

artdaq_core
v3_00_03

Generated by Doxygen 1.8.5

Thu Dec 14 2017 08:34:18

Contents

1	Todo List	1
2	Namespace Index	3
2.1	Namespace List	3
3	Hierarchical Index	5
3.1	Class Hierarchy	5
4	Class Index	7
4.1	Class List	7
5	Namespace Documentation	9
5.1	artdaq Namespace Reference	9
5.1.1	Detailed Description	11
5.1.2	Typedef Documentation	12
5.1.2.1	FragmentPtr	12
5.1.2.2	makeFunc_t	12
5.1.2.3	RawDataType	12
5.1.2.4	RawEvent_ptr	12
5.1.3	Enumeration Type Documentation	12
5.1.3.1	ExceptionHandlerRethrow	12
5.1.4	Function Documentation	13
5.1.4.1	configureMessageFacility	13
5.1.4.2	ExceptionHandler	14
5.1.4.3	fragmentSequenceIDCompare	14
5.1.4.4	generateMessageFacilityConfiguration	15
5.1.4.5	getGlobalQueue	15
5.1.4.6	makeFragmentGenerator	15
5.1.4.7	operator<<	16
5.1.4.8	operator<<	16

5.1.4.9	setMsgFacAppName	16
5.1.4.10	SimpleMemoryReaderApp	16
5.1.4.11	simpleQueueReaderApp	17
5.2	artdaq::detail Namespace Reference	17
5.2.1	Detailed Description	18
5.2.2	Typedef Documentation	18
5.2.2.1	seconds	18
5.2.3	Function Documentation	18
5.2.3.1	memoryUsage	18
5.2.3.2	memoryUsage	19
5.2.3.3	memoryUsage	19
5.3	artdaq::TimeUtils Namespace Reference	19
5.3.1	Detailed Description	20
5.3.2	Typedef Documentation	20
5.3.2.1	seconds	20
5.3.3	Function Documentation	20
5.3.3.1	convertUnixTimeToString	20
5.3.3.2	convertUnixTimeToString	21
5.3.3.3	convertUnixTimeToString	21
5.3.3.4	GetElapsedTime	21
5.3.3.5	GetElapsedTimeMicroseconds	22
5.3.3.6	GetElapsedTimeMilliseconds	23
5.3.3.7	gettimeofday_us	23
5.4	artdaqcore Namespace Reference	23
5.4.1	Detailed Description	24
6	Class Documentation	25
6.1	artdaq::ConcurrentQueue< T, EnqPolicy > Class Template Reference	25
6.1.1	Detailed Description	26
6.1.2	Constructor & Destructor Documentation	26
6.1.2.1	ConcurrentQueue	26
6.1.2.2	~ConcurrentQueue	26
6.1.3	Member Function Documentation	27
6.1.3.1	addExternallyDroppedEvents	27
6.1.3.2	capacity	27
6.1.3.3	clear	27
6.1.3.4	deqNowait	27

6.1.3.5	deqTimedWait	28
6.1.3.6	deqWait	29
6.1.3.7	empty	29
6.1.3.8	enqNowait	29
6.1.3.9	enqTimedWait	30
6.1.3.10	enqWait	30
6.1.3.11	full	30
6.1.3.12	getReadyTime	30
6.1.3.13	memory	31
6.1.3.14	queueReaderIsReady	31
6.1.3.15	setCapacity	31
6.1.3.16	setMemory	31
6.1.3.17	setReaderIsReady	32
6.1.3.18	size	32
6.1.3.19	used	32
6.2	artdaq::ContainerFragment Class Reference	32
6.2.1	Detailed Description	34
6.2.2	Constructor & Destructor Documentation	34
6.2.2.1	ContainerFragment	34
6.2.3	Member Function Documentation	34
6.2.3.1	at	34
6.2.3.2	block_count	34
6.2.3.3	dataBegin	34
6.2.3.4	dataEnd	35
6.2.3.5	fragment_type	35
6.2.3.6	fragmentIndex	35
6.2.3.7	fragSize	35
6.2.3.8	lastFragmentIndex	36
6.2.3.9	metadata	36
6.2.3.10	missing_data	36
6.2.3.11	operator[]	36
6.2.3.12	words_per_frag_word_	37
6.3	artdaq::ContainerFragmentLoader Class Reference	37
6.3.1	Detailed Description	38
6.3.2	Constructor & Destructor Documentation	38
6.3.2.1	ContainerFragmentLoader	38
6.3.3	Member Function Documentation	38

6.3.3.1	addFragment	38
6.3.3.2	addFragment	38
6.3.3.3	addFragments	39
6.3.3.4	metadata	40
6.3.3.5	set_fragment_type	40
6.3.3.6	set_missing_data	40
6.4	artdaq::FaillfFull< T > Struct Template Reference	40
6.4.1	Detailed Description	41
6.4.2	Member Function Documentation	41
6.4.2.1	doEnq	41
6.4.2.2	doInsert	42
6.5	artdaq::Fragment Class Reference	42
6.5.1	Detailed Description	47
6.5.2	Member Typedef Documentation	47
6.5.2.1	byte_t	47
6.5.3	Constructor & Destructor Documentation	48
6.5.3.1	Fragment	48
6.5.3.2	Fragment	48
6.5.3.3	Fragment	48
6.5.3.4	Fragment	48
6.5.3.5	Fragment	48
6.5.4	Member Function Documentation	49
6.5.4.1	dataAddress	49
6.5.4.2	dataBegin	49
6.5.4.3	dataBegin	49
6.5.4.4	dataBeginBytes	49
6.5.4.5	dataBeginBytes	50
6.5.4.6	dataEnd	50
6.5.4.7	dataEnd	50
6.5.4.8	dataEndBytes	50
6.5.4.9	dataEndBytes	50
6.5.4.10	dataFrag	51
6.5.4.11	dataFrag	51
6.5.4.12	dataSize	52
6.5.4.13	dataSizeBytes	52
6.5.4.14	empty	52
6.5.4.15	eodFrag	52

6.5.4.16	FragmentBytes	52
6.5.4.17	FragmentBytes	53
6.5.4.18	fragmentID	53
6.5.4.19	hasMetadata	53
6.5.4.20	headerAddress	54
6.5.4.21	headerBegin	54
6.5.4.22	headerBegin	54
6.5.4.23	headerBeginBytes	54
6.5.4.24	headerBeginBytes	54
6.5.4.25	isSystemFragmentType	54
6.5.4.26	isUserFragmentType	55
6.5.4.27	MakeSystemTypeMap	55
6.5.4.28	metadata	55
6.5.4.29	metadata	55
6.5.4.30	metadataAddress	56
6.5.4.31	operator=	56
6.5.4.32	operator=	56
6.5.4.33	print	56
6.5.4.34	reinterpret_cast_checked	57
6.5.4.35	reinterpret_cast_checked	57
6.5.4.36	reserve	58
6.5.4.37	resize	58
6.5.4.38	resize	58
6.5.4.39	resizeBytes	58
6.5.4.40	resizeBytes	58
6.5.4.41	sequenceID	59
6.5.4.42	setFragmentID	59
6.5.4.43	setMetadata	59
6.5.4.44	setSequenceID	59
6.5.4.45	setSystemType	60
6.5.4.46	setTimestamp	60
6.5.4.47	setUserType	60
6.5.4.48	size	60
6.5.4.49	sizeBytes	60
6.5.4.50	swap	61
6.5.4.51	swap	62
6.5.4.52	timestamp	62

6.5.4.53	type	62
6.5.4.54	typeString	62
6.5.4.55	updateMetadata	62
6.5.4.56	version	63
6.6	artdaq::FragmentGenerator Class Reference	63
6.6.1	Detailed Description	64
6.6.2	Member Function Documentation	64
6.6.2.1	fragmentIDs	64
6.6.2.2	getNext	64
6.7	artdaqtest::FragmentGeneratorTest Class Reference	64
6.7.1	Detailed Description	65
6.7.2	Member Function Documentation	65
6.7.2.1	fragmentIDs	65
6.7.2.2	getNext	65
6.8	artdaqcore::GetPackageBuildInfo Struct Reference	66
6.8.1	Detailed Description	66
6.8.2	Member Function Documentation	66
6.8.2.1	getPackageBuildInfo	66
6.9	artdaq::detail::hasMemoryUsed< T > Class Template Reference	66
6.9.1	Detailed Description	67
6.10	artdaq::KeepNewest< T > Struct Template Reference	67
6.10.1	Detailed Description	68
6.10.2	Member Function Documentation	69
6.10.2.1	doEnq	69
6.10.2.2	doInsert	69
6.11	artdaq::ContainerFragment::Metadata Struct Reference	70
6.11.1	Detailed Description	70
6.12	MetadataTypeHuge Struct Reference	70
6.12.1	Detailed Description	71
6.13	MetadataTypeOne Struct Reference	71
6.13.1	Detailed Description	71
6.13.2	Member Data Documentation	71
6.13.2.1	field1	71
6.13.2.2	field2	71
6.13.2.3	field3	71
6.14	MetadataTypeTwo Struct Reference	72
6.14.1	Detailed Description	72

6.14.2	Member Data Documentation	72
6.14.2.1	field1	72
6.14.2.2	field2	72
6.14.2.3	field3	72
6.14.2.4	field4	72
6.14.2.5	field5	73
6.15	artdaq::MonitoredQuantity Class Reference	73
6.15.1	Detailed Description	74
6.15.2	Constructor & Destructor Documentation	74
6.15.2.1	MonitoredQuantity	74
6.15.3	Member Function Documentation	74
6.15.3.1	addSample	74
6.15.3.2	addSample	75
6.15.3.3	addSample	75
6.15.3.4	addSample	75
6.15.3.5	calculateStatistics	75
6.15.3.6	disable	76
6.15.3.7	enable	76
6.15.3.8	ExpectedCalculationInterval	76
6.15.3.9	getCurrentTime	76
6.15.3.10	getStats	76
6.15.3.11	getTimeWindowForRecentResults	76
6.15.3.12	isEnabled	77
6.15.3.13	reset	77
6.15.3.14	setNewTimeWindowForRecentResults	77
6.15.3.15	waitUntilAccumulatorsHaveBeenFlushed	77
6.16	artdaq::MonitoredQuantityStats Struct Reference	78
6.16.1	Detailed Description	80
6.16.2	Member Enumeration Documentation	80
6.16.2.1	DataSetType	80
6.16.3	Member Function Documentation	80
6.16.3.1	getDuration	80
6.16.3.2	getLastSampleValue	81
6.16.3.3	getLastValueRate	81
6.16.3.4	getSampleCount	81
6.16.3.5	getSampleLatency	81
6.16.3.6	getSampleRate	81

6.16.3.7	getValueAverage	82
6.16.3.8	getValueMax	82
6.16.3.9	getValueMin	82
6.16.3.10	getValueRate	82
6.16.3.11	getValueRMS	83
6.16.3.12	getValueSum	83
6.16.3.13	isEnabled	83
6.17	artdaq::PackageBuildInfo Class Reference	84
6.17.1	Detailed Description	84
6.17.2	Member Function Documentation	84
6.17.2.1	getBuildTimestamp	84
6.17.2.2	getPackageName	84
6.17.2.3	getPackageVersion	85
6.17.2.4	setBuildTimestamp	85
6.17.2.5	setPackageName	85
6.17.2.6	setPackageVersion	85
6.18	artdaq::FailIfFull< T >::QueuelsFull Struct Reference	85
6.18.1	Detailed Description	86
6.18.2	Member Function Documentation	86
6.18.2.1	what	86
6.19	artdaq::QuickVec< TT_ > Struct Template Reference	86
6.19.1	Detailed Description	88
6.19.2	Constructor & Destructor Documentation	88
6.19.2.1	QuickVec	88
6.19.2.2	QuickVec	88
6.19.2.3	QuickVec	89
6.19.2.4	QuickVec	89
6.19.3	Member Function Documentation	89
6.19.3.1	begin	89
6.19.3.2	begin	89
6.19.3.3	capacity	89
6.19.3.4	Class_Version	90
6.19.3.5	end	90
6.19.3.6	end	90
6.19.3.7	erase	90
6.19.3.8	insert	91
6.19.3.9	insert	91

6.19.3.10 operator=	91
6.19.3.11 operator[]	92
6.19.3.12 operator[]	92
6.19.3.13 push_back	92
6.19.3.14 reserve	92
6.19.3.15 resize	93
6.19.3.16 resize	93
6.19.3.17 size	93
6.19.3.18 swap	93
6.20 artdaq::RawEvent Class Reference	94
6.20.1 Detailed Description	95
6.20.2 Constructor & Destructor Documentation	95
6.20.2.1 RawEvent	95
6.20.2.2 RawEvent	95
6.20.3 Member Function Documentation	95
6.20.3.1 fragmentTypes	95
6.20.3.2 insertFragment	95
6.20.3.3 isComplete	96
6.20.3.4 numFragments	96
6.20.3.5 print	96
6.20.3.6 releaseProduct	96
6.20.3.7 releaseProduct	96
6.20.3.8 runID	97
6.20.3.9 sequenceID	97
6.20.3.10 subrunID	97
6.20.3.11 wordCount	97
6.21 artdaq::detail::RawEventHeader Struct Reference	98
6.21.1 Detailed Description	98
6.21.2 Constructor & Destructor Documentation	99
6.21.2.1 RawEventHeader	99
6.22 artdaq::detail::RawFragmentHeader Struct Reference	99
6.22.1 Detailed Description	101
6.22.2 Member Function Documentation	101
6.22.2.1 MakeSystemTypeMap	101
6.22.2.2 MakeVerboseSystemTypeMap	102
6.22.2.3 num_words	102
6.22.2.4 setSystemType	102

6.22.2.5	setUserType	102
6.22.2.6	SystemTypeToString	102
6.23	artdaq::detail::RawFragmentHeaderV0 Struct Reference	103
6.23.1	Detailed Description	105
6.23.2	Member Function Documentation	105
6.23.2.1	MakeSystemTypeMap	105
6.23.2.2	MakeVerboseSystemTypeMap	106
6.23.2.3	num_words	106
6.23.2.4	setSystemType	106
6.23.2.5	setUserType	106
6.23.2.6	upgrade	106
6.24	artdaq::RejectNewest< T > Struct Template Reference	107
6.24.1	Detailed Description	107
6.24.2	Member Function Documentation	108
6.24.2.1	doEnq	108
6.24.2.2	doInsert	108
6.25	artdaq::SharedMemoryEventReceiver Class Reference	109
6.25.1	Detailed Description	109
6.25.2	Constructor & Destructor Documentation	109
6.25.2.1	SharedMemoryEventReceiver	109
6.25.3	Member Function Documentation	110
6.25.3.1	GetFragmentsByType	110
6.25.3.2	GetFragmentTypes	110
6.25.3.3	GetRank	110
6.25.3.4	ReadHeader	110
6.25.3.5	ReadReadyCount	111
6.25.3.6	ReadyForRead	111
6.25.3.7	size	111
6.25.3.8	toString	111
6.26	artdaq::SharedMemoryFragmentManager Class Reference	112
6.26.1	Detailed Description	112
6.26.2	Constructor & Destructor Documentation	113
6.26.2.1	SharedMemoryFragmentManager	113
6.26.3	Member Function Documentation	113
6.26.3.1	ReadFragment	113
6.26.3.2	ReadFragmentData	113
6.26.3.3	ReadFragmentHeader	113

6.26.3.4	WriteFragment	114
6.27	artdaq::SharedMemoryManager Class Reference	114
6.27.1	Detailed Description	117
6.27.2	Member Enumeration Documentation	117
6.27.2.1	BufferSemaphoreFlags	117
6.27.3	Constructor & Destructor Documentation	117
6.27.3.1	SharedMemoryManager	117
6.27.4	Member Function Documentation	117
6.27.4.1	BufferDataSize	117
6.27.4.2	CheckBuffer	118
6.27.4.3	Detach	118
6.27.4.4	FlagToString	118
6.27.4.5	GetAttachedCount	119
6.27.4.6	GetBufferCount	119
6.27.4.7	GetBufferForReading	119
6.27.4.8	GetBufferForWriting	119
6.27.4.9	GetBuffersOwnedByManager	119
6.27.4.10	GetBufferStart	120
6.27.4.11	GetBufferTimeout	121
6.27.4.12	GetKey	121
6.27.4.13	GetLastSeenBufferID	121
6.27.4.14	GetMyId	121
6.27.4.15	GetRank	122
6.27.4.16	GetReadPos	122
6.27.4.17	GetWritePos	122
6.27.4.18	IncrementReadPos	122
6.27.4.19	IncrementWritePos	122
6.27.4.20	IsValid	123
6.27.4.21	MarkBufferEmpty	123
6.27.4.22	MarkBufferFull	123
6.27.4.23	MoreDataInBuffer	123
6.27.4.24	Read	123
6.27.4.25	ReadReadyCount	124
6.27.4.26	ReadyForRead	124
6.27.4.27	ReadyForWrite	124
6.27.4.28	ResetBuffer	124
6.27.4.29	ResetReadPos	125

6.27.4.30	ResetWritePos	125
6.27.4.31	SetMinWriteSize	125
6.27.4.32	SetRank	125
6.27.4.33	size	125
6.27.4.34	toString	126
6.27.4.35	Write	126
6.27.4.36	WriteReadyCount	126
6.28	artdaq::SimpleLookupPolicy Class Reference	127
6.28.1	Detailed Description	127
6.28.2	Member Enumeration Documentation	127
6.28.2.1	ArgType	127
6.28.3	Constructor & Destructor Documentation	128
6.28.3.1	SimpleLookupPolicy	128
6.28.4	Member Function Documentation	128
6.28.4.1	operator()	128
6.29	artdaq::SimpleMemoryReader Class Reference	128
6.29.1	Detailed Description	129
6.29.2	Constructor & Destructor Documentation	129
6.29.2.1	SimpleMemoryReader	129
6.30	artdaq::SimpleQueueReader Class Reference	129
6.30.1	Detailed Description	130
6.30.2	Constructor & Destructor Documentation	130
6.30.2.1	SimpleQueueReader	130
6.31	artdaq::StatisticsCollection Class Reference	130
6.31.1	Detailed Description	131
6.31.2	Member Function Documentation	131
6.31.2.1	addMonitoredQuantity	131
6.31.2.2	getInstance	131
6.31.2.3	getMonitoredQuantity	131
6.32	TraceLock Class Reference	132
6.32.1	Detailed Description	132
6.32.2	Constructor & Destructor Documentation	132
6.32.2.1	TraceLock	132

