

SW ROD

Generated by Doxygen 1.8.5

Fri Apr 30 2021 17:10:56

Contents

1	SW ROD	1
2	Hierarchical Index	15
2.1	Class Hierarchy	15
3	Class Index	16
3.1	Class List	16
4	Class Documentation	17
4.1	swrod::BadConfigurationException Class Reference	17
4.1.1	Detailed Description	17
4.2	swrod::Component Class Reference	18
4.2.1	Detailed Description	18
4.2.2	Member Function Documentation	18
4.3	swrod::Core Class Reference	19
4.3.1	Detailed Description	20
4.3.2	Constructor & Destructor Documentation	20
4.3.3	Member Function Documentation	20
4.4	swrod::CustomProcessingFramework Class Reference	23
4.4.1	Detailed Description	23
4.4.2	Constructor & Destructor Documentation	23
4.4.3	Member Function Documentation	23
4.5	swrod::CustomProcessor Class Reference	24
4.5.1	Detailed Description	25
4.5.2	Member Function Documentation	25
4.6	swrod::ROBFragment::DataBlock Class Reference	25
4.6.1	Detailed Description	26
4.6.2	Constructor & Destructor Documentation	26
4.6.3	Member Function Documentation	26
4.7	swrod::DataInput Class Reference	28
4.7.1	Detailed Description	28
4.7.2	Member Function Documentation	28
4.8	swrod::DataInputHandler Class Reference	31
4.8.1	Detailed Description	32
4.8.2	Member Function Documentation	32
4.9	swrod::DataInputHandlerBase Class Reference	33
4.9.1	Detailed Description	34
4.9.2	Constructor & Destructor Documentation	34
4.9.3	Member Function Documentation	35

4.10	swrod::Exception Class Reference	38
4.10.1	Detailed Description	38
4.11	swrod::Factory< Product > Class Template Reference	38
4.11.1	Detailed Description	39
4.12	swrod::helper::FragmentCollator Class Reference	39
4.12.1	Detailed Description	40
4.13	swrod::GetParameterException Class Reference	40
4.13.1	Detailed Description	40
4.14	swrod::ROBFragmentBuilderInterface::hash_compare Struct Reference	40
4.15	swrod::GBTChunk::Header Struct Reference	40
4.15.1	Detailed Description	40
4.16	swrod::IsUnique< T > Struct Template Reference	41
4.17	swrod::IsUnique< L1AInputHandler > Struct Template Reference	41
4.18	swrod::L1AInfo Struct Reference	41
4.18.1	Detailed Description	41
4.19	swrod::L1AInputHandler Class Reference	41
4.19.1	Detailed Description	42
4.19.2	Constructor & Destructor Documentation	42
4.19.3	Member Function Documentation	42
4.20	swrod::ROBFragmentBuilderInterface::L1ID Class Reference	43
4.21	swrod::DataInputHandlerBase::Link Struct Reference	43
4.21.1	Detailed Description	43
4.22	swrod::Factory< Product >::Registrator Struct Reference	43
4.22.1	Constructor & Destructor Documentation	44
4.23	swrod::ROBFragment Class Reference	45
4.23.1	Detailed Description	45
4.23.2	Constructor & Destructor Documentation	45
4.23.3	Member Function Documentation	46
4.24	swrod::ROBFragmentBuilder Class Reference	46
4.24.1	Detailed Description	47
4.24.2	Member Function Documentation	47
4.25	swrod::ROBFragmentBuilderInterface Class Reference	47
4.25.1	Detailed Description	49
4.25.2	Member Function Documentation	49
4.26	swrod::ROBFragmentConsumer Class Reference	51
4.26.1	Detailed Description	52
4.26.2	Member Function Documentation	52
4.27	swrod::ROBFragmentConsumerBase Class Reference	53
4.27.1	Detailed Description	54
4.27.2	Member Enumeration Documentation	54

4.27.3 Constructor & Destructor Documentation	55
4.27.4 Member Function Documentation	56
4.28 swrod::ROBFragmentProvider Class Reference	57
4.28.1 Detailed Description	57
4.28.2 Member Function Documentation	57
4.29 swrod::ROBFragmentWorkerBase Class Reference	58
4.29.1 Detailed Description	58
4.29.2 Constructor & Destructor Documentation	58

1 SW ROD

This package provides the implementation of the SW ROD readout application for the ATLAS TDAQ system that followed the [Phase-I SW ROD Architectural Analysis and Design](#) document, which in turn is based on the requirements collected from detector community, which are described in the [Phase-I SW ROD User Requirements](#) document. The package is distributed as part of the ATLAS TDAQ software release.

