameValueList< BObject ∗ > BMemberList

10.32.1 Define Documentation

10.32.1.1 #define BOBJECT_H 1

10.32.2 Typedef Documentation

10.32.2.1 typedef BNameValue<BObject∗> BMember

10.32.2.2 typedef BNameValueList<BObject∗> BMemberList

10.33 /src/cern/tms/beam/libBeam/BPoll-1.cpp File Reference

```
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <BPoll.h>
```

10.34 /src/cern/tms/beam/libBeam/BPoll.cpp File Reference

```
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <BPoll.h>
```

10.35 /src/cern/tms/beam/libBeam/BPoll.h File Reference

```
#include <BList.h>
#include <BError.h>
#include <sys/poll.h>
```

Classes

- class BPoll

  This class provides an interface for polling a number of file descriptors. It uses round robin polling.

Defines

- #define BPOLL_H 1

10.35.1 Define Documentation

10.35.1.1 #define BPOLL_H 1

10.36 /src/cern/tms/beam/libBeam/BRefData.cpp File Reference

```
#include <stdlib.h>
#include <string.h>
#include <BRefData.h>
```

Defines

- #define DEBUG 0
- #define CHUNK 16

10.36.1 Define Documentation

10.36.1.1 #define CHUNK 16

10.36.1.2 #define DEBUG 0

10.37 /src/cern/tms/beam/libBeam/BRefData.h File Reference

Classes

- class BRefData

  Referenced data storage.

Defines

- #define BREFDATA_H 1

10.37.1 Define Documentation

10.37.1.1 #define BREFDATA_H 1

10.38 /src/cern/tms/beam/libBeam/BRtc.cpp File Reference

```
#include <BRtc.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/rtc.h>
```

10.39 /src/cern/tms/beam/libBeam/BRtc.h File Reference

```
#include <BError.h>
#include <BThread.h>
#include <BCond.h>
```

Classes

- class BRtc

  Realtime clock.

- class BRtcThreaded

  Threaded real time clock.

10.40 /src/cern/tms/beam/libBeam/BRWLock.cpp File Reference

```
#include <BRWLock.h>
```

10.41 /src/cern/tms/beam/libBeam/BRWLock.h File Reference

```
#include <pthread.h>
```

Classes

- class BRWLock

  thread read-write locks

Defines

- #define BRWLOCK_H 1

10.41.1 Define Documentation

10.41.1.1 #define BRWLOCK_H 1

10.42 /src/cern/tms/beam/libBeam/BSema.cpp File Reference

```
#include <BSema.h>
#include <errno.h>
#include <sys/time.h>
```

10.43 /src/cern/tms/beam/libBeam/BSema.h File Reference

```
#include <semaphore.h>
```

Classes

- class BSema

    Sempahore class.

Defines

- #define BSEMA_H 1

10.43.1 Define Documentation

10.43.1.1 #define BSEMA_H 1

10.44 /src/cern/tms/beam/libBeam/BSocket.cpp File Reference

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <net/if.h>
#include "BSocket.h"
```

Defines

- #define IP_MTU 14

10.44.1 Define Documentation

10.44.1.1 #define IP_MTU 14

10.45 /src/cern/tms/beam/libBeam/BSocket.h File Reference

```
#include <BString.h>
#include <BError.h>
#include <BTypes.h>
#include <stdint.h>
#include <sys/types.h>
#include <sys/prctl.h>
```

Classes

- class BSocketAddress
  Socket Address.
- class BSocketAddressINET
  IP aware socket address.
- class BSocket

Defines

- #define BSOCKET_H 1

10.45.1 Define Documentation

10.45.1.1 #define BSOCKET_H 1

10.46 /src/cern/tms/beam/libBeam/BString.cpp File Reference

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdarg.h>
#include <ctype.h>
#include "BString.h"
```

Defines

- #define DEBUG 0
- #define STRIP 0x7f
- #define MINUS '-'

Functions

- static int gmatch (const char ∗s, const char ∗p)
- std::ostream & operator<< (std::ostream &o, BString &s)
- std::istream & operator>> (std::istream &i, BString &s)

10.46.1 Define Documentation

10.46.1.1 #define DEBUG 0

10.46.1.2 #define MINUS '-'

10.46.1.3 #define STRIP 0x7f

10.46.2 Function Documentation

10.46.2.1 static int gmatch (const char ∗ *s*, const char ∗ *p*) `[static]`

10.46.2.2 std::ostream& operator<< (std::ostream & *o*, BString & *s*)

10.46.2.3 std::istream& operator>> (std::istream & *i*, BString & *s*)

10.47 /src/cern/tms/beam/libBeam/BString.h File Reference

```
#include <BRefData.h>
#include <BList.h>
#include <iostream>
```

