artdaq_core

3.10.01

Generated by Doxygen 1.8.5

Thu Sep 5 2024 10:40:39

Contents

Chapter 1

Todo List

Member artdaq::Fragment::dataFrag (sequence_id_t sequenceID, fragment_id_t fragID, InputIterator i, InputIterator e)

Change function access specifier to restrict access

Member artdaq::Fragment::Fragment (const Fragment &)

Decide if Copy constructor should be declared =delete

Member artdaq::Fragment::metadataAddress ()

Change function access specifier to restrict access

Member artdaq::Fragment::operator= (const Fragment &)

Decide if copy-assignment operator should be declared =delete

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

Namespace Documentation

5.1 anonymous_namespace{configureMessageFacility.cc} Namespace Reference

Functions

- fhicl::ParameterSet make_pset (std::string const &config_str)

  Make a fhicl::ParameterSet from a string (shim for compatibility)

5.1.1 Function Documentation

5.1.1.1 fhicl::ParameterSet anonymous_namespace{configureMessageFacility.cc}::make_pset (std::string const & *config_str*)

Make a fhicl::ParameterSet from a string (shim for compatibility)

Parameters

config_str	String to turn into a ParameterSet

Returns

fhicl::ParameterSet created from string

Definition at line 23 of file configureMessageFacility.cc.

5.2 artdaq Namespace Reference

The artdaq namespace.

Namespaces

- detail

  artdaq implementation details namespace
- TimeUtils

  Namespace to hold useful time-converting functions.

Classes

- struct MonitoredQuantityStats

  struct containing MonitoredQuantity data
- class MonitoredQuantity

  This class keeps track of statistics for a set of sample values and provides timing information on the samples.
- struct QuickVec

  A QuickVec behaves like a std::vector, but does no initialization of its data, making it faster at the cost of having to ensure that uninitialized data is not read.
- class SharedMemoryEventReceiver

  SharedMemoryEventReceiver can receive events (as written by SharedMemoryEventManager) from Shared Memory.
- class SharedMemoryFragmentManager

  The SharedMemoryFragmentManager is a SharedMemoryManager that deals with Fragment transfers using a SharedMemoryManager.
- class SharedMemoryManager

  The SharedMemoryManager creates a Shared Memory area which is divided into a number of fixed-size buffers. It provides for multiple readers and multiple writers through a dual semaphore system.
- class StatisticsCollection

  A collection of MonitoredQuantity instances describing low-level statistics of the artdaq system.
- class ContainerFragment

  The artdaq::ContainerFragment class represents a Fragment which contains other Fragments.
- class ContainerFragmentLoader

  A Read-Write version of the ContainerFragment, used for filling ContainerFragment objects with other Fragment objects.
- class Fragment

  A Fragment contains the data from one piece of the DAQ system for one event The artdaq::Fragment is the main data storage class in artdaq. Each Fragment represents the data from one piece of the readout, for one artdaq event. BoardReaders create Fragments and send them to the EventBuilders, where they are assembled into artdaq::RawEvent objects.
- class PackageBuildInfo

  Class holding information about the artdaq package build.
- class RawEvent

  RawEvent is the artdaq view of a generic event, containing a header and zero or more Fragments.
- class ArtdaqFragmentNameHelper

  Default implementation of FragmentNameHelper.
- class FragmentGenerator

  Base class for all FragmentGenerators.
- class FragmentNameHelper

  The FragmentNameHelper translates between Fragments and their instance names (usually by type, but any/all RawFragmentHeader fields, or even Overlays, may be used)
- class SimpleLookupPolicy

  This class is intended to find files using a set lookup order.

Typedefs

- typedef std::shared_ptr
  < MonitoredQuantity > MonitoredQuantityPtr

  A shared_ptr to a MonitoredQuantity instance.
- typedef
  detail::RawFragmentHeader::RawDataType RawDataType

The RawDataType (currently a 64-bit integer) is the basic unit of data representation within artdaq

- typedef std::vector< Fragment > Fragments

    A std::vector of Fragment objects.
- typedef std::unique_ptr< Fragment > FragmentPtr

    A std::unique_ptr to a Fragment object.
- typedef std::list< FragmentPtr > FragmentPtrs

    A std::list of FragmentPtrs.
- typedef std::shared_ptr< RawEvent > RawEvent_ptr

    A shared_ptr to a RawEvent.
- typedef std::unique_ptr
< artdaq::FragmentGenerator > makeFunc_t (fhicl::ParameterSet const &ps)

    Constructs a FragmentGenerator instance, and returns a pointer to it.

Enumerations

- enum ExceptionHandlerRethrow { ExceptionHandlerRethrow::yes, ExceptionHandlerRethrow::no }

    Controls whether the ExceptionHandler will rethrow after printing exception details.

Functions

- bool fragmentSequenceIDCompare (const Fragment &i, const Fragment &j)

    Comparator for Fragment objects, based on their sequence_id.
- std::ostream & operator<< (std::ostream &os, Fragment const &f)

    Prints the given Fragment to the stream.
- std::ostream & operator<< (std::ostream &os, RawEvent const &ev)

    Prints the RawEvent to the given stream.
- std::shared_ptr
< FragmentNameHelper > makeNameHelper (std::string const &plugin_name, std::string const &unidentified_-instance_name, std::vector< std::pair< artdaq::Fragment::type_t, std::string >> extraTypes)

    Create a FragmentNameHelper.
- std::unique_ptr
< FragmentGenerator > makeFragmentGenerator (std::string const &generator_plugin_spec, fhicl::Parameter-Set const &ps)

    Instantiates the FragmentGenerator plugin with the given name, using the given ParameterSet.
- std::string generateMessageFacilityConfiguration (char const *progname, bool useConsole=true, bool print-Debug=false, char const *fileExtraName="")

    Create the MessageFacility configuration Fhicl string.
- void configureTRACE (fhicl::ParameterSet &trace_pset)

    Configure TRACE.
- void configureMessageFacility (char const *progname, bool useConsole=true, bool printDebug=false)

    Configure and start the message facility. Provide the program name so that messages will be appropriately tagged.
- std::string setMsgFacAppName (const std::string &appType, unsigned short port)

    Set the message facility application name using the specified application type and port number.
- void PrintExceptionStackTrace ()

    Print the Exception Stack Trace.
- void ExceptionHandler (ExceptionHandlerRethrow decision, const std::string &optional_message="")

    The ExceptionHandler class prints out all available information about an excection, then optionally re-throws.

5.2.1 Detailed Description

The artdaq namespace.

5.2.2 Typedef Documentation

5.2.2.1 typedef std::unique_ptr<Fragment> artdaq::FragmentPtr

A std::unique_ptr to a Fragment object.

To reduce move or copy operations, most artdaq processing is done using FragmentPtr objects.

Definition at line 54 of file Fragment.hh.

5.2.2.2 typedef std::unique_ptr<artdaq::FragmentGenerator> artdaq::makeFunc_t(fhicl::ParameterSet const &ps)

Constructs a FragmentGenerator instance, and returns a pointer to it.

Parameters

ps	Parameter set for initializing the FragmentGenerator

Returns

A smart pointer to the FragmentGenerator

Definition at line 19 of file GeneratorMacros.hh.

5.2.2.3 typedef detail::RawFragmentHeader::RawDataType artdaq::RawDataType

The RawDataType (currently a 64-bit integer) is the basic unit of data representation within *artdaq*

The RawDataType (currently a 64-bit integer) is the basic unit of data representation within *artdaq* Copied from RawFragmentHeader into Fragment

Definition at line 40 of file Fragment.hh.

5.2.3 Enumeration Type Documentation

5.2.3.1 enum artdaq::ExceptionHandlerRethrow [strong]

Controls whether the ExceptionHandler will rethrow after printing exception details.

Enumerator

yes	Rethrow the exception after sending details to MessageFacility.
no	Consume the exception and proceed.

Definition at line 10 of file ExceptionHandler.hh.

5.2.4 Function Documentation

5.2.4.1 void artdaq::configureMessageFacility (char const * *progname*, bool *useConsole* = `true`, bool *printDebug* = `false`)

Configure and start the message facility. Provide the program name so that messages will be appropriately tagged.

Parameters

progname	The name of the program
useConsole	Should console output be activated? Default = true
printDebug	Whether Debug-level messages should be printed to console. Default = false

Definition at line 260 of file configureMessageFacility.cc.

5.2.4.2 void artdaq::configureTRACE (fhicl::ParameterSet & *trace_pset*)

Configure TRACE.

Parameters

trace_pset	A fhicl::ParameterSet with the contents of the TRACE table

Definition at line 174 of file configureMessageFacility.cc.

5.2.4.3 void artdaq::ExceptionHandler (ExceptionHandlerRethrow *decision,* const std::string & *optional_message* = "")

The ExceptionHandler class prints out all available information about an excection, then optionally re-throws.

Parameters

decision	Controls whether the ExceptionHandler will rethrow (ExceptionHandlerRethrow::yes) or not (ExceptionHandlerRethow::no)
optional_-message	An optional std::string giving more information about where the exception was originally caught

JCF, 5/28/15

The ExceptionHandler() function is designed to be called within a catch-all block:

```
try {
    // ...Code that might throw an exception...
} catch (...) {
       ExceptionHandler(artdaq::ExceptionHandlerRethrow::yes
      ,
       "Optional string providing additional info");
}
```

Where above, you could switch out `artdaq::ExceptionHandlerRethrow::yes` with `artdaq::-ExceptionHandlerRethrow::no`, depending on what you wish to do

The details of ExceptionHandler() are as follows:

- If an optional string is passed to it, use messagefacility to write the string with mf::LogError()
- Apply a set of different catch-blocks to the original exception, printing out as much information as possible contained within the different exception types (art::Exception, cet::exception, boost::exception and std::exception), again using mf::LogError()
- If artdaq::ExceptionHandlerRethrow::yes was passed to ExceptionHandler(), re-throw the exception rather than swallow it

Definition at line 40 of file ExceptionHandler.cc.

5.2.4.4 bool artdaq::fragmentSequenceIDCompare (const Fragment & *i,* const Fragment & *j*)

Comparator for Fragment objects, based on their sequence_id.

Parameters

i	First Fragment to compare
j	Second Fragment to comapre

Returns

i.sequenceID() < j.sequenceID()

Definition at line 8 of file Fragment.cc.

5.2.4.5 std::string artdaq::generateMessageFacilityConfiguration (char const ∗ *progname*, bool *useConsole* = `true`, bool *printDebug* = `false`, char const ∗ *fileExtraName* = "")

Create the MessageFacility configuration Fhicl string.

Parameters

progname	The name of the program
useConsole	Should console output be activated? Default = true
printDebug	Whether Debug-level messages should be printed to console. Default = false
fileExtraName	Additonal name to be printed after progname in file names within output directory for progname (e.g. "-art")

Returns

Fhicl string with generated MessageFacility configuration

Exceptions

cet::exception	if log path or ARTDAQ_LOG_FHICL do not exist

Definition at line 29 of file configureMessageFacility.cc.

5.2.4.6 std::unique_ptr< artdaq::FragmentGenerator > artdaq::makeFragmentGenerator (std::string const & *generator_plugin_spec*, fhicl::ParameterSet const & *ps*)

Instantiates the FragmentGenerator plugin with the given name, using the given ParameterSet.

Parameters

generator_plugin-_spec	Name of the Generator plugin (omit _generator.so)
ps	The ParameterSet used to initialize the FragmentGenerator

Returns

A smart pointer to the FragmentGenerator instance

Definition at line 7 of file makeFragmentGenerator.cc.

5.2.4.7 std::shared_ptr<FragmentNameHelper> artdaq::makeNameHelper (std::string const & *plugin_name*, std::string const & *unidentified_instance_name*, std::vector< std::pair< artdaq::Fragment::type_t, std::string >> *extraTypes*) `[inline]`

Create a FragmentNameHelper.

Parameters

plugin_name	Name of the FragmentNameHelper plugin to load
unidentified_-instance_name	String to use for when the FragmentNameHelper cannot determine the Fragment name
extraTypes	Additional types to register with the FragmentNameHelper

Returns

FragmentNameHelper shared_ptr handle

Definition at line 183 of file FragmentNameHelper.hh.

5.2.4.8 std::ostream & artdaq::operator<< (std::ostream & *os*, artdaq::Fragment const & *f*) [inline]

Prints the given Fragment to the stream.

Parameters

os	Stream to print Fragment to
f	Fragment to print

Returns

Reference to the stream

Definition at line 1257 of file Fragment.hh.

5.2.4.9 std::ostream& artdaq::operator<< (std::ostream & *os*, RawEvent const & *ev*) [inline]

Prints the RawEvent to the given stream.

Parameters

os	Stream to print RawEvent to
ev	RawEvent to print

Returns

Stream reference

Definition at line 334 of file RawEvent.hh.

5.2.4.10 std::string artdaq::setMsgFacAppName (const std::string & *appType*, unsigned short *port*)

Set the message facility application name using the specified application type and port number.

Parameters

appType	Application name

port	XMLRPC port of this application instance

Returns

Name of the application as set for MessageFacility

Definition at line 290 of file configureMessageFacility.cc.

5.3 artdaq::detail Namespace Reference

artdaq implementation details namespace

Classes

- struct RawFragmentHeader

  The RawFragmentHeader class contains the basic fields used by artdaq for routing Fragment objects through the system.
- struct RawFragmentHeaderV0

  The RawFragmentHeaderV0 class contains the basic fields used by artdaq for routing Fragment objects through the system.
- struct RawFragmentHeaderV1

  The RawFragmentHeaderV1 class contains the basic fields used by artdaq for routing Fragment objects through the system.
- struct RawEventHeader

  The header information used to identify key properties of the RawEvent object.

Functions

- std::ostream & operator<< (std::ostream &os, RawEventHeader const &evh)

  Prints the RawEventHeader to the given stream.

5.3.1 Detailed Description

artdaq implementation details namespace

5.3.2 Function Documentation

5.3.2.1 std::ostream& artdaq::detail::operator<< (std::ostream & *os*, RawEventHeader const & *evh*) `[inline]`

Prints the RawEventHeader to the given stream.

Parameters

os	Stream to print RawEventHeader to
evh	RawEventHeader to print

Returns

Stream reference

Definition at line 85 of file RawEvent.hh.

5.4 artdaq::TimeUtils Namespace Reference

Namespace to hold useful time-converting functions.

Typedefs

- typedef std::chrono::duration
  $<$ double, std::ratio$<$ 1 $>$ $>$ seconds

Functions

- double GetElapsedTime (std::chrono::steady_clock::time_point then, std::chrono::steady_clock::time_point now=std::chrono::steady_clock::now())

  Get the number of seconds in the given interval
- size_t GetElapsedTimeMicroseconds (std::chrono::steady_clock::time_point then, std::chrono::steady_clock-::time_point now=std::chrono::steady_clock::now())

  Gets the number of microseconds in the given time interval
- size_t GetElapsedTimeMilliseconds (std::chrono::steady_clock::time_point then, std::chrono::steady_clock::time-_point now=std::chrono::steady_clock::now())

  Gets the number of milliseconds in the given time interval
- struct timespec get_realtime_clock ()

  Get the current time of day as a pair of seconds and nanoseconds (from clock_gettime(CLOCK_REALTIME, ...) system call)
- constexpr double GetElapsedTime (struct timespec const &then, struct timespec now=get_realtime_clock())

  Get the elapsed time between two struct timespec instances.
- std::string convertUnixTimeToString (time_t inputUnixTime)

  Converts a Unix time to its string representation, in UTC.
- std::string convertUnixTimeToString (struct timeval const &inputUnixTime)

  Converts a Unix time to its string representation, in UTC.
- std::string convertUnixTimeToString (struct timespec const &inputUnixTime)

  Converts a Unix time to its string representation, in UTC.
- uint64_t gettimeofday_us ()

  Get the current time of day in microseconds (from gettimeofday system call)
- double convertUnixTimeToSeconds (time_t inputUnixTime)

  Converts a Unix time to double.
- double convertUnixTimeToSeconds (struct timeval const &inputUnixTime)

  Converts a Unix time to double.
- double convertUnixTimeToSeconds (struct timespec const &inputUnixTime)

  Converts a Unix time to double.

5.4.1 Detailed Description

Namespace to hold useful time-converting functions.

5.4.2 Typedef Documentation

5.4.2.1 typedef std::chrono::duration<double, std::ratio<1> > artdaq::TimeUtils::seconds

We shall use artdaq::detail::seconds as our "standard" duration type. Note that this differs from std::chrono::seconds, which has a representation in some integer type of at least 35 bits.

daqrate::duration dur(1.0) represents a duration of 1 second. daqrate::duration dur2(0.001) represents a duration of 1 millisecond.

Definition at line 22 of file TimeUtils.hh.

5.4.3 Function Documentation

5.4.3.1 double artdaq::TimeUtils::convertUnixTimeToSeconds (time_t *inputUnixTime*)

Converts a Unix time to double.

Parameters

inputUnixTime	A time_t Unix time variable

Returns

double representation of Unix time (seconds since epoch)

Definition at line 66 of file TimeUtils.cc.

5.4.3.2 double artdaq::TimeUtils::convertUnixTimeToSeconds (struct timeval const & *inputUnixTime*)

Converts a Unix time to double.

Parameters

inputUnixTime	A struct timeval Unix time variable

Returns

double representation of Unix time (in seconds)

Definition at line 72 of file TimeUtils.cc.

5.4.3.3 double artdaq::TimeUtils::convertUnixTimeToSeconds (struct timespec const & *inputUnixTime*)

Converts a Unix time to double.

Parameters

inputUnixTime	A struct timespec Unix time variable

Returns

double representation of Unix time (in seconds)

Definition at line 78 of file TimeUtils.cc.

5.4.3.4 std::string artdaq::TimeUtils::convertUnixTimeToString (time_t *inputUnixTime*)

Converts a Unix time to its string representation, in UTC.

Parameters

inputUnixTime	A time_t Unix time variable

Returns

std::string representation of Unix time, in UTC

Definition at line 7 of file TimeUtils.cc.

5.4.3.5 std::string artdaq::TimeUtils::convertUnixTimeToString (struct timeval const & *inputUnixTime*)

Converts a Unix time to its string representation, in UTC.

Parameters

inputUnixTime	A struct timeval Unix time variable

Returns

std::string representation of Unix time, in UTC

Definition at line 18 of file TimeUtils.cc.

5.4.3.6 std::string artdaq::TimeUtils::convertUnixTimeToString (struct timespec const & *inputUnixTime*)

Converts a Unix time to its string representation, in UTC.

Parameters

inputUnixTime	A struct timespec Unix time variable

Returns

std::string representation of Unix time, in UTC

Definition at line 35 of file TimeUtils.cc.

5.4.3.7 struct timespec artdaq::TimeUtils::get_realtime_clock ( )

Get the current time of day as a pair of seconds and nanoseconds (from clock_gettime(CLOCK_REALTIME, ...) system call)

Returns

Pair of seconds, nanoseconds wallclock time

Definition at line 58 of file TimeUtils.cc.

5.4.3.8 double artdaq::TimeUtils::GetElapsedTime (std::chrono::steady_clock::time_point *then*, std::chrono::steady_clock::time_point *now* = `std::chrono::steady_clock::now()`) `[inline]`

Get the number of seconds in the given interval

Parameters

then	std::chrono::steady_clock::time_point representing start of interval
now	std::chrono::steady_clock::time_point representing end of interval. Defaults to std::chrono-::steady_clock::now()

Returns

Seconds in time interval, expressed as double

Definition at line 30 of file TimeUtils.hh.

5.4.3.9 constexpr double artdaq::TimeUtils::GetElapsedTime (struct timespec const & *then,* struct timespec *now =* `get_realtime_clock()`) `[inline]`

Get the elapsed time between two struct timespec instances.

Note that struct timespec instances from get_realtime_clock are subject to clock adjustments and should not be relied on as precision timers!

Parameters

then	Timespec representing beginning of interval
now	Timespec representing end of interval. Defaults to get_realtime_clock()

Returns

Elapseed time between then and now as double, in seconds

Definition at line 71 of file TimeUtils.hh.

5.4.3.10 size_t artdaq::TimeUtils::GetElapsedTimeMicroseconds (std::chrono::steady_clock::time_point *then,* std::chrono::steady_clock::time_point *now =* `std::chrono::steady_clock::now()`) `[inline]`

Gets the number of microseconds in the given time interval

Parameters

then	std::chrono::steady_clock::time_point representing start of interval
now	std::chrono::steady_clock::time_point representing end of interval. Defaults to std::chrono-::steady_clock::now()

Returns

Microseconds in time interval

Definition at line 41 of file TimeUtils.hh.

5.4.3.11 size_t artdaq::TimeUtils::GetElapsedTimeMilliseconds (std::chrono::steady_clock::time_point *then,* std::chrono::steady_clock::time_point *now =* `std::chrono::steady_clock::now()`) `[inline]`

Gets the number of milliseconds in the given time interval

Parameters

then	std::chrono::steady_clock::time_point representing start of interval
now	std::chrono::steady_clock::time_point representing end of interval. Defaults to std::chrono::steady_clock::now()

Returns

Milliseconds in time interval

Definition at line 52 of file TimeUtils.hh.

5.4.3.12 uint64_t artdaq::TimeUtils::gettimeofday_us ( )

Get the current time of day in microseconds (from gettimeofday system call)

Returns

The current time of day in microseconds

Definition at line 51 of file TimeUtils.cc.

5.5 artdaqcore Namespace Reference

Namespace used to differentiate the artdaq_core version of GetPackageBuildInfo from other versions present in the system.

Classes

- struct GetPackageBuildInfo

  Wrapper around the artdaqcore::GetPackageBuildInfo::getPackageBuildInfo function.

5.5.1 Detailed Description

Namespace used to differentiate the artdaq_core version of GetPackageBuildInfo from other versions present in the system.

Chapter 6

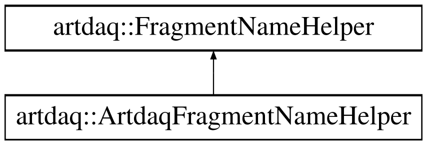
Class Documentation

6.1 artdaq::ArtdaqFragmentNameHelper Class Reference

Default implementation of FragmentNameHelper.

`#include <artdaq-core/Plugins/ArtdaqFragmentNameHelper.hh>`

Inheritance diagram for artdaq::ArtdaqFragmentNameHelper:



Public Member Functions

- virtual ~ArtdaqFragmentNameHelper ()

    DefaultArtdaqFragmentNameHelper Destructor.

- ArtdaqFragmentNameHelper (std::string unidentified_instance_name, std::vector< std::pair< artdaq::Fragment::type_t, std::string >> extraTypes)

    ArtdaqFragmentNameHelper Constructor.

Additional Inherited Members

6.1.1 Detailed Description

Default implementation of FragmentNameHelper.

Definition at line 15 of file ArtdaqFragmentNameHelper.hh.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 artdaq::ArtdaqFragmentNameHelper::ArtdaqFragmentNameHelper (std::string *unidentified_instance_name*, std::vector< std::pair< **artdaq::Fragment::type_t**, std::string >> *extraTypes*)

ArtdaqFragmentNameHelper Constructor.

Parameters

unidentified_-instance_name	Name to use for unidentified Fragments
extraTypes	Additional types to register

Definition at line 6 of file Artdaq_fragmentNameHelper.cc.

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

- artdaq_core/artdaq-core/Plugins/ArtdaqFragmentNameHelper.hh
- artdaq_core/artdaq-core/Plugins/Artdaq_fragmentNameHelper.cc

6.2 mfplugins::ELGenFileOutput::Config Struct Reference

Parameters used to configure GenFileOutput.

Public Attributes

- fhicl::TableFragment
  < ELdestination::Config > elDestConfig

  ELDestination common configuration parameters.
- fhicl::Atom< bool > append

  "append" (Default: true"): Whether to append to the file or recreate it
- fhicl::Atom< std::string > baseDir

  "directory" (Default: "/tmp"): The directory into which files will be saved
- fhicl::Atom< std::string > sep

  "seperator" (Default: "-"): Separator to use after optional replacement parameters
- fhicl::Atom< std::string > timePattern

  "timestamp_pattern" (Default: "%Y%m%d%H%M%S"): Pattern to use for t strftime replacement
- fhicl::Atom< std::string > filePattern

  "pattern" (Default: "%N-%?H%t-%p.log"): Pattern to use for file naming.

6.2.1 Detailed Description

Parameters used to configure GenFileOutput.

Definition at line 30 of file GenFile_mfPlugin.cc.

6.2.2 Member Data Documentation

6.2.2.1 fhicl::Atom<bool> mfplugins::ELGenFileOutput::Config::append

Initial value:

```
= fhicl::Atom<bool>{
            fhicl::Name{"append"}, fhicl::Comment{"Whether to append to the file or recreate it"}, true}
```

"append" (Default: true"): Whether to append to the file or recreate it

Definition at line 35 of file GenFile_mfPlugin.cc.