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 &`)=`default`

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 &`)=`default`

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:

artdaq	The artdaq namespace	9
artdaq::detail	Artdaq implementation details namespace	17
artdaq::TimeUtils	Namespace to hold useful time-converting functions	19
artdaqcore	Namespace used to differentiate the artdaq_core version of GetPackageBuildInfo from other versions present in the system	23

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

artdaq::ConcurrentQueue< T, EnqPolicy >	25
artdaq::ContainerFragment	32
artdaq::ContainerFragmentLoader	37
std::exception	
artdaq::FailIfFull< T >::QueuesFull	85
artdaq::FailIfFull< T >	40
filepath_maker	
artdaq::SimpleLookupPolicy	127
artdaq::Fragment	42
artdaq::FragmentGenerator	63
artdaqtest::FragmentGeneratorTest	64
artdaqcore::GetPackageBuildInfo	66
artdaq::detail::hasMemoryUsed< T >	66
artdaq::KeepNewest< T >	67
artdaq::ContainerFragment::Metadata	70
MetadataTypeHuge	70
MetadataTypeOne	71
MetadataTypeTwo	72
artdaq::MonitoredQuantityStats	78
artdaq::MonitoredQuantity	73
artdaq::PackageBuildInfo	84
artdaq::QuickVec< TT_ >	86
artdaq::QuickVec< RawDataType >	86
artdaq::RawEvent	94
artdaq::detail::RawEventHeader	98
artdaq::detail::RawFragmentHeader	99
artdaq::detail::RawFragmentHeaderV0	103
artdaq::RejectNewest< T >	107
artdaq::SharedMemoryEventReceiver	109
artdaq::SharedMemoryManager	114
artdaq::SharedMemoryFragmentManager	112
artdaq::SimpleMemoryReader	128
artdaq::SimpleQueueReader	129

artdaq::StatisticsCollection	130
TraceLock	132

Chapter 4

Class Index

4.1 Class List

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

artdaq::ConcurrentQueue< T, EnqPolicy >	25
artdaq::ContainerFragment	
The artdaq::ContainerFragment class represents a Fragment which contains other Fragments	32
artdaq::ContainerFragmentLoader	
A Read-Write version of the ContainerFragment , used for filling ContainerFragment objects with other Fragment objects	37
artdaq::FaillfFull< T >	
ConcurrentQueue policy to throw an exception when the queue is full	40
artdaq::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	42
artdaq::FragmentGenerator	
Base class for all FragmentGenerators	63
artdaqtest::FragmentGeneratorTest	
Tests the functionality of the artdaq::FragmentGenerator class	64
artdaqcore::GetPackageBuildInfo	
Wrapper around the artdaqcore::GetPackageBuildInfo::getPackageBuildInfo function	66
artdaq::detail::hasMemoryUsed< T >	66
artdaq::KeepNewest< T >	
ConcurrentQueue policy to discard oldest elements when the queue is full	67
artdaq::ContainerFragment::Metadata	
Contains the information necessary for retrieving Fragment objects from the ContainerFragment	70
MetadataTypeHuge	
Test Metadata that is very large	70
MetadataTypeOne	
Test Metadata with three fields in two long words	71
MetadataTypeTwo	
Test Metadata with five fields, mixing field sizes	72
artdaq::MonitoredQuantity	
This class keeps track of statistics for a set of sample values and provides timing information on the samples	73

artdaq::MonitoredQuantityStats	
Struct containing MonitoredQuantity data	78
artdaq::PackageBuildInfo	
Class holding information about the <i>artdaq</i> package build	84
artdaq::FaillfFull< T >::QueuelsFull	
Exception thrown by FaillfFull policy when an enqueue operation is attempted on a full queue	85
artdaq::QuickVec< TT_ >	
A QuickVec behaves like a <code>std::vector</code> , but does no initialization of its data, making it faster at the cost of having to ensure that uninitialized data is not read	86
artdaq::RawEvent	
RawEvent is the <i>artdaq</i> view of a generic event, containing a header and zero or more Fragments	94
artdaq::detail::RawEventHeader	
The header information used to identify key properties of the RawEvent object	98
artdaq::detail::RawFragmentHeader	
Basic fields used by <i>artdaq</i> for routing Fragment objects through the system	99
artdaq::detail::RawFragmentHeaderV0	
Basic fields used by <i>artdaq</i> for routing Fragment objects through the system	103
artdaq::RejectNewest< T >	
ConcurrentQueue policy to discard new elements when the queue is full	107
artdaq::SharedMemoryEventReceiver	
SharedMemoryEventReceiver can receive events (as written by SharedMemoryEventManager) from Shared Memory	109
artdaq::SharedMemoryFragmentManager	
The SharedMemoryFragmentManager is a SharedMemoryManager that deals with Fragment transfers using a SharedMemoryManager	112
artdaq::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	114
artdaq::SimpleLookupPolicy	
This class is intended to find files using a set lookup order	127
artdaq::SimpleMemoryReader	
SimpleMemoryReader will continue to read RawEvent objects off the queue until it encounters a null pointer, at which point it stops	128
artdaq::SimpleQueueReader	
SimpleQueueReader will continue to read RawEvent objects off the queue until it encounters a null pointer, at which point it stops	129
artdaq::StatisticsCollection	
A collection of MonitoredQuantity instances describing low-level statistics of the <i>artdaq</i> system	130
TraceLock	
Allows a user to debug the acquisition and releasing of locks, by wrapping the <code>unique_lock<std::mutex></code> API with TRACE calls	132

Chapter 5

Namespace Documentation

5.1 artdaq Namespace Reference

The artdaq namespace.

Namespaces

- [detail](#)
artdaq implementation details namespace
- [TimeUtils](#)
Namespace to hold useful time-converting functions.

Classes

- struct [FaillfFull](#)
[ConcurrentQueue](#) policy to throw an exception when the queue is full.
- struct [KeepNewest](#)
[ConcurrentQueue](#) policy to discard oldest elements when the queue is full.
- struct [RejectNewest](#)
[ConcurrentQueue](#) policy to discard new elements when the queue is full.
- class [ConcurrentQueue](#)
- 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 [SimpleMemoryReader](#)

[SimpleMemoryReader](#) will continue to read [RawEvent](#) objects off the queue until it encounters a null pointer, at which point it stops.

- class [SimpleQueueReader](#)

[SimpleQueueReader](#) will continue to read [RawEvent](#) objects off the queue until it encounters a null pointer, at which point it stops.

- 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. Board-Readers 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 [FragmentGenerator](#)

Base class for all FragmentGenerators.

- class [SimpleLookupPolicy](#)

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

Typedefs

- typedef std::shared_ptr< [RawEvent](#) > [RawEvent_ptr](#)

A smart pointer to a [RawEvent](#) object.

- typedef
[artdaq::ConcurrentQueue](#)
< [RawEvent_ptr](#) > [RawEventQueue](#)

A [ConcurrentQueue](#) of [RawEvent](#) objects.

- 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::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

- [RawEventQueue](#) & [getGlobalQueue](#) ([RawEventQueue::SizeType](#) maxSize=std::numeric_limits< [RawEventQueue::SizeType](#) >::max())

The first thread to call [getGlobalQueue\(\)](#) causes the creation of the queue. The queue will be destroyed at static destruction time.

- int [SimpleMemoryReaderApp](#) (int argc, char **argv)

An application which pops items off a RawEventQueue using the [SimpleMemoryReader](#).

- int [simpleQueueReaderApp](#) (int argc, char **argv)

An application which pops items off a RawEventQueue using the [SimpleQueueReader](#).

- bool [fragmentSequenceIDCompare](#) ([Fragment](#) i, [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::unique_ptr< [FragmentGenerator](#) > [makeFragmentGenerator](#) (std::string const &generator_plugin_spec, fhicl::ParameterSet const &ps)

Instantiates the [FragmentGenerator](#) plugin with the given name, using the given ParameterSet.

- std::string [generateMessageFacilityConfiguration](#) (char const *progrname, bool useConsole=true, bool printDebug=false)

Create the MessageFacility configuration Fhicl string.

- void [configureMessageFacility](#) (char const *progrname, bool useConsole=true, bool printDebug=false)

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

- void [setMsgFacAppName](#) (const std::string &appType, unsigned short port)

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

- void [ExceptionHandler](#) ([ExceptionHandlerRethrow](#) decision, std::string optional_message="")

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

Variables

- static const int [CONTAINER_FRAGMENT_COUNT_MAX](#) = 100

The maximum capacity of the [ContainerFragment](#) (in fragments)

5.1.1 Detailed Description

The artdaq namespace.

5.1.2 Typedef Documentation

5.1.2.1 `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 53 of file `Fragment.hh`.

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

Constructs a [FragmentGenerator](#) instance, and returns a pointer to it.

Parameters

<i>ps</i>	Parameter set for initializing the FragmentGenerator
-----------	--

Returns

A smart pointer to the [FragmentGenerator](#)

Definition at line 16 of file `GeneratorMacros.hh`.

5.1.2.3 `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 39 of file `Fragment.hh`.

5.1.2.4 `typedef std::shared_ptr< RawEvent > artdaq::RawEvent_ptr`

A smart pointer to a [RawEvent](#) object.

A `shared_ptr` to a [RawEvent](#).

Definition at line 13 of file `GlobalQueue.hh`.

5.1.3 Enumeration Type Documentation

5.1.3.1 `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 11 of file `ExceptionHandler.hh`.

5.1.4 Function Documentation

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

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

Parameters

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

Definition at line 187 of file configureMessageFacility.cc.

5.1.4.2 void artdaq::ExceptionHandler (ExceptionHandlerRethrow *decision*, std::string *optional_message* = " ")

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

Parameters

<i>decision</i>	Controls whether the ExceptionHandler will rethrow (ExceptionHandlerRethrow::yes) or not (ExceptionHandlerRethrow::no)
<i>optional_message</i>	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 11 of file ExceptionHandler.cc.

5.1.4.3 bool artdaq::fragmentSequenceIDCompare (Fragment *i*, Fragment *j*)

Comparator for [Fragment](#) objects, based on their sequence_id.

Parameters

<i>i</i>	First Fragment to compare
----------	---

<i>j</i>	Second Fragment to compare
----------	--

Returns

`i.sequenceID() < j.sequenceID()`

Definition at line 8 of file `Fragment.cc`.

5.1.4.4 `std::string artdaq::generateMessageFacilityConfiguration (char const * progrname, bool useConsole = true, bool printDebug = false)`

Create the MessageFacility configuration Fhicl string.

Parameters

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

Returns

Fhicl string with generated MessageFacility configuration

Exceptions

<i>cet::exception</i>	if log path or ARTDAQ_LOG_FHICL do not exist
-----------------------	--

Definition at line 17 of file `configureMessageFacility.cc`.

5.1.4.5 `RawEventQueue & artdaq::getGlobalQueue (RawEventQueue::SizeType maxSize = std::numeric_limits< RawEventQueue::SizeType >::max())`

The first thread to call [getGlobalQueue\(\)](#) causes the creation of the queue. The queue will be destroyed at static destruction time.

Parameters

<i>maxSize</i>	Maximum number of elements in the queue
----------------	---

Returns

Reference to the global RawEventQueue

Definition at line 10 of file `GlobalQueue.cc`.

5.1.4.6 `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

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

Returns

A smart pointer to the [FragmentGenerator](#) instance

Definition at line 8 of file `makeFragmentGenerator.cc`.

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

Prints the given [Fragment](#) to the stream.

Parameters

<i>os</i>	Stream to print Fragment to
<i>f</i>	Fragment to print

Returns

Reference to the stream

Definition at line 1198 of file `Fragment.hh`.

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

Prints the [RawEvent](#) to the given stream.

Parameters

<i>os</i>	Stream to print RawEvent to
<i>ev</i>	RawEvent to print

Returns

Stream reference

Definition at line 288 of file `RawEvent.hh`.

5.1.4.9 `void artdaq::setMsgFacAppName (const std::string & appType, unsigned short port)`

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

Parameters

<i>appType</i>	Application name
<i>port</i>	XMLRPC port of this application instance

Definition at line 207 of file `configureMessageFacility.cc`.

5.1.4.10 `int artdaq::SimpleMemoryReaderApp (int argc, char ** argv)`

An application which pops items off a RawEventQueue using the [SimpleMemoryReader](#).

Parameters

<i>argc</i>	Number of arguments (OS-provided)
<i>argv</i>	Array of argument strings (OS-provided)

Returns

Status code: 0 success, 1 for any error

SimpleMemoryReaderApp is a function that can be used in place of artapp(), to read [RawEvent](#) objects from the shared [RawEvent](#) queue. Note that it ignores both of its arguments.

Definition at line 12 of file SimpleMemoryReader.cc.

5.1.4.11 int artdaq::simpleQueueReaderApp (int *argc*, char ** *argv*)

An application which pops items off a RawEventQueue using the [SimpleQueueReader](#).

Parameters

<i>argc</i>	Number of arguments (OS-provided)
<i>argv</i>	Array of argument strings (OS-provided)

Returns

Status code: 0 success, 1 for any error

simpleQueueReaderApp is a function that can be used in place of artapp(), to read [RawEvent](#) objects from the shared [RawEvent](#) queue. Note that it ignores both of its arguments.

Definition at line 12 of file SimpleQueueReader.cc.

5.2 artdaq::detail Namespace Reference

artdaq implementation details namespace

Classes

- class [hasMemoryUsed](#)
- 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 [RawEventHeader](#)

The header information used to identify key properties of the [RawEvent](#) object.

Typedefs

- typedef std::chrono::duration< double > [seconds](#)
- typedef size_t [MemoryType](#)

Basic unit of data storage and pointer types.

Functions

- `template<typename T >`
[MemoryType](#) [memoryUsage](#) (const std::pair< T, size_t > &t)
Returns the memory used by an object.
- `template<typename T >`
`std::enable_if< hasMemoryUsed`
`< T >::value, MemoryType >`
`::type memoryUsage (const T &t)`
Returns the memory used by an object. Uses [hasMemoryUsed](#) to determine if there is a `memoryUsed` function in object.
- `template<typename T >`
`std::enable_if<!hasMemoryUsed`
`< T >::value, MemoryType >`
`::type memoryUsage (const T &t)`
Returns the memory used by an object, as obtained through `sizeof`.

5.2.1 Detailed Description

artdaq implementation details namespace Class template [ConcurrentQueue](#) provides a FIFO that can be used to communicate data between multiple producer and consumer threads in an application.

The template policy `EnqPolicy` determines the behavior of the `enqNowait` function. In all cases, this function will return promptly (that is, it will not wait for a full queue to become not-full). However, what is done in the case of the queue being full depends on the policy chosen:

[FaillfFull](#): a `std::exception` is thrown if the queue is full.

[KeepNewest](#): the head of the FIFO is popped (and destroyed), and the new item is added to the FIFO. The function returns the number of popped (dropped) element.

[RejectNewest](#): the new item is not put onto the FIFO. The function returns the dropped event count (1) if the item cannot be added.

5.2.2 Typedef Documentation

5.2.2.1 `typedef std::chrono::duration<double> artdaq::detail::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 61 of file `ConcurrentQueue.hh`.

5.2.3 Function Documentation

5.2.3.1 `template<typename T > MemoryType artdaq::detail::memoryUsage (const std::pair< T, size_t > & t)`

Returns the memory used by an object.

Template Parameters

<i>T</i>	The type of the object
----------	------------------------

Parameters

<i>t</i>	A pair of object and size_t
----------	-----------------------------

Returns

If the object has a memoryUsed function, the result of that function. Otherwise 0.

Definition at line 110 of file ConcurrentQueue.hh.

5.2.3.2 `template<typename T > std::enable_if<hasMemoryUsed<T>::value, MemoryType>::type
artdaq::detail::memoryUsage (const T & t)`

Returns the memory used by an object. Uses [hasMemoryUsed](#) to determine if there is a memoryUsed function in object.

Template Parameters

<i>T</i>	Type of object t
----------	------------------

Parameters

<i>t</i>	Object to retrieve memory usage from
----------	--------------------------------------

Returns

Self-reported memory usage of object

Definition at line 129 of file ConcurrentQueue.hh.

5.2.3.3 `template<typename T > std::enable_if<!hasMemoryUsed<T>::value, MemoryType>::type
artdaq::detail::memoryUsage (const T & t)`

Returns the memory used by an object, as obtained through sizeof.

Template Parameters

<i>T</i>	Type of object t
----------	------------------

Parameters

<i>t</i>	Object to retrieve memory usage from
----------	--------------------------------------

Returns

sizeof(t)

Definition at line 148 of file ConcurrentQueue.hh.

5.3 artdaq::TimeUtils Namespace Reference

Namespace to hold useful time-converting functions.

Typedefs

- typedef std::chrono::duration< double, std::ratio< 1 > > [seconds](#)

Functions

- constexpr 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
- constexpr 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
- constexpr 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
- 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)

5.3.1 Detailed Description

Namespace to hold useful time-converting functions.

5.3.2 Typedef Documentation

5.3.2.1 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 24 of file TimeUtils.hh.

5.3.3 Function Documentation

5.3.3.1 std::string artdaq::TimeUtils::convertUnixTimeToString (time_t inputUnixTime)

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

Parameters

<i>inputUnixTime</i>	A time_t Unix time variable
----------------------	-----------------------------

Returns

std::string representation of Unix time, in UTC

Definition at line 7 of file TimeUtils.cc.

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

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

Parameters

<i>inputUnixTime</i>	A struct timeval Unix time variable
----------------------	-------------------------------------

Returns

std::string representation of Unix time, in UTC

Definition at line 18 of file TimeUtils.cc.

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

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

Parameters

<i>inputUnixTime</i>	A struct timespec Unix time variable
----------------------	--------------------------------------

Returns

std::string representation of Unix time, in UTC

Definition at line 35 of file TimeUtils.cc.

5.3.3.4 constexpr 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

<i>then</i>	std::chrono::steady_clock::time_point representing start of interval
<i>now</i>	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 32 of file TimeUtils.hh.

5.3.3.5 `constexpr 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

<i>then</i>	std::chrono::steady_clock::time_point representing start of interval
<i>now</i>	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 43 of file TimeUtils.hh.

5.3.3.6 `constexpr 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

<i>then</i>	std::chrono::steady_clock::time_point representing start of interval
<i>now</i>	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 54 of file TimeUtils.hh.

5.3.3.7 `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.

5.4 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.

5.4.1 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::ConcurrentQueue< T, EnqPolicy > Class Template Reference

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

Public Types

- typedef EnqPolicy::ValueType [ValueType](#)
Type of values stored in [ConcurrentQueue](#).
- typedef EnqPolicy::SequenceType [SequenceType](#)
Type of sequence used by [ConcurrentQueue](#).
- typedef SequenceType::size_type [SizeType](#)
Type for indexes in sequence.