Introduction

The SW ROD is envisaged to act in the ATLAS data flow chain as the data-handling interface between the FELIX readout system and the ATLAS High-Level Trigger (HLT). The SW ROD implements ROB fragment building and formatting, which in the Run-1/2 systems were done by the detector specific Readout Driver (ROD) components. The SW ROD also furnishes the same buffering and HLT request processing capabilities as provided by the Readout System (ROS) of the legacy DAQ.

SW ROD Application

SW ROD functionality is provided by a number of homogeneous SW processes running on a cluster of commodity computers. All processes originate from the same binary executable, called **swrod_application**, but they diverge by using different configuration parameters such as a set of FELIX E-Links for receiving data, data processing algorithms, HLT request processing parameters and so on.

The SW ROD application is fully integrated with the TDAQ online infrastructure. It implements the Run Control Finite State Machine (FSM) and is configured using the standard TDAQ Configuration service. It also implements the Event Monitoring Sampler interface and publishes its operational statistics to the TDAQ Information Service.

Integrating Custom Processing into the SW ROD Application

The SW ROD Application provides a framework for executing detector specific code that is required for:

- extracting TTC information (L1ID and BCID) from input data packets;
- verifying the integrity of input data packets;
- applying custom processing to fully assembled ROB fragments.

These procedures have to be implemented by a custom detector specific shared library, which should provide three functions with well-defined signatures as described in the following sections. Such a library is advertised to the SW ROD Application via an object of the **SwRodCustomProcessingLib** OKS configuration class that shall define the names of the custom functions. Such an object has to be linked with each data channel, aka ROD, that is represented by an instance of the **SwRodDataChannel** OKS class.

NOTE: Custom functions must be declared using **extern "C"** specifier that guarantees that the functions run-time symbolic references will be the same as the function names.

TTC Information Extraction Function

A function that implements TTC information extraction is mandatory and shall be implemented along the lines demonstrated by the following example:

```
extern "C"
std::tuple<uint32_t, uint32_t, uint16_t> testTriggerInfoExtractor(const uint8_t * data, uint32_t size) {
    uint32_t type = *((uint32_t*)data) & 0x3;
    if (type == 3) {
        throw swrod::Exception(ERS_HERE, "Data packet is corrupt - incorrect packet type");
    }
    if (type == 1) {
        return std::tuple(
            *((uint8_t*)(data + 1)), // L1ID
            0xff, // L1ID size is 8 bits
            0xffff); // no BCID
    }
    else {
        return std::tuple(
            *((uint16_t*)(data + 2)), // L1ID
            0xffff, // L1ID size is 16 bits
            *((uint16_t*)data) >> 4); // BCID
    }
}
```

A TTC information extraction function has two parameters - a pointer to the memory block that contains data packet to be processed and the size of this data packet in bytes. The first parameter points to the beginning of the data packet payload meaning that any extra data like for example FELIX header or communication protocol header are omitted. This function has to return a 3-tuple with the values taken from the given data packet:

- 32-bit value that contains all available bits of the extended L1ID from the data packet.
- 32-bit mask that defines how many bits of extended L1ID are provided by the given data packet. The bits which are present in the current L1ID value shall be set to 1 in this mask, while all other bits have to be set to 0. For example if the data packet contains only the 8 least significant bits of the extended L1ID then the mask value has to be set to **0xFF**. If a full extended L1ID is present in the given packet then the mask has to be set to **0xFFFFFFFF**
- 16-bit BCID value from the given data packet. If no BCID is present in the data packet the value must be set to **0xFFFF**.

This function may also throw [swrod::Exception](#) if TTC information can not be reliably extracted from the given packet. Any implementation of the SW ROD fragment assembling algorithm has to handle such an exception gracefully. A custom TTC information extraction function may use existing [swrod::Exception](#) class that is declared in the [swrod/exceptions.h](#) file or it can declare a custom exception that must inherit from [swrod::Exception](#).

Data Integrity Check Function

Optionally a custom plugin library can provide a function that can be used to check integrity of a given data packet. If data packet format contains some kind of CRC value that was calculated using a known algorithm then such a function can apply the same algorithm to the given packet and compare its result with that value:

```
extern "C"
std::optional<bool> testDataIntegrityChecker(const uint8_t * data, uint32_t size) {
    if (!contains_checksum_value(packet)) {
        return std::nullopt; // packet has no checksum
    }
    else {
        return (calculate_checksum(packet) == get_checksum_value(packet));
    }
}
```

It is up to the fragment building algorithm implementation how to use this function. For performance reasons the default algorithm implementations call this function only in case of a TTC information mismatch between data and L1A packets.