6.2.2.2 fhicl::Atom<std::string> mfplugins::ELGenFileOutput::Config::baseDir

Initial value:

```
= fhicl::Atom<std::string>{
            fhicl::Name{"directory"}, fhicl::Comment{"The directory into which files will be saved"}, "/tmp
       "}
```

"directory" (Default: "/tmp"): The directory into which files will be saved

Definition at line 38 of file GenFile_mfPlugin.cc.

6.2.2.3 fhicl::Atom<std::string> mfplugins::ELGenFileOutput::Config::filePattern

Initial value:

```
= fhicl::Atom<std::string>{fhicl::Name{"pattern"}, fhicl::Comment{"Pattern to use for file naming.\n"

           " Supported parameters are:\n"

           " %%: Print a % sign\n"

           " %N: Print the executable name, as retrieved from /proc/<pid>/exe\n"

           " %?N: Print the executable name only if it does not already appear in the parsed format. "

           "Format is parsed left-to-right.\n"

           " These options add a seperator AFTER if they are filled and if they are not the last token in "

           "the file pattern, before the last '.' character.\n"

           " %H: Print the hostname, without any domain specifiers (i.e. work.fnal.gov will become work)\n"

           " %?H: Print the hostname only if it does not already appear in the parsed format.\n"

           " %p: Print the PID of the application configuring MessageFacility\n"

           " %t: Print the timestamp using the format specified by timestamp_pattern\n"

           " %T: Print the timestamp in ISO format"},
                                                                   "%N-%?H%t-%p.log"}
```

"pattern" (Default: "%N-%?H%t-%p.log"): Pattern to use for file naming.

" Supported parameters are:\n" %%: Print a % sign N: Print the executable name, as retrieved from /proc/$pid/exe %?N: Print the executable name only if it does not already appear in the parsed format. Format is parsed left-to-right. These options add a seperator AFTER if they are filled and if they are not the last token in the file pattern, before the last '.' character. H: Print the hostname, without any domain specifiers (i.e. work.fnal.gov will become work) %?H: Print the hostname only if it does not already appear in the parsed format. p: Print the PID of the application configuring MessageFacility t: Print the timestamp using the format specified by timestamp_pattern T: Print the timestamp in ISO format

Definition at line 60 of file GenFile_mfPlugin.cc.

6.2.2.4 fhicl::Atom<std::string> mfplugins::ELGenFileOutput::Config::sep

Initial value:

```
= fhicl::Atom<std::string>{
            fhicl::Name{"seperator"}, fhicl::Comment{"Separator to use after optional replacement
       parameters"}, "-"}
```

"seperator" (Default: "-"): Separator to use after optional replacement parameters

Definition at line 41 of file GenFile_mfPlugin.cc.

6.2.2.5 fhicl::Atom<std::string> mfplugins::ELGenFileOutput::Config::timePattern

Initial value:

```
= fhicl::Atom<std::string>{
            fhicl::Name{"timestamp_pattern"}, fhicl::Comment{"Pattern to use for %t strftime replacement"},
        "%Y%m%d%H%M%S"}
```

"timestamp_pattern" (Default: "%Y%m%d%H%M%S"): Pattern to use for t strftime replacement

Definition at line 44 of file GenFile_mfPlugin.cc.

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

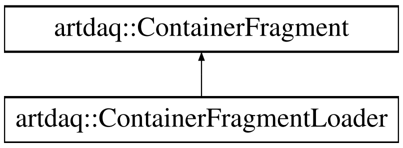
- artdaq_core/artdaq-core/Utilities/GenFile_mfPlugin.cc

6.3 artdaq::ContainerFragment Class Reference

The artdaq::ContainerFragment class represents a Fragment which contains other Fragments.

`#include <artdaq-core/Data/ContainerFragment.hh>`

Inheritance diagram for artdaq::ContainerFragment:



Classes

- struct Metadata

    Contains the information necessary for retrieving Fragment objects from the ContainerFragment.
- struct MetadataV0

    Contains the information necessary for retrieving Fragment objects from the ContainerFragment.

Public Member Functions

- Metadata const ∗ UpgradeMetadata (MetadataV0 const ∗in) const

    Upgrade the Metadata of a fixed-size ContainerFragment to the new standard.
- ContainerFragment (Fragment const &f)
- Metadata const ∗ metadata () const

    const getter function for the Metadata
- Metadata::count_t block_count () const

    Gets the number of fragments stored in the ContainerFragment.
- Fragment::type_t fragment_type () const

    Get the Fragment::type_t of stored Fragment objects.
- bool missing_data () const

    Gets the flag if the ContainerFragment knows that it is missing data.

- void const ∗ dataBegin () const

  Gets the start of the data.
- void const ∗ dataEnd () const

  Gets the last Fragment in the ContainerFragment.
- FragmentPtr at (size_t index) const

  Gets a specific Fragment from the ContainerFragment.
- size_t fragSize (size_t index) const

  Gets the size of the Fragment at the specified location in the ContainerFragment, in bytes.
- FragmentPtr operator[] (size_t index) const

  Alias to ContainerFragment::at()
- size_t fragmentIndex (size_t index) const

  Get the offset of a Fragment within the ContainerFragment.
- size_t lastFragmentIndex () const

  Returns the offset of the last Fragment in the ContainerFragment.

Static Public Attributes

- static constexpr uint8_t CURRENT_VERSION = 1

  The current version of the ContainerFragmentHeader.
- static constexpr size_t CONTAINER_MAGIC = 0x00BADDEED5B1BEE5

  Marker word used in index.

Protected Member Functions

- const size_t ∗ create_index_ () const

  Create an index for the currently-contained Fragments.
- void reset_index_ptr_ () const

  Reset the index pointer to a newly-created index.
- const size_t ∗ get_index_ () const

  Get a pointer to the index.

Static Protected Member Functions

- static constexpr size_t words_per_frag_word_ ()

  Gets the ratio between the fundamental data storage type and the representation within the Fragment.

6.3.1 Detailed Description

The artdaq::ContainerFragment class represents a Fragment which contains other Fragments.

Definition at line 20 of file ContainerFragment.hh.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 artdaq::ContainerFragment::ContainerFragment (**Fragment const & *f***) `[inline]`, `[explicit]`

Parameters

f	The Fragment object to use for data storage

The constructor simply sets its const private member "artdaq_Fragment_" to refer to the artdaq::Fragment object

Definition at line 103 of file ContainerFragment.hh.

6.3.3 Member Function Documentation

6.3.3.1 FragmentPtr artdaq::ContainerFragment::at (size_t *index*) const [inline]

Gets a specific Fragment from the ContainerFragment.

Parameters

index	The Fragment index to return

Returns

Pointer to the specified Fragment in the ContainerFragment

Exceptions

cet::exception	if the index is out-of-range

Definition at line 166 of file ContainerFragment.hh.

6.3.3.2 Metadata::count_t artdaq::ContainerFragment::block_count ( ) const [inline]

Gets the number of fragments stored in the ContainerFragment.

Returns

The number of Fragment objects stored in the ContainerFragment

Definition at line 130 of file ContainerFragment.hh.

6.3.3.3 const size_t∗ artdaq::ContainerFragment::create_index_ ( ) const [inline], [protected]

Create an index for the currently-contained Fragments.

Returns

Array of block_count size_t words containing index

Definition at line 259 of file ContainerFragment.hh.

6.3.3.4 void const∗ artdaq::ContainerFragment::dataBegin ( ) const [inline]

Gets the start of the data.

Returns

Pointer to the first Fragment in the ContainerFragment

Definition at line 146 of file ContainerFragment.hh.

6.3.3.5 void const∗ artdaq::ContainerFragment::dataEnd ( ) const [inline]

Gets the last Fragment in the ContainerFragment.

Returns

Pointer to the last Fragment in the ContainerFragment

Definition at line 155 of file ContainerFragment.hh.

6.3.3.6 Fragment::type_t artdaq::ContainerFragment::fragment_type ( ) const [inline]

Get the Fragment::type_t of stored Fragment objects.

Returns

The Fragment::type_t of stored Fragment objects

Definition at line 135 of file ContainerFragment.hh.

6.3.3.7 size_t artdaq::ContainerFragment::fragmentIndex (size_t *index*) const [inline]

Get the offset of a Fragment within the ContainerFragment.

Parameters

index	The Fragment index

Returns

The offset of the requested Fragment within the ContainerFragment

Exceptions

cet::exception	if the index is out-of-range

Definition at line 223 of file ContainerFragment.hh.

6.3.3.8 size_t artdaq::ContainerFragment::fragSize (size_t *index*) const [inline]

Gets the size of the Fragment at the specified location in the ContainerFragment, in bytes.

Parameters

index	The Fragment index

Returns

The size of the Fragment at the specified location in the ContainerFragment, in bytes

Exceptions

cet::exception	if the index is out-of-range

Definition at line 195 of file ContainerFragment.hh.

6.3.3.9 const size_t* artdaq::ContainerFragment::get_index_ ( ) const [inline], [protected]

Get a pointer to the index.

Returns

pointer to size_t array of Fragment offsets in payload, terminating with CONTAINER_MAGIC

Definition at line 300 of file ContainerFragment.hh.

6.3.3.10 size_t artdaq::ContainerFragment::lastFragmentIndex ( ) const [inline]

Returns the offset of the last Fragment in the ContainerFragment.

Returns

The offset of the last Fragment in the ContainerFragment

Definition at line 240 of file ContainerFragment.hh.

6.3.3.11 Metadata const* artdaq::ContainerFragment::metadata ( ) const [inline]

const getter function for the Metadata

Returns

const pointer to the Metadata

Definition at line 114 of file ContainerFragment.hh.

6.3.3.12 bool artdaq::ContainerFragment::missing_data ( ) const [inline]

Gets the flag if the ContainerFragment knows that it is missing data.

Returns

The flag if the ContainerFragment knows that it is missing data

Definition at line 140 of file ContainerFragment.hh.

6.3.3.13 FragmentPtr artdaq::ContainerFragment::operator[] (size_t *index*) const [inline]

Alias to ContainerFragment::at()

Parameters

index	The Fragment index to return

Returns

Pointer to the specified Fragment in the ContainerFragment

Exceptions

cet::exception	if the index is out-of-range

Definition at line 212 of file ContainerFragment.hh.

6.3.3.14 Metadata const∗ artdaq::ContainerFragment::UpgradeMetadata (MetadataV0 const ∗ *in*) const [inline]

Upgrade the Metadata of a fixed-size ContainerFragment to the new standard.

Parameters

in	Metadata to upgrade

Returns

Upgraded Metadata

Definition at line 82 of file ContainerFragment.hh.

6.3.3.15 static constexpr size_t artdaq::ContainerFragment::words_per_frag_word_ ( ) [inline], [static], [protected]

Gets the ratio between the fundamental data storage type and the representation within the Fragment.

Returns

The ratio between the fundamental data storage type and the representation within the Fragment

Definition at line 250 of file ContainerFragment.hh.

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

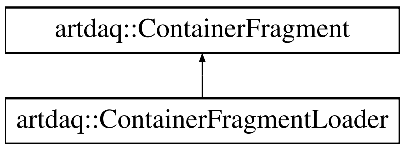
- artdaq_core/artdaq-core/Data/ContainerFragment.hh

6.4 artdaq::ContainerFragmentLoader Class Reference

A Read-Write version of the ContainerFragment, used for filling ContainerFragment objects with other Fragment objects.

`#include <artdaq-core/Data/ContainerFragmentLoader.hh>`

Inheritance diagram for artdaq::ContainerFragmentLoader:



Public Member Functions

- ContainerFragmentLoader (Fragment &f, Fragment::type_t expectedFragmentType)

  Constructs the ContainerFragmentLoader.
- Metadata ∗ metadata ()

  Get the ContainerFragment metadata (includes information about the location of Fragment objects within the Container-Fragment)
- void set_fragment_type (Fragment::type_t type)

  Sets the type of Fragment which this ContainerFragment should contain.
- void set_missing_data (bool isDataMissing)

  Sets the missing_data flag.
- void addFragment (artdaq::Fragment &frag, bool allowDifferentTypes=false)

  Add a Fragment to the ContainerFragment by reference.
- void addFragment (artdaq::FragmentPtr &frag, bool allowDifferentTypes=false)

  Add a Fragment to the ContainerFragment by smart pointer.
- void addFragments (artdaq::Fragments &frags, bool allowDifferentTypes=false)

  Add a collection of Fragment objects to the ContainerFragment.
- void addFragments (artdaq::FragmentPtrs &frags, bool allowDifferentTypes=false)

  Add a collection of Fragment objects to the ContainerFragment.
- detail::RawFragmentHeader ∗ appendFragment (size_t nwords)

  Create a Fragment at the end of the ContainerFragment with the given size nwords Size (in RawDataType words) of the new Fragment.
- void resizeLastFragment (size_t nwords)

  Resize the last Fragment in the container.
- detail::RawFragmentHeader ∗ **lastFragmentHeader** ()

Additional Inherited Members

6.4.1 Detailed Description

A Read-Write version of the ContainerFragment, used for filling ContainerFragment objects with other Fragment objects.

Definition at line 27 of file ContainerFragmentLoader.hh.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 artdaq::ContainerFragmentLoader::ContainerFragmentLoader (**artdaq::Fragment &** *f*, **Fragment::type_t** *expectedFragmentType* = **Fragment::EmptyFragmentType**) `[inline]`, `[explicit]`

Constructs the ContainerFragmentLoader.

Parameters

f	A Fragment object containing a Fragment header.
expected-FragmentType	The type of fragment which will be put into this ContainerFragment

Exceptions

cet::exception	if the Fragment input has inconsistent Header information

Definition at line 123 of file ContainerFragmentLoader.hh.

6.4.3 Member Function Documentation

6.4.3.1 void artdaq::ContainerFragmentLoader::addFragment (**artdaq::Fragment** & *frag*, bool *allowDifferentTypes* = `false`) `[inline]`

Add a Fragment to the ContainerFragment by reference.

Parameters

frag	A Fragment object to be added to the ContainerFragment

Exceptions

cet::exception	If the Fragment to be added has a different type than expected

Definition at line 166 of file ContainerFragmentLoader.hh.

6.4.3.2 void artdaq::ContainerFragmentLoader::addFragment (**artdaq::FragmentPtr** & *frag*, bool *allowDifferentTypes* = `false`) `[inline]`

Add a Fragment to the ContainerFragment by smart pointer.

Parameters

frag	A FragmentPtr to a Fragment to be added to the ContainerFragment

Definition at line 296 of file ContainerFragmentLoader.hh.

6.4.3.3 void artdaq::ContainerFragmentLoader::addFragments (**artdaq::Fragments** & *frags*, bool *allowDifferentTypes* = `false`) `[inline]`

Add a collection of Fragment objects to the ContainerFragment.

Parameters

frags	An artdaq::Fragments object containing Fragments to be added to the ContainerFragment

Definition at line 198 of file ContainerFragmentLoader.hh.

6.4.3.4 void artdaq::ContainerFragmentLoader::addFragments (**artdaq::FragmentPtrs** & *frags*, bool *allowDifferentTypes* = `false`) `[inline]`

Add a collection of Fragment objects to the ContainerFragment.

Parameters

frags	An artdaq::FragmentPtrs object containing Fragments to be added to the ContainerFragment

Definition at line 301 of file ContainerFragmentLoader.hh.

6.4.3.5 artdaq::detail::RawFragmentHeader ∗ artdaq::ContainerFragmentLoader::appendFragment (size_t *nwords*) [inline]

Create a Fragment at the end of the ContainerFragment with the given size nwords Size (in RawDataType words) of the new Fragment.

Returns

Pointer to created Fragment's header (payload starts at ptr + 1)

Definition at line 239 of file ContainerFragmentLoader.hh.

6.4.3.6 Metadata∗ artdaq::ContainerFragmentLoader::metadata ( ) [inline]

Get the ContainerFragment metadata (includes information about the location of Fragment objects within the Container-Fragment)

Returns

The ContainerFragment metadata

Definition at line 43 of file ContainerFragmentLoader.hh.

6.4.3.7 void artdaq::ContainerFragmentLoader::resizeLastFragment (size_t *nwords*) [inline]

Resize the last Fragment in the container.

Parameters

nwords	New size (in RawDataType words) of last Fragment in container

Definition at line 275 of file ContainerFragmentLoader.hh.

6.4.3.8 void artdaq::ContainerFragmentLoader::set_fragment_type (Fragment::type_t *type*) [inline]

Sets the type of Fragment which this ContainerFragment should contain.

Parameters

type	The Fragment::type_t identifying the type of Fragment objects stored in this ContainerFragment

Definition at line 53 of file ContainerFragmentLoader.hh.

6.4.3.9 void artdaq::ContainerFragmentLoader::set_missing_data (bool *isDataMissing*) [inline]

Sets the missing_data flag.

Parameters

isDataMissing	The value of the missing_data flag

The ContainerFragment::Metadata::missing_data flag is used for FragmentGenerators to indicate that the fragment is incomplete, but the generator does not have the correct data to fill it. This happens for Window-mode FragmentGenerators when the window requested is before the start of the FragmentGenerator's buffers, for example.

Definition at line 66 of file ContainerFragmentLoader.hh.

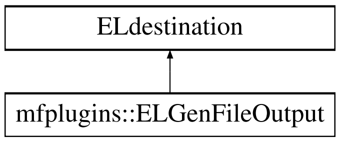
The documentation for this class was generated from the following file:

- artdaq_core/artdaq-core/Data/ContainerFragmentLoader.hh

6.5 mfplugins::ELGenFileOutput Class Reference

Message Facility destination which generates the output file name based on some combination of PID, hostname, application name, and/or timestamp

Inheritance diagram for mfplugins::ELGenFileOutput:



Classes

- struct Config

  Parameters used to configure GenFileOutput.

Public Types

- using Parameters = fhicl::WrappedTable< Config >

  Used for ParameterSet validation.

Public Member Functions

- ELGenFileOutput (Parameters const &pset)

  ELGenFileOutput Constructor.
- void routePayload (const std::ostringstream &o, const ErrorObj &e) override

  Serialize a MessageFacility message to the output stream.
- void flush () override

  Flush any text in the ostream buffer to disk.

6.5.1 Detailed Description

Message Facility destination which generates the output file name based on some combination of PID, hostname, application name, and/or timestamp

Definition at line 24 of file GenFile_mfPlugin.cc.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 mfplugins::ELGenFileOutput::ELGenFileOutput (**Parameters** const & *pset*) [explicit]

ELGenFileOutput Constructor.

Parameters

pset	Validated ParameterSet used to configure GenFileOutput

Definition at line 109 of file GenFile_mfPlugin.cc.

6.5.3 Member Function Documentation

6.5.3.1 void mfplugins::ELGenFileOutput::routePayload (const std::ostringstream & *o*, const ErrorObj & *e*) [override]

Serialize a MessageFacility message to the output stream.

Parameters

o	Stringstream object containing message data
e	MessageFacility object containing header information

Definition at line 257 of file GenFile_mfPlugin.cc.

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

- artdaq_core/artdaq-core/Utilities/GenFile_mfPlugin.cc

6.6 artdaq::Fragment Class Reference

A Fragment contains the data from one piece of the DAQ system for one event The artdaq::Fragment is the main data storage class in artdaq. Each Fragment represents the data from one piece of the readout, for one artdaq event. BoardReaders create Fragments and send them to the EventBuilders, where they are assembled into artdaq::RawEvent objects.

`#include <artdaq-core/Data/Fragment.hh>`

Public Types

- typedef uint8_t byte_t

    For byte representation.

- typedef
  detail::RawFragmentHeader::version_t version_t

    typedef for version_t from RawFragmentHeader

- typedef
  detail::RawFragmentHeader::type_t type_t

  typedef for type_t from RawFragmentHeader
- typedef
  detail::RawFragmentHeader::sequence_id_t sequence_id_t

  typedef for sequence_id_t from RawFragmentHeader
- typedef
  detail::RawFragmentHeader::fragment_id_t fragment_id_t

  typedef for fragment_id_t from RawFragmentHeader
- typedef
  detail::RawFragmentHeader::timestamp_t timestamp_t

  typedef for timestamp_t from RawFragmentHeader
- typedef QuickVec< RawDataType >
  ::reference reference

  Alias reference type from QuickVec<RawDataType>
- typedef QuickVec< RawDataType >
  ::iterator iterator

  Alias iterator type from QuickVec<RawDataType>
- typedef QuickVec< RawDataType >
  ::const_iterator const_iterator

  Alias const_iterator type from QuickVec<RawDataType>
- typedef QuickVec< RawDataType >
  ::value_type value_type

  Alias value_type type from QuickVec<RawDataType>
- typedef QuickVec< RawDataType >
  ::difference_type difference_type

  Alias difference_type type from QuickVec<RawDataType>
- typedef QuickVec< RawDataType >
  ::size_type size_type

  Alias size_type type from QuickVec<RawDataType>

Public Member Functions

- Fragment ()

  Create a Fragment with all header values zeroed.
- Fragment (const Fragment &)

  Default copy constructor.
- Fragment (Fragment &&) noexcept

  Move Constructor.
- Fragment & operator= (const Fragment &)

  Default copy-assignment operator.
- Fragment & operator= (Fragment &&) noexcept

  Move-assignment operator.
- Fragment (std::size_t n)

  Create a Fragment ready to hold n words (RawDataTypes) of payload, and with all values zeroed.
- template<class T >
  Fragment (std::size_t payload_size, sequence_id_t sequence_id, fragment_id_t fragment_id, type_t type, const T &metadata, timestamp_t timestamp=Fragment::InvalidTimestamp)

Create a Fragment with the given header values.

- Fragment (sequence_id_t sequenceID, fragment_id_t fragID, type_t type=Fragment::DataFragmentType, timestamp_t timestamp=Fragment::InvalidTimestamp)

  Create a fragment with the given event id and fragment id, and with no data payload.
- void print (std::ostream &os) const

  Print out summary information for this Fragment to the given stream.
- std::size_t size () const

  Gets the size of the Fragment, from the Fragment header.
- version_t version () const

  Version of the Fragment, from the Fragment header.
- type_t type () const

  Type of the Fragment, from the Fragment header.
- std::string typeString () const

  Print the type of the Fragment.
- sequence_id_t sequenceID () const

  Sequence ID of the Fragment, from the Fragment header.
- fragment_id_t fragmentID () const

  Fragment ID of the Fragment, from the Fragment header.
- timestamp_t timestamp () const

  Timestamp of the Fragment, from the Fragment header.
- void setUserType (type_t utype)

  Sets the type of the Fragment, checking that it is a valid user type.
- void setSystemType (type_t stype)

  Sets the type of the Fragment, checking that it is a valid system type.
- void setSequenceID (sequence_id_t sequence_id)

  Sets the Sequence ID of the Fragment.
- void setFragmentID (fragment_id_t fragment_id)

  Sets the Fragment ID of the Fragment.
- void setTimestamp (timestamp_t timestamp)

  Sets the Timestamp of the Fragment.
- void touch ()

  Update the access time of the Fragment.
- struct timespec atime () const

  Get the last access time of the Fragment.
- struct timespec getLatency (bool touch)

  Get the difference between the current time and the last access time of the Fragment.
- std::size_t sizeBytes () const

  Size of vals_ vector (header + (optional) metadata + payload) in bytes.
- std::size_t dataSize () const

  Return the number of RawDataType words in the data payload. This does not include the number of words in the header or the metadata.
- std::size_t dataSizeBytes () const

  Return the number of bytes in the data payload. This does not include the number of bytes in the header or the metadata.
- bool hasMetadata () const

  Test whether this Fragment has metadata.
- template<class T >
  T ∗ metadata ()

Return a pointer to the metadata. This throws an exception if the Fragment contains no metadata.