Public Member Functions

- [ConcurrentQueue](#) ([SizeType](#) maxSize=std::numeric_limits< [SizeType](#) >::max(), [detail::MemoryType](#) maxMemory=std::numeric_limits< [detail::MemoryType](#) >::max())
[ConcurrentQueue](#) is always bounded. By default, the bound is absurdly large.
- [~ConcurrentQueue](#) ()
- EnqPolicy::ReturnType [enqNowait](#) (T const &item)
Add a copy if item to the queue, according to the rules determined by the EnqPolicy.
- void [enqWait](#) (T const &item)
Add a copy of item to the queue.
- bool [enqTimedWait](#) (T const &item, [detail::seconds](#) const &wait)
Add a copy of item to the queue, waiting for the queue to be non-full.
- bool [deqNowait](#) ([ValueType](#) &item)
Assign the value at the head of the queue to item and then remove the head of the queue.
- void [deqWait](#) ([ValueType](#) &item)
Assign the value of the head of the queue to item and then remove the head of the queue.
- bool [deqTimedWait](#) ([ValueType](#) &item, [detail::seconds](#) const &wait)
Assign the value at the head of the queue to item and then remove the head of the queue.
- bool [empty](#) () const

- `bool full () const`
- `SizeType size () const`
- `SizeType capacity () const`
Return the capacity of the queue, that is, the maximum number of items it can contain.
- `bool setCapacity (SizeType capacity)`
- `detail::MemoryType used () const`
Return the memory in bytes used by items in the queue.
- `detail::MemoryType memory () const`
Return the memory of the queue in bytes, that is, the maximum memory the items in the queue may occupy.
- `bool setMemory (detail::MemoryType maxMemory)`
Reset the memory usage in bytes of the queue. A value of 0 disabled the memory check. This can only be done if the queue is empty.
- `SizeType clear ()`
Remove all items from the queue. This changes the size to zero but does not change the capacity.
- `void addExternallyDroppedEvents (SizeType dropped)`
Adds the passed count to the counter of dropped events.
- `bool queueReaderIsReady () const`
Is the reader connected and ready for items to appear on the queue?
- `void setReaderIsReady (bool rdy=true)`
Set the ready flag for the reader.
- `std::chrono::steady_clock::time_point getReadyTime () const`
Gets the time at which the queue became ready.

6.1.1 Detailed Description

`template<class T, class EnqPolicy = FaillfFull<T>>class artdaq::ConcurrentQueue< T, EnqPolicy >`

`ConcurrentQueue<T>` class template declaration.

Definition at line 414 of file `ConcurrentQueue.hh`.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 `template<class T , class EnqPolicy > artdaq::ConcurrentQueue< T, EnqPolicy >::ConcurrentQueue (SizeType maxSize = std::numeric_limits<SizeType>::max(), detail::MemoryType maxMemory = std::numeric_limits<detail::MemoryType>::max()) [explicit]`

`ConcurrentQueue` is always bounded. By default, the bound is absurdly large.

Parameters

<i>maxSize</i>	Maximum size of the <code>ConcurrentQueue</code> (default: <code>SizeType::max</code>)
<i>maxMemory</i>	Maximum memory size of the <code>ConcurrentQueue</code> (default: <code>MemoryType::max</code>)

Definition at line 708 of file `ConcurrentQueue.hh`.

6.1.2.2 `template<class T , class EnqPolicy > artdaq::ConcurrentQueue< T, EnqPolicy >::~~ConcurrentQueue ()`

Applications should arrange to make sure that the destructor of a `ConcurrentQueue` is not called while some other thread is using that queue. There is some protection against doing this, but it seems impossible to make sufficient protection.

Definition at line 723 of file `ConcurrentQueue.hh`.

6.1.3 Member Function Documentation

6.1.3.1 `template<class T, class EnqPolicy > void artdaq::ConcurrentQueue< T, EnqPolicy >::addExternallyDroppedEvents (SizeType dropped)`

Adds the passed count to the counter of dropped events.

Parameters

<i>dropped</i>	Number of events dropped by code outside ConcurrentQueue
----------------	--

Definition at line 894 of file ConcurrentQueue.hh.

6.1.3.2 `template<class T, class EnqPolicy > ConcurrentQueue< T, EnqPolicy >::SizeType artdaq::ConcurrentQueue< T, EnqPolicy >::capacity () const`

Return the capacity of the queue, that is, the maximum number of items it can contain.

Returns

The maximum number of items the [ConcurrentQueue](#) can contain

Definition at line 837 of file ConcurrentQueue.hh.

6.1.3.3 `template<class T, class EnqPolicy > ConcurrentQueue< T, EnqPolicy >::SizeType artdaq::ConcurrentQueue< T, EnqPolicy >::clear ()`

Remove all items from the queue. This changes the size to zero but does not change the capacity.

Returns

The number of cleared events.

Definition at line 881 of file ConcurrentQueue.hh.

6.1.3.4 `template<class T, class EnqPolicy > bool artdaq::ConcurrentQueue< T, EnqPolicy >::deqNowait (ValueType & item)`

Assign the value at the head of the queue to item and then remove the head of the queue.

Parameters

<i>item</i>	Reference to output item
-------------	--------------------------

Returns

If the dequeue operation was successful

Assign the value at the head of the queue to item and then remove the head of the queue. If successful, return true; on failure, return false. This function will fail without waiting if the queue is empty. This function may throw any exception thrown by the assignment operator of type EnqPolicy::ValueType.

Definition at line 776 of file ConcurrentQueue.hh.

6.1.3.5 `template<class T, class EnqPolicy> bool artdaq::ConcurrentQueue< T, EnqPolicy>::deqTimedWait (ValueType & item, detail::seconds const & wait)`

Assign the value at the head of the queue to item and then remove the head of the queue.

Parameters

<i>item</i>	Reference to output item
<i>wait</i>	Maximum number of seconds to wait for dequeue

Returns

Whether an item was dequeued in the given interval

Assign the value at the head of the queue to item and then remove the head of the queue. If the queue is empty wait until it has become non-empty or until timeDuration has passed. Return true if an item has been removed from the queue or false if the timeout has expired. This may throw any exception thrown by the assignment operator of type EnqPolicy::ValueType.

Definition at line 796 of file ConcurrentQueue.hh.

6.1.3.6 `template<class T, class EnqPolicy > void artdaq::ConcurrentQueue< T, EnqPolicy >::deqWait (ValueType & item)`

Assign the value of the head of the queue to item and then remove the head of the queue.

Parameters

<i>item</i>	Reference to output item
-------------	--------------------------

Assign the value of the head of the queue to item and then remove the head of the queue. If the queue is empty wait until it has become non-empty. This may throw any exception thrown by the assignment operator of type EnqPolicy::ValueType.

Definition at line 786 of file ConcurrentQueue.hh.

6.1.3.7 `template<class T, class EnqPolicy > bool artdaq::ConcurrentQueue< T, EnqPolicy >::empty () const`

Return true if the queue is empty, and false if it is not.

Returns

True if the queue is empty

Definition at line 813 of file ConcurrentQueue.hh.

6.1.3.8 `template<class T, class EnqPolicy > EnqPolicy::ReturnType artdaq::ConcurrentQueue< T, EnqPolicy >::enqNowait (T const & item)`

Add a copy of item to the queue, according to the rules determined by the EnqPolicy.

Copying a [ConcurrentQueue](#) is illegal, as is assigning to a [ConcurrentQueue](#). The copy constructor and copy assignment operator are both private and deleted.

Parameters

<i>item</i>	Item to enqueue
-------------	-----------------

Returns

EnqPolicy::ReturnType result of enqueue operation

Add a copy of item to the queue, according to the rules determined by the EnqPolicy; see documentation above the the provided EnqPolicy choices. This may throw any exception thrown by the assignment operator of type T, or badAlloc.

Definition at line 736 of file ConcurrentQueue.hh.

6.1.3.9 `template<class T, class EnqPolicy > bool artdaq::ConcurrentQueue< T, EnqPolicy >::enqTimedWait (T const & item, detail::seconds const & wait)`

Add a copy of item to the queue, waiting for the queue to be non-full.

Parameters

<i>item</i>	Item to enqueue
<i>wait</i>	Maximum time (in seconds) to wait for queue to be non-full

Returns

Whether the item was added before the timeout expired

Add a copy of item to the queue. If the queue is full wait until it becomes non-full or until timeDuration has passed. Return true if the items has been put onto the queue or false if the timeout has expired. This may throw any exception thrown by the assignment operator of T, or badAlloc.

Definition at line 758 of file ConcurrentQueue.hh.

6.1.3.10 `template<class T, class EnqPolicy > void artdaq::ConcurrentQueue< T, EnqPolicy >::enqWait (T const & item)`

Add a copy of item to the queue.

Parameters

<i>item</i>	Item to enqueue
-------------	-----------------

Add a copy of item to the queue. If the queue is full wait until it becomes non-full. This may throw any exception thrown by the assignment operator of type T, or badAlloc.

Definition at line 747 of file ConcurrentQueue.hh.

6.1.3.11 `template<class T , class EnqPolicy > bool artdaq::ConcurrentQueue< T, EnqPolicy >::full () const`

Return true if the queue is full, and false if it is not.

Returns

True if the queue is full

Definition at line 821 of file ConcurrentQueue.hh.

6.1.3.12 `template<class T, class EnqPolicy = FaillfFull<T>> std::chrono::steady_clock::time_point artdaq::ConcurrentQueue< T, EnqPolicy >::getReadyTime () const [inline]`

Gets the time at which the queue became ready.

Returns

Time at which the queue became ready

Definition at line 618 of file ConcurrentQueue.hh.

6.1.3.13 `template<class T, class EnqPolicy > detail::MemoryType artdaq::ConcurrentQueue< T, EnqPolicy >::memory () const`

Return the memory of the queue in bytes, that is, the maximum memory the items in the queue may occupy.

Returns

The amount of memory allocated by the queue

Definition at line 863 of file ConcurrentQueue.hh.

6.1.3.14 `template<class T, class EnqPolicy = FailIfFull<T>> bool artdaq::ConcurrentQueue< T, EnqPolicy >::queueReaderIsReady () const [inline]`

Is the reader connected and ready for items to appear on the queue?

Returns

If the reader is connected and ready for items to be written to the queue.

Definition at line 599 of file ConcurrentQueue.hh.

6.1.3.15 `template<class T, class EnqPolicy > bool artdaq::ConcurrentQueue< T, EnqPolicy >::setCapacity (SizeType capacity)`

Reset the capacity of the queue. This can only be done if the queue is empty. This function returns false if the queue was not modified, and true if it was modified.

Parameters

<i>capacity</i>	The new capacity of the queue
-----------------	-------------------------------

Returns

True if the queue was modified

Definition at line 845 of file ConcurrentQueue.hh.

6.1.3.16 `template<class T, class EnqPolicy > bool artdaq::ConcurrentQueue< T, EnqPolicy >::setMemory (detail::MemoryType maxMemory)`

Reset the memory usage in bytes of the queue. A value of 0 disabled the memory check. This can only be done if the queue is empty.

Parameters

<i>maxMemory</i>	Sets the maximum amount of memory used by the queue
------------------	---

Returns

This function returns false if the queue was not modified, and true if it was modified.

Definition at line 871 of file ConcurrentQueue.hh.

```
6.1.3.17 template<class T, class EnqPolicy = FaillfFull<T>> void artdaq::ConcurrentQueue< T, EnqPolicy
>::setReaderIsReady( bool rdy = true ) [inline]
```

Set the ready flag for the reader.

Parameters

<i>rdy</i>	Value of the ready flag (default: true)
------------	---

Sets the ready flag for the reader, and the time that the reader became ready or unready. Used to help *artdaq* wait for *art* to finish initializing.

Definition at line 608 of file ConcurrentQueue.hh.

```
6.1.3.18 template<class T, class EnqPolicy > ConcurrentQueue< T, EnqPolicy >::SizeType artdaq::ConcurrentQueue<
T, EnqPolicy >::size( ) const
```

Return the size of the queue, that is, the number of items it contains.

Returns

The number of items in the queue

Definition at line 829 of file ConcurrentQueue.hh.

```
6.1.3.19 template<class T, class EnqPolicy > detail::MemoryType artdaq::ConcurrentQueue< T, EnqPolicy >::used( )
const
```

Return the memory in bytes used by items in the queue.

Returns

The amount of memory in use by queue items

Definition at line 855 of file ConcurrentQueue.hh.

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

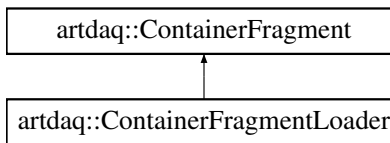
- `artdaq-core/artdaq-core/Core/ConcurrentQueue.hh`

6.2 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](#).

Public Member Functions

- [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 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](#).

6.2.1 Detailed Description

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

Definition at line 30 of file `ContainerFragment.hh`.

6.2.2 Constructor & Destructor Documentation

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

Parameters

<i>f</i>	The <code>Fragment</code> 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 61 of file `ContainerFragment.hh`.

6.2.3 Member Function Documentation

6.2.3.1 `FragmentPtr artdaq::ContainerFragment::at (size_t index) const` `[inline]`

Gets a specific `Fragment` from the `ContainerFragment`.

Parameters

<i>index</i>	The <code>Fragment</code> index to return
--------------	---

Returns

Pointer to the specified `Fragment` in the `ContainerFragment`

Exceptions

<code>cet::exception</code>	if the index is out-of-range
-----------------------------	------------------------------

Definition at line 109 of file `ContainerFragment.hh`.

6.2.3.2 `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 73 of file `ContainerFragment.hh`.

6.2.3.3 `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 89 of file `ContainerFragment.hh`.

6.2.3.4 `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 98 of file `ContainerFragment.hh`.

6.2.3.5 `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 78 of file `ContainerFragment.hh`.

6.2.3.6 `size_t artdaq::ContainerFragment::fragmentIndex (size_t index) const` `[inline]`

Get the offset of a [Fragment](#) within the [ContainerFragment](#).

Parameters

<i>index</i>	The Fragment index
--------------	------------------------------------

Returns

The offset of the requested [Fragment](#) within the [ContainerFragment](#)

Exceptions

<i>cet::exception</i>	if the index is out-of-range
-----------------------	------------------------------

Definition at line 154 of file `ContainerFragment.hh`.

6.2.3.7 `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

<i>index</i>	The Fragment index
--------------	------------------------------------

Returns

The size of the [Fragment](#) at the specified location in the [ContainerFragment](#), in bytes

Exceptions

<i>cet::exception</i>	if the index is out-of-range
-----------------------	------------------------------

Definition at line 126 of file ContainerFragment.hh.

6.2.3.8 `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 168 of file ContainerFragment.hh.

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

const getter function for the [Metadata](#)

Returns

const pointer to the [Metadata](#)

Definition at line 67 of file ContainerFragment.hh.

6.2.3.10 `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 83 of file ContainerFragment.hh.

6.2.3.11 `FragmentPtr artdaq::ContainerFragment::operator[] (size_t index) const` `[inline]`

Alias to [ContainerFragment::at\(\)](#)

Parameters

<i>index</i>	The Fragment index to return
--------------	--

Returns

Pointer to the specified [Fragment](#) in the [ContainerFragment](#)

Exceptions

<i>cet::exception</i>	if the index is out-of-range
-----------------------	------------------------------

Definition at line 143 of file ContainerFragment.hh.

6.2.3.12 `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 179 of file ContainerFragment.hh.

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

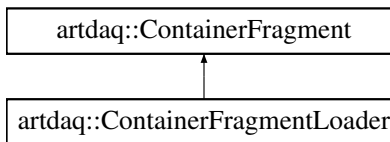
- artdaq-core/artdaq-core/Data/ContainerFragment.hh

6.3 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)
Add a [Fragment](#) to the [ContainerFragment](#) by reference.
- void [addFragment](#) ([artdaq::FragmentPtr](#) &frag)
Add a [Fragment](#) to the [ContainerFragment](#) by smart pointer.
- void [addFragments](#) ([artdaq::FragmentPtrs](#) &frags)
Add a collection of [Fragment](#) objects to the [ContainerFragment](#).

Additional Inherited Members

6.3.1 Detailed Description

A Read-Write version of the [ContainerFragment](#), used for filling [ContainerFragment](#) objects with other [Fragment](#) objects.
Definition at line 28 of file ContainerFragmentLoader.hh.

6.3.2 Constructor & Destructor Documentation

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

Constructs the [ContainerFragmentLoader](#).

Parameters

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

Exceptions

<i>cet::exception</i>	if the Fragment input has inconsistent Header information
-----------------------	---

Definition at line 105 of file ContainerFragmentLoader.hh.

6.3.3 Member Function Documentation

6.3.3.1 `void artdaq::ContainerFragmentLoader::addFragment (artdaq::Fragment & frag) [inline]`

Add a [Fragment](#) to the [ContainerFragment](#) by reference.

Parameters

<i>frag</i>	A Fragment object to be added to the ContainerFragment
-------------	--

Exceptions

<i>cet::exception</i>	If the Fragment to be added has a different type than expected
-----------------------	--

Definition at line 146 of file ContainerFragmentLoader.hh.

6.3.3.2 `void artdaq::ContainerFragmentLoader::addFragment (artdaq::FragmentPtr & frag) [inline]`

Add a [Fragment](#) to the [ContainerFragment](#) by smart pointer.

Parameters

<i>frag</i>	A FragmentPtr to a Fragment to be added to the ContainerFragment
-------------	--

Definition at line 173 of file ContainerFragmentLoader.hh.

6.3.3.3 void artdaq::ContainerFragmentLoader::addFragments (artdaq::FragmentPtrs & *frags*) [inline]

Add a collection of [Fragment](#) objects to the [ContainerFragment](#).

Parameters

<i>frags</i>	An artdaq::FragmentPtrs object containing Fragments to be added to the ContainerFragment
--------------	--

Definition at line 178 of file ContainerFragmentLoader.hh.

6.3.3.4 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 46 of file ContainerFragmentLoader.hh.

6.3.3.5 void artdaq::ContainerFragmentLoader::set_fragment_type ([Fragment::type_t](#) type) [inline]

Sets the type of [Fragment](#) which this [ContainerFragment](#) should contain.

Parameters

<i>type</i>	The Fragment::type_t identifying the type of Fragment objects stored in this ContainerFragment
-------------	--

Definition at line 56 of file ContainerFragmentLoader.hh.

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

Sets the `missing_data` flag.

Parameters

<i>isDataMissing</i>	The value of the <code>missing_data</code> 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 69 of file ContainerFragmentLoader.hh.