Custom Processing

The last function that may be provided by the custom plugin library is also optional and should be implemented only if a detector specific processing has to be applied to the ROB fragments produced by the fragment building algorithm. In this case such a function has to provide a factory for the instances of a detector specific class that implements `swrod::CustomProcessor` interface. The following code shows a possible implementation of such a function:

```
extern "C"
swrod::CustomProcessor * createTestCustomProcessor(const boost::property_tree::ptree
    & config) {
    return new TestCustomProcessor(config.get<int64_t>("CustomLongIntAttribute"));
}
```

Note: The `config` object passed to this function contains configuration parameters declared in the corresponding instance of the `SwRodDataChannel` OKS class. It is possible to add arbitrary number of custom parameters to the `config` object by creating a new OKS class that inherits `SwRodDataChannel` and declares these parameters as its attributes. To get the value of a custom attribute one should use the `boost::property_tree::ptree::get()` function providing the corresponding attribute type as template parameter and the attribute name as the function argument.

A custom class that inherits the `swrod::CustomProcessor` interface has to implement the `processROBFragment(swrod::ROBFragment &)` pure virtual function. For scalability reasons SW ROD may create multiple instances of the `swrod::CustomProcessor` class as defined by the `WorkersNumber` attribute of the corresponding `SwRodCustomProcessor` configuration object. Each instance will be used in a dedicated thread that implies that a `swrod::CustomProcessor` interface implementation should not worry about thread safety unless it uses some global resources. It is guaranteed that each object of the custom class will be used by exactly one thread so there is no need to protect access to the object's local attributes.

The `processROBFragment(swrod::ROBFragment &)` function will be called for every ROB fragment produced by the fragment building algorithm of the corresponding ROB. The ROB fragment is passed to this function as a reference to an instance of the `swrod::ROBFragment` class. This class contains a number of constant attributes, which describe read-only properties of this fragment. There is also a number of mutable attributes, which can be freely modified by the function implementation, namely:

- **m_detector_type** - 32-bit value that will go to the *Detector Event Type* field of the ROD header
- **m_rod_minor_version** - 16-bit value that will go to the lower 2 bytes of the *Format Version Number* field of the ROD header
- **m_status_front** - if set to true the ROD fragment status words will be placed right after the ROD header, otherwise they will be put at the end of the ROD fragment. Default value is true.
- **m_status_words** - an originally empty vector of status words. Custom processor may add any number of 32-bit words to this vector. All these words will be added at the end of the ROD fragment header.
- **m_data** - this vector contains the data payload of the ROD fragment that may be split into a number of memory blocks. Normally this number is equal to the number of data receiving threads, but this is not always guaranteed as this may depend on a particular fragment building algorithm implementation. Each memory chunk is represented by an object of `swrod::ROBFragment::DataBlock` class that provides API to access and modify the data it contains. A format of the memory chunk depends on the fragment building algorithm that produced this ROB fragment. Detailed information about format of the data fragments produced by the default SW ROD algorithm implementations is given in the next chapters.

ROB Fragment Memory Management

A custom processing function may need to reformat the ROB fragment payload in a way that would increase the size of the data blocks. In such a case it is strongly recommended to set the corresponding `MaxMessageSize` parameter of the `SwRodFragmentBuilder` configuration object such that the memory blocks allocated by the algorithm will have enough space to accommodate the extra amount of data. The size of the memory blocks allocated by default algorithm implementations is equal to a product of the `MaxMessageSize` parameter and the number of input links

that have been used to build this memory block. If, by any reason, such configuration can not be done, then the custom processing function may allocate a new contiguous memory block of sufficient size using either the standard C++ **new** operator or a custom memory management routine and copy reformatted data into the new block. The following example shows how that can be done.

```
class TestCustomProcessor: public swrod::CustomProcessor {
public:
    explicit TestCustomProcessor(uint64_t tag) {
        m_tag = tag;
    }
    void processROBFragment(swrod::ROBFragment & fragment) override {
        fragment.m_status_words.push_back(0x0a);
        fragment.m_status_words.push_back(0x0b);
        fragment.m_status_words.push_back(0x0c);
        fragment.m_rod_minor_version = 15;
        fragment.m_detector_type = 0x222;
        fragment.m_status_front = false;

        for (auto & block : fragment.m_data) {
            swrod::GBTChunk::Header header{0, 0, 0, 0xffffffff};
            if (!block.append(header, &m_tag, sizeof(m_tag))) {
                // The memory block is not large enough to accommodate extra data
                // Allocate a new memory block of sufficient size, copy the
                // original data there and add the custom tag at the end

                // The size is in 4-byte words
                uint32_t size = block.dataSize() +
                    ((sizeof(m_tag) + sizeof(swrod::GBTChunk::Header))>>2);
                uint32_t * data = new uint32_t[size];
                memcpy(data, block.dataBegin(), block.dataSize());

                block = swrod::ROBFragment::DataBlock(data, size, block.
                    dataSize());
                block.append(header, &m_tag, sizeof(m_tag));
            }
        }
    }
private:
    uint64_t m_tag;
};
```

Custom processing function may also change the size of the **m_data** vector by either adding new memory blocks or removing the obsolete ones. If for example **m_data** contains multiple memory blocks that would have to be completely re shuffled, it may be more efficient to allocate a single memory block of sufficient size and copy the content of the original memory chunks into that block using a desired translation. Finally the original memory blocks shall be removed from the **m_data** vector and the new one shall be added to it.