- template<class T >
  T const ∗ metadata () const

  Return a const pointer to the metadata. This throws an exception if the Fragment contains no metadata.
- template<class T >
  void setMetadata (const T &metadata)

  Set the metadata in the Fragment to the contents of the specified structure. This throws an exception if the Fragment already contains metadata.
- template<class T >
  void updateMetadata (const T &metadata)

  Updates existing metadata with a new metadata object.
- void resize (std::size_t sz)

  Resize the data payload to hold sz RawDataType words.
- void resize (std::size_t sz, RawDataType val)

  Resize the data payload to hold sz RawDataType words. Initialize new elements (if any) with val.
- void resizeBytes (std::size_t szbytes)

  Resize the data payload to hold szbytes bytes (padded by the 8-byte RawDataTypes, so, e.g., requesting 14 bytes will actually get you 16)
- void resizeBytesWithCushion (std::size_t szbytes, double growthFactor=1.3)

  Resize the data payload to hold szbytes bytes (padded by the 8-byte RawDataTypes, so, e.g., requesting 14 bytes will actually get you 16) and request additional capacity in the underlying storage (to help avoid extra reallocations)
- void resizeBytes (std::size_t szbytes, byte_t val)

  Resize the data payload to hold sz bytes (padded by the 8-byte RawDataTypes, so, e.g., requesting 14 bytes will actually get you 16). Initialize new elements (if any) with val.
- void autoResize ()

  Resize the fragment to hold the number of words indicated by the header.
- iterator dataBegin ()

  Return an iterator to the beginning of the data payload (after header and metadata)
- iterator dataEnd ()

  Return an iterator to the end of the data payload.
- template<typename T >
  T reinterpret_cast_checked (const RawDataType ∗in) const

  Wrapper around reinterpret_cast.
- template<typename T >
  T reinterpret_cast_checked (RawDataType ∗in)

  Wrapper around reinterpret_cast.
- byte_t ∗ dataBeginBytes ()

  Return Fragment::byte_t∗ pointing at the beginning of the payload.
- byte_t ∗ dataEndBytes ()

  Return Fragment::byte_t∗ pointing at the end of the payload.
- iterator headerBegin ()

  Return an iterator to the beginning of the header (should be used for serialization only: use setters for preference).
- byte_t ∗ headerBeginBytes ()

  Return a Fragment::byte_t pointer pointing to the beginning of the header.
- const_iterator dataBegin () const

  Returns a const_iterator to the beginning of the data payload.
- const_iterator dataEnd () const

  Returns a const_iterator to the end of the data payload.
- const byte_t ∗ dataBeginBytes () const

Return const Fragment::byte_t∗ pointing at the beginning of the payload.
- const byte_t ∗ dataEndBytes () const

  Return const Fragment::byte_t∗ pointing at the end of the payload.
- const_iterator headerBegin () const

  Return an const_iterator to the beginning of the header (should be used for serialization only: use setters for preference).
- const byte_t ∗ headerBeginBytes () const

  Return a const Fragment::byte_t pointer pointing to the beginning of the header.
- size_t headerSizeWords () const

  Get the size of this Fragment's header, in RawDataType words.
- size_t headerSizeBytes () const

  Get the size of this Fragment's header, in bytes.
- void clear ()

  Removes all elements from the payload section of the Fragment.
- bool empty ()

  Determines if the Fragment contains no data.
- void reserve (std::size_t cap)

  Reserves enough memory to hold cap RawDataType words in the Fragment payload.
- void swap (Fragment &other) noexcept

  Swaps two Fragment objects.
- void swap (QuickVec< RawDataType > &other) noexcept

  Swaps two Fragment data vectors.
- RawDataType ∗ dataAddress ()

  Returns a RawDataType pointer to the beginning of the payload.
- RawDataType ∗ metadataAddress ()

  Get the address of the metadata. For internal use only, use metadata() instead.
- RawDataType ∗ headerAddress ()

  Gets the address of the header.
- detail::RawFragmentHeader const fragmentHeader () const

  Get a copy of the RawFragmentHeader from this Fragment.

Static Public Member Functions

- static constexpr bool isUserFragmentType (type_t fragmentType)

  Returns whether the given type is in the range of user types.
- static constexpr bool isSystemFragmentType (type_t fragmentType)

  Returns whether the given type is in the range of system types.
- static std::map< type_t,
  std::string > MakeSystemTypeMap ()

  Returns a map of the most commonly-used system types.
- static FragmentPtr FragmentBytes (std::size_t nbytes)

  Create a Fragment using a static factory function rather than a constructor to allow for the function name "FragmentBytes".
- template<class T >
  static FragmentPtr FragmentBytes (std::size_t payload_size_in_bytes, sequence_id_t sequence_id, fragment_id_t fragment_id, type_t type, const T &metadata, timestamp_t timestamp=Fragment::InvalidTimestamp)

  Create a Fragment with the given header values. Uses static factory function instead of constructor to allow for the function name "FragmentBytes".
- static FragmentPtr eodFrag (size_t nFragsToExpect)

Creates an EndOfData Fragment.

- template<class InputIterator >
  static FragmentPtr dataFrag (sequence_id_t sequenceID, fragment_id_t fragID, InputIterator i, InputIterator e)

  Creates a Fragment, copying data from given location. 12-Apr-2013, KAB - this method is deprecated, please do not use (internal use only)
- static FragmentPtr dataFrag (sequence_id_t sequenceID, fragment_id_t fragID, RawDataType const ∗dataPtr, size_t dataSize, timestamp_t timestamp=Fragment::InvalidTimestamp)

  Crates a Fragment, copying data from given location.

Static Public Attributes

- static constexpr version_t InvalidVersion = detail::RawFragmentHeader::InvalidVersion

  Copy InvalidVersion from RawFragmentHeader.
- static constexpr sequence_id_t InvalidSequenceID = detail::RawFragmentHeader::InvalidSequenceID

  Copy InvalidSequenceID from RawFragmentHeader.
- static constexpr fragment_id_t InvalidFragmentID = detail::RawFragmentHeader::InvalidFragmentID

  Copy InvalidFragmentID from RawFragmentHeader.
- static constexpr timestamp_t InvalidTimestamp = detail::RawFragmentHeader::InvalidTimestamp

  Copy InvalidTimestamp from RawFragmentHeader.
- static constexpr type_t InvalidFragmentType = detail::RawFragmentHeader::InvalidFragmentType

  Copy InvalidFragmentType from RawFragmentHeader.
- static constexpr type_t EndOfDataFragmentType = detail::RawFragmentHeader::EndOfDataFragmentType

  Copy EndOfDataFragmentType from RawFragmentHeader.
- static constexpr type_t DataFragmentType = detail::RawFragmentHeader::DataFragmentType

  Copy DataFragmentType from RawFragmentHeader.
- static constexpr type_t InitFragmentType = detail::RawFragmentHeader::InitFragmentType

  Copy InitFragmentType from RawFragmentHeader.
- static constexpr type_t EndOfRunFragmentType = detail::RawFragmentHeader::EndOfRunFragmentType

  Copy EndOfRunFragmentType from RawFragmentHeader.
- static constexpr type_t EndOfSubrunFragmentType = detail::RawFragmentHeader::EndOfSubrunFragmentType

  Copy EndOfSubrunFragmentType from RawFragmentHeader.
- static constexpr type_t ShutdownFragmentType = detail::RawFragmentHeader::ShutdownFragmentType

  Copy ShutdownFragmentType from RawFragmentHeader.
- static constexpr type_t FirstUserFragmentType = detail::RawFragmentHeader::FIRST_USER_TYPE

  Copy FIRST_USER_TYPE from RawFragmentHeader.
- static constexpr type_t EmptyFragmentType = detail::RawFragmentHeader::EmptyFragmentType

  Copy EmptyFragmentType from RawFragmentHeader.
- static constexpr type_t ContainerFragmentType = detail::RawFragmentHeader::ContainerFragmentType

  Copy ContainerFragmentType from RawFragmentHeader.
- static constexpr type_t ErrorFragmentType = detail::RawFragmentHeader::ErrorFragmentType

  Copy ErrorFragmentType from RawFragmentHeader.

6.6.1 Detailed Description

A Fragment contains the data from one piece of the DAQ system for one event The artdaq::Fragment is the main data storage class in artdaq. Each Fragment represents the data from one piece of the readout, for one artdaq event. BoardReaders create Fragments and send them to the EventBuilders, where they are assembled into artdaq::RawEvent objects.

Definition at line 85 of file Fragment.hh.

6.6.2 Member Typedef Documentation

6.6.2.1 typedef uint8_t **artdaq::Fragment::byte_t**

For byte representation.

JCF, 3/25/14 Add interface functions which allow users to work with the underlying data (a vector of RawDataTypes) in byte representation

Definition at line 100 of file Fragment.hh.

6.6.3 Constructor & Destructor Documentation

6.6.3.1 artdaq::Fragment::Fragment (const **Fragment** & *f*) `[inline]`

Default copy constructor.

Todo Decide if Copy constructor should be declared =delete

Definition at line 783 of file Fragment.hh.

6.6.3.2 artdaq::Fragment::Fragment (**artdaq::Fragment** && *of*) `[inline]`, `[noexcept]`

Move Constructor.

Separate declaration and definition of Move Constructor: `http://stackoverflow.com/questions/33939687` This should generate an exception if artdaq::Fragment is not move-constructible

Definition at line 771 of file Fragment.hh.

6.6.3.3 artdaq::Fragment::Fragment (std::size_t *n*) `[explicit]`

Create a Fragment ready to hold n words (RawDataTypes) of payload, and with all values zeroed.

Parameters

n	The initial size of the Fragment, in RawDataType words

Definition at line 23 of file Fragment.cc.

6.6.3.4 template<class T > artdaq::Fragment::Fragment (std::size_t *payload_size*, **sequence_id_t** *sequence_id*, **fragment_id_t** *fragment_id*, **type_t** *type*, const T & *metadata*, **timestamp_t** *timestamp* = **Fragment::InvalidTimestamp**)

Create a Fragment with the given header values.

Template Parameters

T	Metadata type

Parameters

payload_size	Size of the payload in RawDataType words (Fragment size is header + metadata + payload)
sequence_id	Sequence ID of Fragment
fragment_id	Fragment ID of Fragment
type	Type of Fragment
metadata	Metadata object
timestamp	Timestamp of Fragment

Definition at line 831 of file Fragment.hh.

6.6.3.5 artdaq::Fragment::Fragment (sequence_id_t *sequenceID*, fragment_id_t *fragID*, type_t *type* = Fragment::DataFragmentType, timestamp_t *timestamp* = Fragment::InvalidTimestamp)

Create a fragment with the given event id and fragment id, and with no data payload.

Parameters

sequenceID	Sequence ID of Fragment
fragID	Fragment ID of Fragment
type	Type of Fragment
timestamp	Timestamp of Fragment

Definition at line 42 of file Fragment.cc.

6.6.4 Member Function Documentation

6.6.4.1 struct timespec artdaq::Fragment::atime ( ) const

Get the last access time of the Fragment.

Returns

struct timespec with last access time of the Fragment

Definition at line 941 of file Fragment.hh.

6.6.4.2 artdaq::RawDataType ∗ artdaq::Fragment::dataAddress ( ) [inline]

Returns a RawDataType pointer to the beginning of the payload.

Returns

A RawDataType pointer to the beginning of the payload

Definition at line 1158 of file Fragment.hh.

6.6.4.3 artdaq::Fragment::iterator artdaq::Fragment::dataBegin ( ) [inline]

Return an iterator to the beginning of the data payload (after header and metadata)

Returns

iterator to the beginning of the data payload

Definition at line 1093 of file Fragment.hh.

6.6.4.4 artdaq::Fragment::const_iterator artdaq::Fragment::dataBegin ( ) const [inline]

Returns a const_iterator to the beginning of the data payload.

Returns

A const_iterator to the beginning of the data payload

Definition at line 1112 of file Fragment.hh.

6.6.4.5 byte_t∗ artdaq::Fragment::dataBeginBytes ( ) [inline]

Return Fragment::byte_t∗ pointing at the beginning of the payload.

Returns

byte_t pointer to beginning of data payload

JCF, 3/25/14 – one nice thing about returning a pointer rather than an iterator is that we don't need to take the address of the dereferenced iterator (e.g., via &∗dataBegin()) to get ahold of the memory

Definition at line 551 of file Fragment.hh.

6.6.4.6 const byte_t∗ artdaq::Fragment::dataBeginBytes ( ) const [inline]

Return const Fragment::byte_t∗ pointing at the beginning of the payload.

Returns

const byte_t pointer to beginning of data payload

JCF, 3/25/14 – one nice thing about returning a pointer rather than an iterator is that we don't need to take the address of the dereferenced iterator (e.g., via &∗dataEnd()) to get ahold of the memory

Definition at line 596 of file Fragment.hh.

6.6.4.7 artdaq::Fragment::iterator artdaq::Fragment::dataEnd ( ) [inline]

Return an iterator to the end of the data payload.

Returns

iterator to the end of the data payload

Definition at line 1100 of file Fragment.hh.

6.6.4.8 artdaq::Fragment::const_iterator artdaq::Fragment::dataEnd ( ) const [inline]

Returns a const_iterator to the end of the data payload.

Returns

A const_iterator to the end of the data payload

Definition at line 1119 of file Fragment.hh.

6.6.4.9 byte_t∗ artdaq::Fragment::dataEndBytes ( ) [inline]

Return Fragment::byte_t∗ pointing at the end of the payload.

Returns

byte_t pointer to end of data payload

JCF, 3/25/14 – one nice thing about returning a pointer rather than an iterator is that we don't need to take the address of the dereferenced iterator (e.g., via &∗dataEnd()) to get ahold of the memory

Definition at line 561 of file Fragment.hh.

6.6.4.10 const byte_t∗ artdaq::Fragment::dataEndBytes ( ) const [inline]

Return const Fragment::byte_t∗ pointing at the end of the payload.

Returns

const byte_t pointer to end of data payload

JCF, 3/25/14 – one nice thing about returning a pointer rather than an iterator is that we don't need to take the address of the dereferenced iterator (e.g., via &∗dataEnd()) to get ahold of the memory

Definition at line 609 of file Fragment.hh.

6.6.4.11 template<class InputIterator > static FragmentPtr artdaq::Fragment::dataFrag (sequence_id_t *sequenceID*, fragment_id_t *fragID*, InputIterator *i*, InputIterator *e*) [inline], [static]

Creates a Fragment, copying data from given location. 12-Apr-2013, KAB - this method is deprecated, please do not use (internal use only)

Template Parameters

InputIterator	Type of input iterator

Parameters

sequenceID	Sequence ID of new Fragment
fragID	Fragment ID of new Fragment

i	Beginning of input range
e	End of input range

Returns

FragmentPtr to created Fragment

Todo Change function access specifier to restrict access

Definition at line 711 of file Fragment.hh.

6.6.4.12 artdaq::FragmentPtr artdaq::Fragment::dataFrag (sequence_id_t *sequenceID,* fragment_id_t *fragID,* RawDataType const ∗ *dataPtr,* size_t *dataSize,* timestamp_t *timestamp* = Fragment::InvalidTimestamp) [static]

Crates a Fragment, copying data from given location.

Parameters

sequenceID	Sequence ID of new Fragment
fragID	Fragment ID of new Fragment
dataPtr	Pointer to data to store in Fragment
dataSize	Size of data to store in Fragment
timestamp	Timestamp of created Fragment

Returns

FragmentPtr to created Fragment

Definition at line 88 of file Fragment.cc.

6.6.4.13 std::size_t artdaq::Fragment::dataSize ( ) const [inline]

Return the number of RawDataType words in the data payload. This does not include the number of words in the header or the metadata.

Returns

Number of RawDataType words in the payload section of the Fragment

Definition at line 962 of file Fragment.hh.

6.6.4.14 std::size_t artdaq::Fragment::dataSizeBytes ( ) const [inline]

Return the number of bytes in the data payload. This does not include the number of bytes in the header or the metadata.

Returns

Definition at line 374 of file Fragment.hh.

6.6.4.15 bool artdaq::Fragment::empty ( ) [inline]

Determines if the Fragment contains no data.

Returns

Whether the Fragment's payload is empty

Definition at line 1138 of file Fragment.hh.

6.6.4.16 artdaq::FragmentPtr artdaq::Fragment::eodFrag (size_t *nFragsToExpect*) [static]

Creates an EndOfData Fragment.

Parameters

nFragsToExpect	The number of Fragments the receiver should have at the end of data-taking

Returns

Pointer to created EndOfData Fragment

Definition at line 77 of file Fragment.cc.

6.6.4.17 static FragmentPtr artdaq::Fragment::FragmentBytes (std::size_t *nbytes*) [inline], [static]

Create a Fragment using a static factory function rather than a constructor to allow for the function name "Fragment-Bytes".

Parameters

nbytes	The initial size of the Fragment, in bytes

Returns

FragmentPtr to created Fragment

Definition at line 202 of file Fragment.hh.

6.6.4.18 template<class T > static FragmentPtr artdaq::Fragment::FragmentBytes (std::size_t *payload_size_in_bytes*, sequence_id_t *sequence_id*, fragment_id_t *fragment_id*, type_t *type*, const T & *metadata*, timestamp_t *timestamp* = Fragment::InvalidTimestamp) [inline], [static]

Create a Fragment with the given header values. Uses static factory function instead of constructor to allow for the function name "FragmentBytes".

Template Parameters

T	Metadata type

Parameters

payload_size_in-_bytes	Size of the payload in bytes (Fragment size is header + metadata + payload). Bytes will be rounded to the next factor of RawDataType / sizeof(char)
sequence_id	Sequence ID of Fragment
fragment_id	Fragment ID of Fragment
type	Type of Fragment
metadata	Metadata object
timestamp	Timestamp of Fragment

Returns

FragmentPtr to created Fragment

Definition at line 237 of file Fragment.hh.

6.6.4.19 artdaq::detail::RawFragmentHeader const artdaq::Fragment::fragmentHeader ( ) const [inline]

Get a copy of the RawFragmentHeader from this Fragment.

Returns

Copy of the RawFragmentHeader of this Fragment, upgraded to the latest version

Definition at line 1245 of file Fragment.hh.

6.6.4.20 artdaq::Fragment::fragment_id_t artdaq::Fragment::fragmentID ( ) const [inline]

Fragment ID of the Fragment, from the Fragment header.

Returns

Fragment ID of the Fragment

Definition at line 894 of file Fragment.hh.

6.6.4.21 struct timespec artdaq::Fragment::getLatency (bool *touch*)

Get the difference between the current time and the last access time of the Fragment.

Parameters

touch	Whether to also perform a touch operation

Returns

struct timespec representing the difference between current time and the last access time

Definition at line 946 of file Fragment.hh.

6.6.4.22 bool artdaq::Fragment::hasMetadata ( ) const [inline]

Test whether this Fragment has metadata.

Returns

If a metadata object has been set

Definition at line 969 of file Fragment.hh.

6.6.4.23 artdaq::RawDataType ∗ artdaq::Fragment::headerAddress ( ) [inline]

Gets the address of the header.

Returns

Pointer to the header's location within the vals_ vector

Definition at line 1176 of file Fragment.hh.

6.6.4.24 artdaq::Fragment::iterator artdaq::Fragment::headerBegin ( ) [inline]

Return an iterator to the beginning of the header (should be used for serialization only: use setters for preference).

Returns

an iterator to the beginning of the header

Definition at line 1106 of file Fragment.hh.

6.6.4.25 artdaq::Fragment::const_iterator artdaq::Fragment::headerBegin ( ) const [inline]

Return an const_iterator to the beginning of the header (should be used for serialization only: use setters for preference).

Returns

an const_iterator to the beginning of the header

Definition at line 1125 of file Fragment.hh.

6.6.4.26 byte_t∗ artdaq::Fragment::headerBeginBytes ( ) [inline]

Return a Fragment::byte_t pointer pointing to the beginning of the header.

Returns

byte_t pointer to the beginning of the header

Definition at line 574 of file Fragment.hh.

6.6.4.27 const byte_t* artdaq::Fragment::headerBeginBytes ( ) const [inline]

Return a const Fragment::byte_t pointer pointing to the beginning of the header.

Returns

const byte_t pointer to the beginning of the header

Definition at line 625 of file Fragment.hh.

6.6.4.28 size_t artdaq::Fragment::headerSizeBytes ( ) const [inline]

Get the size of this Fragment's header, in bytes.

Returns

The on-disk or in-memory size of the Fragment header, in bytes

Definition at line 640 of file Fragment.hh.

6.6.4.29 size_t artdaq::Fragment::headerSizeWords ( ) const [inline]

Get the size of this Fragment's header, in RawDataType words.

Returns

The on-disk or in-memory size of the Fragment header, in RawDataType words

Definition at line 1182 of file Fragment.hh.

6.6.4.30 bool constexpr artdaq::Fragment::isSystemFragmentType (type_t *fragmentType*) [inline], [static]

Returns whether the given type is in the range of system types.

Parameters

fragmentType	The type to test

Returns

Whether the given type is in the range of system types

Definition at line 798 of file Fragment.hh.

6.6.4.31 bool constexpr artdaq::Fragment::isUserFragmentType (type_t *fragmentType*) [inline], [static]

Returns whether the given type is in the range of user types.

Parameters

fragmentType	The type to test

Returns

Whether the given type is in the range of user types

Definition at line 791 of file Fragment.hh.

6.6.4.32 static std::map<type_t, std::string> artdaq::Fragment::MakeSystemTypeMap ( ) [inline], [static]

Returns a map of the most commonly-used system types.

Returns

A std::map of the most commonly-used system types

Definition at line 177 of file Fragment.hh.

6.6.4.33 template<class T > T ∗ artdaq::Fragment::metadata ( )

Return a pointer to the metadata. This throws an exception if the Fragment contains no metadata.

Template Parameters

T	Type of the metadata

Returns

Pointer to the metadata

Exceptions

cet::exception	if no metadata is present

Definition at line 975 of file Fragment.hh.

6.6.4.34 template<class T > T const ∗ artdaq::Fragment::metadata ( ) const

Return a const pointer to the metadata. This throws an exception if the Fragment contains no metadata.

Template Parameters

T	Type of the metadata

Returns

const Pointer to the metadata

Exceptions

cet::exception	if no metadata is present

Definition at line 988 of file Fragment.hh.

6.6.4.35 artdaq::RawDataType ∗ artdaq::Fragment::metadataAddress ( ) [inline]

Get the address of the metadata. For internal use only, use metadata() instead.

Returns

Pointer to the metadata's location within the vals_ vector

Exceptions

cet::exception	if no metadata in Fragment

Todo Change function access specifier to restrict access

Definition at line 1165 of file Fragment.hh.

6.6.4.36 artdaq::Fragment & artdaq::Fragment::operator= (const Fragment & *f*) [inline]

Default copy-assignment operator.

Returns

Reference to new Fragment

Todo Decide if copy-assignment operator should be declared =delete

Definition at line 784 of file Fragment.hh.

6.6.4.37 artdaq::Fragment & artdaq::Fragment::operator= (artdaq::Fragment && *of*) [inline], [noexcept]

Move-assignment operator.

Returns

Reference to Fragment

Separate declaration and definition of Move Constructor: `http://stackoverflow.com/questions/33939687`
This should generate an exception if artdaq::Fragment is not move-constructible

Definition at line 774 of file Fragment.hh.

6.6.4.38 void artdaq::Fragment::print (std::ostream & *os*) const

Print out summary information for this Fragment to the given stream.

Parameters

os	Stream to print to

Definition at line 68 of file Fragment.cc.

6.6.4.39 template< typename T > T artdaq::Fragment::reinterpret_cast_checked (const RawDataType ∗ *in*) const [inline]

Wrapper around reinterpret_cast.

Template Parameters

T	Type of output pointer

Parameters

in	input pointer

Returns

Pointer cast to type T

Exceptions

cet::exception	if new pointer does not point to same address as old pointer

JCF, 1/21/15 There's actually not an ironclad guarantee in the C++ standard that the pointer reinterpret_cast<> returns has the same address as the pointer that was casted. It IS tested in the artdaq-core test suite, but since any uncaught, unexpected behavior from reinterpret_cast could be disastrous, I've wrapped it in this function and added a check just to be completely safe.

Please note that for this const-version, you'll need the const- qualifier to the pointer you pass as a parameter (i.e., reinterpret_cast_checked<const PtrType∗>, not reinterpret_cast_checked<PtrType∗>)

Definition at line 503 of file Fragment.hh.

6.6.4.40 template< typename T > T artdaq::Fragment::reinterpret_cast_checked (RawDataType ∗ *in*) [inline]

Wrapper around reinterpret_cast.

Template Parameters

T	Type of output pointer

Parameters

in	input pointer

Returns

Pointer cast to type T

Exceptions

cet::exception	if new pointer does not point to same address as old pointer

JCF, 1/21/15 There's actually not an ironclad guarantee in the C++ standard that the pointer reinterpret_cast<> returns has the same address as the pointer that was casted. It IS tested in the artdaq-core test suite, but since any uncaught, unexpected behavior from reinterpret_cast could be disastrous, I've wrapped it in this function and added a check just to be completely safe.

Definition at line 531 of file Fragment.hh.

6.6.4.41 void artdaq::Fragment::reserve (std::size_t *cap*) [inline]

Reserves enough memory to hold cap RawDataType words in the Fragment payload.

Parameters

cap	The new capacity of the Fragment payload, in RawDataType words.

Definition at line 1145 of file Fragment.hh.

6.6.4.42 void artdaq::Fragment::resize (std::size_t *sz*) [inline]

Resize the data payload to hold sz RawDataType words.

Parameters

sz	The new size of the payload portion of the Fragment, in RawDataType words

Definition at line 1035 of file Fragment.hh.

6.6.4.43 void artdaq::Fragment::resize (std::size_t *sz*, RawDataType *val*) [inline]

Resize the data payload to hold sz RawDataType words. Initialize new elements (if any) with val.

Parameters

sz	The new size of the payload portion of the Fragment, in RawDataType words
val	Value with which to initialize any new elements

Definition at line 1043 of file Fragment.hh.

6.6.4.44 void artdaq::Fragment::resizeBytes (std::size_t *szbytes*) [inline]

Resize the data payload to hold szbytes bytes (padded by the 8-byte RawDataTypes, so, e.g., requesting 14 bytes will actually get you 16)

Parameters

szbytes	The new size of the payload portion of the Fragment, in bytes

Definition at line 1052 of file Fragment.hh.