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

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

6.4 artdaq::FailIfFull< T > Struct Template Reference

[ConcurrentQueue](#) policy to throw an exception when the queue is full.

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

Classes

- struct [QueuesFull](#)

Exception thrown by [FailIfFull](#) policy when an enqueue operation is attempted on a full queue.

Public Types

- typedef bool [ReturnType](#)
Type returned by doEnq.
- typedef T [ValueType](#)
Type of values stored in queue.
- typedef std::list< T > [SequenceType](#)
Type of sequences of values.
- typedef SequenceType::size_type [SizeType](#)
Size type of sequences.

Static Public Member Functions

- static void [doInsert](#) (T const &item, [SequenceType](#) &elements, [SizeType](#) &size, [detail::MemoryType](#) const &item-Size, [detail::MemoryType](#) &used, std::condition_variable &nonempty)
Inserts element into the [ConcurrentQueue](#).
- static [ReturnType](#) [doEnq](#) (T const &item, [SequenceType](#) &elements, [SizeType](#) &size, [SizeType](#) &capacity, [detail::MemoryType](#) &used, [detail::MemoryType](#) &memory, size_t &elementsDropped, std::condition_variable &nonempty)
Attempts to enqueue an item.

Static Public Attributes

- [artdaq::FaillfFull::QueuelsFull](#) [queuelsFull](#) {}
Instance of [QueuelsFull](#) exception.

6.4.1 Detailed Description

template<class T>struct artdaq::FaillfFull< T >

[ConcurrentQueue](#) policy to throw an exception when the queue is full.

Template Parameters

<i>T</i>	Type of element to store in queue
----------	-----------------------------------

Definition at line 160 of file ConcurrentQueue.hh.

6.4.2 Member Function Documentation

- 6.4.2.1 template<class T > static [ReturnType](#) artdaq::FaillfFull< T >::doEnq (T const & *item*, [SequenceType](#) & *elements*, [SizeType](#) & *size*, [SizeType](#) & *capacity*, [detail::MemoryType](#) & *used*, [detail::MemoryType](#) & *memory*, size_t & *elementsDropped*, std::condition_variable & *nonempty*) [\[inline\]](#), [\[static\]](#)

Attempts to enqueue an item.

Parameters

<i>item</i>	Item to enqueue
<i>elements</i>	The ConcurrentQueue data
<i>size</i>	Number of elements in the ConcurrentQueue
<i>capacity</i>	Maximum number of elements in the ConcurrentQueue
<i>used</i>	Memory used by the ConcurrentQueue elements
<i>memory</i>	Amount of memory available for use by the ConcurrentQueue
<i>elements-Dropped</i>	Number of failed insert operations
<i>nonempty</i>	Condition variable to notify readers of new data on the queue

Returns

Whether the enqueue operation succeeded

Exceptions

<i>queueIsFull</i>	if the queue is full
--------------------	----------------------

Definition at line 222 of file [ConcurrentQueue.hh](#).

```
6.4.2.2 template<class T > static void artdaq::FailIfFull< T >::doInsert ( T const & item, SequenceType & elements,
    SizeType & size, detail::MemoryType const & itemSize, detail::MemoryType & used, std::condition_variable &
    nonempty ) [inline], [static]
```

Inserts element into the [ConcurrentQueue](#).

Parameters

<i>item</i>	Item to insert
<i>elements</i>	The ConcurrentQueue data
<i>size</i>	Number of elements in the ConcurrentQueue
<i>itemSize</i>	Size of the newly-inserted element
<i>used</i>	Memory used by the ConcurrentQueue elements
<i>nonempty</i>	Condition variable to notify readers of new data on queue

Definition at line 193 of file [ConcurrentQueue.hh](#).

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

- [artdaq-core/artdaq-core/Core/ConcurrentQueue.hh](#)

6.5 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
Default copy constructor.
- [Fragment](#) ([Fragment](#) &&) noexcept
Move Constructor.
- [Fragment](#) & operator= (const [Fragment](#) &)=default
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*.

- `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 &md)`

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 &md)`

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 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.
- `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.

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 nFragmentsToExpect)`
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)`
Creates 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.

6.5.1 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 84 of file Fragment.hh.

6.5.2 Member Typedef Documentation

6.5.2.1 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 99 of file Fragment.hh.

6.5.3 Constructor & Destructor Documentation

6.5.3.1 `artdaq::Fragment::Fragment (const Fragment &) [default]`

Default copy constructor.

Todo Decide if Copy constructor should be declared `=delete`

6.5.3.2 `artdaq::Fragment::Fragment (Fragment &&) [inline],[default],[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

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

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

Parameters

<code>n</code>	The initial size of the <code>Fragment</code> , in <code>RawDataType</code> words
----------------	---

Definition at line 22 of file `Fragment.cc`.

6.5.3.4 `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

<code>T</code>	Metadata type
----------------	---------------

Parameters

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

Definition at line 767 of file `Fragment.hh`.

6.5.3.5 `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

<i>sequenceID</i>	Sequence ID of Fragment
<i>fragID</i>	Fragment ID of Fragment
<i>type</i>	Type of Fragment
<i>timestamp</i>	Timestamp of Fragment

Definition at line 39 of file `Fragment.cc`.

6.5.4 Member Function Documentation

6.5.4.1 `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 1101 of file `Fragment.hh`.

6.5.4.2 `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 1025 of file `Fragment.hh`.

6.5.4.3 `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 1047 of file `Fragment.hh`.

6.5.4.4 `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 522 of file `Fragment.hh`.

6.5.4.5 `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 567 of file `Fragment.hh`.

6.5.4.6 `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 1033 of file `Fragment.hh`.

6.5.4.7 `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 1055 of file `Fragment.hh`.

6.5.4.8 `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 532 of file `Fragment.hh`.

6.5.4.9 `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 580 of file Fragment.hh.

6.5.4.10 `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

<i>InputIterator</i>	Type of input iterator
----------------------	------------------------

Parameters

<i>sequenceID</i>	Sequence ID of new Fragment
<i>fragID</i>	Fragment ID of new Fragment
<i>i</i>	Beginning of input range
<i>e</i>	End of input range

Returns

FragmentPtr to created [Fragment](#)

Todo Change function access specifier to restrict access

Definition at line 669 of file Fragment.hh.

6.5.4.11 `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

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

Returns

FragmentPtr to created [Fragment](#)

Definition at line 85 of file Fragment.cc.

6.5.4.12 `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 894 of file Fragment.hh.

6.5.4.13 `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 355 of file Fragment.hh.

6.5.4.14 `bool artdaq::Fragment::empty ()` `[inline]`

Determines if the [Fragment](#) contains no data.

Returns

Whether the [Fragment's](#) payload is empty

Definition at line 1078 of file Fragment.hh.

6.5.4.15 `artdaq::FragmentPtr artdaq::Fragment::eodFrag (size_t nFragmentsToExpect)` `[static]`

Creates an EndOfData [Fragment](#).

Parameters

<i>nFragmentsToExpect</i>	The number of Fragments the receiver should have at the end of data-taking
---------------------------	--

Returns

Pointer to created EndOfData [Fragment](#)

Definition at line 74 of file Fragment.cc.

6.5.4.16 `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

<i>nbytes</i>	The initial size of the Fragment , in bytes
---------------	---

Returns

FragmentPtr to created [Fragment](#)

Definition at line 200 of file Fragment.hh.

```
6.5.4.17 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

<i>T</i>	Metadata type
----------	---------------

Parameters

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

Returns

FragmentPtr to created [Fragment](#)

Definition at line 235 of file Fragment.hh.

```
6.5.4.18 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 833 of file Fragment.hh.

```
6.5.4.19 bool artdaq::Fragment::hasMetadata ( ) const [inline]
```

Test whether this [Fragment](#) has metadata.

Returns

If a metadata object has been set

Definition at line 902 of file Fragment.hh.

6.5.4.20 `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 1121 of file `Fragment.hh`.

6.5.4.21 `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 1040 of file `Fragment.hh`.

6.5.4.22 `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 1062 of file `Fragment.hh`.

6.5.4.23 `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 545 of file `Fragment.hh`.

6.5.4.24 `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 596 of file `Fragment.hh`.

6.5.4.25 `bool constexpr artdaq::Fragment::isSystemFragmentType (type_t fragmentType) [inline], [static]`

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

Parameters

<i>fragmentType</i>	The type to test
---------------------	------------------

Returns

Whether the given type is in the range of system types

Definition at line 734 of file Fragment.hh.

6.5.4.26 `bool constexpr artdaq::Fragment::isUserFragmentType (type_t fragmentType) [inline], [static]`

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

Parameters

<i>fragmentType</i>	The type to test
---------------------	------------------

Returns

Whether the given type is in the range of user types

Definition at line 724 of file Fragment.hh.

6.5.4.27 `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 175 of file Fragment.hh.

6.5.4.28 `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

<i>T</i>	Type of the metadata
----------	----------------------

Returns

Pointer to the metadata

Exceptions

<i>cet::exception</i>	if no metadata is present
-----------------------	---------------------------

Definition at line 909 of file Fragment.hh.

6.5.4.29 `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

<i>T</i>	Type of the metadata
----------	----------------------

Returns

const Pointer to the metadata

Exceptions

<i>cet::exception</i>	if no metadata is present
-----------------------	---------------------------

Definition at line 923 of file Fragment.hh.

6.5.4.30 `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

<i>cet::exception</i>	if no metadata in Fragment
-----------------------	--

Todo Change function access specifier to restrict access

Definition at line 1109 of file Fragment.hh.

6.5.4.31 `Fragment& artdaq::Fragment::operator= (const Fragment &)` [default]

Default copy-assignment operator.

Returns

Reference to new [Fragment](#)

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

6.5.4.32 `artdaq::Fragment & artdaq::Fragment::operator= (Fragment &&)` [inline], [default], [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

6.5.4.33 `void artdaq::Fragment::print (std::ostream & os) const`

Print out summary information for this [Fragment](#) to the given stream.

Parameters

<i>os</i>	Stream to print to
-----------	--------------------

Definition at line 65 of file Fragment.cc.

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

Wrapper around reinterpret_cast.

Template Parameters

<i>T</i>	Type of output pointer
----------	------------------------

Parameters

<i>in</i>	input pointer
-----------	---------------

Returns

Pointer cast to type T

Exceptions

<i>cet::exception</i>	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 474 of file Fragment.hh.

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

Wrapper around reinterpret_cast.

Template Parameters

<i>T</i>	Type of output pointer
----------	------------------------

Parameters

<i>in</i>	input pointer
-----------	---------------

Returns

Pointer cast to type T

Exceptions

<i>cet::exception</i>	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 502 of file `Fragment.hh`.

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

Reserves enough memory to hold `cap` `RawDataType` words in the [Fragment](#) payload.

Parameters

<i>cap</i>	The new capacity of the Fragment payload, in <code>RawDataType</code> words.
------------	--

Definition at line 1086 of file `Fragment.hh`.

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

Resize the data payload to hold `sz` `RawDataType` words.

Parameters

<i>sz</i>	The new size of the payload portion of the Fragment , in <code>RawDataType</code> words
-----------	---

Definition at line 973 of file `Fragment.hh`.

6.5.4.38 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

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

Definition at line 982 of file `Fragment.hh`.

6.5.4.39 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

<i>szbytes</i>	The new size of the payload portion of the Fragment , in bytes
----------------	--

Definition at line 990 of file `Fragment.hh`.

6.5.4.40 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

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

Definition at line 998 of file `Fragment.hh`.

6.5.4.41 `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 826 of file `Fragment.hh`.

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

Sets the [Fragment](#) ID of the [Fragment](#).

Parameters

<i>fragment_id</i>	The Fragment ID to set
--------------------	--

Definition at line 869 of file `Fragment.hh`.

6.5.4.43 `template<class T> void artdaq::Fragment::setMetadata (const T & md)`

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

<i>T</i>	Type of the metadata
----------	----------------------

Parameters

<i>md</i>	Metadata to store in Fragment
-----------	---

Exceptions

<i>cet::exception</i>	if metadata already present in Fragment
-----------------------	---

Definition at line 936 of file `Fragment.hh`.

6.5.4.44 `void artdaq::Fragment::setSequenceID (sequence_id_t sequence_id)` `[inline]`

Sets the Sequence ID of the [Fragment](#).

Parameters

<i>sequence_id</i>	The sequence ID to set
--------------------	------------------------

Definition at line 861 of file Fragment.hh.

6.5.4.45 `void artdaq::Fragment::setSystemType (type_t stype) [inline]`

Sets the type of the [Fragment](#), checking that it is a valid system type.

Parameters

<i>stype</i>	The System type to set
--------------	------------------------

Definition at line 854 of file Fragment.hh.

6.5.4.46 `void artdaq::Fragment::setTimestamp (timestamp_t timestamp) [inline]`

Sets the Timestamp of the [Fragment](#).

Parameters

<i>timestamp</i>	The Timestamp to set
------------------	----------------------

Definition at line 876 of file Fragment.hh.

6.5.4.47 `void artdaq::Fragment::setUserType (type_t utype) [inline]`

Sets the type of the [Fragment](#), checking that it is a valid user type.

Parameters

<i>utype</i>	The User type to set
--------------	----------------------

Definition at line 847 of file Fragment.hh.

6.5.4.48 `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 798 of file Fragment.hh.

6.5.4.49 `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 340 of file Fragment.hh.

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

Swaps two [Fragment](#) objects.

Parameters

<i>other</i>	Fragment to swap with
--------------	---------------------------------------

Definition at line 1094 of file `Fragment.hh`.

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

Swaps two [Fragment](#) data vectors.

Parameters

<i>other</i>	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 630 of file `Fragment.hh`.

6.5.4.52 `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 840 of file `Fragment.hh`.

6.5.4.53 `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 812 of file `Fragment.hh`.

6.5.4.54 `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 819 of file `Fragment.hh`.

6.5.4.55 `template<class T> void artdaq::Fragment::updateMetadata (const T & md)`

Updates existing metadata with a new metadata object.

Template Parameters

<i>T</i>	Type of the metadata
----------	----------------------

Parameters

<i>md</i>	Metadata to set
-----------	-----------------

Exceptions

<i>cet::exception</i>	if no metadata stored in Fragment
<i>cet::exception</i>	if new metadata has different size than existing metadata

Definition at line 953 of file `Fragment.hh`.

6.5.4.56 `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 805 of file `Fragment.hh`.

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

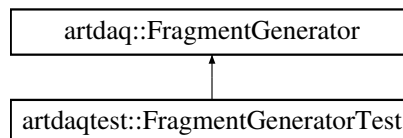
- `artdaq-core/artdaq-core/Data/Fragment.hh`
- `artdaq-core/artdaq-core/Data/Fragment.cc`

6.6 artdaq::FragmentGenerator Class Reference

Base class for all `FragmentGenerators`.

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

Inheritance diagram for `artdaq::FragmentGenerator`:



Public Member Functions

- [FragmentGenerator](#) ()=default
Default Constructor.
- virtual [~FragmentGenerator](#) ()=default
Default Destructor.
- 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?

6.6.1 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 25 of file [FragmentGenerator.hh](#).

6.6.2 Member Function Documentation

6.6.2.1 `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](#).

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

Obtain the next collection of Fragments.

Parameters

<i>output</i>	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](#).

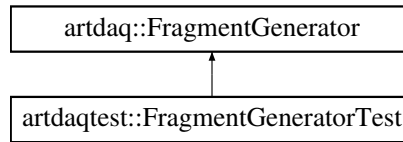
The documentation for this class was generated from the following file:

- [artdaq-core/artdaq-core/Generators/FragmentGenerator.hh](#)

6.7 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?

6.7.1 Detailed Description

Tests the functionality of the [artdaq::FragmentGenerator](#) class.

Definition at line 15 of file `FragmentGenerator_t.cc`.

6.7.2 Member Function Documentation

6.7.2.1 `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 26 of file `FragmentGenerator_t.cc`.

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

Obtain the next collection of Fragments.

Parameters

<i>output</i>	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 21 of file `FragmentGenerator_t.cc`.

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

- `artdaq-core/test/Generators/FragmentGenerator_t.cc`

6.8 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.

6.8.1 Detailed Description

Wrapper around the [artdaqcore::GetPackageBuildInfo::getPackageBuildInfo](#) function.

Definition at line 15 of file `GetPackageBuildInfo.hh`.

6.8.2 Member Function Documentation

6.8.2.1 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`

6.9 artdaq::detail::hasMemoryUsed< T > Class Template Reference

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

Static Public Attributes

- static const bool [value](#) = (sizeof(test<T>(nullptr)) == sizeof(TrueType))

Use SFINAE to figure out if the class used to instantiate the [ConcurrentQueue](#) template has a method `memoryUsed` returning the number of bytes occupied by the class itself.

6.9.1 Detailed Description

```
template<typename T>class artdaq::detail::hasMemoryUsed< T >
```

This template is using SFINAE to figure out if the class used to instantiate the [ConcurrentQueue](#) template has a method `memoryUsed` returning the number of bytes occupied by the class itself.

Definition at line 71 of file `ConcurrentQueue.hh`.

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

- `artdaq-core/artdaq-core/Core/ConcurrentQueue.hh`

6.10 artdaq::KeepNewest< T > Struct Template Reference

[ConcurrentQueue](#) policy to discard oldest elements when the queue is full.

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

Public Types

- typedef std::pair< T, size_t > [ValueType](#)
Type of elements stored in the queue.
- typedef std::list< T > [SequenceType](#)
Type of sequences of items.
- typedef SequenceType::size_type [SizeType](#)
Size type of sequences.
- typedef [SizeType](#) [ReturnType](#)
Type returned by `doEnq`.

Static Public Member Functions

- static void [doInsert](#) (T const &item, [SequenceType](#) &elements, [SizeType](#) &size, [detail::MemoryType](#) const &item-Size, [detail::MemoryType](#) &used, std::condition_variable &nonempty)
Inserts element into the [ConcurrentQueue](#).
- static [ReturnType](#) [doEnq](#) (T const &item, [SequenceType](#) &elements, [SizeType](#) &size, [SizeType](#) &capacity, [detail::MemoryType](#) &used, [detail::MemoryType](#) &memory, size_t &elementsDropped, std::condition_variable &nonempty)
Attempts to enqueue an item.

6.10.1 Detailed Description

```
template<class T>struct artdaq::KeepNewest< T >
```

[ConcurrentQueue](#) policy to discard oldest elements when the queue is full.

Template Parameters

<i>T</i>	Type of element to store in queue
----------	-----------------------------------

Definition at line 255 of file ConcurrentQueue.hh.