Fragment Building Algorithms

A SW ROD fragment building algorithm implementation is supposed to collect data packets, which have the same L1 Trigger Identifier (L1ID), from a given set of input links and combine them along with the corresponding TTC information into a new object of the **swrod::ROBFragment** class. This class provides *serialize()* function that can be used to convert the **swrod::ROBFragment** data into a number of contiguous memory chunks, which contain a ROB event fragment that is formatted in accordance to the *ATLAS Event Format specification*. The high-level structure of a ROB fragment is shown in the following table.

Field	Description
ROB Header	Information is mostly taken from L1A packet
ROD Header	Some information is duplicated from the ROB header. Other information can be added by detector custom processing
ROD Data	The format is detector specific and may also depend on the fragment building algorithm implementation
ROD Trailer	Contains detector specific information that can be added by by detector custom processing

Data for the ROB and ROD headers are mostly taken from the Level 1 Trigger Accept (L1A) message that corresponds to the given L1ID and are normally independent of the algorithm implementation. The ROD data portion of the fragment contains an assembly of the data chunks received from the detector Front-End electronics, which

internally use detector specific format. The way in which these data chunks are combined depends on the algorithm implementation.

SW ROD package provides two fragment aggregation algorithms that can be used to handle data received from FELIX in the following ways:

- GBT mode algorithm aggregates data chunks from all e-links associated with the given `SwRodDataChannel` object into a single ROD Data block using chunks' L1IDs for alignment. The algorithm takes at most one data chunk from a given e-link for the same ROB fragment. Receiving more than one data chunk in a row with the same L1ID from the same e-link is considered an error. The algorithm expects that every individual data chunk has a size that is a multiple of 4 bytes. If this is not the case for a given data chunk then the algorithm adds padding zeros at the end of the chunk to make it 4-byte aligned.
- FULL mode algorithm treats every individual data chunk from any e-link associated with the given `SwRodDataChannel` as a completely built ROD Data, that may also optionally contain ROD Header and Trailer. This algorithm also expects that every incoming data chunk has a size that is multiple of 4 bytes. If this is not the case the algorithm adds padding zeros at the end of the chunk to make it 4-byte aligned.

A new custom algorithm can be implemented and added to the SW ROD as a plugin if required.

Both GBT and FULL mode algorithms can be used either in TTC-driven or in data-driven mode. For the latter no Level1 Accept packets are required by the algorithms. The mode of operation depends on the state of the **L1A-Handler** relationships of the **SwRodConfiguration** and **SwRodModule** configuration objects. If both relationships are empty then event builders will not expect to receive L1A messages and will run in data-driven mode. Otherwise they will subscribe for L1 Accept messages and use them for fragment building.

SwRodFragmentBuilder configuration class

This is an abstract class that defines a number of parameters that are common for any fragment building algorithm implementation. These parameters are:

- **BuildersNumber** - The number of threads that are notified when a fragment data aggregation is complete and take care of creating a new instance of `swrod::ROBFragment` class for this data and passing it to the ROB Fragment Consumers. These threads are used to disentangle fragment builders from fragment consumers and reduce a possible impact of slow consumers on fragment building performance. If **BuildersNumber** number is set to zero, then no building threads are created, in which case one of the fragment builder's working threads will execute the above procedure.
- **FlushBufferAtStop** - Defines fragment builder behavior at the Stop-of-Run command. If set to 1, the algorithm stops data processing immediately upon receiving the SoR command. The data which were present in the internal buffers are flushed. If set to 0, the algorithm keeps processing data from its internal buffers until they get empty.
- **L1AWaitTimeout** - Building a ROB fragment may require information from the corresponding L1 Accept packet. The timeout defines the maximum time in milliseconds to wait for L1A packet when some information from this packet is required. If the required L1A packet does not arrive within this timeout fragment building procedure continues using default values for the missing information.
- **MaxMessageSize** - The maximum size in bytes of a single data hunk that may arrive from an individual input link. The algorithm will use this value to preallocate memory blocks for the ROB fragments to be built. The size of these blocks is equal to a product of the **MaxMessageSize** and a number of input links which provide data to be aggregated to a single ROB fragment. A value of this parameter has to be carefully chosen. If it is too low this can cause ROB fragments truncation. If it is too high this may result in excessive memory footprint, which may affect the algorithm performance.
- **ReadyQueueSize** - The size of the queue that is used to hold references to completely aggregated data blocks. This queue is used by the building threads that are defined via **BuildersNumber** parameters. The building threads take the references from this queue and use them to create new instances of the `swrod::ROBFragment` class. If **BuildersNumber** is set to zero, this parameter is ignored as no queue will be used in this case.