6.6.4.45 void artdaq::Fragment::resizeBytes (std::size_t *szbytes*, byte_t *val*) [inline]

Resize the data payload to hold sz bytes (padded by the 8-byte RawDataTypes, so, e.g., requesting 14 bytes will actually get you 16). Initialize new elements (if any) with val.

Parameters

szbytes	The new size of the payload portion of the Fragment, in bytes
val	Value with which to initialize any new elements

Definition at line 1069 of file Fragment.hh.

6.6.4.46 void artdaq::Fragment::resizeBytesWithCushion (std::size_t *szbytes*, double *growthFactor* = `1.3`) `[inline]`

Resize the data payload to hold szbytes bytes (padded by the 8-byte RawDataTypes, so, e.g., requesting 14 bytes will actually get you 16) and request additional capacity in the underlying storage (to help avoid extra reallocations)

Parameters

szbytes	The new size of the payload portion of the Fragment, in bytes
growthFactor	The requested growth factor in the capacity of storage

Definition at line 1059 of file Fragment.hh.

6.6.4.47 artdaq::Fragment::sequence_id_t artdaq::Fragment::sequenceID ( ) const `[inline]`

Sequence ID of the Fragment, from the Fragment header.

Returns

Sequence ID of the Fragment

Definition at line 888 of file Fragment.hh.

6.6.4.48 void artdaq::Fragment::setFragmentID (fragment_id_t *fragment_id*) `[inline]`

Sets the Fragment ID of the Fragment.

Parameters

fragment_id	The Fragment ID to set

Definition at line 925 of file Fragment.hh.

6.6.4.49 template<class T > void artdaq::Fragment::setMetadata (const T & *metadata*)

Set the metadata in the Fragment to the contents of the specified structure. This throws an exception if the Fragment already contains metadata.

Template Parameters

T	Type of the metadata

Parameters

metadata	Metadata to store in Fragment

Exceptions

cet::exception	if metadata already present in Fragment

Definition at line 999 of file Fragment.hh.

6.6.4.50 void artdaq::Fragment::setSequenceID (sequence_id_t *sequence_id*) [inline]

Sets the Sequence ID of the Fragment.

Parameters

sequence_id	The sequence ID to set

Definition at line 918 of file Fragment.hh.

6.6.4.51 void artdaq::Fragment::setSystemType (type_t *stype*) [inline]

Sets the type of the Fragment, checking that it is a valid system type.

Parameters

stype	The System type to set

Definition at line 912 of file Fragment.hh.

6.6.4.52 void artdaq::Fragment::setTimestamp (timestamp_t *timestamp*) [inline]

Sets the Timestamp of the Fragment.

Parameters

timestamp	The Timestamp to set

Definition at line 931 of file Fragment.hh.

6.6.4.53 void artdaq::Fragment::setUserType (type_t *utype*) [inline]

Sets the type of the Fragment, checking that it is a valid user type.

Parameters

utype	The User type to set

Definition at line 906 of file Fragment.hh.

6.6.4.54 std::size_t artdaq::Fragment::size ( ) const [inline]

Gets the size of the Fragment, from the Fragment header.

Returns

Number of words in the Fragment. Includes header, metadata, and payload

Definition at line 863 of file Fragment.hh.

6.6.4.55 std::size_t artdaq::Fragment::sizeBytes ( ) const [inline]

Size of vals_ vector (header + (optional) metadata + payload) in bytes.

Returns

The size of the Fragment in bytes, including header, metadata, and payload

Definition at line 360 of file Fragment.hh.

6.6.4.56 void artdaq::Fragment::swap (Fragment & *other*) [inline], [noexcept]

Swaps two Fragment objects.

Parameters

other	Fragment to swap with

Definition at line 1152 of file Fragment.hh.

6.6.4.57 void artdaq::Fragment::swap (QuickVec< RawDataType > & *other*) [inline], [noexcept]

Swaps two Fragment data vectors.

Parameters

other	The data vector to swap with

Since all Fragment header information is stored in the data vector, this is equivalent to swapping two Fragment objects

Definition at line 671 of file Fragment.hh.

6.6.4.58 artdaq::Fragment::timestamp_t artdaq::Fragment::timestamp ( ) const [inline]

Timestamp of the Fragment, from the Fragment header.

Returns

Timestamp of the Fragment

Definition at line 900 of file Fragment.hh.

6.6.4.59 artdaq::Fragment::type_t artdaq::Fragment::type ( ) const [inline]

Type of the Fragment, from the Fragment header.

Returns

Type of the Fragment

Definition at line 876 of file Fragment.hh.

6.6.4.60 std::string artdaq::Fragment::typeString ( ) const [inline]

Print the type of the Fragment.

Returns

String representation of the Fragment type. For system types, the name will be included in parentheses

Definition at line 882 of file Fragment.hh.

6.6.4.61 template<class T > void artdaq::Fragment::updateMetadata (const T & *metadata*)

Updates existing metadata with a new metadata object.

Template Parameters

T	Type of the metadata

Parameters

metadata	Metadata to set

Exceptions

cet::exception	if no metadata stored in Fragment
cet::exception	if new metadata has different size than existing metadata

Definition at line 1015 of file Fragment.hh.

6.6.4.62 artdaq::Fragment::version_t artdaq::Fragment::version ( ) const [inline]

Version of the Fragment, from the Fragment header.

Returns

Version of the Fragment

Definition at line 869 of file Fragment.hh.

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

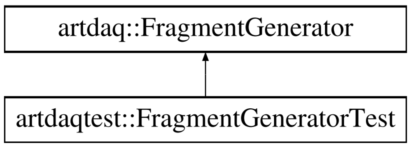
- artdaq_core/artdaq-core/Data/Fragment.hh
- artdaq_core/artdaq-core/Data/Fragment.cc

6.7 artdaq::FragmentGenerator Class Reference

Base class for all FragmentGenerators.

`#include <artdaq-core/Plugins/FragmentGenerator.hh>`

Inheritance diagram for artdaq::FragmentGenerator:



Public Member Functions

- FragmentGenerator ()=default

  Default Constructor.
- virtual bool getNext (FragmentPtrs &output)=0

  Obtain the next collection of Fragments.
- virtual std::vector
  < Fragment::fragment_id_t > fragmentIDs ()=0

  Which fragment IDs does this FragmentGenerator generate?

6.7.1 Detailed Description

Base class for all FragmentGenerators.

FragmentGenerator is an abstract class that defines the interface for obtaining events in artdaq. Subclasses are to override the (private) virtual functions; users of FragmentGenerator are to invoke the public (non-virtual) functions.

Definition at line 24 of file FragmentGenerator.hh.

6.7.2 Member Function Documentation

6.7.2.1 virtual std::vector<**Fragment::fragment_id_t**> artdaq::FragmentGenerator::fragmentIDs ( ) `[pure virtual]`

Which fragment IDs does this FragmentGenerator generate?

Returns

A std::vector of fragment_id_t

Each FragmentGenerator is responsible for one or more Fragment IDs. Fragment IDs should be unique in an event, and consistent for a given piece of hardware.

Implemented in artdaqtest::FragmentGeneratorTest.

6.7.2.2 virtual bool artdaq::FragmentGenerator::getNext (**FragmentPtrs** & *output*) `[pure virtual]`

Obtain the next collection of Fragments.

Parameters

output	New FragmentPtr objects will be added to this FragmentPtrs object.

Returns

False indicates end-of-data

Obtain the next collection of Fragments. Return false to indicate end-of-data. Fragments may or may not be in the same event; Fragments may or may not have the same FragmentID. Fragments will all be part of the same Run and SubRun.

Implemented in artdaqtest::FragmentGeneratorTest.

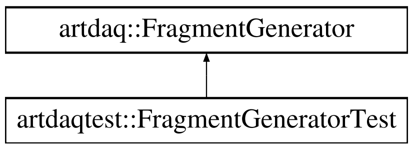
The documentation for this class was generated from the following file:

- artdaq_core/artdaq-core/Plugins/FragmentGenerator.hh

6.8 artdaqtest::FragmentGeneratorTest Class Reference

Tests the functionality of the artdaq::FragmentGenerator class.

Inheritance diagram for artdaqtest::FragmentGeneratorTest:



Public Member Functions

- bool getNext (artdaq::FragmentPtrs &output) override

  Obtain the next collection of Fragments.
- std::vector
  < artdaq::Fragment::fragment_id_t > fragmentIDs () override

  Which fragment IDs does this FragmentGenerator generate?

6.8.1 Detailed Description

Tests the functionality of the artdaq::FragmentGenerator class.

Definition at line 14 of file FragmentGenerator_t.cc.

6.8.2 Member Function Documentation

6.8.2.1 std::vector<**artdaq::Fragment::fragment_id_t**> artdaqtest::FragmentGeneratorTest::fragmentIDs ( ) [inline], [override], [virtual]

Which fragment IDs does this FragmentGenerator generate?

Returns

A std::vector of fragment_id_t

Each FragmentGenerator is responsible for one or more Fragment IDs. Fragment IDs should be unique in an event, and consistent for a given piece of hardware.

Implements artdaq::FragmentGenerator.

Definition at line 24 of file FragmentGenerator_t.cc.

6.8.2.2 bool artdaqtest::FragmentGeneratorTest::getNext (artdaq::FragmentPtrs & *output*) [inline], [override], [virtual]

Obtain the next collection of Fragments.

Parameters

output	New FragmentPtr objects will be added to this FragmentPtrs object.

Returns

False indicates end-of-data

Obtain the next collection of Fragments. Return false to indicate end-of-data. Fragments may or may not be in the same event; Fragments may or may not have the same FragmentID. Fragments will all be part of the same Run and SubRun.

Implements artdaq::FragmentGenerator.

Definition at line 19 of file FragmentGenerator_t.cc.

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

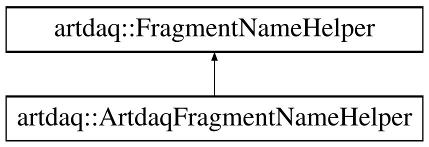
- artdaq_core/test/Plugins/FragmentGenerator_t.cc

6.9 artdaq::FragmentNameHelper Class Reference

The FragmentNameHelper translates between Fragments and their instance names (usually by type, but any/all Raw-FragmentHeader fields, or even Overlays, may be used)

```
#include <artdaq-core/Plugins/FragmentNameHelper.hh>
```

Inheritance diagram for artdaq::FragmentNameHelper:



Public Member Functions

- virtual ∼FragmentNameHelper ()=default

  Default virtual destructor.

- FragmentNameHelper (std::string unidentified_instance_name, std::vector< std::pair< artdaq::Fragment::type_t, std::string >> extraTypes)

  FragmentNameHelper Constructor.
- void SetBasicTypes (std::map< artdaq::Fragment::type_t, std::string > const &type_map)

  Sets the basic types to be translated. (Should not include "container" types.)
- void AddExtraType (artdaq::Fragment::type_t type_id, std::string const &type_name)

  Adds an additional type to be translated.
- std::string GetUnidentifiedInstanceName () const

  Get the configured unidentified_instance_name.
- virtual std::string GetInstanceNameForType (artdaq::Fragment::type_t type_id) const

  Returns the basic translation for the specified type. Must be implemented by derived classes.
- virtual std::set< std::string > GetAllProductInstanceNames () const

  Returns the full set of product instance names which may be present in the data, based on the types that have been specified in the SetBasicTypes() and AddExtraType() methods. This does include "container" types, if the container type mapping is part of the basic types. Must be implemented by derived classes.
- virtual std::pair< bool,
  std::string > GetInstanceNameForFragment (artdaq::Fragment const &fragment) const

  Returns the product instance name for the specified fragment, based on the types that have been specified in the SetBasicTypes() and AddExtraType() methods. This does include the use of "container" types, if the container type mapping is part of the basic types. If no mapping is found, the specified unidentified_instance_name should be returned. Must be implemented by derived classes.

Protected Attributes

- std::map
  < artdaq::Fragment::type_t,
  std::string > type_map_

  Map relating Fragment Type to strings.
- std::string unidentified_instance_name_

  The name to use for unknown Fragment types.

6.9.1 Detailed Description

The FragmentNameHelper translates between Fragments and their instance names (usually by type, but any/all Raw-FragmentHeader fields, or even Overlays, may be used)

Definition at line 35 of file FragmentNameHelper.hh.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 artdaq::FragmentNameHelper::FragmentNameHelper (std::string *unidentified_instance_name,* std::vector< std::pair< artdaq::Fragment::type_t, std::string >> *extraTypes*) [inline]

FragmentNameHelper Constructor.

Parameters

unidentified_-instance_name	Name to use for unidentified Fragments
extraTypes	Additional types to register

Definition at line 48 of file FragmentNameHelper.hh.

6.9.3 Member Function Documentation

6.9.3.1 std::string artdaq::FragmentNameHelper::GetUnidentifiedInstanceName ( ) const [inline]

Get the configured unidentified_instance_name.

Returns

The configured unidentified_instance_name

Definition at line 82 of file FragmentNameHelper.hh.

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

- artdaq_core/artdaq-core/Plugins/FragmentNameHelper.hh

6.10 artdaqcore::GetPackageBuildInfo Struct Reference

Wrapper around the artdaqcore::GetPackageBuildInfo::getPackageBuildInfo function.

`#include <artdaq-core/BuildInfo/GetPackageBuildInfo.hh>`

Static Public Member Functions

- static artdaq::PackageBuildInfo getPackageBuildInfo ()

  Gets the version number and build timestmap for artdaq_core.

6.10.1 Detailed Description

Wrapper around the artdaqcore::GetPackageBuildInfo::getPackageBuildInfo function.

Definition at line 14 of file GetPackageBuildInfo.hh.

6.10.2 Member Function Documentation

6.10.2.1 static artdaq::PackageBuildInfo artdaqcore::GetPackageBuildInfo::getPackageBuildInfo ( ) [static]

Gets the version number and build timestmap for artdaq_core.

Returns

An artdaq::PackageBuildInfo object containing the version number and build timestamp for artdaq_core

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

- artdaq_core/artdaq-core/BuildInfo/GetPackageBuildInfo.hh

6.11 artdaq::ContainerFragment::Metadata Struct Reference

Contains the information necessary for retrieving Fragment objects from the ContainerFragment.

`#include <artdaq-core/Data/ContainerFragment.hh>`

Public Types

- typedef uint8_t data_t
  Basic unit of data-retrieval.
- typedef uint64_t count_t
  Size of block_count variables.

Public Attributes

- count_t block_count: 16
  The number of Fragment objects stored in the ContainerFragment.
- count_t fragment_type: 8
  The Fragment::type_t of stored Fragment objects.
- count_t version: 4
  Version number of ContainerFragment.
- count_t missing_data: 1
  Flag if the ContainerFragment knows that it is missing data.
- count_t has_index: 1
  Whether the ContainerFragment has an index at the end of the payload.
- count_t unused_flag1: 1
  Unused.
- count_t unused_flag2: 1
  Unused.
- count_t unused: 32
  Unused.
- uint64_t index_offset
  Index starts this many bytes after the beginning of the payload (is also the total size of contained Fragments)

Static Public Attributes

- static size_t const size_words = 16ul
  Size of the Metadata object.

6.11.1 Detailed Description

Contains the information necessary for retrieving Fragment objects from the ContainerFragment.

Definition at line 56 of file ContainerFragment.hh.

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

- artdaq_core/artdaq-core/Data/ContainerFragment.hh

6.12 MetadataTypeHuge Struct Reference

Test Metadata that is very large.

Public Attributes

- uint64_t fields [300]
  300 long words

6.12.1 Detailed Description

Test Metadata that is very large.

Definition at line 32 of file Fragment_t.cc.

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

- artdaq_core/test/Data/Fragment_t.cc

6.13 MetadataTypeOne Struct Reference

Test Metadata with three fields in two long words.

Public Attributes

- uint64_t field1
- uint32_t field2
- uint32_t field3

6.13.1 Detailed Description

Test Metadata with three fields in two long words.

Definition at line 10 of file Fragment_t.cc.

6.13.2 Member Data Documentation

6.13.2.1 uint64_t MetadataTypeOne::field1

1. A 64-bit field

Definition at line 12 of file Fragment_t.cc.

6.13.2.2 uint32_t MetadataTypeOne::field2

1. A 32-bit field

Definition at line 13 of file Fragment_t.cc.

6.13.2.3 uint32_t MetadataTypeOne::field3

1. A 32-bit field

Definition at line 14 of file Fragment_t.cc.

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

- artdaq_core/test/Data/Fragment_t.cc

6.14 MetadataTypeTwo Struct Reference

Test Metadata with five fields, mixing field sizes.

Public Attributes

- uint64_t field1
- uint32_t field2
- uint32_t field3
- uint64_t field4
- uint16_t field5

6.14.1 Detailed Description

Test Metadata with five fields, mixing field sizes.

Definition at line 20 of file Fragment_t.cc.

6.14.2 Member Data Documentation

6.14.2.1 uint64_t MetadataTypeTwo::field1

1. A 64-bit field

Definition at line 22 of file Fragment_t.cc.

6.14.2.2 uint32_t MetadataTypeTwo::field2

1. A 32-bit field

Definition at line 23 of file Fragment_t.cc.

6.14.2.3 uint32_t MetadataTypeTwo::field3

1. A 32-bit field

Definition at line 24 of file Fragment_t.cc.

6.14.2.4 uint64_t MetadataTypeTwo::field4

1. A 64-bit field

Definition at line 25 of file Fragment_t.cc.

6.14.2.5 uint16_t MetadataTypeTwo::field5

1. A 16-bit field

Definition at line 26 of file Fragment_t.cc.

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

- artdaq_core/test/Data/Fragment_t.cc

6.15 artdaq::ContainerFragment::MetadataV0 Struct Reference

Contains the information necessary for retrieving Fragment objects from the ContainerFragment.

```
#include <artdaq-core/Data/ContainerFragment.hh>
```

Public Types

- typedef uint8_t data_t

  Basic unit of data-retrieval.
- typedef uint64_t count_t

  Size of block_count variables.

Public Attributes

- count_t block_count: 55

  The number of Fragment objects stored in the ContainerFragment.
- count_t fragment_type: 8

  The Fragment::type_t of stored Fragment objects.
- count_t missing_data: 1

  Flag if the ContainerFragment knows that it is missing data.
- size_t index [CONTAINER_FRAGMENT_COUNT_MAX]

  Offset of each Fragment within the ContainerFragment.

Static Public Attributes

- static constexpr int CONTAINER_FRAGMENT_COUNT_MAX = 100

  The maximum capacity of the ContainerFragment (in fragments)
- static size_t const size_words = 8ul + CONTAINER_FRAGMENT_COUNT_MAX ∗ sizeof(size_t) / sizeof(data_t)

  Size of the Metadata object.

6.15.1 Detailed Description

Contains the information necessary for retrieving Fragment objects from the ContainerFragment.

Definition at line 31 of file ContainerFragment.hh.

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

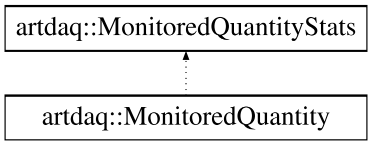
- artdaq_core/artdaq-core/Data/ContainerFragment.hh

6.16 artdaq::MonitoredQuantity Class Reference

This class keeps track of statistics for a set of sample values and provides timing information on the samples.

`#include <artdaq-core/Core/MonitoredQuantity.hh>`

Inheritance diagram for artdaq::MonitoredQuantity:



Public Member Functions

- MonitoredQuantity (DURATION_T expectedCalculationInterval, DURATION_T timeWindowForRecentResults)

  Instantiates a MonitoredQuantity object.
- void addSample (const double value=1.0)

  Adds the specified doubled valued sample value to the monitor instance.
- void addSample (const int value=1)

  Adds the specified integer valued sample value to the monitor instance.
- void addSample (const uint32_t value=1)

  Adds the specified uint32_t valued sample value to the monitor instance.
- void addSample (const uint64_t value=1)

  Adds the specified uint64_t valued sample value to the monitor instance.
- bool calculateStatistics (TIME_POINT_T currentTime=getCurrentTime())

  Forces a calculation of the statistics for the monitored quantity.
- void reset ()
- void enable ()
- void disable ()
- bool isEnabled () const

  Tests whether the monitor is currently enabled.
- void setNewTimeWindowForRecentResults (DURATION_T interval)

  Specifies a new time interval to be used when calculating "recent" statistics.
- DURATION_T getTimeWindowForRecentResults () const

  Returns the length of the time window that has been specified for recent results.
- DURATION_T ExpectedCalculationInterval () const

  Returns the expected interval between calculateStatistics calls.

- bool waitUntilAccumulatorsHaveBeenFlushed (DURATION_T timeout) const

  Blocks while the MonitoredQuantity is flushed, up to timeout duration.
- void getStats (MonitoredQuantityStats &stats) const

  Write all our collected statistics into the given Stats struct.
- TIME_POINT_T getLastCalculationTime () const

  Access the last calculation time.
- DURATION_T getFullDuration () const

  Access the full duration of the statistics.
- double getRecentValueSum () const

  Access the sum of the value samples in the "recent" time span.
- double getRecentValueAverage () const

  Access the average of the value samples in the "recent" time span.
- size_t getFullSampleCount () const

  Access the count of samples for the entire history of the MonitoredQuantity.

Static Public Member Functions

- static TIME_POINT_T getCurrentTime ()

  Returns the current point in time.

6.16.1 Detailed Description

This class keeps track of statistics for a set of sample values and provides timing information on the samples.

Definition at line 158 of file MonitoredQuantity.hh.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 MonitoredQuantity::MonitoredQuantity (DURATION_T *expectedCalculationInterval*, DURATION_T *timeWindowForRecentResults*) `[explicit]`

Instantiates a MonitoredQuantity object.

Parameters

expected-Calculation-Interval	How often calculateStatistics is expected to be called
timeWindowFor-RecentResults	Defines the meaning of DataSetType::RECENT

Definition at line 8 of file MonitoredQuantity.cc.

6.16.3 Member Function Documentation

6.16.3.1 void MonitoredQuantity::addSample (const double *value* = `1.0`)

Adds the specified doubled valued sample value to the monitor instance.

Parameters

value	The sample value to add

Definition at line 17 of file MonitoredQuantity.cc.

6.16.3.2 void MonitoredQuantity::addSample (const int *value* = `1`)

Adds the specified integer valued sample value to the monitor instance.

Parameters

value	The sample value to add

Definition at line 33 of file MonitoredQuantity.cc.

6.16.3.3 void MonitoredQuantity::addSample (const uint32_t *value* = `1`)

Adds the specified uint32_t valued sample value to the monitor instance.

Parameters

value	The sample value to add

Definition at line 38 of file MonitoredQuantity.cc.

6.16.3.4 void MonitoredQuantity::addSample (const uint64_t *value* = `1`)

Adds the specified uint64_t valued sample value to the monitor instance.

Parameters

value	The sample value to add

Definition at line 43 of file MonitoredQuantity.cc.

6.16.3.5 bool MonitoredQuantity::calculateStatistics (TIME_POINT_T *currentTime* = getCurrentTime`()`)

Forces a calculation of the statistics for the monitored quantity.

Parameters

currentTime	Time point to use for calculating statistics (if synchronized at a higher level)

Returns

Whether the statisics were calculated

Forces a calculation of the statistics for the monitored quantity. The frequency of the updates to the statistics is driven by how often this method is called. It is expected that this method will be called once per interval specified by expected-CalculationInterval

Definition at line 48 of file MonitoredQuantity.cc.

6.16.3.6 void MonitoredQuantity::disable ( )

Disables the monitor.

Definition at line 272 of file MonitoredQuantity.cc.

6.16.3.7 void MonitoredQuantity::enable ( )

Enables the monitor (and resets the statistics to provide a fresh start).

Definition at line 263 of file MonitoredQuantity.cc.

6.16.3.8 DURATION_T artdaq::MonitoredQuantity::ExpectedCalculationInterval ( ) const [inline]

Returns the expected interval between calculateStatistics calls.

Returns

The expected interval between calculateStatistics calls

Definition at line 256 of file MonitoredQuantity.hh.

6.16.3.9 MonitoredQuantity::TIME_POINT_T MonitoredQuantity::getCurrentTime ( ) [static]

Returns the current point in time.

Returns

The current point in time.

Returns the current point in time. A negative value indicates that an error occurred when fetching the time from the operating system.

Definition at line 374 of file MonitoredQuantity.cc.

6.16.3.10 void MonitoredQuantity::getStats (MonitoredQuantityStats & *stats*) const

Write all our collected statistics into the given Stats struct.

Parameters

stats	Destination for copy of collected statistics

Definition at line 331 of file MonitoredQuantity.cc.

6.16.3.11 DURATION_T artdaq::MonitoredQuantity::getTimeWindowForRecentResults ( ) const [inline]

Returns the length of the time window that has been specified for recent results.

Returns

The length of the time windows for recent results

Returns the length of the time window that has been specified for recent results. (This may be different than the actual length of the recent time window which is affected by the interval of calls to the calculateStatistics() method. Use a getDuration(RECENT) call to determine the actual recent time window.)