6.10.2 Member Function Documentation

6.10.2.1 `template<class T> static ReturnT artdaq::KeepNewest< T >::doEnq (T const & item, SequenceType & elements, SizeType & size, SizeType & capacity, detail::MemoryType & used, detail::MemoryType & memory, size_t & elementsDropped, std::condition_variable & nonempty) [inline],[static]`

Attempts to enqueue an item.

Parameters

<i>item</i>	Item to enqueue
<i>elements</i>	The ConcurrentQueue data
<i>size</i>	Number of elements in the ConcurrentQueue
<i>capacity</i>	Maximum number of elements in the ConcurrentQueue
<i>used</i>	Memory used by the ConcurrentQueue elements
<i>memory</i>	Amount of memory available for use by the ConcurrentQueue
<i>elements-Dropped</i>	Number of failed insert operations
<i>nonempty</i>	Condition variable to notify readers of new data on the queue

Returns

Number of elements removed from the queue

Definition at line 300 of file ConcurrentQueue.hh.

6.10.2.2 `template<class T> static void artdaq::KeepNewest< T >::doInsert (T const & item, SequenceType & elements, SizeType & size, detail::MemoryType const & itemSize, detail::MemoryType & used, std::condition_variable & nonempty) [inline],[static]`

Inserts element into the [ConcurrentQueue](#).

Parameters

<i>item</i>	Item to insert
<i>elements</i>	The ConcurrentQueue data
<i>size</i>	Number of elements in the ConcurrentQueue
<i>itemSize</i>	Size of the newly-inserted element
<i>used</i>	Memory used by the ConcurrentQueue elements
<i>nonempty</i>	Condition variable to notify readers of new data on queue

Definition at line 272 of file ConcurrentQueue.hh.

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

- artdaq-core/artdaq-core/Core/ConcurrentQueue.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](#): 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 [size_t](#) const [size_words](#) = 8ul + CONTAINER_FRAGMENT_COUNT_MAX * sizeof([size_t](#)) / sizeof([data_t](#))
Size of the [Metadata](#) object.

6.11.1 Detailed Description

Contains the information necessary for retrieving [Fragment](#) objects from the [ContainerFragment](#).

Definition at line 37 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 34 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 11 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 13 of file Fragment_t.cc.

6.13.2.2 uint32_t MetadataTypeOne::field2

1. A 32-bit field

Definition at line 14 of file Fragment_t.cc.

6.13.2.3 uint32_t MetadataTypeOne::field3

1. A 32-bit field

Definition at line 15 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 22 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 24 of file `Fragment_t.cc`.

6.14.2.2 uint32_t MetadataTypeTwo::field2

1. A 32-bit field

Definition at line 25 of file `Fragment_t.cc`.

6.14.2.3 uint32_t MetadataTypeTwo::field3

1. A 32-bit field

Definition at line 26 of file `Fragment_t.cc`.

6.14.2.4 uint64_t MetadataTypeTwo::field4

1. A 64-bit field

Definition at line 27 of file `Fragment_t.cc`.

6.14.2.5 uint16_t MetadataTypeTwo::field5

1. A 16-bit field

Definition at line 28 of file Fragment_t.cc.

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

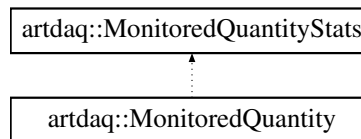
- artdaq-core/test/Data/Fragment_t.cc

6.15 artdaq::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.
- long long [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.

6.15.1 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.

6.15.2 Constructor & Destructor Documentation

- 6.15.2.1 [MonitoredQuantity::MonitoredQuantity](#) (DURATION_T *expectedCalculationInterval*, DURATION_T *timeWindowForRecentResults*) [explicit]

Instantiates a [MonitoredQuantity](#) object.

Parameters

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

Definition at line 9 of file MonitoredQuantity.cc.

6.15.3 Member Function Documentation

- 6.15.3.1 void [MonitoredQuantity::addSample](#) (const double *value* = 1.0)

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

Parameters

<i>value</i>	The sample value to add
--------------	-------------------------

Definition at line 19 of file MonitoredQuantity.cc.

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

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

Parameters

<i>value</i>	The sample value to add
--------------	-------------------------

Definition at line 35 of file MonitoredQuantity.cc.

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

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

Parameters

<i>value</i>	The sample value to add
--------------	-------------------------

Definition at line 40 of file MonitoredQuantity.cc.

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

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

Parameters

<i>value</i>	The sample value to add
--------------	-------------------------

Definition at line 45 of file MonitoredQuantity.cc.

6.15.3.5 bool MonitoredQuantity::calculateStatistics (TIME_POINT_T *currentTime* = getCurrentTime ())

Forces a calculation of the statistics for the monitored quantity.

Parameters

<i>currentTime</i>	Time point to use for calculating statistics (if synchronized at a higher level)
--------------------	--

Returns

Whether the statistics 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 50 of file MonitoredQuantity.cc.

6.15.3.6 void MonitoredQuantity::disable ()

Disables the monitor.

Definition at line 274 of file MonitoredQuantity.cc.

6.15.3.7 void MonitoredQuantity::enable ()

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

Definition at line 265 of file MonitoredQuantity.cc.

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

Returns the expected interval between calculateStatistics calls.

Returns

The expected interval between calculateStatistics calls

Definition at line 259 of file MonitoredQuantity.hh.

6.15.3.9 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 381 of file MonitoredQuantity.cc.

6.15.3.10 void MonitoredQuantity::getStats (MonitoredQuantityStats & stats) const

Write all our collected statistics into the given Stats struct.

Parameters

<i>stats</i>	Destination for copy of collected statistics
--------------	--

Definition at line 338 of file MonitoredQuantity.cc.

6.15.3.11 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 250 of file `MonitoredQuantity.hh`.

6.15.3.12 `bool artdaq::MonitoredQuantity::isEnabled () const` `[inline]`

Tests whether the monitor is currently enabled.

Returns

Whether the monitor is currently enabled.

Definition at line 230 of file `MonitoredQuantity.hh`.

6.15.3.13 `void MonitoredQuantity::reset ()`

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

Definition at line 253 of file `MonitoredQuantity.cc`.

6.15.3.14 `void MonitoredQuantity::setNewTimeWindowForRecentResults (DURATION_T interval)`

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

Parameters

<i>interval</i>	The new time interval for calculating "recent" statistics.
-----------------	--

Definition at line 281 of file `MonitoredQuantity.cc`.

6.15.3.15 `bool MonitoredQuantity::waitUntilAccumulatorsHaveBeenFlushed (DURATION_T timeout) const`

Blocks while the [MonitoredQuantity](#) is flushed, up to timeout duration.

Parameters

<i>timeout</i>	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 318 of file `MonitoredQuantity.cc`.

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

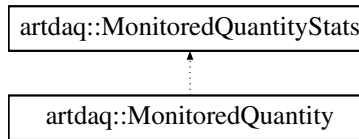
- `artdaq-core/artdaq-core/Core/MonitoredQuantity.hh`
- `artdaq-core/artdaq-core/Core/MonitoredQuantity.cc`

6.16 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

- long long [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 calculateStatistics calls)
- bool [isEnabled](#) () const
- Access the enable flag.

Public Attributes

- long long [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 samples.
- double [fullValueRate](#)
The sum of all samples over the full duration of sampling.
- [DURATION_T](#) [fullDuration](#)
The full duration of sampling.
- long long [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< long long > [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 calculateStatistics 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.

6.16.1 Detailed Description

struct containing [MonitoredQuantity](#) data

Definition at line 15 of file MonitoredQuantity.hh.

6.16.2 Member Enumeration Documentation

6.16.2.1 `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.

6.16.3 Member Function Documentation

6.16.3.1 `DURATION_T artdaq::MonitoredQuantityStats::getDuration (DataSetType t = DataSetType::FULL) const` `[inline]`

Returns the duration of the requested interval.

Parameters

<code>t</code>	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.

6.16.3.2 `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.

6.16.3.3 `double artdaq::MonitoredQuantityStats::getLastValueRate () const [inline]`

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

Returns

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

Definition at line 145 of file MonitoredQuantity.hh.

6.16.3.4 `long long artdaq::MonitoredQuantityStats::getSampleCount (DataSetType t = DataSetType::FULL) const [inline]`

Returns the sample count for the requested interval.

Parameters

<code>t</code>	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.

6.16.3.5 `double artdaq::MonitoredQuantityStats::getSampleLatency (DataSetType t = DataSetType::FULL) const [inline]`

Parameters

<code>t</code>	Which interval to return, DataSetType::FULL (default) or DataSetType::RECENT
----------------	--

Returns

Definition at line 129 of file MonitoredQuantity.hh.

6.16.3.6 `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.

6.16.3.7 `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.

6.16.3.8 `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.

6.16.3.9 `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.

6.16.3.10 `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.

6.16.3.11 `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.

6.16.3.12 `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.

6.16.3.13 `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

6.17 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 str)
Sets the package name.
- void [setPackageVersion](#) (std::string str)
Sets the package version.
- void [setBuildTimestamp](#) (std::string str)
Sets the build timestamp.

6.17.1 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 18 of file `PackageBuildInfo.hh`.

6.17.2 Member Function Documentation

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

Gets the build timestamp.

Returns

The timestamp of the build

Definition at line 42 of file `PackageBuildInfo.hh`.

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

Gets the package name.

Returns

The package name

Definition at line 30 of file `PackageBuildInfo.hh`.

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

Gets the package version.

Returns

The package version

Definition at line 36 of file PackageBuildInfo.hh.

6.17.2.4 `void artdaq::PackageBuildInfo::setBuildTimestamp (std::string str)` `[inline]`

Sets the build timestamp.

Parameters

<i>str</i>	The timestamp of the build
------------	----------------------------

Definition at line 60 of file PackageBuildInfo.hh.

6.17.2.5 `void artdaq::PackageBuildInfo::setPackageName (std::string str)` `[inline]`

Sets the package name.

Parameters

<i>str</i>	The package name
------------	------------------

Definition at line 48 of file PackageBuildInfo.hh.

6.17.2.6 `void artdaq::PackageBuildInfo::setPackageVersion (std::string str)` `[inline]`

Sets the package version.

Parameters

<i>str</i>	The package version
------------	---------------------

Definition at line 54 of file PackageBuildInfo.hh.

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

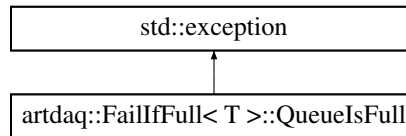
- artdaq-core/artdaq-core/Data/PackageBuildInfo.hh

6.18 artdaq::FailIfFull< T >::QueueIsFull Struct Reference

Exception thrown by [FailIfFull](#) policy when an enqueue operation is attempted on a full queue.

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

Inheritance diagram for artdaq::FailIfFull< T >::QueueIsFull:



Public Member Functions

- virtual const char * [what](#) () const throw ()
Describe exception.

6.18.1 Detailed Description

template<class T>struct artdaq::FailIfFull< T >::QueueIsFull

Exception thrown by [FailIfFull](#) policy when an enqueue operation is attempted on a full queue.

Definition at line 171 of file ConcurrentQueue.hh.

6.18.2 Member Function Documentation

6.18.2.1 template<class T> virtual const char* artdaq::FailIfFull< T >::QueueIsFull::what () const throw () [inline], [virtual]

Describe exception.

Returns

Description of [QueueIsFull](#) exception

Definition at line 177 of file ConcurrentQueue.hh.

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

- artdaq-core/artdaq-core/Core/ConcurrentQueue.hh

6.19 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 type
- 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 [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 *.*

6.19.1 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 86 of file QuickVec.hh.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 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 338 of file QuickVec.hh.

6.19.2.2 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

<i>sz</i>	Size of QuickVec object to allocate
<i>val</i>	Value with which to initialize elements

Definition at line 347 of file QuickVec.hh.

6.19.2.3 `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

<i>other</i>	The vector to copy
--------------	--------------------

Definition at line 118 of file QuickVec.hh.

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

Copy Constructor.

Parameters

<i>other</i>	QuickVec to copy
--------------	----------------------------------

Definition at line 138 of file QuickVec.hh.

6.19.3 Member Function Documentation

6.19.3.1 `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 387 of file QuickVec.hh.

6.19.3.2 `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 390 of file QuickVec.hh.

6.19.3.3 `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 capacity 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 384 of file QuickVec.hh.

6.19.3.4 `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 326 of file QuickVec.hh.

6.19.3.5 `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 393 of file QuickVec.hh.

6.19.3.6 `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 396 of file QuickVec.hh.

6.19.3.7 `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

<i>first</i>	First element to erase
<i>last</i>	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 484 of file QuickVec.hh.

6.19.3.8 `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

<i>position</i>	Position at which to insert
<i>nn</i>	Number of copies of val to insert
<i>val</i>	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 443 of file QuickVec.hh.

6.19.3.9 `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

<i>position</i>	Position at which to insert
<i>first</i>	const_iterator to first element to insert
<i>last</i>	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 463 of file QuickVec.hh.

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

Copy assignment operator.

Parameters

<i>other</i>	QuickVec to copy
--------------	----------------------------------

Returns

Reference to new [QuickVec](#) object

Definition at line 153 of file QuickVec.hh.

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

Returns a reference to a given element.

Parameters

<i>idx</i>	Element to return
------------	-------------------

Returns

Reference to element

Definition at line 367 of file QuickVec.hh.

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

Returns a const reference to a given element.

Parameters

<i>idx</i>	Element to return
------------	-------------------

Returns

const reference to element

Definition at line 374 of file QuickVec.hh.

6.19.3.13 `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

<i>val</i>	Value to add to the QuickVec
------------	--

Definition at line 513 of file QuickVec.hh.

6.19.3.14 `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

<i>size</i>	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 399 of file QuickVec.hh.

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

Resizes the [QuickVec](#).

Parameters

<i>size</i>	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 415 of file QuickVec.hh.

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

Resizes the [QuickVec](#), initializes new elements with *val*.

Parameters

<i>size</i>	New size of the QuickVec
<i>val</i>	Value with which to initialize elements

Definition at line 432 of file QuickVec.hh.

6.19.3.17 `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 381 of file QuickVec.hh.

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

Exchanges references to two [QuickVec](#) objects.

Parameters

<i>other</i>	Other QuickVec to swap with
--------------	---

Definition at line 501 of file QuickVec.hh.

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

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

6.20 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::sequence_id_t](#) [sequence_id_t](#)
Field size should be the same as the [Fragment::sequence_id](#) field.

Public Member Functions

- [RawEvent](#) ([run_id_t](#) run, [subrun_id_t](#) subrun, [sequence_id_t](#) event)
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](#).
- [sequence_id_t](#) [sequenceID](#) () const
Retrieve the sequence id 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](#).

6.20.1 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 64 of file RawEvent.hh.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 `artdaq::RawEvent::RawEvent (run_id_t run, subrun_id_t subrun, sequence_id_t event) [inline]`

Constructs a [RawEvent](#) with the given parameters.

Parameters

<i>run</i>	The current Run number
<i>subrun</i>	The current Subrun number
<i>event</i>	The current sequence_id

Definition at line 184 of file RawEvent.hh.

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

Constructs a [RawEvent](#) using the given RawEventHeader.

Parameters

<i>hdr</i>	Header to use for initializing RawEvent
------------	---

Definition at line 188 of file RawEvent.hh.

6.20.3 Member Function Documentation

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

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

Parameters

<i>type_list</i>	Any Fragment types not included in this list will be added
------------------	--

Definition at line 243 of file RawEvent.hh.

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

Insert the given (pointer to a) [Fragment](#) into this [RawEvent](#).

Parameters

<i>pfrag</i>	The FragmentPtr to insert into the RawEvent
--------------	---

Exceptions

<i>cet::exception</i>	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 193 of file RawEvent.hh.

6.20.3.3 `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 222 of file RawEvent.hh.

6.20.3.4 `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 206 of file RawEvent.hh.

6.20.3.5 `void artdaq::RawEvent::print (std::ostream & os) const`

Print summary information about this [RawEvent](#) to the given stream.

Parameters

<i>os</i>	The target stream for summary information
-----------	---

Definition at line 6 of file RawEvent.cc.

6.20.3.6 `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 225 of file RawEvent.hh.

6.20.3.7 `std::unique_ptr< Fragments > artdaq::RawEvent::releaseProduct (Fragment::type_t type) [inline]`

Release Fragments from the [RawEvent](#).

Parameters

<i>type</i>	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 261 of file `RawEvent.hh`.

6.20.3.8 `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 219 of file `RawEvent.hh`.

6.20.3.9 `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 221 of file `RawEvent.hh`.

6.20.3.10 `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 220 of file `RawEvent.hh`.

6.20.3.11 `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 212 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

6.21 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 [Fragment::sequence_id_t](#) [sequence_id_t](#)
Field size should be the same as the [Fragment::sequence_id](#) field.

Public Member Functions

- [RawEventHeader](#) ([run_id_t](#) run, [subrun_id_t](#) subrun, [sequence_id_t](#) event)
Constructs the [RawEventHeader](#) struct with the given parameters.

Public Attributes

- [run_id_t](#) [run_id](#)
Fragments don't know about runs.
- [subrun_id_t](#) [subrun_id](#)
Fragments don't know about subruns.
- [sequence_id_t](#) [sequence_id](#)
[RawEvent](#) [sequence_id](#) should be the same as its component [Fragment](#) [sequence_ids](#).
- bool [is_complete](#)
Does the event contain the expected number of [Fragment](#) objects?

6.21.1 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 30 of file RawEvent.hh.

6.21.2 Constructor & Destructor Documentation

6.21.2.1 `artdaq::detail::RawEventHeader::RawEventHeader (run_id_t run, subrun_id_t subrun, sequence_id_t event)`
`[inline]`

Constructs the [RawEventHeader](#) struct with the given parameters.

Parameters

<i>run</i>	The current Run number
<i>subrun</i>	The current Subrun number
<i>event</i>	The current sequence_id

Definition at line 47 of file RawEvent.hh.

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

- `artdaq-core/artdaq-core/Data/RawEvent.hh`

6.22 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 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.

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. 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](#): 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 [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 32-bits.

6.22.1 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 33 of file RawFragmentHeader.hh.

6.22.2 Member Function Documentation

6.22.2.1 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 68 of file RawFragmentHeader.hh.

6.22.2.2 `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 81 of file RawFragmentHeader.hh.

6.22.2.3 `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 175 of file RawFragmentHeader.hh.

6.22.2.4 `void artdaq::detail::RawFragmentHeader::setSystemType (uint8_t stype)` `[inline]`

Sets the type field to the specified system type.