- **ResynchTimeout** - This value defines time in milliseconds to wait for building of the ROB fragments that correspond to the last L1ID produced before the Trigger was put on hold. This timeout is used for the stopless recovery procedure to make sure that the fragment builder will be properly synchronized with the rest of the read-out system when the Trigger is resumed.

GBT Fragment Building Algorithm

The **ROD data** produced by this algorithm is split into a number of a contiguous memory blocks, with each block containing copies of data packets received from a subset of the E-Links associated with the given ROB in the SW ROD configuration. The number of such blocks is equal to the number of data receiving threads defined in the respective configuration via the **WorkersNumber** attribute of the **SwRodModule** class. By default the algorithm uses one reader thread for all the E-Links and therefore puts all data packets into a single memory block. A memory block contains a sequence of data packets received from the associated E-Links with each packet preceded with a meta-information header that has the following structure:

- **size** - 16-bit value that contains a total size (in 4-byte words) of the data packet including the size of the header itself.
- **felix_status** - 8-bit status of the data packet provided by FELIX
- **swrod_status** - 8 bit status of the data packet assigned by the fragment building algorithm. This status may contain a combination of the following flags:
 - **swrod::GBTChunk::Status::Ok (0)** - no errors detected for this packet
 - **swrod::GBTChunk::Status::Corrupt (1)** - the custom TTC information extraction has thrown exception for this packet
 - **swrod::GBTChunk::Status::CRCError (2)** - the custom data integrity checking function return false for this packet
 - **swrod::GBTChunk::Status::L1IdMismatch (4)** - the packets L1 ID does not match L1 ID from the L1A packet
 - **swrod::GBTChunk::Status::BCIdMismatch (8)** - the packets BC ID does not match BC ID from the L1A packet
- **link_id** - 32-bit detector resource ID that identifies the origin of the data packet

Note: The *link_id* field contains **DetectorResourceId** value that corresponds to the FELIX E-Link ID as defined by the respective instance of the **SwRodInputLink** OKS configuration class.

SwRodGBTModeBuilder configuration class

This OKS configuration class inherits from the **SwRodFragmentBuilder** and adds a few configuration parameters that are specific for the FULL mode algorithm, namely:

- **BufferSize** - Defines the maximum size of the main aggregation buffer in terms of a number of ROB fragments, which can be built simultaneously. This parameter also indirectly defines the data input timeout - the time to wait before terminating aggregation of a ROB fragment that misses chunks from one or more input links. The timeout value is not explicitly defined in the SW ROD configuration but can be estimated by dividing the **BufferSize** value by the average input data rate for the current data taking session:

`Timeout(seconds) = BufferSize / InputRate(Hz)`

- **MinimumBufferSize** - Defines the minimum size of the main aggregation buffer in terms of a number of ROB fragments. The buffer will always try to shrink itself to this value (but not beyond) whenever the number of concurrently built events goes down.
- **DropCorruptedPackets** - This parameter defines what to do with data chunks that can not be unanimously attributed to any ROB fragment due to containing corrupt data or arriving too late. If this parameter is set to 1 then such data chunks will be dropped. Otherwise the chunks will be assigned to the first of the currently being built ROB fragments, in which case the corresponding error bits, that explain the origin of the error, will be set to the **swrod_status** status word of the chunk's local header and a corresponding error bits will be set to status word of the ROB fragment header.

- **RecoveryDepth** - If an incoming data chunk contains L1ID and BCID values that don't match the algorithm's expectations, the algorithm will check if this is caused by a number of previous packets from the same input link been missed. For that it will try to match the chunk's L1ID and BCID to one of the known L1 Accept packets. This parameter defines a number of L1A packets to be checked as this is the only way to terminate this procedure if no match can be found.

Error Handling

The default GBT event aggregation algorithm implements error handling as described by the [SW ROD Error Use Cases](#) wiki page. The algorithm may produce incomplete ROB fragments, i.e. fragments that miss data packets from one or several E-Links. This may happen if data from these E-Links did not arrive within timeout, that is proportional to the algorithm's buffer size.

FULL Mode Fragment Building Algorithm

This algorithm receives data packets from the given set of E-Links defined in the corresponding SW ROD configuration. It assumes that each data packet contains a fully aggregated **ROD data** payload that may optionally include **ROD header** and **ROD trailer**. The latter is controlled by the **RODHeaderPresent** attribute of the **SwRodFullModeBuilder** configuration object. The algorithm combines these data packets with the corresponding TTC information into objects of the **swrod::ROBFragment** class.