Definition at line 247 of file MonitoredQuantity.hh.

6.16.3.12 bool artdaq::MonitoredQuantity::isEnabled ( ) const [inline]

Tests whether the monitor is currently enabled.

Returns

Whether the monitor is currently enabled.

Definition at line 227 of file MonitoredQuantity.hh.

6.16.3.13 void MonitoredQuantity::reset ( )

Resets the monitor (zeroes out all counters and restarts the time interval).

Definition at line 251 of file MonitoredQuantity.cc.

6.16.3.14 void MonitoredQuantity::setNewTimeWindowForRecentResults (DURATION_T *interval*)

Specifies a new time interval to be used when calculating "recent" statistics.

Parameters

interval	The new time interval for calculating "recent" statistics.

Definition at line 279 of file MonitoredQuantity.cc.

6.16.3.15 bool MonitoredQuantity::waitUntilAccumulatorsHaveBeenFlushed (DURATION_T *timeout*) const

Blocks while the MonitoredQuantity is flushed, up to timeout duration.

Parameters

timeout	How long to wait for the MonitoredQuantity to be emptied, in seconds

Returns

Whether the MonitoredQuantity was emptied within the specified timeout

Definition at line 312 of file MonitoredQuantity.cc.

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

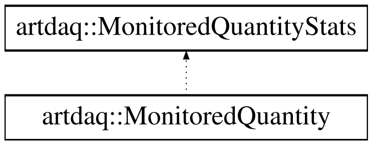
- artdaq_core/artdaq-core/Core/MonitoredQuantity.hh
- artdaq_core/artdaq-core/Core/MonitoredQuantity.cc

6.17 artdaq::MonitoredQuantityStats Struct Reference

struct containing MonitoredQuantity data

`#include <artdaq-core/Core/MonitoredQuantity.hh>`

Inheritance diagram for artdaq::MonitoredQuantityStats:



Public Types

- enum DataSetType { DataSetType::FULL = 0, DataSetType::RECENT = 1 }

  Which data points to return (all or only recent)
- typedef double DURATION_T

  A Duration.
- typedef double TIME_POINT_T

  A point in time.

Public Member Functions

- size_t getSampleCount (DataSetType t=DataSetType::FULL) const

  Returns the sample count for the requested interval.
- double getValueSum (DataSetType t=DataSetType::FULL) const

  Returns the sum of values in the requested interval.
- double getValueAverage (DataSetType t=DataSetType::FULL) const

  Returns the average of the values in the requested interval.
- double getValueRate (DataSetType t=DataSetType::FULL) const

  Returns the sum of the values in the requested interval, divided by the duration of the requested interval.
- double getValueRMS (DataSetType t=DataSetType::FULL) const

  Returns the RMS of the values in the requested interval.
- double getValueMin (DataSetType t=DataSetType::FULL) const

  Returns the smallest of the values in the requested interval.
- double getValueMax (DataSetType t=DataSetType::FULL) const

  Returns the largest of the values in the requested interval.
- DURATION_T getDuration (DataSetType t=DataSetType::FULL) const

  Returns the duration of the requested interval.
- double getSampleRate (DataSetType t=DataSetType::FULL) const

  Returns the sample rate in the requested interval.
- double getSampleLatency (DataSetType t=DataSetType::FULL) const
- double getLastSampleValue () const

  Accessor for the last sample value recorded.
- double getLastValueRate () const

  Accessor for the lastValueRate (Sum of last samples over interval between calculateStatisics calls)
- bool isEnabled () const

  Access the enable flag.

Public Attributes

- size_t fullSampleCount

  The total number of samples represented.
- double fullSampleRate

  The total number of samples over the full duration of sampling.
- double fullValueSum

  The sum of all samples.
- double fullValueSumOfSquares

  The sum of the squares of all samples.
- double fullValueAverage

  The average of all samples.
- double fullValueRMS

  The RMS of all samples.
- double fullValueMin

  The smallest value of all samples.
- double fullValueMax

  The largest value of all sampels.
- double fullValueRate

  The sum of all samples over the full duration of sampling.
- DURATION_T fullDuration

  The full duration of sampling.
- size_t recentSampleCount

  The number of samples in the "recent" time window.
- double recentSampleRate

  The number of samples in the "recent" time window, divided by the length of that window.
- double recentValueSum

  The sum of the "recent" samples.
- double recentValueSumOfSquares

  The sum of the squares of the "recent" samples.
- double recentValueAverage

  The average of the "recent" samples.
- double recentValueRMS

  The RMS of the 'recent" samples.
- double recentValueMin

  The smallest value of the "recent" samples.
- double recentValueMax

  The largest value of the "recent" samples.
- double recentValueRate

  The sum of the "recent" samples, divided by the length of the "recent" time window.
- DURATION_T recentDuration

  The length of the "recent" time window.
- std::vector< size_t > recentBinnedSampleCounts

  Sample counts for each instance of calculateStatistics in _intervalForRecentStats (rolling window)
- std::vector< double > recentBinnedValueSums

  Sums for each instance of calculateStatistics in _intervalForRecentStats (rolling window)
- std::vector< DURATION_T > recentBinnedDurations

Duration between each instance of calcualteStatistics in _intervalForRecentStats (rolling window)
- std::vector< TIME_POINT_T > recentBinnedEndTimes

  Last sample time in each instance of calculateStatistics in _intervalForRecentStats (rolling window)
- double lastSampleValue

  Value of the most recent sample.
- double lastValueRate

  Latest rate point (sum of values over calculateStatistics interval)
- TIME_POINT_T lastCalculationTime

  Last time calculateStatistics was called.
- bool enabled

  Whether the MonitoredQuantity is collecting data.

6.17.1 Detailed Description

struct containing MonitoredQuantity data

Definition at line 15 of file MonitoredQuantity.hh.

6.17.2 Member Enumeration Documentation

6.17.2.1 enum artdaq::MonitoredQuantityStats::DataSetType [strong]

Which data points to return (all or only recent)

Enumerator

FULL	the full data set (all samples)
RECENT	recent data only

Definition at line 23 of file MonitoredQuantity.hh.

6.17.3 Member Function Documentation

6.17.3.1 DURATION_T artdaq::MonitoredQuantityStats::getDuration (DataSetType *t* = DataSetType::FULL) const [inline]

Returns the duration of the requested interval.

Parameters

t	Which interval to return, DataSetType::FULL (default) or DataSetType::RECENT

Returns

The duration of the requested interval

Definition at line 115 of file MonitoredQuantity.hh.

6.17.3.2 double artdaq::MonitoredQuantityStats::getLastSampleValue ( ) const [inline]

Accessor for the last sample value recorded.

Returns

The last sample value recorded

Definition at line 139 of file MonitoredQuantity.hh.

6.17.3.3 double artdaq::MonitoredQuantityStats::getLastValueRate ( ) const [inline]

Accessor for the lastValueRate (Sum of last samples over interval between calculateStatisics calls)

Returns

The lastValueRate (Sum of last samples over interval between calculateStatisics calls)

Definition at line 145 of file MonitoredQuantity.hh.

6.17.3.4 size_t artdaq::MonitoredQuantityStats::getSampleCount (DataSetType *t* = DataSetType::FULL) const [inline]

Returns the sample count for the requested interval.

Parameters

t	Which interval to return, DataSetType::FULL (default) or DataSetType::RECENT

Returns

The sample count for the requested interval

Definition at line 66 of file MonitoredQuantity.hh.

6.17.3.5 double artdaq::MonitoredQuantityStats::getSampleLatency (DataSetType *t* = DataSetType::FULL) const [inline]

Parameters

t	Which interval to return, DataSetType::FULL (default) or DataSetType::RECENT

Returns

Definition at line 129 of file MonitoredQuantity.hh.

6.17.3.6 double artdaq::MonitoredQuantityStats::getSampleRate (DataSetType *t* = DataSetType::FULL) const [inline]

Returns the sample rate in the requested interval.

Parameters

t	Which interval to return, DataSetType::FULL (default) or DataSetType::RECENT

Returns

The sample rate in the requested interval

Definition at line 122 of file MonitoredQuantity.hh.

6.17.3.7 double artdaq::MonitoredQuantityStats::getValueAverage (DataSetType *t* = DataSetType::FULL) const [inline]

Returns the average of the values in the requested interval.

Parameters

t	Which interval to return, DataSetType::FULL (default) or DataSetType::RECENT

Returns

The average of the values in the requested interval

Definition at line 80 of file MonitoredQuantity.hh.

6.17.3.8 double artdaq::MonitoredQuantityStats::getValueMax (DataSetType *t* = DataSetType::FULL) const [inline]

Returns the largest of the values in the requested interval.

Parameters

t	Which interval to return, DataSetType::FULL (default) or DataSetType::RECENT

Returns

The largest of the values in the requested interval

Definition at line 108 of file MonitoredQuantity.hh.

6.17.3.9 double artdaq::MonitoredQuantityStats::getValueMin (DataSetType *t* = DataSetType::FULL) const [inline]

Returns the smallest of the values in the requested interval.

Parameters

t	Which interval to return, DataSetType::FULL (default) or DataSetType::RECENT

Returns

The smallest of the values in the requested interval

Definition at line 101 of file MonitoredQuantity.hh.

6.17.3.10 double artdaq::MonitoredQuantityStats::getValueRate (DataSetType *t* = DataSetType::FULL) const [inline]

Returns the sum of the values in the requested interval, divided by the duration of the requested interval.

Parameters

t	Which interval to return, DataSetType::FULL (default) or DataSetType::RECENT

Returns

The sum of the values in the requested interval, divided by the duration of the requested interval

Definition at line 87 of file MonitoredQuantity.hh.

6.17.3.11 double artdaq::MonitoredQuantityStats::getValueRMS (DataSetType *t* = DataSetType::FULL) const [inline]

Returns the RMS of the values in the requested interval.

Parameters

t	Which interval to return, DataSetType::FULL (default) or DataSetType::RECENT

Returns

The RMS of the values in the requested interval

Definition at line 94 of file MonitoredQuantity.hh.

6.17.3.12 double artdaq::MonitoredQuantityStats::getValueSum (DataSetType *t* = DataSetType::FULL) const [inline]

Returns the sum of values in the requested interval.

Parameters

t	Which interval to return, DataSetType::FULL (default) or DataSetType::RECENT

Returns

The sum of values in the requested interval

Definition at line 73 of file MonitoredQuantity.hh.

6.17.3.13 bool artdaq::MonitoredQuantityStats::isEnabled ( ) const [inline]

Access the enable flag.

Returns

The current value of the enable flag

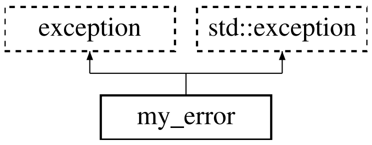
Definition at line 151 of file MonitoredQuantity.hh.

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

- artdaq_core/artdaq-core/Core/MonitoredQuantity.hh

6.18 my_error Struct Reference

Inheritance diagram for my_error:



6.18.1 Detailed Description

Definition at line 17 of file ExceptionHandler_t.cc.

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

- artdaq_core/test/Utilities/ExceptionHandler_t.cc

6.19 artdaq::PackageBuildInfo Class Reference

Class holding information about the *artdaq* package build.

```
#include <artdaq-core/Data/PackageBuildInfo.hh>
```

Public Member Functions

- PackageBuildInfo ()

    Default Constructor.
- std::string getPackageName () const

    Gets the package name.
- std::string getPackageVersion () const

    Gets the package version.
- std::string getBuildTimestamp () const

    Gets the build timestamp.
- void setPackageName (std::string const &str)

    Sets the package name.
- void setPackageVersion (std::string const &str)

    Sets the package version.
- void setBuildTimestamp (std::string const &str)

    Sets the build timestamp.

6.19.1 Detailed Description

Class holding information about the *artdaq* package build.

The PackageBuildInfo class contains the name of the package, the version, and the timestamp of the build. *artdaq* stores this information in each data file.

Definition at line 17 of file PackageBuildInfo.hh.

6.19.2 Member Function Documentation

6.19.2.1 std::string artdaq::PackageBuildInfo::getBuildTimestamp ( ) const [inline]

Gets the build timestamp.

Returns

The timestamp of the build

Definition at line 41 of file PackageBuildInfo.hh.

6.19.2.2 std::string artdaq::PackageBuildInfo::getPackageName ( ) const [inline]

Gets the package name.

Returns

The package name

Definition at line 29 of file PackageBuildInfo.hh.

6.19.2.3 std::string artdaq::PackageBuildInfo::getPackageVersion ( ) const [inline]

Gets the package version.

Returns

The package version

Definition at line 35 of file PackageBuildInfo.hh.

6.19.2.4 void artdaq::PackageBuildInfo::setBuildTimestamp (std::string const & *str*) [inline]

Sets the build timestamp.

Parameters

str	The timestamp of the build

Definition at line 59 of file PackageBuildInfo.hh.

6.19.2.5 void artdaq::PackageBuildInfo::setPackageName (std::string const & *str*) [inline]

Sets the package name.

Parameters

str	The package name

Definition at line 47 of file PackageBuildInfo.hh.

6.19.2.6 void artdaq::PackageBuildInfo::setPackageVersion (std::string const & *str*) [inline]

Sets the package version.

Parameters

str	The package version

Definition at line 53 of file PackageBuildInfo.hh.

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

- artdaq_core/artdaq-core/Data/PackageBuildInfo.hh

6.20 artdaq::QuickVec< TT_ > Struct Template Reference

A QuickVec behaves like a std::vector, but does no initialization of its data, making it faster at the cost of having to ensure that uninitialized data is not read.

```
#include <artdaq-core/Core/QuickVec.hh>
```

Public Types

- typedef TT_ ∗ iterator

  Iterator is pointer-to-member type.
- typedef const TT_ ∗ const_iterator

  const_iterator is const-pointer-to-member type
- typedef TT_ & reference

  reference is reference-to-member tpye
- typedef const TT_ & const_reference

  const_reference is const-reference-to-member type
- typedef TT_ value_type

  value_type is member type
- typedef ptrdiff_t difference_type

  difference_type is ptrdiff_t
- typedef size_t size_type

  size_type is size_t

Public Member Functions

- QuickVec (size_t sz)

  Allocates a QuickVec object, doing no initialization of allocated memory.
- QuickVec (size_t sz, TT_ val)

  Allocates a QuickVec object, initializing each element to the given value.
- virtual ∼QuickVec () noexcept

  Destructor calls free on data.
- QuickVec (std::vector< TT_ > &other)

  Copies the contents of a std::vector into a new QuickVec object.
- void clear ()

  Sets the size to 0. QuickVec does not reinitialize memory, so no further action will be taken.
- QuickVec (const QuickVec &other)

  Copy Constructor.
- QuickVec< TT_ > & operator= (const QuickVec &other)

Copy assignment operator.

- TT_ & operator[] (int idx)

  Returns a reference to a given element.
- const TT_ & operator[] (int idx) const

  Returns a const reference to a given element.
- size_t size () const

  Accesses the current size of the QuickVec.
- size_t capacity () const

  Accesses the current capacity of the QuickVec.
- iterator begin ()

  Gets an iterator to the beginning of the QuickVec.
- const_iterator begin () const

  Gets a const_iterator to the beginning of the QuickVec.
- iterator end ()

  Gets an iterator to the end of the QuickVec.
- const_iterator end () const

  Gets a const_iterator to the end of the QuickVec.
- void reserve (size_t size)

  Allocates memory for the QuickVec so that its capacity is at least size.
- void resize (size_t size)

  Resizes the QuickVec.
- void resizeWithCushion (size_t size, double growthFactor=1.3)

  Resizes the QuickVec and requests additional capacity.
- void resize (size_t size, TT_ val)

  Resizes the QuickVec, initializes new elements with val.
- iterator insert (const_iterator position, size_t nn, const TT_ &val)

  Inserts an element into the QuickVec.
- iterator insert (const_iterator position, const_iterator first, const_iterator last)

  Inserts a range of elements into the QuickVec.
- iterator erase (const_iterator first, const_iterator last)

  Erases elements in given range from the QuickVec.
- void swap (QuickVec &other) noexcept

  Exchanges references to two QuickVec objects.
- void push_back (const value_type &val)

  Adds a value to the QuickVec, resizing if necessary (adds 10% capacity)

Static Public Member Functions

- static short Class_Version ()

  Returns the current version of the template code \ $*$.

6.20.1 Detailed Description

template<typename TT_>struct artdaq::QuickVec< TT_ >

A QuickVec behaves like a std::vector, but does no initialization of its data, making it faster at the cost of having to ensure that uninitialized data is not read.

Template Parameters

*TT_*	The data type stored in the QuickVec

Definition at line 90 of file QuickVec.hh.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 template<typename TT_ > artdaq::QuickVec< TT_ >::QuickVec (size_t *sz*) [inline]

Allocates a QuickVec object, doing no initialization of allocated memory.

Parameters

sz	Size of QuickVec object to allocate

Definition at line 348 of file QuickVec.hh.

6.20.2.2 template<typename TT_> artdaq::QuickVec< TT_ >::QuickVec (size_t *sz*, TT_ *val*) [inline]

Allocates a QuickVec object, initializing each element to the given value.

Parameters

sz	Size of QuickVec object to allocate
val	Value with which to initialize elements

Definition at line 357 of file QuickVec.hh.

6.20.2.3 template<typename TT_> artdaq::QuickVec< TT_ >::QuickVec (std::vector< TT_ > & *other*) [inline]

Copies the contents of a std::vector into a new QuickVec object.

Parameters

other	The vector to copy

Definition at line 122 of file QuickVec.hh.

6.20.2.4 template<typename TT_> artdaq::QuickVec< TT_ >::QuickVec (const QuickVec< TT_ > & *other*) [inline]

Copy Constructor.

Parameters

other	QuickVec to copy

Definition at line 141 of file QuickVec.hh.

6.20.3 Member Function Documentation

6.20.3.1 template<typename TT_ > QuickVec< TT_ >::iterator artdaq::QuickVec< TT_ >::begin ( ) [inline]

Gets an iterator to the beginning of the QuickVec.

Returns

An iterator to the beginning of the QuickVec

Definition at line 398 of file QuickVec.hh.

6.20.3.2 template<typename TT_ > QuickVec< TT_ >::const_iterator artdaq::QuickVec< TT_ >::begin ( ) const [inline]

Gets a const_iterator to the beginning of the QuickVec.

Returns

A const_iterator to the beginning of the QuickVec

Definition at line 401 of file QuickVec.hh.

6.20.3.3 template<typename TT_ > size_t artdaq::QuickVec< TT_ >::capacity ( ) const [inline]

Accesses the current capacity of the QuickVec.

Returns

The current capacity of the QuickVec

Accesses the current capcity of the QuickVec. Like a vector, the capacity of a QuickVec object is defined as the maximum size it can hold before it must reallocate more memory.

Definition at line 395 of file QuickVec.hh.

6.20.3.4 template<typename TT_> static short artdaq::QuickVec< TT_ >::Class_Version ( ) [inline], [static]

Returns the current version of the template code \ ∗.

\ ∗

Returns

The current version of the QuickVec \ ∗ \ ∗ Class_Version() MUST be updated every time private member data change. \

Definition at line 336 of file QuickVec.hh.

6.20.3.5 template<typename TT_ > QuickVec< TT_ >::iterator artdaq::QuickVec< TT_ >::end ( ) [inline]

Gets an iterator to the end of the QuickVec.

Returns

An iterator to the end of the QuickVec

Definition at line 404 of file QuickVec.hh.

6.20.3.6 template<typename TT_ > QuickVec< TT_ >::const_iterator artdaq::QuickVec< TT_ >::end ( ) const [inline]

Gets a const_iterator to the end of the QuickVec.

Returns

A const_iterator to the end of the QuickVec

Definition at line 410 of file QuickVec.hh.

6.20.3.7 template<typename TT_ > QuickVec< TT_ >::iterator artdaq::QuickVec< TT_ >::erase (const_iterator *first*, const_iterator *last*) [inline]

Erases elements in given range from the QuickVec.

Parameters

first	First element to erase
last	Last element to erase

Returns

iterator to first element after erase range

Erases elements in given range from the QuickVec. Note that since the underlying data structure resembles a std::vector, erase operations are very inefficient! (O(n))

Definition at line 525 of file QuickVec.hh.

6.20.3.8 template<typename TT_> QuickVec< TT_ >::iterator artdaq::QuickVec< TT_ >::insert (const_iterator *position*, size_t *nn*, const TT_ & *val*) [inline]

Inserts an element into the QuickVec.

Parameters

position	Position at which to isnert
nn	Number of copies of val to insert
val	Value to insert

Returns

Iterator to first inserted element

Inserts an element (or copies thereof) into the QuickVec. Note that since the underlying data structure resembles a std::vector, insert operations are very inefficient!

Definition at line 488 of file QuickVec.hh.

6.20.3.9 template<typename TT_> QuickVec< TT_ >::iterator artdaq::QuickVec< TT_ >::insert (const_iterator *position*, const_iterator *first*, const_iterator *last*) [inline]

Inserts a range of elements into the QuickVec.

Parameters

position	Position at which to insert
first	const_iterator to first element to insert
last	const_iterator to last element to insert

Returns

Iterator to first inserted element

Inserts elements into the QuickVec. Note that since the underlying data structure resembles a std::vector, insert operations are very inefficient!

Definition at line 506 of file QuickVec.hh.

6.20.3.10 template<typename TT_> QuickVec<TT_>& artdaq::QuickVec< TT_ >::operator= (const QuickVec< TT_ > & *other*) [inline]

Copy assignment operator.

Parameters

other	QuickVec to copy

Returns

Reference to new QuickVec object

Definition at line 155 of file QuickVec.hh.

6.20.3.11 template<typename TT_ > TT_ & artdaq::QuickVec< TT_ >::operator[] (int *idx*) [inline]

Returns a reference to a given element.

Parameters

idx	Element to return

Returns

Reference to element

Definition at line 378 of file QuickVec.hh.

6.20.3.12 template<typename TT_ > const TT_ & artdaq::QuickVec< TT_ >::operator[] (int *idx*) const [inline]

Returns a const reference to a given element.

Parameters

idx	Element to return

Returns

const reference to element

Definition at line 385 of file QuickVec.hh.

6.20.3.13 template<typename TT_ > void artdaq::QuickVec< TT_ >::push_back (const value_type & *val*) [inline]

Adds a value to the QuickVec, resizing if necessary (adds 10% capacity)

Parameters

val	Value to add to the QuickVec

Definition at line 551 of file QuickVec.hh.

6.20.3.14 template<typename TT_ > void artdaq::QuickVec< TT_ >::reserve (size_t *size*) [inline]

Allocates memory for the QuickVec so that its capacity is at least size.

Parameters

size	The new capacity of the QuickVec

Allocates memory for the QuickVec so that its capacity is at least size. If the QuickVec is already at or above size in capacity, no allocation is performed.

Definition at line 416 of file QuickVec.hh.

6.20.3.15 template<typename TT_ > void artdaq::QuickVec< TT_ >::resize (size_t *size*) [inline]

Resizes the QuickVec.

Parameters

size	New size of the QuickVec

If size is smaller than the current size of the QuickVec, then it will change its size_ parameter (no reallocation, capacity does not change). If size is greater than the capacity of the QuickVec, a reallocation will occur.

Definition at line 432 of file QuickVec.hh.

6.20.3.16 template<typename TT_> void artdaq::QuickVec< TT_ >::resize (size_t *size,* TT_ *val*) [inline]

Resizes the QuickVec, initializes new elements with val.

Parameters

size	New size of the QuickVec
val	Value with which to initialize elements

Definition at line 476 of file QuickVec.hh.

6.20.3.17 template<typename TT_ > void artdaq::QuickVec< TT_ >::resizeWithCushion (size_t *size*, double *growthFactor* = 1.3) [inline]

Resizes the QuickVec and requests additional capacity.

Parameters

size	New size of the QuickVec
growthFactor	Factor to use when allocating additional capacity

This method updates the size of the QuickVec. If the new size is within the current capacity, no realloction takes place. If not, then the reallocation reserves additional capacity as a cushion against future needs to reallocate, based on the specified growth factor.

Definition at line 451 of file QuickVec.hh.

6.20.3.18 template< typename TT_ > size_t artdaq::QuickVec< TT_ >::size ( ) const [inline]

Accesses the current size of the QuickVec.

Returns

The current size of the QuickVec

Definition at line 392 of file QuickVec.hh.

6.20.3.19 template< typename TT_ > void artdaq::QuickVec< TT_ >::swap (QuickVec< TT_ > & *other*) [inline], [noexcept]

Exchanges references to two QuickVec objects.

Parameters

other	Other QuickVec to swap with

Definition at line 541 of file QuickVec.hh.