Parameters

<i>stype</i>	The type code to set
--------------	----------------------

Exceptions

<i>cet::exception</i>	if stype is not in the allowed range for system types
-----------------------	---

Definition at line 205 of file RawFragmentHeader.hh.

6.22.2.5 `void artdaq::detail::RawFragmentHeader::setUserType (uint8_t utype)` `[inline]`

Sets the type field to the specified user type.

Parameters

<i>utype</i>	The type code to set
--------------	----------------------

Exceptions

<i>cet::exception</i>	if utype is not in the allowed range for user types
-----------------------	---

Definition at line 191 of file RawFragmentHeader.hh.

6.22.2.6 `static std::string artdaq::detail::RawFragmentHeader::SystemTypeToString (type_t type)` `[inline], [static]`

Print a system type's string name.

Parameters

<i>type</i>	Type to print
-------------	---------------

Returns

String with "Name" of type

Definition at line 100 of file RawFragmentHeader.hh.

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

- artdaq-core/artdaq-core/Data/detail/RawFragmentHeader.hh

6.23 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 unsigned long long [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.

6.23.1 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 [RawFragmentHeader](#), provided for compatibility

Definition at line 35 of file [RawFragmentHeaderV0.hh](#).

6.23.2 Member Function Documentation

6.23.2.1 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 70 of file [RawFragmentHeaderV0.hh](#).

6.23.2.2 `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 83 of file RawFragmentHeaderV0.hh.

6.23.2.3 `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 158 of file RawFragmentHeaderV0.hh.

6.23.2.4 `void artdaq::detail::RawFragmentHeaderV0::setSystemType (uint8_t stype)` `[inline]`

Sets the type field to the specified system type.

Parameters

<i>stype</i>	The type code to set
--------------	----------------------

Exceptions

<i>cet::exception</i>	if stype is not in the allowed range for system types
-----------------------	---

Definition at line 187 of file RawFragmentHeaderV0.hh.

6.23.2.5 `void artdaq::detail::RawFragmentHeaderV0::setUserType (uint8_t utype)` `[inline]`

Sets the type field to the specified user type.

Parameters

<i>utype</i>	The type code to set
--------------	----------------------

Exceptions

<i>cet::exception</i>	if utype is not in the allowed range for user types
-----------------------	---

Definition at line 173 of file RawFragmentHeaderV0.hh.

6.23.2.6 `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 (guaranteed to be smaller) is cast to the new header. In the case of added fields, they will remain marked Invalid.

Definition at line 200 of file RawFragmentHeaderV0.hh.

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

- artdaq-core/artdaq-core/Data/detail/RawFragmentHeaderV0.hh

6.24 artdaq::RejectNewest< T > Struct Template Reference

[ConcurrentQueue](#) policy to discard new elements when the queue is full.

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

Public Types

- typedef std::pair< T, size_t > [ValueType](#)
Type of elements stored in the queue.
- typedef std::list< T > [SequenceType](#)
Type of sequences of items.
- typedef SequenceType::size_type [SizeType](#)
Size type of sequences.
- typedef [SizeType](#) [ReturnType](#)
Type returned by doEnq.

Static Public Member Functions

- static void [doInsert](#) (T const &item, [SequenceType](#) &elements, [SizeType](#) &size, [detail::MemoryType](#) const &item-Size, [detail::MemoryType](#) &used, std::condition_variable &nonempty)
Inserts element into the [ConcurrentQueue](#).
- static [ReturnType](#) [doEnq](#) (T const &item, [SequenceType](#) &elements, [SizeType](#) &size, [SizeType](#) &capacity, [detail::MemoryType](#) &used, [detail::MemoryType](#) &memory, size_t &elementsDropped, std::condition_variable &nonempty)
Attempts to enqueue an item.

6.24.1 Detailed Description

```
template<class T>struct artdaq::RejectNewest< T >
```

[ConcurrentQueue](#) policy to discard new elements when the queue is full.

Template Parameters

<i>T</i>	Type of element to store in queue
----------	-----------------------------------

Definition at line 343 of file ConcurrentQueue.hh.

6.24.2 Member Function Documentation

6.24.2.1 `template<class T> static ReturnT artdaq::RejectNewest< T >::doEnq (T const & item, SequenceType & elements, SizeType & size, SizeType & capacity, detail::MemoryType & used, detail::MemoryType & memory, size_t & elementsDropped, std::condition_variable & nonempty) [inline],[static]`

Attempts to enqueue an item.

Parameters

<i>item</i>	Item to enqueue
<i>elements</i>	The ConcurrentQueue data
<i>size</i>	Number of elements in the ConcurrentQueue
<i>capacity</i>	Maximum number of elements in the ConcurrentQueue
<i>used</i>	Memory used by the ConcurrentQueue elements
<i>memory</i>	Amount of memory available for use by the ConcurrentQueue
<i>elements-Dropped</i>	Number of failed insert operations
<i>nonempty</i>	Condition variable to notify readers of new data on the queue

Returns

Number of elements removed from the queue (1 if new element was rejected, 0 otherwise)

Definition at line 388 of file ConcurrentQueue.hh.

6.24.2.2 `template<class T> static void artdaq::RejectNewest< T >::doInsert (T const & item, SequenceType & elements, SizeType & size, detail::MemoryType const & itemSize, detail::MemoryType & used, std::condition_variable & nonempty) [inline],[static]`

Inserts element into the [ConcurrentQueue](#).

Parameters

<i>item</i>	Item to insert
<i>elements</i>	The ConcurrentQueue data
<i>size</i>	Number of elements in the ConcurrentQueue
<i>itemSize</i>	Size of the newly-inserted element
<i>used</i>	Memory used by the ConcurrentQueue elements
<i>nonempty</i>	Condition variable to notify readers of new data on queue

Definition at line 360 of file ConcurrentQueue.hh.

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

- artdaq-core/artdaq-core/Core/ConcurrentQueue.hh

6.25 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 [ReadReadyCount](#) ()
Get the count of available buffers, both broadcasts and data.
- size_t [size](#) ()
Get the size of the data buffer.

6.25.1 Detailed Description

[SharedMemoryEventReceiver](#) can receive events (as written by SharedMemoryEventManager) from Shared Memory.

Definition at line 15 of file SharedMemoryEventReceiver.hh.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 artdaq::SharedMemoryEventReceiver::SharedMemoryEventReceiver (uint32_t shm_key, uint32_t broadcast_shm_key)

Connect to a Shared Memory segment using the given parameters.

Parameters

<i>shm_key</i>	Key of the Shared Memory segment
<i>broadcast_shm_key</i>	Key of the broadcast Shared Memory segment

Definition at line 11 of file SharedMemoryEventReceiver.cc.

6.25.3 Member Function Documentation

6.25.3.1 `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

<i>err</i>	Flag used to indicate if an error has occurred
<i>type</i>	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 119 of file SharedMemoryEventReceiver.cc.

6.25.3.2 `std::set< artdaq::Fragment::type_t > artdaq::SharedMemoryEventReceiver::GetFragmentTypes (bool & err)`

Get a set of [Fragment](#) Types present in the event.

Parameters

<i>err</i>	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 95 of file SharedMemoryEventReceiver.cc.

6.25.3.3 `int artdaq::SharedMemoryEventReceiver::GetRank () [inline]`

Returns the Rank of the writing process.

Returns

The rank of the writer process

Definition at line 74 of file SharedMemoryEventReceiver.hh.

6.25.3.4 `artdaq::detail::RawEventHeader * artdaq::SharedMemoryEventReceiver::ReadHeader (bool & err)`

Get the Event header.

Parameters

<i>err</i>	Flag used to indicate if an error has occurred
------------	--

Returns

Pointer to RawEventHeader from buffer

Definition at line 76 of file SharedMemoryEventReceiver.cc.

6.25.3.5 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 80 of file SharedMemoryEventReceiver.hh.

6.25.3.6 bool artdaq::SharedMemoryEventReceiver::ReadyForRead (bool *broadcast* = false, size_t *timeout_us* = 1000000)

Determine whether an event is available for reading.

Parameters

<i>broadcast</i>	(Default false) Whether to wait for a broadcast buffer only
<i>timeout_us</i>	(Default 1000000) Time to wait for buffer to become available.

Returns

Whether an event is available for reading

Definition at line 22 of file SharedMemoryEventReceiver.cc.

6.25.3.7 size_t artdaq::SharedMemoryEventReceiver::size () [inline]

Get the size of the data buffer.

Returns

The size of the data buffer

Definition at line 86 of file SharedMemoryEventReceiver.hh.

6.25.3.8 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 176 of file SharedMemoryEventReceiver.cc.

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

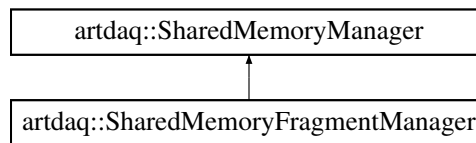
- artdaq-core/artdaq-core/Core/SharedMemoryEventReceiver.hh
- artdaq-core/artdaq-core/Core/SharedMemoryEventReceiver.cc

6.26 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:



Public Member Functions

- [SharedMemoryFragmentManager](#) (uint32_t shm_key, size_t buffer_count, size_t max_buffer_size, size_t buffer_timeout_us=100 * 1000000)
SharedMemoryFragmentManager Constructor.
- virtual [~SharedMemoryFragmentManager](#) ()=default
SharedMemoryFragmentManager destructor.
- int [WriteFragment](#) ([Fragment](#) &&fragment, bool overwrite)
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.

Additional Inherited Members

6.26.1 Detailed Description

The [SharedMemoryFragmentManager](#) is a [SharedMemoryManager](#) that deals with [Fragment](#) transfers using a [Shared-MemoryManager](#).

Definition at line 11 of file SharedMemoryFragmentManager.hh.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 `artdaq::SharedMemoryFragmentManager::SharedMemoryFragmentManager (uint32_t shm_key, size_t buffer_count, size_t max_buffer_size, size_t buffer_timeout_us = 100 * 1000000)`

[SharedMemoryFragmentManager](#) Constructor.

Parameters

<i>shm_key</i>	The key to use when attaching/creating the shared memory segment
<i>buffer_count</i>	The number of buffers in the shared memory
<i>max_buffer_size</i>	The size of each buffer
<i>buffer_timeout_us</i>	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 5 of file SharedMemoryFragmentManager.cc.

6.26.3 Member Function Documentation

6.26.3.1 `int artdaq::SharedMemoryFragmentManager::ReadFragment (Fragment & fragment)`

Read a [Fragment](#) from the Shared Memory.

Parameters

<i>fragment</i>	Output Fragment object
-----------------	--

Returns

0 on success

Definition at line 32 of file SharedMemoryFragmentManager.cc.

6.26.3.2 `int artdaq::SharedMemoryFragmentManager::ReadFragmentData (RawDataType * destination, size_t words)`

Read [Fragment](#) Data from the Shared Memory.

Parameters

<i>destination</i>	Destination for data
<i>words</i>	RawDataType Word count to read

Returns

0 on success

Definition at line 61 of file SharedMemoryFragmentManager.cc.

6.26.3.3 `int artdaq::SharedMemoryFragmentManager::ReadFragmentHeader (detail::RawFragmentHeader & header)`

Read a [Fragment](#) Header from the Shared Memory.

Parameters

<i>header</i>	Output Fragment Header
---------------	--

Returns

0 on success

Definition at line 46 of file SharedMemoryFragmentManager.cc.

6.26.3.4 `int artdaq::SharedMemoryFragmentManager::WriteFragment (Fragment && fragment, bool overwrite)`

Write a [Fragment](#) to the Shared Memory.

Parameters

<i>fragment</i>	Fragment to write
<i>overwrite</i>	Whether to set the overwrite flag

Returns

0 on success

Definition at line 12 of file SharedMemoryFragmentManager.cc.

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

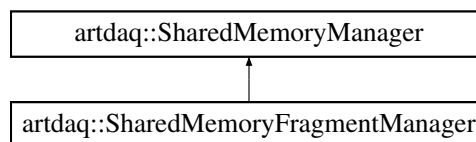
- `artdaq-core/artdaq-core/Core/SharedMemoryFragmentManager.hh`
- `artdaq-core/artdaq-core/Core/SharedMemoryFragmentManager.cc`

6.27 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`:



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.
- void [Attach](#) ()
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.
- 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](#) ()
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.
- 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.
- void [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)
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?
- 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, std::string category="", std::string message="")
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)

Static Public Member Functions

- static std::string [FlagToString](#) ([BufferSemaphoreFlags](#) flag)
Convert a BufferSemaphoreFlags variable to its string representation.

6.27.1 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 18 of file SharedMemoryManager.hh.

6.27.2 Member Enumeration Documentation

6.27.2.1 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 25 of file SharedMemoryManager.hh.

6.27.3 Constructor & Destructor Documentation

6.27.3.1 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

<i>shm_key</i>	The key to use when attaching/creating the shared memory segment
<i>buffer_count</i>	The number of buffers in the shared memory
<i>buffer_size</i>	The size of each buffer
<i>buffer_timeout_us</i>	The maximum amount of time a buffer can be left untouched by its owner before being returned to its previous state.
<i>destructive_read_mode</i>	Whether a read operation empties the buffer (default: true, false for broadcast mode)

Definition at line 9 of file SharedMemoryManager.cc.

6.27.4 Member Function Documentation

6.27.4.1 size_t artdaq::SharedMemoryManager::BufferDataSize (int buffer)

Get the current size of the buffer's data.

Parameters

<i>buffer</i>	Buffer ID of buffer
---------------	---------------------

Returns

Current size of data in the buffer

Definition at line 335 of file SharedMemoryManager.cc.

6.27.4.2 `bool artdaq::SharedMemoryManager::CheckBuffer (int buffer, BufferSemaphoreFlags flags)`

Check both semaphore conditions (Mode flag and manager ID) for a given buffer.

Parameters

<i>buffer</i>	Buffer ID of buffer
<i>flags</i>	Expected Mode flag

Returns

Whether the buffer passed the check

Definition at line 398 of file SharedMemoryManager.cc.

6.27.4.3 `void artdaq::SharedMemoryManager::Detach (bool throwException = false, std::string category = "", std::string message = "")`

Detach from the Shared Memory segment, optionally throwing a cet::exception with the specified properties.

Parameters

<i>throwException</i>	Whether to throw an exception after detaching
<i>category</i>	Category for the cet::exception
<i>message</i>	Message for the cet::exception

Definition at line 616 of file SharedMemoryManager.cc.

6.27.4.4 `static std::string artdaq::SharedMemoryManager::FlagToString (BufferSemaphoreFlags flag) [inline], [static]`

Convert a BufferSemaphoreFlags variable to its string representation.

Parameters

<i>flag</i>	BufferSemaphoreFlags variable to convert
-------------	--

Returns

String representation of flag

Definition at line 38 of file SharedMemoryManager.hh.

6.27.4.5 uint16_t artdaq::SharedMemoryManager::GetAttachedCount () const [inline]

Get the number of attached SharedMemoryManagers.

Returns

The number of attached SharedMemoryManagers

Definition at line 199 of file SharedMemoryManager.hh.

6.27.4.6 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 305 of file SharedMemoryManager.hh.

6.27.4.7 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 130 of file SharedMemoryManager.cc.

6.27.4.8 int artdaq::SharedMemoryManager::GetBufferForWriting (bool *overwrite*)

Finds a buffer that is ready to be written to, and reserves it for the calling manager.

Parameters

<i>overwrite</i>	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 194 of file SharedMemoryManager.cc.

6.27.4.9 std::deque< int > artdaq::SharedMemoryManager::GetBuffersOwnedByManager ()

Get the list of all buffers currently owned by this manager instance.

Returns

A std::deque<int> of buffer IDs currently owned by this manager instance.

Definition at line 317 of file SharedMemoryManager.cc.

6.27.4.10 `void * artdaq::SharedMemoryManager::GetBufferStart (int buffer)`

Get a pointer to the start position of the buffer.

Parameters

<i>buffer</i>	Buffer ID of buffer
---------------	---------------------

Returns

void* pointer to buffer start position

Definition at line 568 of file SharedMemoryManager.cc.

6.27.4.11 `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 299 of file SharedMemoryManager.hh.

6.27.4.12 `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 264 of file SharedMemoryManager.hh.

6.27.4.13 `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 311 of file SharedMemoryManager.hh.

6.27.4.14 `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 210 of file SharedMemoryManager.hh.

6.27.4.15 `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 216 of file SharedMemoryManager.hh.

6.27.4.16 `void * artdaq::SharedMemoryManager::GetReadPos (int buffer)`

Get a pointer to the current read position of the buffer.

Parameters

<i>buffer</i>	Buffer ID of buffer
---------------	---------------------

Returns

void* pointer to the buffer's current read position

Definition at line 557 of file SharedMemoryManager.cc.

6.27.4.17 `void * artdaq::SharedMemoryManager::GetWritePos (int buffer)`

Get a pointer to the current write position of the buffer.

Parameters

<i>buffer</i>	Buffer ID of buffer
---------------	---------------------

Returns

void* pointer to buffer's current write position

Definition at line 562 of file SharedMemoryManager.cc.

6.27.4.18 `void artdaq::SharedMemoryManager::IncrementReadPos (int buffer, size_t read)`

Increment the read position for a given buffer.

Parameters

<i>buffer</i>	Buffer ID of buffer
<i>read</i>	Number of bytes by which to increment read position

Definition at line 364 of file SharedMemoryManager.cc.

6.27.4.19 `void artdaq::SharedMemoryManager::IncrementWritePos (int buffer, size_t written)`

Increment the write position for a given buffer.

Parameters

<i>buffer</i>	Buffer ID of buffer
<i>written</i>	Number of bytes by which to increment write position

Definition at line 377 of file SharedMemoryManager.cc.

6.27.4.20 `bool artdaq::SharedMemoryManager::IsValid () const [inline]`

Is the shared memory pointer valid?

Returns

Whether the shared memory pointer is valid

Definition at line 228 of file SharedMemoryManager.hh.

6.27.4.21 `void artdaq::SharedMemoryManager::MarkBufferEmpty (int buffer, bool force = false)`

Release a buffer from a reader, marking it Empty and ready to accept more data.

Parameters

<i>buffer</i>	Buffer ID of buffer
<i>force</i>	Force buffer to empty state (only if manager_id_ == 0)

Definition at line 420 of file SharedMemoryManager.cc.

6.27.4.22 `void artdaq::SharedMemoryManager::MarkBufferFull (int buffer, int destination = -1)`

Release a buffer from a writer, marking it Full and ready for a reader.