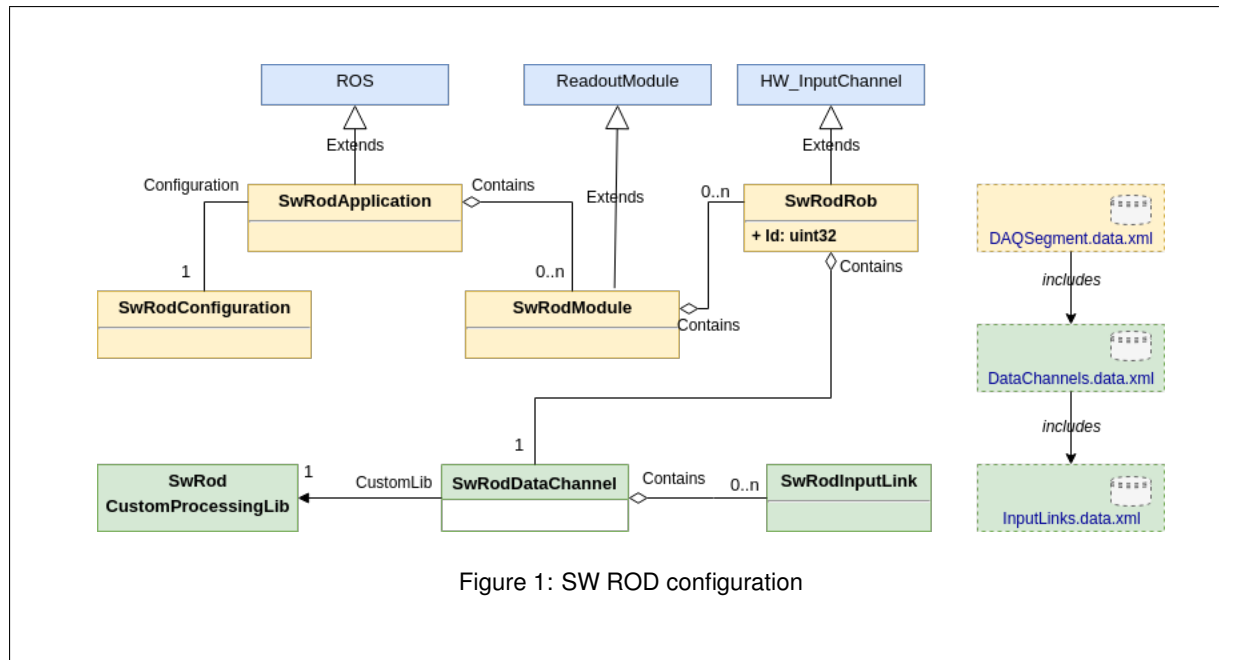
SwRodFullModeBuilder configuration class

This OKS configuration class inherits from the **SwRodFragmentBuilder** and adds the **RODHeaderPresent** configuration parameter. If this parameter is set to **1** then the algorithm assumes that each incoming data packet will already contain **ROD header** and **ROD trailer** and therefore will not add these pieces by itself. Otherwise the algorithm will generate both **ROD header** and **ROD trailer** for every incoming data packet.

Configuring a SW ROD Application

A SW ROD application has to be configured using OKS configuration service. For convenience all OKS classes that can be used for that have their names started with **SwRod** prefix. These classes are defined in the **daq/schema/swrod.schema.xml** OKS schema file that has to be included by any SW ROD OKS configuration file. A fully functional example of a SW ROD configuration can be found in the **data/SwRodSegment.data.xml** file located in the **swrod** package.

SW ROD configuration schema facilitates splitting of a SW ROD application configuration between a number of files, which can be maintained independently by detector and TDAQ experts. The aim is to make a clear distinction between detector specific part of the configuration and the common part that has to be maintained centrally by the TDAQ. The following diagram shows a recommended way of handling a SW ROD configuration. The green boxes in this diagram represent classes that should be instantiated in the detector specific part of configuration. The boxes with yellow background denote the classes which have to be used for creating the TDAQ portion of the SW ROD configuration that will be under responsibility of the TDAQ team. More details will be given in the following sections.



Note: **SwRodApplication**, **SwRodModule** and **SwRodRob** classes inherit from the legacy read-out configuration classes to make the new SW ROD configuration compatible with the legacy one.