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

- artdaq_core/artdaq-core/Core/QuickVec.hh

6.21 artdaq::RawEvent Class Reference

RawEvent is the artdaq view of a generic event, containing a header and zero or more Fragments.

```
#include <artdaq-core/Data/RawEvent.hh>
```

Public Types

- typedef detail::RawEventHeader::run_id_t run_id_t

  Run numbers are 32 bits.
- typedef detail::RawEventHeader::subrun_id_t subrun_id_t

  Subrun numbers are 32 bits.
- typedef detail::RawEventHeader::event_id_t event_id_t

  Event numbers are 32 bits.

- typedef
  detail::RawEventHeader::sequence_id_t sequence_id_t

  Field size should be the same as the Fragment::sequence_id field.
- typedef
  detail::RawEventHeader::timestamp_t timestamp_t

  Field size should be the same as the Fragment::timestamp field.

Public Member Functions

- RawEvent (run_id_t run, subrun_id_t subrun, event_id_t event, sequence_id_t seq, timestamp_t ts)

  Constructs a RawEvent with the given parameters.
- RawEvent (detail::RawEventHeader hdr)

  Constructs a RawEvent using the given RawEventHeader.
- void insertFragment (FragmentPtr &&pfrag)

  Insert the given (pointer to a) Fragment into this RawEvent.
- void markComplete ()

  Mark the event as complete.
- size_t numFragments () const

  Return the number of fragments this RawEvent contains.
- size_t wordCount () const

  Return the sum of the word counts of all fragments in this RawEvent.
- run_id_t runID () const

  Retrieve the run number from the RawEventHeader.
- subrun_id_t subrunID () const

  Retrieve the subrun number from the RawEventHeader.
- event_id_t eventID () const

  Retrieve the event number from the RawEventHeader.
- sequence_id_t sequenceID () const

  Retrieve the sequence id from the RawEventHeader.
- timestamp_t timestamp () const

  Retrieve the timestamp from the RawEventHeader.
- bool isComplete () const

  Retrieve the value of the complete flag from the RawEventHeader.
- void print (std::ostream &os) const

  Print summary information about this RawEvent to the given stream.
- std::unique_ptr< Fragments > releaseProduct ()

  Release all the Fragments from this RawEvent.
- void fragmentTypes (std::vector< Fragment::type_t > &type_list)

  Fills in a list of unique fragment types from this event.
- std::unique_ptr< Fragments > releaseProduct (Fragment::type_t type)

  Release Fragments from the RawEvent.

6.21.1 Detailed Description

RawEvent is the artdaq view of a generic event, containing a header and zero or more Fragments.

RawEvent should be a class, not a struct; it should be enforcing invariants (the contained Fragments should all have the correct event id).

Definition at line 100 of file RawEvent.hh.

6.21.2 Constructor & Destructor Documentation

6.21.2.1 artdaq::RawEvent::RawEvent (**run_id_t** *run,* **subrun_id_t** *subrun,* **event_id_t** *event,* **sequence_id_t** *seq,* **timestamp_t** *ts*) `[inline]`

Constructs a RawEvent with the given parameters.

Parameters

run	The current Run number
subrun	The current Subrun number
event	The current Event number
seq	The current sequence_id
ts	The timestamp for the event

Definition at line 235 of file RawEvent.hh.

6.21.2.2 artdaq::RawEvent::RawEvent (**detail::RawEventHeader** *hdr*) `[inline]`, `[explicit]`

Constructs a RawEvent using the given RawEventHeader.

Parameters

hdr	Header to use for initializing RawEvent

Definition at line 239 of file RawEvent.hh.

6.21.3 Member Function Documentation

6.21.3.1 RawEvent::event_id_t artdaq::RawEvent::eventID ( ) const `[inline]`

Retrieve the event number from the RawEventHeader.

Returns

The event number stored in the RawEventHeader

Definition at line 270 of file RawEvent.hh.

6.21.3.2 void artdaq::RawEvent::fragmentTypes (std::vector< **Fragment::type_t** > & *type_list*) `[inline]`

Fills in a list of unique fragment types from this event.

Parameters

type_list	Any Fragment types not included in this list will be added

Definition at line 292 of file RawEvent.hh.

6.21.3.3 void artdaq::RawEvent::insertFragment (**FragmentPtr** && *pfrag*) `[inline]`

Insert the given (pointer to a) Fragment into this RawEvent.

Parameters

pfrag	The FragmentPtr to insert into the RawEvent

Exceptions

cet::exception	if pfrag is nullptr

Insert the given (pointer to a) Fragment into this RawEvent. This takes ownership of the Fragment referenced by the FragmentPtr, unless an exception is thrown.

Definition at line 244 of file RawEvent.hh.

6.21.3.4 bool artdaq::RawEvent::isComplete ( ) const [inline]

Retrieve the value of the complete flag from the RawEventHeader.

Returns

The value of RawEventHeader::is_complete

Definition at line 273 of file RawEvent.hh.

6.21.3.5 size_t artdaq::RawEvent::numFragments ( ) const [inline]

Return the number of fragments this RawEvent contains.

Returns

The number of Fragment objects in this RawEvent

Definition at line 256 of file RawEvent.hh.

6.21.3.6 void artdaq::RawEvent::print (std::ostream & *os*) const

Print summary information about this RawEvent to the given stream.

Parameters

os	The target stream for summary information

Definition at line 18 of file RawEvent.cc.

6.21.3.7 std::unique_ptr< **Fragments** > artdaq::RawEvent::releaseProduct ( ) [inline]

Release all the Fragments from this RawEvent.

Returns

A pointer to a Fragments object (owns the Fragment data contained)

Release all the Fragments from this RawEvent, returning them to the caller through a unique_ptr that manages a vector into which the Fragments have been moved.

Definition at line 275 of file RawEvent.hh.

6.21.3.8 std::unique_ptr< Fragments > artdaq::RawEvent::releaseProduct (Fragment::type_t *type*) [inline]

Release Fragments from the RawEvent.

Parameters

type	The type of Fragments to release

Returns

A pointer to a Fragments object (owns the Fragment data contained)

Release the Fragments from this RawEvent with the specified fragment type, returning them to the caller through a unique_ptr that manages a vector into which the Fragments have been moved. PLEASE NOTE that releaseProduct and releaseProduct(type_t) can not both be used on the same RawEvent since each one gives up ownership of the fragments within the event.

Definition at line 309 of file RawEvent.hh.

6.21.3.9 RawEvent::run_id_t artdaq::RawEvent::runID ( ) const [inline]

Retrieve the run number from the RawEventHeader.

Returns

The run number stored in the RawEventHeader

Definition at line 268 of file RawEvent.hh.

6.21.3.10 RawEvent::sequence_id_t artdaq::RawEvent::sequenceID ( ) const [inline]

Retrieve the sequence id from the RawEventHeader.

Returns

The sequence id stored in the RawEventHeader

Definition at line 271 of file RawEvent.hh.

6.21.3.11 RawEvent::subrun_id_t artdaq::RawEvent::subrunID ( ) const [inline]

Retrieve the subrun number from the RawEventHeader.

Returns

The subrun number stored in the RawEventHeader

Definition at line 269 of file RawEvent.hh.

6.21.3.12 RawEvent::timestamp_t artdaq::RawEvent::timestamp ( ) const [inline]

Retrieve the timestamp from the RawEventHeader.

Returns

The timestamp stored in the RawEventHeader

Definition at line 272 of file RawEvent.hh.

6.21.3.13 size_t artdaq::RawEvent::wordCount ( ) const [inline]

Return the sum of the word counts of all fragments in this RawEvent.

Returns

The sum of the word counts of all Fragment objects in this RawEvent

Definition at line 261 of file RawEvent.hh.

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

- artdaq_core/artdaq-core/Data/RawEvent.hh
- artdaq_core/artdaq-core/Data/RawEvent.cc

6.22 artdaq::detail::RawEventHeader Struct Reference

The header information used to identify key properties of the RawEvent object.

```
#include <artdaq-core/Data/RawEvent.hh>
```

Public Types

- typedef uint32_t run_id_t
  Run numbers are 32 bits.
- typedef uint32_t subrun_id_t
  Subrun numbers are 32 bits.
- typedef uint32_t event_id_t
  Event numbers are 32 bits.
- typedef uint64_t sequence_id_t
  Field size should be the same as the Fragment::sequence_id field.
- typedef uint64_t timestamp_t
  Field size should be the same as the Fragment::timestamp field.

Public Member Functions

- RawEventHeader ()
  Default constructor. Provided for ROOT compatibility.
- RawEventHeader (run_id_t run, subrun_id_t subrun, event_id_t event, sequence_id_t seq, timestamp_t ts)
  Constructs the RawEventHeader struct with the given parameters.
- void print (std::ostream &os) const
  Print a RawEventHeader to the given stream.

Public Attributes

- run_id_t run_id
  Fragments don't know about runs.
- subrun_id_t subrun_id

Fragments don't know about subruns.

- event_id_t event_id

    Event number should be either sequence ID or Timestamp of component Fragments.
- sequence_id_t sequence_id

    RawEvent sequence_id should be the same as its component Fragment sequence_ids.
- timestamp_t timestamp

    The timestamp of the first Fragment received for this event.
- bool is_complete

    Does the event contain the expected number of Fragment objects?
- uint8_t version

    Version number of the RawEventHeader.

Static Public Attributes

- static constexpr uint8_t CURRENT_VERSION = 0

    Current version of the RawEventHeader.

6.22.1 Detailed Description

The header information used to identify key properties of the RawEvent object.

RawEventHeader is the artdaq generic event header. It contains the information necessary for routing of raw events inside the artdaq code, but is not intended for use by any experiment.

Definition at line 26 of file RawEvent.hh.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 artdaq::detail::RawEventHeader::RawEventHeader (**run_id_t** *run,* **subrun_id_t** *subrun,* **event_id_t** *event,* **sequence_id_t** *seq,* **timestamp_t** *ts*) `[inline]`

Constructs the RawEventHeader struct with the given parameters.

Parameters

run	The current Run number
subrun	The current Subrun number
event	The current event number
seq	The current Sequence ID
ts	The current Timestamp

Definition at line 57 of file RawEvent.hh.

6.22.3 Member Function Documentation

6.22.3.1 void artdaq::detail::RawEventHeader::print (std::ostream & *os*) const

Print a RawEventHeader to the given stream.

Parameters

os	Output stream to print to

Definition at line 5 of file RawEvent.cc.

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

- artdaq_core/artdaq-core/Data/RawEvent.hh
- artdaq_core/artdaq-core/Data/RawEvent.cc

6.23 artdaq::detail::RawFragmentHeader Struct Reference

The RawFragmentHeader class contains the basic fields used by *artdaq* for routing Fragment objects through the system.

```
#include <artdaq-core/Data/detail/RawFragmentHeader.hh>
```

Public Types

- typedef unsigned long long RawDataType

  The RawDataType (currently an unsigned long long) is the basic unit of data representation within artdaq
- typedef uint16_t version_t

  version field is 16 bits
- typedef uint64_t sequence_id_t

  sequence_id field is 48 bits
- typedef uint8_t type_t

  type field is 8 bits
- typedef uint16_t fragment_id_t

  fragment_id field is 16 bits
- typedef uint8_t metadata_word_count_t

  metadata_word_count field is 8 bits
- typedef uint64_t timestamp_t

  timestamp field is 32 bits

Public Member Functions

- void setUserType (uint8_t utype)

  Sets the type field to the specified user type.
- void setSystemType (uint8_t stype)

  Sets the type field to the specified system type.
- void touch ()

  Update the atime fields of the RawFragmentHeader to current time.
- struct timespec atime () const

  Get the last access time of this RawFragmentHeader.
- struct timespec getLatency (bool touch)

  Get the elapsed time between now and the last access time of the RawFragmentHeader, optionally resetting it.
- bool operator== (const detail::RawFragmentHeader &other) const

  Check if two RawFragmentHeader objects are equal.

Static Public Member Functions

- static std::map< type_t,
  std::string > MakeSystemTypeMap ()

  Returns a map of the most-commonly used system types.
- static std::map< type_t,
  std::string > MakeVerboseSystemTypeMap ()

  Returns a map of all system types.
- static std::string SystemTypeToString (type_t type)

  Print a system type's string name.
- static constexpr std::size_t num_words ()

  Returns the number of RawDataType words present in the header.

Public Attributes

- RawDataType word_count: 32

  number of RawDataType words in this Fragment
- RawDataType version: 16

  The version of the fragment.
- RawDataType type: 8

  The type of the fragment, either system or user-defined.
- RawDataType metadata_word_count: 8

  The number of RawDataType words in the user-defined metadata.
- RawDataType sequence_id: 48

  The 48-bit sequence_id uniquely identifies events within the artdaq system.
- RawDataType fragment_id: 16

  The fragment_id uniquely identifies a particular piece of hardware within the artdaq system.
- RawDataType timestamp: 64

  The 64-bit timestamp field is the output of a user-defined clock used for building time-correlated events.
- RawDataType valid: 1

  Flag for whether the Fragment has been transported correctly through the artdaq system.
- RawDataType complete: 1

  Flag for whether the Fragment completely represents an event for its hardware.
- RawDataType atime_ns: 30

  Last access time of the Fragment, nanosecond part.
- RawDataType atime_s: 32

  Last access time of the Fragment, second part (measured from epoch)

Static Public Attributes

- static constexpr type_t INVALID_TYPE = 0

  Marks a Fragment as Invalid.
- static constexpr type_t FIRST_USER_TYPE = 1

  The first user-accessible type.
- static constexpr type_t LAST_USER_TYPE = 224

  The last user-accessible type (types above this number are system types.
- static constexpr type_t FIRST_SYSTEM_TYPE = 225

The first system type.

- static constexpr type_t LAST_SYSTEM_TYPE = 255

  The last system type.
- static constexpr type_t InvalidFragmentType = INVALID_TYPE

  Marks a Fragment as Invalid.
- static constexpr type_t EndOfDataFragmentType = FIRST_SYSTEM_TYPE

  This Fragment indicates the end of data to art
- static constexpr type_t DataFragmentType = FIRST_SYSTEM_TYPE + 1

  This Fragment holds data. Used for RawEvent Fragments sent from the EventBuilder to the Aggregator.
- static constexpr type_t InitFragmentType = FIRST_SYSTEM_TYPE + 2

  This Fragment holds the necessary data for initializing art
- static constexpr type_t EndOfRunFragmentType = FIRST_SYSTEM_TYPE + 3

  This Fragment indicates the end of a run to art
- static constexpr type_t EndOfSubrunFragmentType = FIRST_SYSTEM_TYPE + 4

  This Fragment indicates the end of a subrun to art
- static constexpr type_t ShutdownFragmentType = FIRST_SYSTEM_TYPE + 5

  This Fragment indicates a system shutdown to art
- static constexpr type_t EmptyFragmentType = FIRST_SYSTEM_TYPE + 6

  This Fragment contains no data and serves as a placeholder for when no data from a FragmentGenerator is expected.
- static constexpr type_t ContainerFragmentType = FIRST_SYSTEM_TYPE + 7

  This Fragment is a ContainerFragment and analysis code should unpack it.
- static constexpr type_t ErrorFragmentType = FIRST_SYSTEM_TYPE + 8

  This Fragment has experienced some error, and no attempt should be made to read it.
- static const version_t InvalidVersion = 0xFFFF

  The version field is currently 16-bits.
- static const version_t CurrentVersion = 0x2

  The CurrentVersion field should be incremented whenever the RawFragmentHeader changes.
- static const sequence_id_t InvalidSequenceID = 0xFFFFFFFFFFFF

  The sequence_id field is currently 48-bits.
- static const fragment_id_t InvalidFragmentID = 0xFFFF

  The fragment_id field is currently 16-bits.
- static const timestamp_t InvalidTimestamp = 0xFFFFFFFFFFFFFFFF

  The timestamp field is currently 64-bits.

6.23.1 Detailed Description

The RawFragmentHeader class contains the basic fields used by *artdaq* for routing Fragment objects through the system.

The RawFragmentHeader class contains the basic fields used by *artdaq* for routing Fragment objects through the system. It also contains static value definitions of values used in those fields.

Definition at line 31 of file RawFragmentHeader.hh.

6.23.2 Member Typedef Documentation

6.23.2.1 typedef unsigned long long artdaq::detail::RawFragmentHeader::RawDataType

The RawDataType (currently an unsigned long long) is the basic unit of data representation within *artdaq*

ELF, 7/30/2020: This typedef apparently cannot be changed without breaking compatibility with older data files. I have tried and failed to deal with such a change in classes_def.xml.

Definition at line 39 of file RawFragmentHeader.hh.

6.23.3 Member Function Documentation

6.23.3.1 struct timespec artdaq::detail::RawFragmentHeader::atime ( ) const

Get the last access time of this RawFragmentHeader.

Returns

struct timespec representing last access time of this RawFragmentHeader

Definition at line 250 of file RawFragmentHeader.hh.

6.23.3.2 struct timespec artdaq::detail::RawFragmentHeader::getLatency (bool *touch*)

Get the elapsed time between now and the last access time of the RawFragmentHeader, optionally resetting it.

Parameters

touch	Whether to also update the access time to current time

Returns

struct timespec representing interval between now and last access time of this RawFragmentHeader

Definition at line 258 of file RawFragmentHeader.hh.

6.23.3.3 static std::map< type_t, std::string > artdaq::detail::RawFragmentHeader::MakeSystemTypeMap ( ) [inline], [static]

Returns a map of the most-commonly used system types.

Returns

A map of the system types used in the *artdaq* data stream

Definition at line 70 of file RawFragmentHeader.hh.

6.23.3.4 static std::map< type_t, std::string > artdaq::detail::RawFragmentHeader::MakeVerboseSystemTypeMap ( ) [inline], [static]

Returns a map of all system types.

Returns

A map of all defined system types

Definition at line 84 of file RawFragmentHeader.hh.

6.23.3.5 constexpr std::size_t artdaq::detail::RawFragmentHeader::num_words ( ) [inline], [static]

Returns the number of RawDataType words present in the header.

Returns

The number of RawDataType words present in the header

Definition at line 205 of file RawFragmentHeader.hh.

6.23.3.6 bool artdaq::detail::RawFragmentHeader::operator== (const detail::RawFragmentHeader & *other*) const [inline]

Check if two RawFragmentHeader objects are equal.

Parameters

other	RawFragmentHeader to compare

Returns

Whether the two RawFragmentHeaders are identical

Definition at line 183 of file RawFragmentHeader.hh.

6.23.3.7 void artdaq::detail::RawFragmentHeader::setSystemType (uint8_t *stype*) [inline]

Sets the type field to the specified system type.

Parameters

stype	The type code to set

Exceptions

cet::exception	if stype is not in the allowed range for system types

Definition at line 232 of file RawFragmentHeader.hh.

6.23.3.8 void artdaq::detail::RawFragmentHeader::setUserType (uint8_t *utype*) [inline]

Sets the type field to the specified user type.

Parameters

utype	The type code to set

Exceptions

cet::exception	if utype is not in the allowed range for user types

Definition at line 219 of file RawFragmentHeader.hh.

6.23.3.9 static std::string artdaq::detail::RawFragmentHeader::SystemTypeToString (type_t *type*) [inline], [static]

Print a system type's string name.

Parameters

type	Type to print

Returns

String with "Name" of type

Definition at line 103 of file RawFragmentHeader.hh.

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

- artdaq_core/artdaq-core/Data/detail/RawFragmentHeader.hh

6.24 artdaq::detail::RawFragmentHeaderV0 Struct Reference

The RawFragmentHeaderV0 class contains the basic fields used by *artdaq* for routing Fragment objects through the system.

```
#include <artdaq-core/Data/detail/RawFragmentHeaderV0.hh>
```

Public Types

- typedef uint64_t RawDataType

  The RawDataType (currently a 64-bit integer) is the basic unit of data representation within artdaq
- typedef uint16_t version_t

  version field is 16 bits
- typedef uint64_t sequence_id_t

  sequence_id field is 48 bits
- typedef uint8_t type_t

  type field is 8 bits
- typedef uint16_t fragment_id_t

  fragment_id field is 16 bits
- typedef uint8_t metadata_word_count_t

  metadata_word_count field is 8 bits
- typedef uint32_t timestamp_t

  timestamp field is 32 bits

Public Member Functions

- void setUserType (uint8_t utype)

  Sets the type field to the specified user type.
- void setSystemType (uint8_t stype)

  Sets the type field to the specified system type.
- RawFragmentHeader upgrade () const

  Upgrades the RawFragmentHeaderV0 to a RawFragmentHeader (Current version)

Static Public Member Functions

- static std::map< type_t,
  std::string > MakeSystemTypeMap ()

  Returns a map of the most-commonly used system types.
- static std::map< type_t,
  std::string > MakeVerboseSystemTypeMap ()

  Returns a map of all system types.
- static constexpr std::size_t num_words ()

  Returns the number of RawDataType words present in the header.

Public Attributes

- RawDataType word_count: 32

  number of RawDataType words in this Fragment
- RawDataType version: 16

  The version of the fragment. Currently always InvalidVersion.
- RawDataType type: 8

  The type of the fragment, either system or user-defined.
- RawDataType metadata_word_count: 8

  The number of RawDataType words in the user-defined metadata.
- RawDataType sequence_id: 48

  The 48-bit sequence_id uniquely identifies events within the artdaq system.
- RawDataType fragment_id: 16

  The fragment_id uniquely identifies a particular piece of hardware within the artdaq system.
- RawDataType timestamp: 32

  The 64-bit timestamp field is the output of a user-defined clock used for building time-correlated events.
- RawDataType unused1: 16

  Extra space.
- RawDataType unused2: 16

  Extra space.

Static Public Attributes

- static constexpr type_t INVALID_TYPE = 0

  Marks a Fragment as Invalid.
- static constexpr type_t FIRST_USER_TYPE = 1

  The first user-accessible type.
- static constexpr type_t LAST_USER_TYPE = 224

  The last user-accessible type (types above this number are system types.
- static constexpr type_t FIRST_SYSTEM_TYPE = 225

  The first system type.
- static constexpr type_t LAST_SYSTEM_TYPE = 255

  The last system type.
- static constexpr type_t InvalidFragmentType = INVALID_TYPE

  Marks a Fragment as Invalid.
- static constexpr type_t EndOfDataFragmentType = FIRST_SYSTEM_TYPE

  This Fragment indicates the end of data to art
- static constexpr type_t DataFragmentType = FIRST_SYSTEM_TYPE + 1

  This Fragment holds data. Used for RawEvent Fragments sent from the EventBuilder to the Aggregator.
- static constexpr type_t InitFragmentType = FIRST_SYSTEM_TYPE + 2

  This Fragment holds the necessary data for initializing art
- static constexpr type_t EndOfRunFragmentType = FIRST_SYSTEM_TYPE + 3

  This Fragment indicates the end of a run to art
- static constexpr type_t EndOfSubrunFragmentType = FIRST_SYSTEM_TYPE + 4

  This Fragment indicates the end of a subrun to art
- static constexpr type_t ShutdownFragmentType = FIRST_SYSTEM_TYPE + 5

  This Fragment indicates a system shutdown to art
- static constexpr type_t EmptyFragmentType = FIRST_SYSTEM_TYPE + 6

  This Fragment contains no data and serves as a placeholder for when no data from a FragmentGenerator is expected.
- static constexpr type_t ContainerFragmentType = FIRST_SYSTEM_TYPE + 7

  This Fragment is a ContainerFragment and analysis code should unpack it.
- static const version_t InvalidVersion = 0xFFFF

  The version field is currently 16-bits.
- static const version_t CurrentVersion = 0x0

  The CurrentVersion field should be incremented whenever the RawFragmentHeader changes.
- static const sequence_id_t InvalidSequenceID = 0xFFFFFFFFFFFF

  The sequence_id field is currently 48-bits.
- static const fragment_id_t InvalidFragmentID = 0xFFFF

  The fragment_id field is currently 16-bits.
- static const timestamp_t InvalidTimestamp = 0xFFFFFFFF

  The timestamp field is currently 32-bits.

6.24.1 Detailed Description

The RawFragmentHeaderV0 class contains the basic fields used by *artdaq* for routing Fragment objects through the system.

The RawFragmentHeaderV0 class contains the basic fields used by *artdaq* for routing Fragment objects through the system. It also contains static value definitions of values used in those fields. This is an old version of RawFragment-Header, provided for compatibility

Definition at line 33 of file RawFragmentHeaderV0.hh.

6.24.2 Member Function Documentation

6.24.2.1 static std::map<type_t, std::string> artdaq::detail::RawFragmentHeaderV0::MakeSystemTypeMap ( ) [inline], [static]

Returns a map of the most-commonly used system types.

Returns

A map of the system types used in the *artdaq* data stream

Definition at line 68 of file RawFragmentHeaderV0.hh.

6.24.2.2 static std::map<type_t, std::string> artdaq::detail::RawFragmentHeaderV0::MakeVerboseSystemTypeMap ( ) [inline], [static]

Returns a map of all system types.

Returns

A map of all defined system types

Definition at line 80 of file RawFragmentHeaderV0.hh.

6.24.2.3 constexpr std::size_t artdaq::detail::RawFragmentHeaderV0::num_words ( ) [inline], [static]

Returns the number of RawDataType words present in the header.