Parameters

<i>buffer</i>	Buffer ID of buffer
<i>destination</i>	If desired, a destination manager ID may be specified for a buffer

Definition at line 405 of file SharedMemoryManager.cc.

6.27.4.23 `bool artdaq::SharedMemoryManager::MoreDataInBuffer (int buffer)`

Determine if more data is available to be read, based on the read position and data size.

Parameters

<i>buffer</i>	Buffer ID of buffer
---------------	---------------------

Returns

Whether more data is available in the given buffer.

Definition at line 389 of file SharedMemoryManager.cc.

6.27.4.24 `bool artdaq::SharedMemoryManager::Read (int buffer, void * data, size_t size)`

Read size bytes of data from buffer into the given pointer.

Parameters

<i>buffer</i>	Buffer ID of buffer
<i>data</i>	Destination pointer for read
<i>size</i>	Size of read, in bytes

Returns

Whether the read was successful

Definition at line 514 of file SharedMemoryManager.cc.

6.27.4.25 `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 274 of file SharedMemoryManager.cc.

6.27.4.26 `bool artdaq::SharedMemoryManager::ReadyForRead () [inline]`

Whether any buffer is ready for read.

Returns

True if there is a buffer available

Definition at line 93 of file SharedMemoryManager.hh.

6.27.4.27 `bool artdaq::SharedMemoryManager::ReadyForWrite (bool overwrite) [inline]`

Whether any buffer is available for write.

Parameters

<i>overwrite</i>	Whether to allow overwriting full buffers
------------------	---

Returns

True if there is a buffer available

Definition at line 100 of file SharedMemoryManager.hh.

6.27.4.28 `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

<i>buffer</i>	Buffer ID of buffer
---------------	---------------------

Returns

Whether the buffer has exceeded the maximum age

Definition at line 450 of file SharedMemoryManager.cc.

6.27.4.29 void artdaq::SharedMemoryManager::ResetReadPos (int *buffer*)

Set the read position of the given buffer to the beginning of the buffer.

Parameters

<i>buffer</i>	Buffer ID of buffer
---------------	---------------------

Definition at line 345 of file SharedMemoryManager.cc.

6.27.4.30 void artdaq::SharedMemoryManager::ResetWritePos (int *buffer*)

Set the write position of the given buffer to the beginning of the buffer.

Parameters

<i>buffer</i>	Buffer ID of buffer
---------------	---------------------

Definition at line 354 of file SharedMemoryManager.cc.

6.27.4.31 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

<i>size</i>	Size (in bytes) after which a buffer will be considered non-empty
-------------	---

Definition at line 322 of file SharedMemoryManager.hh.

6.27.4.32 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

<i>rank</i>	Rank to set
-------------	-------------

Definition at line 222 of file SharedMemoryManager.hh.

6.27.4.33 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 234 of file SharedMemoryManager.hh.

6.27.4.34 `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 529 of file SharedMemoryManager.cc.

6.27.4.35 `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

<i>buffer</i>	Buffer ID of buffer
<i>data</i>	Source pointer for write
<i>size</i>	Size of write, in bytes

Returns

Amount of data written, in bytes

Definition at line 492 of file SharedMemoryManager.cc.

6.27.4.36 `size_t artdaq::SharedMemoryManager::WriteReadyCount (bool overwrite)`

Count the number of buffers that are ready for writing.

Parameters

<i>overwrite</i>	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 298 of file SharedMemoryManager.cc.

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

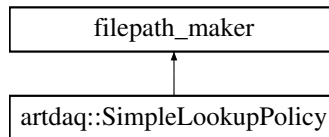
- artdaq-core/artdaq-core/Core/SharedMemoryManager.hh
- artdaq-core/artdaq-core/Core/SharedMemoryManager.cc

6.28 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.

6.28.1 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 21 of file SimpleLookupPolicy.hh.

6.28.2 Member Enumeration Documentation

6.28.2.1 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 28 of file SimpleLookupPolicy.hh.

6.28.3 Constructor & Destructor Documentation

6.28.3.1 `artdaq::SimpleLookupPolicy::SimpleLookupPolicy (std::string const & paths, ArgType argType = ArgType::ENV_VAR)`

Constructor.

Parameters

<i>paths</i>	Either the name of an environment variable or a list of directories to search
<i>argType</i>	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 5 of file SimpleLookupPolicy.cc.

6.28.4 Member Function Documentation

6.28.4.1 `std::string artdaq::SimpleLookupPolicy::operator() (std::string const & filename) [override]`

Perform the file lookup.

Parameters

<i>filename</i>	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 41 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`

6.29 artdaq::SimpleMemoryReader Class Reference

[SimpleMemoryReader](#) will continue to read [RawEvent](#) objects off the queue until it encounters a null pointer, at which point it stops.

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

Public Member Functions

- [SimpleMemoryReader](#) (uint32_t shm_key, uint32_t broadcast_key, std::size_t expectedEventCount=0)

Constructs a [SimpleMemoryReader](#).

- void [run](#) ()

Run until a null pointer is popped off of the RawEventQueue. Throws an exception if expectedEventCount is set and a different number of events come off the queue.

Exceptions

std::string	When the expectedEventCount is set and a different number of events come off the queue.
-------------	---

6.29.1 Detailed Description

[SimpleMemoryReader](#) will continue to read [RawEvent](#) objects off the queue until it encounters a null pointer, at which point it stops.

Definition at line 26 of file SimpleMemoryReader.hh.

6.29.2 Constructor & Destructor Documentation

6.29.2.1 `artdaq::SimpleMemoryReader::SimpleMemoryReader (uint32_t shm_key, uint32_t broadcast_key, std::size_t expectedEventCount = 0) [explicit]`

Constructs a [SimpleMemoryReader](#).

Parameters

<i>shm_key</i>	Key of the shared memory segment
<i>broadcast_key</i>	Key of the broadcast shared memory segment
<i>expectedEvent-Count</i>	The number of events the SimpleMemoryReader should expect

Definition at line 39 of file SimpleMemoryReader.cc.

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

- artdaq-core/artdaq-core/Core/SimpleMemoryReader.hh
- artdaq-core/artdaq-core/Core/SimpleMemoryReader.cc

6.30 artdaq::SimpleQueueReader Class Reference

[SimpleQueueReader](#) will continue to read [RawEvent](#) objects off the queue until it encounters a null pointer, at which point it stops.

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

Public Member Functions

- [SimpleQueueReader](#) (std::size_t expectedEventCount=0)

Constructs a [SimpleQueueReader](#).

- void [run](#) ()

Run until a null pointer is popped off of the RawEventQueue. Throws an exception if expectedEventCount is set and a different number of events come off the queue.

Exceptions

std::string	When the <code>expectedEventCount</code> is set and a different number of events come off the queue.
-------------	--

6.30.1 Detailed Description

[SimpleQueueReader](#) will continue to read [RawEvent](#) objects off the queue until it encounters a null pointer, at which point it stops.

Definition at line 26 of file `SimpleQueueReader.hh`.

6.30.2 Constructor & Destructor Documentation

6.30.2.1 `artdaq::SimpleQueueReader::SimpleQueueReader (std::size_t expectedEventCount = 0) [explicit]`

Constructs a [SimpleQueueReader](#).

Parameters

<i>expectedEvent-Count</i>	The number of events the SimpleQueueReader should expect
----------------------------	--

Definition at line 39 of file `SimpleQueueReader.cc`.

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

- `artdaq-core/artdaq-core/Core/SimpleQueueReader.hh`
- `artdaq-core/artdaq-core/Core/SimpleQueueReader.cc`

6.31 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](#).

6.31.1 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 23 of file `StatisticsCollection.hh`.

6.31.2 Member Function Documentation

6.31.2.1 void `artdaq::StatisticsCollection::addMonitoredQuantity (const std::string & name, MonitoredQuantityPtr mqPtr)`

Registers a new [MonitoredQuantity](#) to be tracked by the [StatisticsCollection](#).

Parameters

<i>name</i>	Name of the MonitoredQuantity (used for lookup)
<i>mqPtr</i>	shared_ptr to MonitoredQuantity

Definition at line 25 of file `StatisticsCollection.cc`.

6.31.2.2 [StatisticsCollection](#) & `artdaq::StatisticsCollection::getInstance ()` `[static]`

Returns the singleton instance of the [StatisticsCollection](#).

Returns

[StatisticsCollection](#) instance.

Definition at line 5 of file `StatisticsCollection.cc`.

6.31.2.3 [MonitoredQuantityPtr](#) `artdaq::StatisticsCollection::getMonitoredQuantity (const std::string & name) const`

Lookup and return a [MonitoredQuantity](#) from the [StatisticsCollection](#).

Parameters

<i>name</i>	Name of the MonitoredQuantity
-------------	---

Returns

[MonitoredQuantityPtr](#) (nullptr if not found in [StatisticsCollection](#))

Definition at line 33 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`

6.32 TraceLock Class 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](#) (std::mutex &mutex, int level, std::string description)
Construct a [TraceLock](#).
- virtual [~TraceLock](#) ()
Release the [TraceLock](#).

6.32.1 Detailed Description

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 10 of file `TraceLock.hh`.

6.32.2 Constructor & Destructor Documentation

6.32.2.1 `TraceLock::TraceLock (std::mutex & mutex, int level, std::string description)` `[inline]`

Construct a [TraceLock](#).

Parameters

<i>mutex</i>	Mutex to hold lock on
<i>level</i>	Level to TRACE (in the TraceLock TRACE_NAME)
<i>description</i>	Description of lock (to be printed in TRACE calls)

Definition at line 31 of file `TraceLock.hh`.

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

- `artdaq-core/artdaq-core/Utilities/TraceLock.hh`

Index

- ~ConcurrentQueue
 - artdaq::ConcurrentQueue, [26](#)
- addExternallyDroppedEvents
 - artdaq::ConcurrentQueue, [27](#)
- addFragment
 - artdaq::ContainerFragmentLoader, [38](#)
- addFragments
 - artdaq::ContainerFragmentLoader, [38](#)
- addMonitoredQuantity
 - artdaq::StatisticsCollection, [131](#)
- addSample
 - artdaq::MonitoredQuantity, [74](#), [75](#)
- ArgType
 - artdaq::SimpleLookupPolicy, [127](#)
- artdaq, [9](#)
 - configureMessageFacility, [13](#)
 - ExceptionHandler, [14](#)
 - ExceptionHandlerRethrow, [12](#)
 - FragmentPtr, [12](#)
 - fragmentSequenceIDCompare, [14](#)
 - generateMessageFacilityConfiguration, [15](#)
 - getGlobalQueue, [15](#)
 - makeFragmentGenerator, [15](#)
 - makeFunc_t, [12](#)
 - no, [12](#)
 - operator<<, [16](#)
 - RawDataType, [12](#)
 - RawEvent_ptr, [12](#)
 - setMsgFacAppName, [16](#)
 - SimpleMemoryReaderApp, [16](#)
 - simpleQueueReaderApp, [17](#)
 - yes, [12](#)
- artdaq::MonitoredQuantityStats
 - FULL, [80](#)
 - RECENT, [80](#)
- artdaq::SharedMemoryManager
 - Empty, [117](#)
 - Full, [117](#)
 - Reading, [117](#)
 - Writing, [117](#)
- artdaq::SimpleLookupPolicy
 - ENV_VAR, [127](#)
 - PATH_STRING, [127](#)
- artdaq::ConcurrentQueue
 - ~ConcurrentQueue, [26](#)
 - addExternallyDroppedEvents, [27](#)
 - capacity, [27](#)
 - clear, [27](#)
 - ConcurrentQueue, [26](#)
 - deqNowait, [27](#)
 - deqTimedWait, [27](#)
 - deqWait, [29](#)
 - empty, [29](#)
 - enqNowait, [29](#)
 - enqTimedWait, [30](#)
 - enqWait, [30](#)
 - full, [30](#)
 - getReadyTime, [30](#)
 - memory, [31](#)
 - queueReaderIsReady, [31](#)
 - setCapacity, [31](#)
 - setMemory, [31](#)
 - setReaderIsReady, [32](#)
 - size, [32](#)
 - used, [32](#)
- artdaq::ConcurrentQueue< T, EnqPolicy >, [25](#)
- artdaq::ContainerFragment, [32](#)
 - at, [34](#)
 - block_count, [34](#)
 - ContainerFragment, [34](#)
 - dataBegin, [34](#)
 - dataEnd, [34](#)
 - fragSize, [35](#)
 - fragment_type, [35](#)
 - fragmentIndex, [35](#)
 - lastFragmentIndex, [36](#)
 - metadata, [36](#)
 - missing_data, [36](#)
 - words_per_frag_word_, [37](#)
- artdaq::ContainerFragment::Metadata, [70](#)
- artdaq::ContainerFragmentLoader, [37](#)
 - addFragment, [38](#)
 - addFragments, [38](#)
 - ContainerFragmentLoader, [38](#)
 - metadata, [40](#)
 - set_fragment_type, [40](#)
 - set_missing_data, [40](#)
- artdaq::FaillFull
 - doEnq, [41](#)
 - doInsert, [42](#)

- artdaq::FailIfFull< T >, 40
- artdaq::FailIfFull< T >::QueuesFull, 85
- artdaq::FailIfFull::QueuesFull
 - what, 86
- artdaq::Fragment, 42
 - byte_t, 47
 - dataAddress, 49
 - dataBegin, 49
 - dataBeginBytes, 49
 - dataEnd, 50
 - dataEndBytes, 50
 - dataFrag, 51
 - dataSize, 51
 - dataSizeBytes, 52
 - empty, 52
 - eodFrag, 52
 - Fragment, 48
 - FragmentBytes, 52, 53
 - fragmentID, 53
 - hasMetadata, 53
 - headerAddress, 53
 - headerBegin, 54
 - headerBeginBytes, 54
 - isSystemFragmentType, 54
 - isUserFragmentType, 55
 - MakeSystemTypeMap, 55
 - metadata, 55
 - metadataAddress, 56
 - operator=, 56
 - print, 56
 - reinterpret_cast_checked, 57
 - reserve, 58
 - resize, 58
 - resizeBytes, 58
 - sequenceID, 59
 - setFragmentID, 59
 - setMetadata, 59
 - setSequenceID, 59
 - setSystemType, 60
 - setTimestamp, 60
 - setUserType, 60
 - size, 60
 - sizeBytes, 60
 - swap, 60, 62
 - timestamp, 62
 - type, 62
 - typeString, 62
 - updateMetadata, 62
 - version, 63
- artdaq::FragmentGenerator, 63
 - fragmentIDs, 64
 - getNext, 64
- artdaq::KeepNewest
 - doEnq, 69
 - doInsert, 69
- artdaq::KeepNewest< T >, 67
- artdaq::MonitoredQuantity, 73
 - addSample, 74, 75
 - calculateStatistics, 75
 - disable, 75
 - enable, 76
 - ExpectedCalculationInterval, 76
 - getCurrentTime, 76
 - getStats, 76
 - getTimeWindowForRecentResults, 76
 - isEnabled, 77
 - MonitoredQuantity, 74
 - reset, 77
 - setNewTimeWindowForRecentResults, 77
 - waitUntilAccumulatorsHaveBeenFlushed, 77
- artdaq::MonitoredQuantityStats, 78
 - DataSetType, 80
 - getDuration, 80
 - getLastSampleValue, 80
 - getLastValueRate, 81
 - getSampleCount, 81
 - getSampleLatency, 81
 - getSampleRate, 81
 - getValueAverage, 82
 - getValueMax, 82
 - getValueMin, 82
 - getValueRMS, 83
 - getValueRate, 82
 - getValueSum, 83
 - isEnabled, 83
- artdaq::PackageBuildInfo, 84
 - getBuildTimestamp, 84
 - getPackageName, 84
 - getPackageVersion, 84
 - setBuildTimestamp, 85
 - setPackageName, 85
 - setPackageVersion, 85
- artdaq::QuickVec
 - begin, 89
 - capacity, 89
 - Class_Version, 90
 - end, 90
 - erase, 90
 - insert, 91
 - operator=, 91
 - push_back, 92
 - QuickVec, 88, 89
 - reserve, 92
 - resize, 93
 - size, 93
 - swap, 93
- artdaq::QuickVec< TT_ >, 86
- artdaq::RawEvent, 94

- fragmentTypes, 95
- insertFragment, 95
- isComplete, 96
- numFragments, 96
- print, 96
- RawEvent, 95
- releaseProduct, 96
- runID, 97
- sequenceID, 97
- subrunID, 97
- wordCount, 97
- artdaq::RejectNewest
 - doEnq, 108
 - doInsert, 108
- artdaq::RejectNewest< T >, 107
- artdaq::SharedMemoryEventReceiver, 109
 - GetFragmentTypes, 110
 - GetFragmentsByType, 110
 - GetRank, 110
 - ReadHeader, 110
 - ReadReadyCount, 111
 - ReadyForRead, 111
 - SharedMemoryEventReceiver, 109
 - size, 111
 - toString, 111
- artdaq::SharedMemoryFragmentManager, 112
 - ReadFragment, 113
 - ReadFragmentData, 113
 - ReadFragmentHeader, 113
 - SharedMemoryFragmentManager, 113
 - WriteFragment, 114
- artdaq::SharedMemoryManager, 114
 - BufferDataSize, 117
 - BufferSemaphoreFlags, 117
 - CheckBuffer, 118
 - Detach, 118
 - FlagToString, 118
 - GetAttachedCount, 118
 - GetBufferCount, 119
 - GetBufferForReading, 119
 - GetBufferForWriting, 119
 - GetBufferStart, 119
 - GetBufferTimeout, 121
 - GetBuffersOwnedByManager, 119
 - GetKey, 121
 - GetLastSeenBufferID, 121
 - GetMyId, 121
 - GetRank, 121
 - GetReadPos, 122
 - GetWritePos, 122
 - IncrementReadPos, 122
 - IncrementWritePos, 122
 - IsValid, 123
 - MarkBufferEmpty, 123
 - MarkBufferFull, 123
 - MoreDataInBuffer, 123
 - Read, 123
 - ReadReadyCount, 124
 - ReadyForRead, 124
 - ReadyForWrite, 124
 - ResetBuffer, 124
 - ResetReadPos, 125
 - ResetWritePos, 125
 - SetMinWriteSize, 125
 - SetRank, 125
 - SharedMemoryManager, 117
 - size, 125
 - toString, 126
 - Write, 126
 - WriteReadyCount, 126
- artdaq::SimpleLookupPolicy, 127
 - ArgType, 127
 - operator(), 128
 - SimpleLookupPolicy, 128
- artdaq::SimpleMemoryReader, 128
 - SimpleMemoryReader, 129
- artdaq::SimpleQueueReader, 129
 - SimpleQueueReader, 130
- artdaq::StatisticsCollection, 130
 - addMonitoredQuantity, 131
 - getInstance, 131
 - getMonitoredQuantity, 131
- artdaq::TimeUtils, 19
 - convertUnixTimeToString, 20, 21
 - GetElapsedTime, 21
 - GetElapsedTimeMicroseconds, 21
 - GetElapsedTimeMilliseconds, 23
 - gettimeofday_us, 23
 - seconds, 20
- artdaq::detail, 17
 - memoryUsage, 18, 19
 - seconds, 18
- artdaq::detail::RawEventHeader, 98
 - RawEventHeader, 99
- artdaq::detail::RawFragmentHeader, 99
 - MakeSystemTypeMap, 101
 - MakeVerboseSystemTypeMap, 101
 - num_words, 102
 - setSystemType, 102
 - setUserType, 102
 - SystemTypeToString, 102
- artdaq::detail::RawFragmentHeaderV0, 103
 - MakeSystemTypeMap, 105
 - MakeVerboseSystemTypeMap, 105
 - num_words, 106
 - setSystemType, 106
 - setUserType, 106
 - upgrade, 106