Detector Specific Configuration

Detector experts are expected to maintain objects of three classes for a given SW ROD configuration. These classes are:

- **SwRodInputLink** is used to define a set of E-Links for receiving data
- **SwRodDataChannel** defines a mapping of E-Links to ROB fragments
- **SwRodCustomProcessingLib** class provides configuration of a detector specific custom processing plugin

SwRodInputLink class

This class is used to describe a set of input E-Links for a particular detector. It also implements the mapping of FELIX-based E-Link IDs to detector specific Resource IDs. This class has three attributes:

- **FelixId** - ID of this link as defined by FELIX system. Such an ID shall be unique within ATLAS.
- **DetectorResourceId** - ID of the detector read-out element that is connected to this FELIX link. Detector ID shall be unique for a given sub-detector.
- **DetectorResourceName** - human-readable name of the detector read-out element

It is recommended to put all instances of this class into a dedicate OKS configuration file (or a set of files), which then can be effectively shared by different configurations.

SwRodDataChannel class

An object of this class defines a set of input links for a given ATLAS data channel (ROD) as well as a custom processing plugin that has to be used for this channel. It has the following relationships:

- **Contains** - this relationship is inherited from **ResourceSetAND** class and has to contain references to the objects of the **SwRodInputLink** class that represent the corresponding E-Links
- **CustomLib** - a link to an instance of the **SwRodCustomProcessingLib** class

It is recommended to put all instances of this class into a separate OKS configuration file, which has to include files defining objects of the **SwRodInputLink** class.

SwRodCustomProcessingLib class

This class should be used for configuring custom detector specific plugins for the SW ROD. It declares the following attributes:

- **LibraryName** - the name of the shared library that implements the custom detector specific functions.
- **TriggerInfoExtractor** - a name of the mandatory trigger information extraction function.
- **DataIntegrityChecker** - a name of the optional data integrity checking function. If the custom library does not provide this function then this attribute shall be left empty.
- **ProcessorFactory** - a name of the optional custom processor factory function. If the custom library does not provide this function then this attribute shall be left empty.

Each instance of the **SwRodDataChannel** class has to be linked with an appropriate instance of the **SwRodCustomProcessingLib** to provide the necessary information for data processing. It is recommended to keep an instance of this class in the same OKS configuration file that declares the corresponding data channels.

TDAQ Specific Configuration

TDAQ configuration defines a number of TDAQ specific parameters for all SW ROD applications, in particular:

- computers where the SW ROD applications will be running
- a mapping of the data channels (RODs) to the SW ROD applications
- HLT request handling and Event Monitoring configuration parameters

SwRodApplication Class

An instance of the **SwRodApplication** class is an entry point to a SW ROD configuration. It serves multiple purposes:

- defines the standard Run Control Application parameters for the **swrod_application** process
- points to an instance of the **SwRodConfiguration** class that defines the TDAQ specific portion of the SW ROD configuration
- contains a set of **SwRodRob** objects which are used to associate ATLAS ROBs with the corresponding instances of the **SwRodDataChannel** class that are declared by the detector specific portion of the SW ROD configuration

SwRodConfiguration Class

An instance of the **SwRodConfiguration** class provides the following parameters:

- **Plugins** - a list of **SwRodPluginLib** objects, which reference shared libraries that provide implementation of the SW ROD interfaces. By default this list should contain a reference to the **libswrod_core_impl.so** library, which provides default implementations of these interfaces.
- **L1AHandler** - a pointer to an instance of a class that inherits **SwRodL1AInputHandler** interface. By default this relationship should point to an instance of the **SwRodDefaultL1AHandler** class, which provides default implementation of L1 Accept message handler.
- **Consumers** - a list of objects implementing **SwRodFragmentConsumer** interface. Consumers from this list will get all fragments for all ROBs produced by the current SW ROD application. Note that the order of objects in this list matters. ROB fragments will be passed to the consumers in the same order as they are linked to the **SwRodConfiguration** instance. For example if the **SwRodEventSampler** consumer precedes the **SwRodCustomProcessor** one then any monitoring task connected to the current SW ROD application will ROB fragments without custom processing been applied.
- **InputMethod** - a reference to an object implementing **SwRodInputMethod** interface. For getting data from FELIX one should reference to an instance of the **SwRodNetioInput** or **SwRodNetioNextInput** class depending on the version of the FELIX software being used.