Classes

- class BString

Defines

- #define BSTRING_H 1

Functions

- std::ostream & operator<< (std::ostream &o, BString &s)
- std::istream & operator>> (std::istream &i, BString &s)

10.47.1 Define Documentation

10.47.1.1 #define BSTRING_H 1

10.47.2 Function Documentation

10.47.2.1 std::ostream& operator<< (std::ostream & *o*, BString & *s*)

10.47.2.2 std::istream& operator>> (std::istream & *i*, BString & *s*)

10.48 /src/cern/tms/beam/libBeam/BThread.cpp File Reference

```
#include <BThread.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
```

10.49 /src/cern/tms/beam/libBeam/BThread.h File Reference

```
#include <pthread.h>
```

Classes

- class BThread

Defines

- #define BTHREAD_H 1

10.49.1 Define Documentation

10.49.1.1 #define BTHREAD_H 1

10.50 /src/cern/tms/beam/libBeam/BTimer.cpp File Reference

```
#include <BTimer.h>
#include <sys/time.h>
```

10.51 /src/cern/tms/beam/libBeam/BTimer.h File Reference

```
#include <BMutex.h>
```

Classes

- class BTimer

  Stopwatch style timer.

10.52 /src/cern/tms/beam/libBeam/BTypes.h File Reference

```
#include <stdint.h>
#include <sys/types.h>
#include <vector>
```

Defines

- #define BTYPES_H 1

Typedefs

- typedef int8_t BInt8
- typedef uint8_t BUInt8
- typedef int16_t BInt16
- typedef uint16_t BUInt16
- typedef int32_t BInt32
- typedef uint32_t BUInt32
- typedef int64_t BInt64
- typedef uint64_t BUInt64
- typedef float BFloat
- typedef double BDouble
- typedef size_t BSize
- typedef uint32_t BUInt
- typedef std::vector< float > BArrayFloat
- typedef std::vector< double > BArrayDouble
- typedef int8_t Int8
- typedef uint8_t UInt8
- typedef int16_t Int16
- typedef uint16_t UInt16
- typedef int32_t Int32
- typedef uint32_t UInt32
- typedef int64_t Int64
- typedef uint64_t UInt64
- typedef float Float
- typedef double Double

10.52.1 Define Documentation

10.52.1.1 #define BTYPES_H 1

10.52.2 Typedef Documentation

10.52.2.1 typedef std::vector<double> BArrayDouble

10.52.2.2 typedef std::vector<float> BArrayFloat

10.52.2.3 typedef double BDouble

10.52.2.4 typedef float BFloat

10.52.2.5 typedef int16_t BInt16

10.52.2.6 typedef int32_t BInt32

10.52.2.7 typedef int64_t BInt64

10.52.2.8 typedef int8_t BInt8

10.52.2.9 typedef size_t BSize

10.52.2.10 typedef uint32_t BUInt

10.52.2.11 typedef uint16_t BUInt16

10.52.2.12 typedef uint32_t BUInt32

10.52.2.13 typedef uint64_t BUInt64

10.52.2.14 typedef uint8_t BUInt8

10.52.2.15 typedef double Double

10.52.2.16 typedef float Float

10.52.2.17 typedef int16_t Int16

10.52.2.18 typedef int32_t Int32

10.52.2.19 typedef int64_t Int64

10.52.2.20 typedef int8_t Int8

10.52.2.21 typedef uint16_t UInt16

10.52.2.22 typedef uint32_t UInt32

10.52.2.23 typedef uint64_t UInt64

10.52.2.24 typedef uint8_t UInt8

10.53 /src/cern/tms/beam/libBeam/BUrl.cpp File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <BUrl.h>
#include <curl/curl.h>
```

10.54 /src/cern/tms/beam/libBeam/BUrl.h File Reference

```
#include <stdio.h>
#include <BString.h>
#include <BError.h>
```

Classes

- class BUrl

  Basic access to a Url.

Defines

- #define BURL_H 1

10.54.1 Define Documentation

10.54.1.1 #define BURL_H 1

10.55 overview.dox File Reference

10.56 SigGen.cpp File Reference

```
#include <SigGen.h>
#include <math.h>
#include <time.h>
```

Defines

- #define DEBUG 0
- #define dprintf(fmt, a...)

10.56.1 Define Documentation

10.56.1.1 #define DEBUG 0

10.56.1.2 #define dprintf(fmt, a...)

10.57 SigGen.h File Reference