Returns

The number of RawDataType words present in the header

Definition at line 152 of file RawFragmentHeaderV0.hh.

6.24.2.4 void artdaq::detail::RawFragmentHeaderV0::setSystemType (uint8_t *stype*) [inline]

Sets the type field to the specified system type.

Parameters

stype	The type code to set

Exceptions

cet::exception	if stype is not in the allowed range for system types

Definition at line 179 of file RawFragmentHeaderV0.hh.

6.24.2.5 void artdaq::detail::RawFragmentHeaderV0::setUserType (uint8_t *utype*) [inline]

Sets the type field to the specified user type.

Parameters

utype	The type code to set

Exceptions

cet::exception	if utype is not in the allowed range for user types

Definition at line 166 of file RawFragmentHeaderV0.hh.

6.24.2.6 artdaq::detail::RawFragmentHeader artdaq::detail::RawFragmentHeaderV0::upgrade ( ) const [inline]

Upgrades the RawFragmentHeaderV0 to a RawFragmentHeader (Current version)

Returns

Current-version RawFragmentHeader

The constraints on RawFragmentHeader upgrades are that no field may shrink in size or be deleted. Therefore, there will always be an upgrade path from old RawFragmentHeaders to new ones. By convention, all fields are initialized to the Invalid defines, and then the old data (guarenteed to be smaller) is cast to the new header. In the case of added fields, they will remain marked Invalid.

Definition at line 191 of file RawFragmentHeaderV0.hh.

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

- artdaq_core/artdaq-core/Data/detail/RawFragmentHeaderV0.hh

6.25 artdaq::detail::RawFragmentHeaderV1 Struct Reference

The RawFragmentHeaderV1 class contains the basic fields used by *artdaq* for routing Fragment objects through the system.

`#include <artdaq-core/Data/detail/RawFragmentHeaderV1.hh>`

Public Types

- typedef uint64_t RawDataType

    The RawDataType (currently a 64-bit integer) is the basic unit of data representation within artdaq
- typedef uint16_t version_t

    version field is 16 bits
- typedef uint64_t sequence_id_t

    sequence_id field is 48 bits
- typedef uint8_t type_t

    type field is 8 bits
- typedef uint16_t fragment_id_t

    fragment_id field is 16 bits
- typedef uint8_t metadata_word_count_t

    metadata_word_count field is 8 bits
- typedef uint64_t timestamp_t

    timestamp field is 32 bits

Public Member Functions

- void setUserType (uint8_t utype)

  Sets the type field to the specified user type.
- void setSystemType (uint8_t stype)

  Sets the type field to the specified system type.
- RawFragmentHeader upgrade () const

  Upgrades the RawFragmentHeaderV1 to a RawFragmentHeader (Current version)

Static Public Member Functions

- static std::map< type_t,
  std::string > MakeSystemTypeMap ()

  Returns a map of the most-commonly used system types.
- static std::map< type_t,
  std::string > MakeVerboseSystemTypeMap ()

  Returns a map of all system types.
- static std::string SystemTypeToString (type_t type)

  Print a system type's string name.
- static constexpr std::size_t num_words ()

  Returns the number of RawDataType words present in the header.

Public Attributes

- RawDataType word_count: 32

  number of RawDataType words in this Fragment
- RawDataType version: 16

  The version of the fragment.
- RawDataType type: 8

  The type of the fragment, either system or user-defined.
- RawDataType metadata_word_count: 8

  The number of RawDataType words in the user-defined metadata.
- RawDataType sequence_id: 48

  The 48-bit sequence_id uniquely identifies events within the artdaq system.
- RawDataType fragment_id: 16

  The fragment_id uniquely identifies a particular piece of hardware within the artdaq system.
- RawDataType timestamp: 64

  The 64-bit timestamp field is the output of a user-defined clock used for building time-correlated events.

Static Public Attributes

- static constexpr type_t INVALID_TYPE = 0

  Marks a Fragment as Invalid.
- static constexpr type_t FIRST_USER_TYPE = 1

  The first user-accessible type.
- static constexpr type_t LAST_USER_TYPE = 224

  The last user-accessible type (types above this number are system types.

- static constexpr type_t FIRST_SYSTEM_TYPE = 225

  The first system type.
- static constexpr type_t LAST_SYSTEM_TYPE = 255

  The last system type.
- static constexpr type_t InvalidFragmentType = INVALID_TYPE

  Marks a Fragment as Invalid.
- static constexpr type_t EndOfDataFragmentType = FIRST_SYSTEM_TYPE

  This Fragment indicates the end of data to art
- static constexpr type_t DataFragmentType = FIRST_SYSTEM_TYPE + 1

  This Fragment holds data. Used for RawEvent Fragments sent from the EventBuilder to the Aggregator.
- static constexpr type_t InitFragmentType = FIRST_SYSTEM_TYPE + 2

  This Fragment holds the necessary data for initializing art
- static constexpr type_t EndOfRunFragmentType = FIRST_SYSTEM_TYPE + 3

  This Fragment indicates the end of a run to art
- static constexpr type_t EndOfSubrunFragmentType = FIRST_SYSTEM_TYPE + 4

  This Fragment indicates the end of a subrun to art
- static constexpr type_t ShutdownFragmentType = FIRST_SYSTEM_TYPE + 5

  This Fragment indicates a system shutdown to art
- static constexpr type_t EmptyFragmentType = FIRST_SYSTEM_TYPE + 6

  This Fragment contains no data and serves as a placeholder for when no data from a FragmentGenerator is expected.
- static constexpr type_t ContainerFragmentType = FIRST_SYSTEM_TYPE + 7

  This Fragment is a ContainerFragment and analysis code should unpack it.
- static const version_t InvalidVersion = 0xFFFF

  The version field is currently 16-bits.
- static const version_t CurrentVersion = 0x1

  The CurrentVersion field should be incremented whenever the RawFragmentHeaderV1 changes.
- static const sequence_id_t InvalidSequenceID = 0xFFFFFFFFFFFF

  The sequence_id field is currently 48-bits.
- static const fragment_id_t InvalidFragmentID = 0xFFFF

  The fragment_id field is currently 16-bits.
- static const timestamp_t InvalidTimestamp = 0xFFFFFFFFFFFFFFFF

  The timestamp field is currently 64-bits.

6.25.1 Detailed Description

The RawFragmentHeaderV1 class contains the basic fields used by *artdaq* for routing Fragment objects through the system.

The RawFragmentHeaderV1 class contains the basic fields used by *artdaq* for routing Fragment objects through the system. It also contains static value definitions of values used in those fields. This is an old version of RawFragment-Header, provided for compatibility

Definition at line 32 of file RawFragmentHeaderV1.hh.

6.25.2 Member Function Documentation

6.25.2.1 static std::map<type_t, std::string> artdaq::detail::RawFragmentHeaderV1::MakeSystemTypeMap ( ) [inline], [static]

Returns a map of the most-commonly used system types.

Returns

A map of the system types used in the *artdaq* data stream

Definition at line 67 of file RawFragmentHeaderV1.hh.

6.25.2.2 static std::map<type_t, std::string> artdaq::detail::RawFragmentHeaderV1::MakeVerboseSystemTypeMap ( ) [inline], [static]

Returns a map of all system types.

Returns

A map of all defined system types

Definition at line 79 of file RawFragmentHeaderV1.hh.

6.25.2.3 constexpr std::size_t artdaq::detail::RawFragmentHeaderV1::num_words ( ) [inline], [static]

Returns the number of RawDataType words present in the header.

Returns

The number of RawDataType words present in the header

Definition at line 184 of file RawFragmentHeaderV1.hh.

6.25.2.4 void artdaq::detail::RawFragmentHeaderV1::setSystemType (uint8_t *stype*) [inline]

Sets the type field to the specified system type.

Parameters

stype	The type code to set

Exceptions

cet::exception	if stype is not in the allowed range for system types

Definition at line 211 of file RawFragmentHeaderV1.hh.

6.25.2.5 void artdaq::detail::RawFragmentHeaderV1::setUserType (uint8_t *utype*) [inline]

Sets the type field to the specified user type.

Parameters

utype	The type code to set

Exceptions

cet::exception	if utype is not in the allowed range for user types

Definition at line 198 of file RawFragmentHeaderV1.hh.

6.25.2.6 static std::string artdaq::detail::RawFragmentHeaderV1::SystemTypeToString (type_t *type*) [inline], [static]

Print a system type's string name.

Parameters

type	Type to print

Returns

String with "Name" of type

Definition at line 97 of file RawFragmentHeaderV1.hh.

6.25.2.7 artdaq::detail::RawFragmentHeader artdaq::detail::RawFragmentHeaderV1::upgrade ( ) const [inline]

Upgrades the RawFragmentHeaderV1 to a RawFragmentHeader (Current version)

Returns

Current-version RawFragmentHeader

The constraints on RawFragmentHeader upgrades are that no field may shrink in size or be deleted. Therefore, there will always be an upgrade path from old RawFragmentHeaders to new ones. By convention, all fields are initialized to the Invalid defines, and then the old data (guarenteed to be smaller) is cast to the new header. In the case of added fields, they will remain marked Invalid.

Definition at line 223 of file RawFragmentHeaderV1.hh.

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

- artdaq_core/artdaq-core/Data/detail/RawFragmentHeaderV1.hh

6.26 artdaq::SharedMemoryEventReceiver Class Reference

SharedMemoryEventReceiver can receive events (as written by SharedMemoryEventManager) from Shared Memory.

`#include <artdaq-core/Core/SharedMemoryEventReceiver.hh>`

Public Member Functions

- SharedMemoryEventReceiver (uint32_t shm_key, uint32_t broadcast_shm_key)

    Connect to a Shared Memory segment using the given parameters.

- virtual ∼SharedMemoryEventReceiver ()=default

  SharedMemoryEventReceiver Destructor.
- bool ReadyForRead (bool broadcast=false, size_t timeout_us=1000000)

  Determine whether an event is available for reading.
- detail::RawEventHeader ∗ ReadHeader (bool &err)

  Get the Event header.
- std::set< Fragment::type_t > GetFragmentTypes (bool &err)

  Get a set of Fragment Types present in the event.
- std::unique_ptr< Fragments > GetFragmentsByType (bool &err, Fragment::type_t type)

  Get a pointer to the Fragments of a given type in the event.
- std::string toString ()

  Write out information about the Shared Memory to a string.
- void ReleaseBuffer ()

  Release the buffer currently being read to the Empty state.
- int GetRank ()

  Returns the Rank of the writing process.
- int GetMyId ()

  Returns the ID of the reading process.
- bool IsEndOfData ()

  Determine if the end of data has been reached (from data shared memory segment)
- int ReadReadyCount ()

  Get the count of available buffers, both broadcasts and data.
- size_t size ()

  Get the size of the data buffer.

6.26.1 Detailed Description

SharedMemoryEventReceiver can receive events (as written by SharedMemoryEventManager) from Shared Memory.

Definition at line 14 of file SharedMemoryEventReceiver.hh.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 artdaq::SharedMemoryEventReceiver::SharedMemoryEventReceiver (uint32_t *shm_key,* uint32_t *broadcast_shm_key*)

Connect to a Shared Memory segment using the given parameters.

Parameters

shm_key	Key of the Shared Memory segment
broadcast_shm_-key	Key of the broadcast Shared Memory segment

Definition at line 9 of file SharedMemoryEventReceiver.cc.

6.26.3 Member Function Documentation

6.26.3.1 std::unique_ptr< artdaq::Fragments > artdaq::SharedMemoryEventReceiver::GetFragmentsByType (bool & *err,* Fragment::type_t *type*)

Get a pointer to the Fragments of a given type in the event.

Parameters

err	Flag used to indicate if an error has occurred
type	Type of Fragments to get. (Use InvalidFragmentType to get all Fragments)

Returns

std::unique_ptr to a Fragments object containing returned Fragment objects

Definition at line 141 of file SharedMemoryEventReceiver.cc.

6.26.3.2 std::set< artdaq::Fragment::type_t > artdaq::SharedMemoryEventReceiver::GetFragmentTypes (bool & *err*)

Get a set of Fragment Types present in the event.

Parameters

err	Flag used to indicate if an error has occurred

Returns

std::set of Fragment::type_t of all Fragment types in the event

Definition at line 109 of file SharedMemoryEventReceiver.cc.

6.26.3.3 int artdaq::SharedMemoryEventReceiver::GetMyId ( ) `[inline]`

Returns the ID of the reading process.

Returns

The ID of the reading process

Definition at line 79 of file SharedMemoryEventReceiver.hh.

6.26.3.4 int artdaq::SharedMemoryEventReceiver::GetRank ( ) `[inline]`

Returns the Rank of the writing process.

Returns

The rank of the writer process

Definition at line 73 of file SharedMemoryEventReceiver.hh.

6.26.3.5 bool artdaq::SharedMemoryEventReceiver::IsEndOfData ( ) `[inline]`

Determine if the end of data has been reached (from data shared memory segment)

Returns

Whether the EndOfData flag has been raised by the data shared memory segment

Definition at line 85 of file SharedMemoryEventReceiver.hh.

6.26.3.6 artdaq::detail::RawEventHeader $*$ artdaq::SharedMemoryEventReceiver::ReadHeader (bool & *err*)

Get the Event header.

Parameters

err	Flag used to indicate if an error has occurred

Returns

Pointer to RawEventHeader from buffer

Definition at line 90 of file SharedMemoryEventReceiver.cc.

6.26.3.7 int artdaq::SharedMemoryEventReceiver::ReadReadyCount ( ) [inline]

Get the count of available buffers, both broadcasts and data.

Returns

The sum of the available data buffer count and the available broadcast buffer count

Definition at line 91 of file SharedMemoryEventReceiver.hh.

6.26.3.8 bool artdaq::SharedMemoryEventReceiver::ReadyForRead (bool *broadcast* = false, size_t *timeout_us* = 1000000)

Determine whether an event is available for reading.

Parameters

broadcast	(Default false) Whether to wait for a broadcast buffer only
timeout_us	(Default 1000000) Time to wait for buffer to become available.

Returns

Whether an event is available for reading

Definition at line 20 of file SharedMemoryEventReceiver.cc.

6.26.3.9 size_t artdaq::SharedMemoryEventReceiver::size ( ) [inline]

Get the size of the data buffer.

Returns

The size of the data buffer

Definition at line 97 of file SharedMemoryEventReceiver.hh.

6.26.3.10 std::string artdaq::SharedMemoryEventReceiver::toString ( )

Write out information about the Shared Memory to a string.

Returns

String containing information about the current Shared Memory buffers

Definition at line 210 of file SharedMemoryEventReceiver.cc.

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

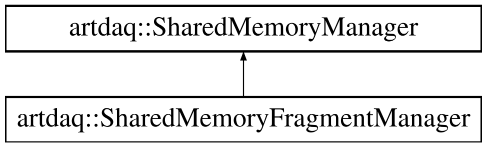
- artdaq_core/artdaq-core/Core/SharedMemoryEventReceiver.hh
- artdaq_core/artdaq-core/Core/SharedMemoryEventReceiver.cc

6.27 artdaq::SharedMemoryFragmentManager Class Reference

The SharedMemoryFragmentManager is a SharedMemoryManager that deals with Fragment transfers using a Shared-MemoryManager.

`#include <artdaq-core/Core/SharedMemoryFragmentManager.hh>`

Inheritance diagram for artdaq::SharedMemoryFragmentManager:


artdaq::SharedMemoryManager
artdaq::SharedMemoryFragmentManager

Public Member Functions

- SharedMemoryFragmentManager (uint32_t shm_key, size_t buffer_count=0, size_t max_buffer_size=0, size_t buffer_timeout_us=100 ∗1000000)

  SharedMemoryFragmentManager Constructor.
- virtual ∼SharedMemoryFragmentManager ()=default

  SharedMemoryFragmentManager destructor.
- SharedMemoryFragmentManager (SharedMemoryFragmentManager const &)=delete

  Copy Constructor is deleted.
- SharedMemoryFragmentManager (SharedMemoryFragmentManager &&)=delete

  Move Constructor is deleted.
- SharedMemoryFragmentManager & operator= (SharedMemoryFragmentManager const &)=delete

  Copy Assignment Operator is deleted.
- SharedMemoryFragmentManager & operator= (SharedMemoryFragmentManager &&)=delete

  Move Assignment Operator is deleted.
- int WriteFragment (Fragment &&fragment, bool overwrite, size_t timeout_us)

  Write a Fragment to the Shared Memory.
- int ReadFragment (Fragment &fragment)

  Read a Fragment from the Shared Memory.
- int ReadFragmentHeader (detail::RawFragmentHeader &header)

  Read a Fragment Header from the Shared Memory.
- int ReadFragmentData (RawDataType ∗destination, size_t words)

  Read Fragment Data from the Shared Memory.
- bool ReadyForWrite (bool overwrite) override

  Check if a buffer is ready for writing, and if so, reserves it for use.

Additional Inherited Members

6.27.1 Detailed Description

The SharedMemoryFragmentManager is a SharedMemoryManager that deals with Fragment transfers using a Shared-MemoryManager.

Definition at line 11 of file SharedMemoryFragmentManager.hh.

6.27.2 Constructor & Destructor Documentation

6.27.2.1 artdaq::SharedMemoryFragmentManager::SharedMemoryFragmentManager (uint32_t *shm_key*, size_t *buffer_count* = `0`, size_t *max_buffer_size* = `0`, size_t *buffer_timeout_us* = `100 * 1000000`)

SharedMemoryFragmentManager Constructor.

Parameters

shm_key	The key to use when attaching/creating the shared memory segment
buffer_count	The number of buffers in the shared memory
max_buffer_size	The size of each buffer
buffer_timeout_-us	The maximum amount of time a buffer may be locked before being returned to its previous state. This timer is reset upon any operation by the owning SharedMemoryManager.

Definition at line 6 of file SharedMemoryFragmentManager.cc.

6.27.3 Member Function Documentation

6.27.3.1 int artdaq::SharedMemoryFragmentManager::ReadFragment (Fragment & *fragment*)

Read a Fragment from the Shared Memory.

Parameters

fragment	Output Fragment object

Returns

0 on success

Definition at line 89 of file SharedMemoryFragmentManager.cc.

6.27.3.2 int artdaq::SharedMemoryFragmentManager::ReadFragmentData (RawDataType $*$ *destination*, size_t *words*)

Read Fragment Data from the Shared Memory.

Parameters

destination	Destination for data
words	RawDataType Word count to read

Returns

0 on success

Definition at line 136 of file SharedMemoryFragmentManager.cc.

6.27.3.3 int artdaq::SharedMemoryFragmentManager::ReadFragmentHeader (detail::RawFragmentHeader & *header*)

Read a Fragment Header from the Shared Memory.

Parameters

header	Output Fragment Header

Returns

0 on success

Definition at line 106 of file SharedMemoryFragmentManager.cc.

6.27.3.4 bool artdaq::SharedMemoryFragmentManager::ReadyForWrite (bool *overwrite*) `[override]`, `[virtual]`

Check if a buffer is ready for writing, and if so, reserves it for use.

Parameters

overwrite	Whether to overwrite Full buffers (non-reliable mode)

Returns

True if SharedMemoryFragmentManager is ready for Fragment data

Reimplemented from artdaq::SharedMemoryManager.

Definition at line 12 of file SharedMemoryFragmentManager.cc.

6.27.3.5 int artdaq::SharedMemoryFragmentManager::WriteFragment (Fragment && *fragment*, bool *overwrite*, size_t *timeout_us*)

Write a Fragment to the Shared Memory.

Parameters

fragment	Fragment to write
overwrite	Whether to set the overwrite flag
timeout_us	Time to wait for shared memory to be free (0: No timeout) (Timeout does not apply if overwrite == false)

Returns

0 on success

Definition at line 24 of file SharedMemoryFragmentManager.cc.

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

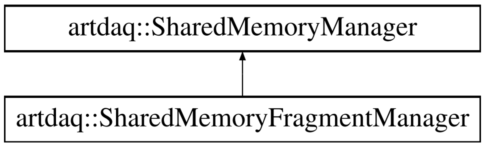
- artdaq_core/artdaq-core/Core/SharedMemoryFragmentManager.hh
- artdaq_core/artdaq-core/Core/SharedMemoryFragmentManager.cc

6.28 artdaq::SharedMemoryManager Class Reference

The SharedMemoryManager creates a Shared Memory area which is divided into a number of fixed-size buffers. It provides for multiple readers and multiple writers through a dual semaphore system.

`#include <artdaq-core/Core/SharedMemoryManager.hh>`

Inheritance diagram for artdaq::SharedMemoryManager:


artdaq::SharedMemoryManager
artdaq::SharedMemoryFragmentManager

Public Types

- enum BufferSemaphoreFlags { BufferSemaphoreFlags::Empty, BufferSemaphoreFlags::Writing, BufferSemaphoreFlags::Full, BufferSemaphoreFlags::Reading }

  The BufferSemaphoreFlags enumeration represents the different possible "states" of a given shared memory buffer.

Public Member Functions

- SharedMemoryManager (uint32_t shm_key, size_t buffer_count=0, size_t buffer_size=0, uint64_t buffer_timeout_us=100 $*$1000000, bool destructive_read_mode=true)

  SharedMemoryManager Constructor.
- virtual ∼SharedMemoryManager () noexcept

  SharedMemoryManager Destructor.
- bool Attach (size_t timeout_usec=0)

  Reconnect to the shared memory segment.
- int GetBufferForReading ()

  Finds a buffer that is ready to be read, and reserves it for the calling manager.
- int GetBufferForWriting (bool overwrite)

  Finds a buffer that is ready to be written to, and reserves it for the calling manager.
- bool ReadyForRead ()

  Whether any buffer is ready for read.
- virtual bool ReadyForWrite (bool overwrite)

  Whether any buffer is available for write.
- size_t ReadReadyCount ()

  Count the number of buffers that are ready for reading.
- size_t WriteReadyCount (bool overwrite)

  Count the number of buffers that are ready for writing.
- std::deque< int > GetBuffersOwnedByManager (bool locked=true)

  Get the list of all buffers currently owned by this manager instance.
- size_t BufferDataSize (int buffer)

  Get the current size of the buffer's data.
- size_t BufferSize ()

  Get the size of of a single buffer.

- void ResetReadPos (int buffer)

    Set the read position of the given buffer to the beginning of the buffer.
- void ResetWritePos (int buffer)

    Set the write position of the given buffer to the beginning of the buffer.
- void IncrementReadPos (int buffer, size_t read)

    Increment the read position for a given buffer.
- bool IncrementWritePos (int buffer, size_t written)

    Increment the write position for a given buffer.
- bool MoreDataInBuffer (int buffer)

    Determine if more data is available to be read, based on the read position and data size.
- bool CheckBuffer (int buffer, BufferSemaphoreFlags flags)

    Check both semaphore conditions (Mode flag and manager ID) for a given buffer.
- void MarkBufferFull (int buffer, int destination=-1)

    Release a buffer from a writer, marking it Full and ready for a reader.
- void MarkBufferEmpty (int buffer, bool force=false, bool detachOnException=true)

    Release a buffer from a reader, marking it Empty and ready to accept more data.
- bool ResetBuffer (int buffer)

    Resets the buffer from Reading to Full. This operation will only have an effect if performed by the owning manager or if the buffer has timed out.
- void GetNewId ()

    Assign a new ID to the current SharedMemoryManager, if one has not yet been assigned.
- uint16_t GetAttachedCount () const

    Get the number of attached SharedMemoryManagers.
- void ResetAttachedCount () const

    Reset the attached manager count to 0.
- int GetMyId () const

    Get the ID number of the current SharedMemoryManager.
- int GetRank () const

    Get the rank of the owner of the Shared Memory (artdaq assigns rank to each artdaq process for data flow)
- void SetRank (int rank)

    Set the rank stored in the Shared Memory, if the current instance is the owner of the shared memory.
- bool IsValid () const

    Is the shared memory pointer valid?
- bool IsEndOfData () const

    Determine whether the Shared Memory is marked for destruction (End of Data)
- size_t size () const

    Get the number of buffers in the shared memory segment.
- size_t Write (int buffer, void ∗data, size_t size)

    Write size bytes of data from the given pointer to a buffer.
- bool Read (int buffer, void ∗data, size_t size)

    Read size bytes of data from buffer into the given pointer.
- virtual std::string toString ()

    Write information about the SharedMemory to a string.
- uint32_t GetKey () const

    Get the key of the shared memory attached to this SharedMemoryManager.
- void ∗ GetReadPos (int buffer)

    Get a pointer to the current read position of the buffer.