- artdaq::detail::hasMemoryUsed< T >, 66
- artdaqcore, 23
- artdaqcore::GetPackageBuildInfo, 66
 - getPackageBuildInfo, 66
- artdaqtest::FragmentGeneratorTest, 64
 - fragmentIDs, 65
 - getNext, 65
- at
 - artdaq::ContainerFragment, 34
- begin
 - artdaq::QuickVec, 89
- block_count
 - artdaq::ContainerFragment, 34
- BufferDataSize
 - artdaq::SharedMemoryManager, 117
- BufferSemaphoreFlags
 - artdaq::SharedMemoryManager, 117
- byte_t
 - artdaq::Fragment, 47
- calculateStatistics
 - artdaq::MonitoredQuantity, 75
- capacity
 - artdaq::ConcurrentQueue, 27
 - artdaq::QuickVec, 89
- CheckBuffer
 - artdaq::SharedMemoryManager, 118
- Class_Version
 - artdaq::QuickVec, 90
- clear
 - artdaq::ConcurrentQueue, 27
- ConcurrentQueue
 - artdaq::ConcurrentQueue, 26
- configureMessageFacility
 - artdaq, 13
- ContainerFragment
 - artdaq::ContainerFragment, 34
- ContainerFragmentLoader
 - artdaq::ContainerFragmentLoader, 38
- convertUnixTimeToString
 - artdaq::TimeUtils, 20, 21
- dataAddress
 - artdaq::Fragment, 49
- dataBegin
 - artdaq::ContainerFragment, 34
 - artdaq::Fragment, 49
- dataBeginBytes
 - artdaq::Fragment, 49
- dataEnd
 - artdaq::ContainerFragment, 34
 - artdaq::Fragment, 50
- dataEndBytes
 - artdaq::Fragment, 50
- dataFrag
 - artdaq::Fragment, 51
- DataSetType
 - artdaq::MonitoredQuantityStats, 80
- dataSize
 - artdaq::Fragment, 51
- dataSizeBytes
 - artdaq::Fragment, 52
- deqNowait
 - artdaq::ConcurrentQueue, 27
- deqTimedWait
 - artdaq::ConcurrentQueue, 27
- deqWait
 - artdaq::ConcurrentQueue, 29
- Detach
 - artdaq::SharedMemoryManager, 118
- disable
 - artdaq::MonitoredQuantity, 75
- doEnq
 - artdaq::FailIfFull, 41
 - artdaq::KeepNewest, 69
 - artdaq::RejectNewest, 108
- doInsert
 - artdaq::FailIfFull, 42
 - artdaq::KeepNewest, 69
 - artdaq::RejectNewest, 108
- ENV_VAR
 - artdaq::SimpleLookupPolicy, 127
- Empty
 - artdaq::SharedMemoryManager, 117
- empty
 - artdaq::ConcurrentQueue, 29
 - artdaq::Fragment, 52
- enable
 - artdaq::MonitoredQuantity, 76
- end
 - artdaq::QuickVec, 90
- enqNowait
 - artdaq::ConcurrentQueue, 29
- enqTimedWait
 - artdaq::ConcurrentQueue, 30
- enqWait
 - artdaq::ConcurrentQueue, 30
- eodFrag
 - artdaq::Fragment, 52
- erase
 - artdaq::QuickVec, 90
- ExceptionHandler
 - artdaq, 14
- ExceptionHandlerRethrow
 - artdaq, 12
- ExpectedCalculationInterval
 - artdaq::MonitoredQuantity, 76

- FULL
 - artdaq::MonitoredQuantityStats, [80](#)
- field1
 - MetadataTypeOne, [71](#)
 - MetadataTypeTwo, [72](#)
- field2
 - MetadataTypeOne, [71](#)
 - MetadataTypeTwo, [72](#)
- field3
 - MetadataTypeOne, [71](#)
 - MetadataTypeTwo, [72](#)
- field4
 - MetadataTypeTwo, [72](#)
- field5
 - MetadataTypeTwo, [72](#)
- FlagToString
 - artdaq::SharedMemoryManager, [118](#)
- fragSize
 - artdaq::ContainerFragment, [35](#)
- Fragment
 - artdaq::Fragment, [48](#)
- fragment_type
 - artdaq::ContainerFragment, [35](#)
- FragmentBytes
 - artdaq::Fragment, [52, 53](#)
- fragmentID
 - artdaq::Fragment, [53](#)
- fragmentIDs
 - artdaq::FragmentGenerator, [64](#)
 - artdaqtest::FragmentGeneratorTest, [65](#)
- fragmentIndex
 - artdaq::ContainerFragment, [35](#)
- FragmentPtr
 - artdaq, [12](#)
- fragmentSequenceIDCompare
 - artdaq, [14](#)
- fragmentTypes
 - artdaq::RawEvent, [95](#)
- Full
 - artdaq::SharedMemoryManager, [117](#)
- full
 - artdaq::ConcurrentQueue, [30](#)
- generateMessageFacilityConfiguration
 - artdaq, [15](#)
- GetAttachedCount
 - artdaq::SharedMemoryManager, [118](#)
- GetBufferCount
 - artdaq::SharedMemoryManager, [119](#)
- GetBufferForReading
 - artdaq::SharedMemoryManager, [119](#)
- GetBufferForWriting
 - artdaq::SharedMemoryManager, [119](#)
- GetBufferStart
 - artdaq::SharedMemoryManager, [119](#)
- GetBufferTimeout
 - artdaq::SharedMemoryManager, [121](#)
- GetBuffersOwnedByManager
 - artdaq::SharedMemoryManager, [119](#)
- getBuildTimestamp
 - artdaq::PackageBuildInfo, [84](#)
- getCurrentTime
 - artdaq::MonitoredQuantity, [76](#)
- getDuration
 - artdaq::MonitoredQuantityStats, [80](#)
- GetElapsedTime
 - artdaq::TimeUtils, [21](#)
- GetElapsedTimeMicroseconds
 - artdaq::TimeUtils, [21](#)
- GetElapsedTimeMilliseconds
 - artdaq::TimeUtils, [23](#)
- GetFragmentTypes
 - artdaq::SharedMemoryEventReceiver, [110](#)
- GetFragmentsByType
 - artdaq::SharedMemoryEventReceiver, [110](#)
- getGlobalQueue
 - artdaq, [15](#)
- getInstance
 - artdaq::StatisticsCollection, [131](#)
- GetKey
 - artdaq::SharedMemoryManager, [121](#)
- getLastSampleValue
 - artdaq::MonitoredQuantityStats, [80](#)
- GetLastSeenBufferID
 - artdaq::SharedMemoryManager, [121](#)
- getLastValueRate
 - artdaq::MonitoredQuantityStats, [81](#)
- getMonitoredQuantity
 - artdaq::StatisticsCollection, [131](#)
- GetMyId
 - artdaq::SharedMemoryManager, [121](#)
- getNext
 - artdaq::FragmentGenerator, [64](#)
 - artdaqtest::FragmentGeneratorTest, [65](#)
- getPackageBuildInfo
 - artdaqcore::GetPackageBuildInfo, [66](#)
- getPackageName
 - artdaq::PackageBuildInfo, [84](#)
- getPackageVersion
 - artdaq::PackageBuildInfo, [84](#)
- GetRank
 - artdaq::SharedMemoryEventReceiver, [110](#)
 - artdaq::SharedMemoryManager, [121](#)
- GetReadPos
 - artdaq::SharedMemoryManager, [122](#)
- getReadyTime
 - artdaq::ConcurrentQueue, [30](#)
- getSampleCount

- artdaq::MonitoredQuantityStats, [81](#)
- getSampleLatency
 - artdaq::MonitoredQuantityStats, [81](#)
- getSampleRate
 - artdaq::MonitoredQuantityStats, [81](#)
- getStats
 - artdaq::MonitoredQuantity, [76](#)
- getTimeWindowForRecentResults
 - artdaq::MonitoredQuantity, [76](#)
- getValueAverage
 - artdaq::MonitoredQuantityStats, [82](#)
- getValueMax
 - artdaq::MonitoredQuantityStats, [82](#)
- getValueMin
 - artdaq::MonitoredQuantityStats, [82](#)
- getValueRMS
 - artdaq::MonitoredQuantityStats, [83](#)
- getValueRate
 - artdaq::MonitoredQuantityStats, [82](#)
- getValueSum
 - artdaq::MonitoredQuantityStats, [83](#)
- GetWritePos
 - artdaq::SharedMemoryManager, [122](#)
- gettimeofday_us
 - artdaq::TimeUtils, [23](#)
- hasMetadata
 - artdaq::Fragment, [53](#)
- headerAddress
 - artdaq::Fragment, [53](#)
- headerBegin
 - artdaq::Fragment, [54](#)
- headerBeginBytes
 - artdaq::Fragment, [54](#)
- IncrementReadPos
 - artdaq::SharedMemoryManager, [122](#)
- IncrementWritePos
 - artdaq::SharedMemoryManager, [122](#)
- insert
 - artdaq::QuickVec, [91](#)
- insertFragment
 - artdaq::RawEvent, [95](#)
- isComplete
 - artdaq::RawEvent, [96](#)
- isEnabled
 - artdaq::MonitoredQuantity, [77](#)
 - artdaq::MonitoredQuantityStats, [83](#)
- isSystemFragmentType
 - artdaq::Fragment, [54](#)
- isUserFragmentType
 - artdaq::Fragment, [55](#)
- IsValid
 - artdaq::SharedMemoryManager, [123](#)
- lastFragmentIndex
 - artdaq::ContainerFragment, [36](#)
- makeFragmentGenerator
 - artdaq, [15](#)
- makeFunc_t
 - artdaq, [12](#)
- MakeSystemTypeMap
 - artdaq::detail::RawFragmentHeader, [101](#)
 - artdaq::detail::RawFragmentHeaderV0, [105](#)
 - artdaq::Fragment, [55](#)
- MakeVerboseSystemTypeMap
 - artdaq::detail::RawFragmentHeader, [101](#)
 - artdaq::detail::RawFragmentHeaderV0, [105](#)
- MarkBufferEmpty
 - artdaq::SharedMemoryManager, [123](#)
- MarkBufferFull
 - artdaq::SharedMemoryManager, [123](#)
- memory
 - artdaq::ConcurrentQueue, [31](#)
- memoryUsage
 - artdaq::detail, [18](#), [19](#)
- metadata
 - artdaq::ContainerFragment, [36](#)
 - artdaq::ContainerFragmentLoader, [40](#)
 - artdaq::Fragment, [55](#)
- metadataAddress
 - artdaq::Fragment, [56](#)
- MetadataTypeHuge, [70](#)
- MetadataTypeOne, [71](#)
 - field1, [71](#)
 - field2, [71](#)
 - field3, [71](#)
- MetadataTypeTwo, [72](#)
 - field1, [72](#)
 - field2, [72](#)
 - field3, [72](#)
 - field4, [72](#)
 - field5, [72](#)
- missing_data
 - artdaq::ContainerFragment, [36](#)
- MonitoredQuantity
 - artdaq::MonitoredQuantity, [74](#)
- MoreDataInBuffer
 - artdaq::SharedMemoryManager, [123](#)
- no
 - artdaq, [12](#)
- num_words
 - artdaq::detail::RawFragmentHeader, [102](#)
 - artdaq::detail::RawFragmentHeaderV0, [106](#)
- numFragments
 - artdaq::RawEvent, [96](#)
- operator<<

- artdaq, 16
- operator()
 - artdaq::SimpleLookupPolicy, 128
- operator=
 - artdaq::Fragment, 56
 - artdaq::QuickVec, 91
- PATH_STRING
 - artdaq::SimpleLookupPolicy, 127
- print
 - artdaq::Fragment, 56
 - artdaq::RawEvent, 96
- push_back
 - artdaq::QuickVec, 92
- queueReaderIsReady
 - artdaq::ConcurrentQueue, 31
- QuickVec
 - artdaq::QuickVec, 88, 89
- RECENT
 - artdaq::MonitoredQuantityStats, 80
- RawDataType
 - artdaq, 12
- RawEvent
 - artdaq::RawEvent, 95
- RawEvent_ptr
 - artdaq, 12
- RawEventHeader
 - artdaq::detail::RawEventHeader, 99
- Read
 - artdaq::SharedMemoryManager, 123
- ReadFragment
 - artdaq::SharedMemoryFragmentManager, 113
- ReadFragmentData
 - artdaq::SharedMemoryFragmentManager, 113
- ReadFragmentHeader
 - artdaq::SharedMemoryFragmentManager, 113
- ReadHeader
 - artdaq::SharedMemoryEventReceiver, 110
- ReadReadyCount
 - artdaq::SharedMemoryEventReceiver, 111
 - artdaq::SharedMemoryManager, 124
- Reading
 - artdaq::SharedMemoryManager, 117
- ReadyForRead
 - artdaq::SharedMemoryEventReceiver, 111
 - artdaq::SharedMemoryManager, 124
- ReadyForWrite
 - artdaq::SharedMemoryManager, 124
- reinterpret_cast_checked
 - artdaq::Fragment, 57
- releaseProduct
 - artdaq::RawEvent, 96
- reserve
 - artdaq::Fragment, 58
 - artdaq::QuickVec, 92
- reset
 - artdaq::MonitoredQuantity, 77
- ResetBuffer
 - artdaq::SharedMemoryManager, 124
- ResetReadPos
 - artdaq::SharedMemoryManager, 125
- ResetWritePos
 - artdaq::SharedMemoryManager, 125
- resize
 - artdaq::Fragment, 58
 - artdaq::QuickVec, 93
- resizeBytes
 - artdaq::Fragment, 58
- runID
 - artdaq::RawEvent, 97
- seconds
 - artdaq::detail, 18
 - artdaq::TimeUtils, 20
- sequenceID
 - artdaq::Fragment, 59
 - artdaq::RawEvent, 97
- set_fragment_type
 - artdaq::ContainerFragmentLoader, 40
- set_missing_data
 - artdaq::ContainerFragmentLoader, 40
- setBuildTimestamp
 - artdaq::PackageBuildInfo, 85
- setCapacity
 - artdaq::ConcurrentQueue, 31
- setFragmentID
 - artdaq::Fragment, 59
- setMemory
 - artdaq::ConcurrentQueue, 31
- setMetadata
 - artdaq::Fragment, 59
- SetMinWriteSize
 - artdaq::SharedMemoryManager, 125
- setMsgFacAppName
 - artdaq, 16
- setNewTimeWindowForRecentResults
 - artdaq::MonitoredQuantity, 77
- setPackageName
 - artdaq::PackageBuildInfo, 85
- setPackageVersion
 - artdaq::PackageBuildInfo, 85
- SetRank
 - artdaq::SharedMemoryManager, 125
- setReaderIsReady
 - artdaq::ConcurrentQueue, 32
- setSequenceID
 - artdaq::Fragment, 59

- setSystemType
 - artdaq::detail::RawFragmentHeader, 102
 - artdaq::detail::RawFragmentHeaderV0, 106
 - artdaq::Fragment, 60
- setTimestamp
 - artdaq::Fragment, 60
- setUserType
 - artdaq::detail::RawFragmentHeader, 102
 - artdaq::detail::RawFragmentHeaderV0, 106
 - artdaq::Fragment, 60
- SharedMemoryEventReceiver
 - artdaq::SharedMemoryEventReceiver, 109
- SharedMemoryFragmentManager
 - artdaq::SharedMemoryFragmentManager, 113
- SharedMemoryManager
 - artdaq::SharedMemoryManager, 117
- SimpleLookupPolicy
 - artdaq::SimpleLookupPolicy, 128
- SimpleMemoryReader
 - artdaq::SimpleMemoryReader, 129
- SimpleMemoryReaderApp
 - artdaq, 16
- SimpleQueueReader
 - artdaq::SimpleQueueReader, 130
- simpleQueueReaderApp
 - artdaq, 17
- size
 - artdaq::ConcurrentQueue, 32
 - artdaq::Fragment, 60
 - artdaq::QuickVec, 93
 - artdaq::SharedMemoryEventReceiver, 111
 - artdaq::SharedMemoryManager, 125
- sizeBytes
 - artdaq::Fragment, 60
- subrunID
 - artdaq::RawEvent, 97
- swap
 - artdaq::Fragment, 60, 62
 - artdaq::QuickVec, 93
- SystemTypeToString
 - artdaq::detail::RawFragmentHeader, 102
- timestamp
 - artdaq::Fragment, 62
- toString
 - artdaq::SharedMemoryEventReceiver, 111
 - artdaq::SharedMemoryManager, 126
- TraceLock, 132
 - TraceLock, 132
 - TraceLock, 132
- type
 - artdaq::Fragment, 62
- typeString
 - artdaq::Fragment, 62
- updateMetadata
 - artdaq::Fragment, 62
- upgrade
 - artdaq::detail::RawFragmentHeaderV0, 106
- used
 - artdaq::ConcurrentQueue, 32
- version
 - artdaq::Fragment, 63
- waitUntilAccumulatorsHaveBeenFlushed
 - artdaq::MonitoredQuantity, 77
- what
 - artdaq::FailIfFull::QueuesFull, 86
- wordCount
 - artdaq::RawEvent, 97
- words_per_frag_word_
 - artdaq::ContainerFragment, 37
- Write
 - artdaq::SharedMemoryManager, 126
- WriteFragment
 - artdaq::SharedMemoryFragmentManager, 114
- WriteReadyCount
 - artdaq::SharedMemoryManager, 126
- Writing
 - artdaq::SharedMemoryManager, 117
- yes
 - artdaq, 12