```
#include <BList.h>
#include <BError.h>
```

Classes

- class BSignal
- class SigGen
- class SigGenSine
- class SigGenSquare
- class SigGenNoise
- class SigGenPulse
- class SigGenBeam

Defines

- #define SigGen_h 1

Typedefs

- typedef float Sample
- typedef BList< BSignal > BSignalList

10.57.1 Define Documentation

10.57.1.1 #define SigGen_h 1

10.57.2 Typedef Documentation

10.57.2.1 typedef BList<BSignal> BSignalList

10.57.2.2 typedef float Sample

10.58 test1.cpp File Reference

```
#include <stdio.h>
#include <TmsLib.h>
```

Functions

- void printCycleParams (Tms::CycleParam p)
- int main ()

10.58.1 Function Documentation

10.58.1.1 int main ()

10.58.1.2 void printCycleParams (Tms::CycleParam *p*)

10.59 TmsC.cc File Reference

```
#include <TmsC.h>
```

Namespaces

- namespace Tms

10.60 TmsC.h File Reference

This file contains the TmsAPi class definitions.

```
#include <stdlib.h>
#include <stdint.h>
#include <Boap.h>
#include <BString.h>
#include <BList.h>
#include <BArray.h>
#include <TmsD.h>
```

Namespaces

- namespace Tms

Classes

- class Tms::PuControl

  This class defines the parameters for a test data capture.
- class Tms::PuProcess

  This interface provides functions to configure and capture data from individual pick-up.
- class Tms::TmsControl

  This interface provides functions to control, test and get statistics from the TMS as a whole.
- class Tms::TmsProcess

  This interface provides functions to capture data from the TMS as a whole.
- class Tms::TmsEvent

  This interface provides functions for events to be sent to clients from the TMS as a whole.

Defines

- #define TMSC_H 1

Variables

- const BUInt32 Tms::apiVersion = 0

10.60.1 Detailed Description

This file contains the TmsAPi class definitions.

10.60.2 Define Documentation

10.60.2.1 #define TMSC_H 1

10.61 TmsCycleParam-1.cc File Reference

```
#include <stdlib.h>
#include <stdint.h>
#include <unistd.h>
#include <errno.h>
#include <math.h>
#include <TmsCycleParam.h>
#include <BFile.h>
#include <BEntry.h>
```

Namespaces

- namespace Tms

10.62 TmsCycleParam.cc File Reference

```
#include <stdlib.h>
#include <stdint.h>
#include <unistd.h>
#include <errno.h>
#include <math.h>
#include <TmsCycleParam.h>
#include <BFile.h>
#include <BEntry.h>
```

Namespaces

- namespace Tms

10.63 TmsCycleParam.h File Reference

```
#include <TmsLib.h>
```

Namespaces

- namespace Tms

Classes

- class Tms::CycleParamState
- class Tms::CycleParamEdit

  Cycle Parameter management class.

Defines

- #define TmsCycleParam_H 1

10.63.1 Define Documentation

10.63.1.1 #define TmsCycleParam_H 1

10.64 TmsD.cc File Reference

```
#include <TmsD.h>
```

Namespaces

- namespace Tms

10.65 TmsD.h File Reference

```
#include <Boap.h>
#include <BList.h>
#include <BArray.h>
```

Namespaces

- namespace Tms

Classes

- class Tms::NameValue
- class Tms::PuChannel

  This class stores a Physical Pick-Up channel id.

- class Tms::PuStatus

  This class stores the status of an individual Pick-Up.

- class Tms::ConfigInfo

  This class describes the configuration of the TMS.

- class Tms::DataInfo

  This class defines the data to be acquired and/or fetched.

- class Tms::DataValue

  This is the definition of a single data value.

- class Tms::Data

  This class stores the raw data.

- class Tms::PuStateTable

  This class defines the Pick-Up state table.

- class Tms::CycleParam

  This class defines the parameters for a PS processing cycle.

- class Tms::CycleParamItem
- class Tms::TestCaptureInfo

  This class defines the parameters for a test data capture.

- class Tms::PupeConfig
- class Tms::CycleInformationPeriod

  Cycle information.

- class Tms::CycleInformation
- class Tms::CycleTypeInformationPeriod

  Cycle Type information.

- class Tms::CycleTypeInformation
- class Tms::Simulation

Defines

- #define TMSD_H 1

Enumerations

- enum Tms::Errors {
  Tms::ErrorOk, Tms::ErrorMisc, Tms::ErrorWarning, Tms::ErrorInit,
  Tms::ErrorConfig, Tms::ErrorParam, Tms::ErrorNotImplemented, Tms::ErrorComms,
  Tms::ErrorCommsTimeout, Tms::ErrorMC, Tms::ErrorFpga, Tms::ErrorStateTable,
  Tms::ErrorCycleNumber, Tms::ErrorDataNotAvailable, Tms::ErrorDataGone, Tms::ErrorDataFuture,
  Tms::ErrorTimeout }
- enum Tms::CyclePeriod {
  Tms::CyclePeriodAll, Tms::CyclePeriodCalibration, Tms::CyclePeriodEvent0, Tms::CyclePeriodEvent1,
  Tms::CyclePeriodEvent2, Tms::CyclePeriodEvent3, Tms::CyclePeriodEvent4, Tms::CyclePeriodEvent5,
  Tms::CyclePeriodEvent6, Tms::CyclePeriodEvent7, Tms::CyclePeriodEvent8, Tms::CyclePeriodEvent9 }
- enum Tms::DataType { Tms::DataTypeRaw }
- enum Tms::DataFunction {
  Tms::DataFunctionRaw, Tms::DataFunctionMean, Tms::DataFunctionMeanAll, Tms::DataFunctionMean0,
  Tms::DataFunctionMean1 }
- enum Tms::TestOutput { Tms::TestOutputFrefLocal, Tms::TestOutputPllL1, Tms::TestOutputPllL2 }
- enum Tms::Priority { Tms::PriorityLow, Tms::PriorityNormal, Tms::PriorityHigh }

10.65.1 Define Documentation

10.65.1.1 #define TMSD_H 1

10.66 TmsEventServerList.cc File Reference