- void ∗ GetWritePos (int buffer)

    Get a pointer to the current write position of the buffer.
- void ∗ GetBufferStart (int buffer)

    Get a pointer to the start position of the buffer.
- void Detach (bool throwException=false, const std::string &category="", const std::string &message="", bool force=false)

    Detach from the Shared Memory segment, optionally throwing a cet::exception with the specified properties.
- uint64_t GetBufferTimeout () const

    Gets the configured timeout for buffers to be declared "stale".
- size_t GetBufferCount () const

    Gets the number of buffers which have been processed through the Shared Memory.
- size_t GetLastSeenBufferID () const

    Gets the highest buffer number either written or read by this SharedMemoryManager.
- size_t GetLowestSeqIDRead () const

    Gets the lowest sequence ID that has been read by any reader, as reported by the readers.
- void SetMinWriteSize (size_t size)

    Sets the threshold after which a buffer should be considered "non-empty" (in case of default headers)
- std::vector< std::pair< int,
  BufferSemaphoreFlags > > GetBufferReport ()

    Get a report on the status of each buffer.
- void TouchBuffer (int buffer)

    Touch the given buffer (update its last_touch_time)

Static Public Member Functions

- static std::string FlagToString (BufferSemaphoreFlags flag)

    Convert a BufferSemaphoreFlags variable to its string represenatation.
- static uint64_t **GetAvailableRAM** ()
- static std::string **PrintBytes** (uint64_t bytes)

6.28.1 Detailed Description

The SharedMemoryManager creates a Shared Memory area which is divided into a number of fixed-size buffers. It provides for multiple readers and multiple writers through a dual semaphore system.

Definition at line 20 of file SharedMemoryManager.hh.

6.28.2 Member Enumeration Documentation

6.28.2.1 enum **artdaq::SharedMemoryManager::BufferSemaphoreFlags** `[strong]`

The BufferSemaphoreFlags enumeration represents the different possible "states" of a given shared memory buffer.

Enumerator

Empty	The buffer is empty, and waiting for a writer.
Writing	The buffer is currently being written to.
Full	The buffer is full, and waiting for a reader.
Reading	The buffer is currently being read from.

Definition at line 26 of file SharedMemoryManager.hh.

6.28.3 Constructor & Destructor Documentation

6.28.3.1 artdaq::SharedMemoryManager::SharedMemoryManager (uint32_t *shm_key,* size_t *buffer_count =* `0`*,* size_t *buffer_size =* `0`*,* uint64_t *buffer_timeout_us =* `100 * 1000000`*,* bool *destructive_read_mode =* `true`)

SharedMemoryManager Constructor.

Parameters

shm_key	The key to use when attaching/creating the shared memory segment
buffer_count	The number of buffers in the shared memory
buffer_size	The size of each buffer
buffer_timeout_-us	The maximum amount of time a buffer can be left untouched by its owner (if 0, buffers do not expire) before being returned to its previous state.
destructive_read-_mode	Whether a read operation empties the buffer (default: true, false for broadcast mode)

Definition at line 71 of file SharedMemoryManager.cc.

6.28.4 Member Function Documentation

6.28.4.1 size_t artdaq::SharedMemoryManager::BufferDataSize (int *buffer*)

Get the current size of the buffer's data.

Parameters

buffer	Buffer ID of buffer

Returns

Current size of data in the buffer, in bytes

Definition at line 778 of file SharedMemoryManager.cc.

6.28.4.2 size_t artdaq::SharedMemoryManager::BufferSize ( ) `[inline]`

Get the size of of a single buffer.

Returns

The configured size of a single buffer, in bytes

Definition at line 133 of file SharedMemoryManager.hh.

6.28.4.3 bool artdaq::SharedMemoryManager::CheckBuffer (int *buffer,* BufferSemaphoreFlags *flags*)

Check both semaphore conditions (Mode flag and manager ID) for a given buffer.

Parameters

buffer	Buffer ID of buffer
flags	Expected Mode flag

Returns

Whether the buffer passed the check

Definition at line 940 of file SharedMemoryManager.cc.

6.28.4.4 void artdaq::SharedMemoryManager::Detach (bool *throwException* = `false`, const std::string & *category* = `""`, const std::string & *message* = `""`, bool *force* = `false`)

Detach from the Shared Memory segment, optionally throwing a cet::exception with the specified properties.

Parameters

throwException	Whether to throw an exception after detaching
category	Category for the cet::exception
message	Message for the cet::exception
force	Whether to mark shared memory for destruction even if not owner (i.e. from signal handler)

Definition at line 1337 of file SharedMemoryManager.cc.

6.28.4.5 static std::string artdaq::SharedMemoryManager::FlagToString (BufferSemaphoreFlags *flag*) `[inline]`, `[static]`

Convert a BufferSemaphoreFlags variable to its string represenatation.

Parameters

flag	BufferSemaphoreFlags variable to convert

Returns

String representation of flag

Definition at line 39 of file SharedMemoryManager.hh.

6.28.4.6 uint16_t artdaq::SharedMemoryManager::GetAttachedCount ( ) const

Get the number of attached SharedMemoryManagers.

Returns

The number of attached SharedMemoryManagers

Definition at line 1126 of file SharedMemoryManager.cc.

6.28.4.7 size_t artdaq::SharedMemoryManager::GetBufferCount ( ) const `[inline]`

Gets the number of buffers which have been processed through the Shared Memory.

Returns

The number of buffers processed by the Shared Memory

Definition at line 329 of file SharedMemoryManager.hh.

6.28.4.8 int artdaq::SharedMemoryManager::GetBufferForReading ( )

Finds a buffer that is ready to be read, and reserves it for the calling manager.

Returns

The id number of the buffer. -1 indicates no buffers available for read.

Definition at line 298 of file SharedMemoryManager.cc.

6.28.4.9 int artdaq::SharedMemoryManager::GetBufferForWriting (bool *overwrite*)

Finds a buffer that is ready to be written to, and reserves it for the calling manager.

Parameters

overwrite	Whether to consider buffers that are in the Full and Reading state as ready for write (non-reliable mode)

Returns

The id number of the buffer. -1 indicates no buffers available for write.

Definition at line 425 of file SharedMemoryManager.cc.

6.28.4.10 std::vector< std::pair< int, artdaq::SharedMemoryManager::BufferSemaphoreFlags > > artdaq::SharedMemoryManager::GetBufferReport ( )

Get a report on the status of each buffer.

Returns

A list of manager_id, semaphore pairs

Definition at line 1281 of file SharedMemoryManager.cc.

6.28.4.11 std::deque< int > artdaq::SharedMemoryManager::GetBuffersOwnedByManager (bool *locked* = `true`)

Get the list of all buffers currently owned by this manager instance.

Parameters

locked	Default = true, Whether to lock search_mutex_ before checking buffer ownership (skipped in Detach)

Returns

A std::deque<int> of buffer IDs currently owned by this manager instance.

Definition at line 730 of file SharedMemoryManager.cc.

6.28.4.12 void ∗ artdaq::SharedMemoryManager::GetBufferStart (int *buffer*)

Get a pointer to the start position of the buffer.

Parameters

buffer	Buffer ID of buffer

Returns

void∗ pointer to buffer start position

Definition at line 1276 of file SharedMemoryManager.cc.

6.28.4.13 uint64_t artdaq::SharedMemoryManager::GetBufferTimeout ( ) const [inline]

Gets the configured timeout for buffers to be declared "stale".

Returns

The buffer timeout, in microseconds

Definition at line 323 of file SharedMemoryManager.hh.

6.28.4.14 uint32_t artdaq::SharedMemoryManager::GetKey ( ) const [inline]

Get the key of the shared memory attached to this SharedMemoryManager.

Returns

The shared memory key

Definition at line 287 of file SharedMemoryManager.hh.

6.28.4.15 size_t artdaq::SharedMemoryManager::GetLastSeenBufferID ( ) const [inline]

Gets the highest buffer number either written or read by this SharedMemoryManager.

Returns

The highest buffer id written or read by this SharedMemoryManager

Definition at line 335 of file SharedMemoryManager.hh.

6.28.4.16 int artdaq::SharedMemoryManager::GetMyId ( ) const [inline]

Get the ID number of the current SharedMemoryManager.

Returns

The ID number of the current SharedMemoryManager

Definition at line 225 of file SharedMemoryManager.hh.

6.28.4.17 int artdaq::SharedMemoryManager::GetRank ( ) const [inline]

Get the rank of the owner of the Shared Memory (artdaq assigns rank to each artdaq process for data flow)

Returns

The rank of the owner of the Shared Memory

Definition at line 231 of file SharedMemoryManager.hh.

6.28.4.18 void ∗ artdaq::SharedMemoryManager::GetReadPos (int *buffer*)

Get a pointer to the current read position of the buffer.

Parameters

buffer	Buffer ID of buffer

Returns

void∗ pointer to the buffer's current read position

Definition at line 1257 of file SharedMemoryManager.cc.

6.28.4.19 void ∗ artdaq::SharedMemoryManager::GetWritePos (int *buffer*)

Get a pointer to the current write position of the buffer.

Parameters

buffer	Buffer ID of buffer

Returns

void∗ pointer to buffer's current write position

Definition at line 1266 of file SharedMemoryManager.cc.

6.28.4.20 void artdaq::SharedMemoryManager::IncrementReadPos (int *buffer,* size_t *read*)

Increment the read position for a given buffer.

Parameters

buffer	Buffer ID of buffer
read	Number of bytes by which to increment read position

Definition at line 854 of file SharedMemoryManager.cc.

6.28.4.21 bool artdaq::SharedMemoryManager::IncrementWritePos (int *buffer,* size_t *written*)

Increment the write position for a given buffer.

Parameters

buffer	Buffer ID of buffer
written	Number of bytes by which to increment write position

Returns

Whether the write is allowed

Definition at line 882 of file SharedMemoryManager.cc.

6.28.4.22 bool artdaq::SharedMemoryManager::IsValid ( ) const [inline]

Is the shared memory pointer valid?

Returns

Whether the shared memory pointer is valid

Definition at line 246 of file SharedMemoryManager.hh.

6.28.4.23 void artdaq::SharedMemoryManager::MarkBufferEmpty (int *buffer*, bool *force* = `false`, bool *detachOnException* = `true`)

Release a buffer from a reader, marking it Empty and ready to accept more data.

Parameters

buffer	Buffer ID of buffer
force	Force buffer to empty state (only if manager_id_ == 0)
detachOn-Exception	Whether to throw exceptions when buffers are not in the expected state (default true)

Definition at line 983 of file SharedMemoryManager.cc.

6.28.4.24 void artdaq::SharedMemoryManager::MarkBufferFull (int *buffer*, int *destination* = `-1`)

Release a buffer from a writer, marking it Full and ready for a reader.

Parameters

buffer	Buffer ID of buffer
destination	If desired, a destination manager ID may be specified for a buffer

Definition at line 954 of file SharedMemoryManager.cc.

6.28.4.25 bool artdaq::SharedMemoryManager::MoreDataInBuffer (int *buffer*)

Determine if more data is available to be read, based on the read position and data size.

Parameters

buffer	Buffer ID of buffer

Returns

Whether more data is available in the given buffer.

Definition at line 918 of file SharedMemoryManager.cc.

6.28.4.26 bool artdaq::SharedMemoryManager::Read (int *buffer,* void ∗ *data,* size_t *size*)

Read size bytes of data from buffer into the given pointer.

Parameters

buffer	Buffer ID of buffer
data	Destination pointer for read
size	Size of read, in bytes

Returns

Whether the read was successful

Definition at line 1180 of file SharedMemoryManager.cc.

6.28.4.27 size_t artdaq::SharedMemoryManager::ReadReadyCount ( )

Count the number of buffers that are ready for reading.

Returns

The number of buffers ready for reading

Definition at line 579 of file SharedMemoryManager.cc.

6.28.4.28 bool artdaq::SharedMemoryManager::ReadyForRead ( )

Whether any buffer is ready for read.

Returns

True if there is a buffer available

Definition at line 651 of file SharedMemoryManager.cc.

6.28.4.29 bool artdaq::SharedMemoryManager::ReadyForWrite (bool *overwrite*) `[virtual]`

Whether any buffer is available for write.

Parameters

overwrite	Whether to allow overwriting full buffers

Returns

True if there is a buffer available

Reimplemented in artdaq::SharedMemoryFragmentManager.

Definition at line 694 of file SharedMemoryManager.cc.

6.28.4.30 bool artdaq::SharedMemoryManager::ResetBuffer (int *buffer*)

Resets the buffer from Reading to Full. This operation will only have an effect if performed by the owning manager or if the buffer has timed out.

Parameters

buffer	Buffer ID of buffer

Returns

Whether the buffer has exceeded the maximum age

Definition at line 1024 of file SharedMemoryManager.cc.

6.28.4.31 void artdaq::SharedMemoryManager::ResetReadPos (int *buffer*)

Set the read position of the given buffer to the beginning of the buffer.

Parameters

buffer	Buffer ID of buffer

Definition at line 803 of file SharedMemoryManager.cc.

6.28.4.32 void artdaq::SharedMemoryManager::ResetWritePos (int *buffer*)

Set the write position of the given buffer to the beginning of the buffer.

Parameters

buffer	Buffer ID of buffer

Definition at line 828 of file SharedMemoryManager.cc.

6.28.4.33 void artdaq::SharedMemoryManager::SetMinWriteSize (size_t *size*) [inline]

Sets the threshold after which a buffer should be considered "non-empty" (in case of default headers)

Parameters

size	Size (in bytes) after which a buffer will be considered non-empty

Definition at line 346 of file SharedMemoryManager.hh.

6.28.4.34 void artdaq::SharedMemoryManager::SetRank (int *rank*) [inline]

Set the rank stored in the Shared Memory, if the current instance is the owner of the shared memory.

Parameters

rank	Rank to set

Definition at line 237 of file SharedMemoryManager.hh.

6.28.4.35 size_t artdaq::SharedMemoryManager::size ( ) const [inline]

Get the number of buffers in the shared memory segment.

Returns

The number of buffers in the shared memory segment

Definition at line 257 of file SharedMemoryManager.hh.

6.28.4.36 std::string artdaq::SharedMemoryManager::toString ( ) [virtual]

Write information about the SharedMemory to a string.

Returns

String describing current state of SharedMemory and buffers

Definition at line 1216 of file SharedMemoryManager.cc.

6.28.4.37 size_t artdaq::SharedMemoryManager::Write (int *buffer,* void ∗ *data,* size_t *size*)

Write size bytes of data from the given pointer to a buffer.

Parameters

buffer	Buffer ID of buffer
data	Source pointer for write
size	Size of write, in bytes

Returns

Amount of data written, in bytes

Definition at line 1144 of file SharedMemoryManager.cc.

6.28.4.38 size_t artdaq::SharedMemoryManager::WriteReadyCount (bool *overwrite*)

Count the number of buffers that are ready for writing.

Parameters

overwrite	Whether to consider buffers that are in the Full and Reading state as ready for write (non-reliable mode)

Returns

The number of buffers ready for writing

Definition at line 617 of file SharedMemoryManager.cc.

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

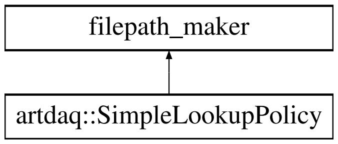
- artdaq_core/artdaq-core/Core/SharedMemoryManager.hh
- artdaq_core/artdaq-core/Core/SharedMemoryManager.cc

6.29 artdaq::SimpleLookupPolicy Class Reference

This class is intended to find files using a set lookup order.

`#include <artdaq-core/Utilities/SimpleLookupPolicy.hh>`

Inheritance diagram for artdaq::SimpleLookupPolicy:



Public Types

- enum ArgType : int { ArgType::ENV_VAR = 0, ArgType::PATH_STRING = 1 }

  Flag if the constructor argument is a list of paths or the name of an environment variable.

Public Member Functions

- SimpleLookupPolicy (std::string const &paths, ArgType argType=ArgType::ENV_VAR)

  Constructor.
- std::string operator() (std::string const &filename) override

  Perform the file lookup.
- virtual ~SimpleLookupPolicy () noexcept

  Default destructor.

6.29.1 Detailed Description

This class is intended to find files using a set lookup order.

This class is intended to find files using the following lookup order:

- the absolute path, if provided
- the current directory
- the specified list of paths

Definition at line 20 of file SimpleLookupPolicy.hh.

6.29.2 Member Enumeration Documentation

6.29.2.1 enum **artdaq::SimpleLookupPolicy::ArgType** : int [strong]

Flag if the constructor argument is a list of paths or the name of an environment variable.

Enumerator

ENV_VAR	Constructor argument is environment variable name.
PATH_STRING	Constructor argument is a list of directories.

Definition at line 26 of file SimpleLookupPolicy.hh.

6.29.3 Constructor & Destructor Documentation

6.29.3.1 artdaq::SimpleLookupPolicy::SimpleLookupPolicy (std::string const & *paths*, ArgType *argType* = ArgType::ENV_VAR)

Constructor.

Parameters

paths	Either the name of an environment variable or a list of directories to search
argType	Flag to determine if paths argument is an environment variable or a list of directories

The SimpleLookupPolicy Constructor instantiates the cet::search_path objects used for file lookup.

Definition at line 7 of file SimpleLookupPolicy.cc.

6.29.4 Member Function Documentation

6.29.4.1 std::string artdaq::SimpleLookupPolicy::operator() (std::string const & *filename*) [override]

Perform the file lookup.

Parameters

filename	The name of the file to find

Returns

The location that the file was found in.

The lookup proceeds in the following order:

- the absolute path, if provided

- the current directory
- the specified list of paths

Definition at line 43 of file SimpleLookupPolicy.cc.

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

- artdaq_core/artdaq-core/Utilities/SimpleLookupPolicy.hh
- artdaq_core/artdaq-core/Utilities/SimpleLookupPolicy.cc

6.30 artdaq::debug::StackTrace Class Reference

Represents the entire stack trace message.

`#include <artdaq-core/Utilities/ExceptionStackTrace.hh>`

Public Types

- using traces_t = std::vector< Trace >

    Trace collection type.

Public Member Functions

- StackTrace (std::string type_name)

    Constructor.
- std::string print () const

    Produces a stack trace summary.
- void resolve ()

    Reads and demangles backtrace symbols.
- StackTrace (StackTrace &&)=default

    Default Move Constructor.
- StackTrace & operator= (StackTrace &&)=default

    Default move assignment operator.
- StackTrace (const StackTrace &)=delete

    Copy Constructor is deleted.
- StackTrace & operator= (const StackTrace &)=delete

    Copy assignment operator is deleted.

Static Public Member Functions

- static std::string demangle (std::string const &symbol)

    Demangles backtrace symbols.

6.30.1 Detailed Description

Represents the entire stack trace message.

Definition at line 97 of file ExceptionStackTrace.hh.

6.30.2 Constructor & Destructor Documentation

6.30.2.1 artdaq::debug::StackTrace::StackTrace (std::string *type_name*) [explicit]

Constructor.

Parameters

type_name	The mangled type of the thrown exception

Definition at line 73 of file ExceptionStackTrace.cc.

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

- artdaq_core/artdaq-core/Utilities/ExceptionStackTrace.hh
- artdaq_core/artdaq-core/Utilities/ExceptionStackTrace.cc

6.31 artdaq::debug::StackTraceCollector Class Reference

Collects stack traces from different threads.

`#include <artdaq-core/Utilities/ExceptionStackTrace.hh>`

Public Types

- using stacktrace_map_t = std::unordered_map< std::thread::id, StackTrace >

  Map relating Thread IDs to their StackTraces.

Public Member Functions

- StackTraceCollector ()

  Constructor.
- StackTraceCollector (const StackTraceCollector &)=delete

  Copy Constructor is deleted.
- StackTraceCollector & operator= (const StackTraceCollector &)=delete

  Copy Assignment is deleted.
- StackTraceCollector (StackTraceCollector &&)=delete

  Move Constructor is deleted.
- StackTraceCollector & operator= (StackTraceCollector &&)=delete

  Move Assignment is deleted.
- template<typename... Args>
  void collect_stacktrace (Args &&...args)

  Adds a stacktrace to the stack_traces_ map.
- std::string print_stacktrace ()

  Produces a stack trace summary.

6.31.1 Detailed Description

Collects stack traces from different threads.

Definition at line 158 of file ExceptionStackTrace.hh.

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

- artdaq_core/artdaq-core/Utilities/ExceptionStackTrace.hh

6.32 artdaq::StatisticsCollection Class Reference

A collection of MonitoredQuantity instances describing low-level statistics of the *artdaq* system.

`#include <artdaq-core/Core/StatisticsCollection.hh>`

Public Member Functions

- virtual ∼StatisticsCollection () noexcept

    StatisticsCollection Destructor.
- void addMonitoredQuantity (const std::string &name, MonitoredQuantityPtr mqPtr)

    Registers a new MonitoredQuantity to be tracked by the StatisticsCollection.
- MonitoredQuantityPtr getMonitoredQuantity (const std::string &name) const

    Lookup and return a MonitoredQuantity from the StatisticsCollection.
- void reset ()

    Reset all MonitoredQuantity object in this StatisticsCollection.
- void requestStop ()

    Stops the statistics calculation thread.
- void run ()

    Start the background thread that performs MonitoredQuantity statistics calculation.

Static Public Member Functions

- static StatisticsCollection & getInstance ()

    Returns the singleton instance of the StatisticsCollection.

6.32.1 Detailed Description

A collection of MonitoredQuantity instances describing low-level statistics of the *artdaq* system.

A collection of MonitoredQuantity instances describing low-level statistics of the *artdaq* system. Periodically (default 1s) calculates statistics for each MonitoredQuantity instance.

Definition at line 22 of file StatisticsCollection.hh.

6.32.2 Member Function Documentation

6.32.2.1 void artdaq::StatisticsCollection::addMonitoredQuantity (const std::string & *name*, **MonitoredQuantityPtr** *mqPtr*)

Registers a new MonitoredQuantity to be tracked by the StatisticsCollection.

Parameters

name	Name of the MonitoredQuantity (used for lookup)
mqPtr	shared_ptr to MonitoredQuantity

Definition at line 54 of file StatisticsCollection.cc.

6.32.2.2 StatisticsCollection & artdaq::StatisticsCollection::getInstance ( ) [static]

Returns the singleton instance of the StatisticsCollection.

Returns

StatisticsCollection instance.

Definition at line 8 of file StatisticsCollection.cc.

6.32.2.3 MonitoredQuantityPtr artdaq::StatisticsCollection::getMonitoredQuantity (const std::string & *name*) const

Lookup and return a MonitoredQuantity from the StatisticsCollection.

Parameters

name	Name of the MonitoredQuantity

Returns

MonitoredQuantityPtr (nullptr if not found in StatisticsCollection)

Definition at line 62 of file StatisticsCollection.cc.

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

- artdaq_core/artdaq-core/Core/StatisticsCollection.hh
- artdaq_core/artdaq-core/Core/StatisticsCollection.cc

6.33 artdaq::debug::Trace Class Reference

Represents one line of the stack trace message.

`#include <artdaq-core/Utilities/ExceptionStackTrace.hh>`

Public Member Functions

- Trace (size_t index, std::string symbol)

  Constructor.
- Trace (Trace &&)=default

  Default Move Constructor.
- Trace (const Trace &)=delete

  Copy Constructor is deleted.
- Trace & operator= (const Trace &)=delete

Copy Assignment operator is deleted.

- Trace & operator= (Trace &&)=delete

  Move Assignment operator is deleted.

- std::string print () const

  Produces a one-line summary.

- void resolve ()

  Reads and demangles backtrace symbols.

6.33.1 Detailed Description

Represents one line of the stack trace message.

Definition at line 30 of file ExceptionStackTrace.hh.

6.33.2 Constructor & Destructor Documentation

6.33.2.1 artdaq::debug::Trace::Trace (size_t *index*, std::string *symbol*) [inline], [explicit]

Constructor.

Parameters

index	The position in a backtrace list
symbol	A backtrace symbol

Definition at line 38 of file ExceptionStackTrace.hh.

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

- artdaq_core/artdaq-core/Utilities/ExceptionStackTrace.hh
- artdaq_core/artdaq-core/Utilities/ExceptionStackTrace.cc

6.34 TraceLock< MUTEX > Class Template Reference

The TraceLock class allows a user to debug the acquisition and releasing of locks, by wrapping the unique_lock<std-::mutex> API with TRACE calls.

```
#include <artdaq-core/Utilities/TraceLock.hh>
```

Public Member Functions

- TraceLock (MUTEX &mutex, int level, std::string const &description)

  Construct a TraceLock.

- virtual ~TraceLock ()

  Release the TraceLock.

6.34.1 Detailed Description

template<typename MUTEX = std::mutex>class TraceLock< MUTEX >

The TraceLock class allows a user to debug the acquisition and releasing of locks, by wrapping the unique_lock<std::mutex> API with TRACE calls.

Definition at line 11 of file TraceLock.hh.

6.34.2 Constructor & Destructor Documentation

6.34.2.1 template<typename MUTEX = std::mutex> TraceLock< MUTEX >::TraceLock (MUTEX & *mutex*, int *level*, std::string const & *description*) [inline]

Construct a TraceLock.

Parameters

mutex	Mutex to hold lock on
level	Level to TRACE (in the TraceLock TRACE_NAME)
description	Description of lock (to be printed in TRACE calls)

Definition at line 20 of file TraceLock.hh.

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

- artdaq_core/artdaq-core/Utilities/TraceLock.hh

Index