```
#include <stdlib.h>
#include <stdint.h>
#include <TmsEventServerList.h>
```

Namespaces

- namespace Tms

10.67 TmsEventServerList.h File Reference

```
#include <TmsD.h>
#include <TmsC.h>
```

Namespaces

- namespace Tms

Classes

- class Tms::TmsEventServerList

Defines

- #define TmsEventServerList_H 1

10.67.1 Define Documentation

10.67.1.1 #define TmsEventServerList_H 1

10.68 tmsFunctions.dox File Reference

Namespaces

- namespace Tms

10.69 TmsLib.cc File Reference

```
#include <stdlib.h>
#include <stdint.h>
#include <unistd.h>
#include <math.h>
#include <TmsLib.h>
#include <BDir.h>
#include <BEntry.h>
#include <BFile.h>
#include <TmsCycleParam.h>
```

Namespaces

- namespace Tms

10.70 TmsLib.h File Reference

```
#include <TmsD.h>
#include <TmsC.h>
```

Namespaces

- namespace Tms

Classes

- union Tms::TmsState

  The Tms State entry.

- union Tms::TmsPhase

  The Tms Phase Table Entry.

- class Tms::CycleParamDb

  Internal CycleParameter management class.

Defines

- #define TmsLib_H 1

Enumerations

- enum Tms::TimingSig {

  Tms::TimingSigClock = 0x01, Tms::TimingSigCycleStart = 0x02, Tms::TimingSigCycleStop = 0x04, Tms::TimingSigCalStart = 0x08,

  Tms::TimingSigCalStop = 0x10, Tms::TimingSigInjection = 0x20, Tms::TimingSigHChange = 0x40, Tms::TimingSigFRef = 0x80 }

  The timing signal bits.

- enum Tms::CaptureClock {

  Tms::ClkAdcDiv_1 = 0x00, Tms::ClkAdcDiv_2 = 0x01, Tms::ClkAdcDiv_5 = 0x02, Tms::ClkAdcDiv_10 = 0x03,

  Tms::ClkAdcDiv_20 = 0x04, Tms::ClkAdcDiv_50 = 0x05, Tms::ClkAdcDiv_100 = 0x06, Tms::ClkAdcDiv_200 = 0x07,

  Tms::ClkAdcDiv_500 = 0x08, Tms::ClkAdcDiv_1000 = 0x09, Tms::ClkAdcDiv_2000 = 0x0A, Tms::ClkAdcDiv_5000 = 0x0B,

  Tms::ClkAdcDiv_10000 = 0x0C, Tms::ClkAdcDiv_20000 = 0x0D, Tms::ClkAdcDiv_50000 = 0x0E, Tms::ClkAdcDiv_100000 = 0x0F,

  Tms::ClkMs = 0x10, Tms::ClkFref = 0x11 }

  The Diagnostics Capture Clock settings.

Variables

- const unsigned int Tms::tmsNumPickups = 40

  The default number of pick ups.

- const unsigned int Tms::tmsPhaseTableSize = 512

  The size of the Phase Table.

10.70.1 Define Documentation

10.70.1.1 #define TmsLib_H 1

10.71 TmsS.cc File Reference

```
#include <TmsC.h>
#include <TmsS.h>
```

Namespaces

- namespace Tms

10.72 TmsT.cc File Reference

```
#include <stdlib.h>
#include <stdint.h>
#include <TmsT.h>
```

Index