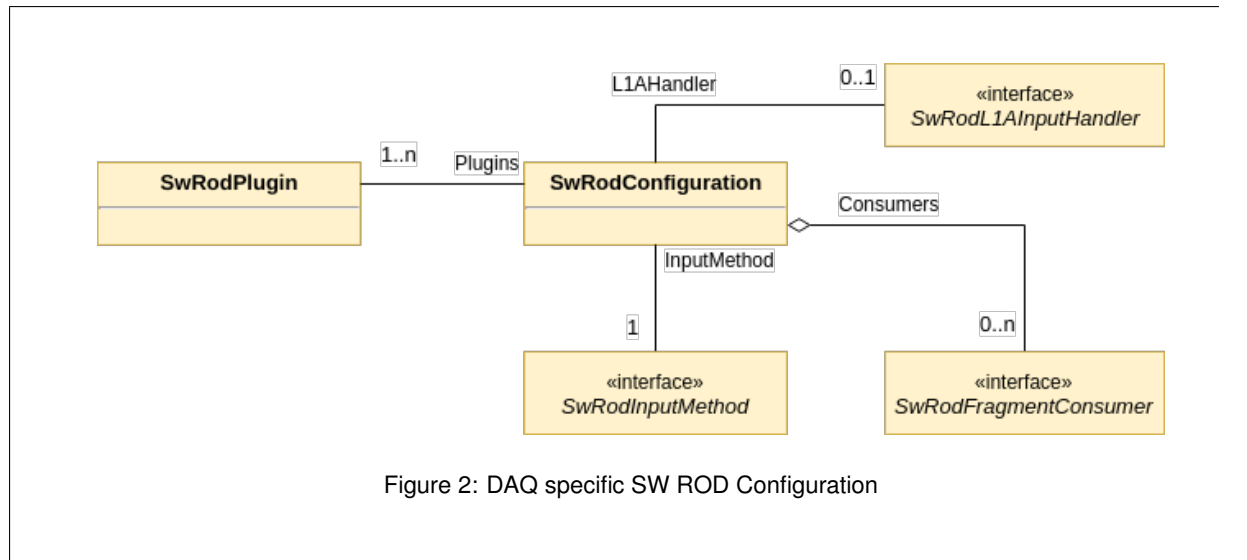


Figure 2: DAQ specific SW ROD Configuration

SwRodFragmentConsumer interface implementations

SwRodFragmentConsumer is an abstract base class for any resource that deals with fully built ROB fragments. It declares three attributes:

- **Type** - this is a string ID of a specific fragment builder type that is used to create an instance of the respective consumer at run time. Each sub-class of the **SwRodFragmentConsumer** shall provide an appropriate type name to be used for its instantiation.
- **WorkersNumber** - number of worker threads for this consumer
- **CPU** - affinity of the worker threads of this consumer will be set to the given CPU cores. This is a string parameter that contains numbers separated by commas and may include ranges. For example: 0,5,7,9-11.

SW ROD provides multiple default implementations of the **SwRodFragmentConsumer** interface that can be configured by using the following OKS classes:

- **SwRodFileWriter** - an instance of this class writes data produced by the SW ROD application to a standard ATLAS raw data file in a form of standard ATLAS events that combine fragments from all ROB handled by this SW ROD application based on their L1IDs.
- **SwRodHLTRequestHandler** - an instance of this class responds to the standard HLT data and clear requests in the same way as ROS. From the HLT point of view a SW ROD application is indistinguishable from a ROS one.
- **SwRodEventSampler** - can be used to create an instance of Event Sampler for the given SW ROD application. Event Sampler collates fragments from all ROB handled by this SW ROD application into a single ATLAS event based in their L1IDs and serves such events to a monitoring application via the standard TDAQ Event Monitoring interface. The monitoring application can connect to the Event Sampler using its *Sampler-Type* and *SamplerName* attributes. The **SamplerType** attribute for the SW ROD Event Sampler is always set to the "SWROD" string while the **SamplerName** attribute is set to the ID of the SW ROD application (i.e. the ID of the corresponding **SwRodApplication** instance in the OKS configuration).
- **SwRodCustomProcessor** - an instance of this class applies detector specific custom processing to ROB fragments.

Fragment Consumers can be attached to the SW ROD application at two levels: per ROB via **SwRodRob::Consumers** relationship as well as per application via the **SwRodConfiguration::Consumers** relationship. However there are some restrictions that should be taken into account when configuring consumers. They are summarized in the following table.

SwRodConsumer	SwRodRob	SwRodConfiguration
SwRodFileWriter	Yes	Yes
SwRodHLTRequestHandler	Yes	No
SwRodEventSampler	No	Yes, limited to one instance per SW ROD
SwRodCustomProcessor	Yes	No

SwRodInputMethod interface implementations

SwRodInputMethod is an abstract class that defines interface for getting input data into SW ROD. SW ROD provides several implementations of this interface:

- **SwRodInternalDataGenerator** - in memory data generator that is used for tests
- **SwRodNetioInput** - is used to get data via NetIO API.

The latter has a number of attributes which can be used to configure NetIO communication protocol

- **Backend** - it has to be either `_"posix_"` or `"fi_verbs"` which define low level communication protocol to be used by NetIO
- **BufferPages** - a number of buffer pages
- **BufferPageSize** - size of a buffer page in bytes
- **FelixBusGroupName** - FELIX Bus group name
- **FelixBusInterface** - name of the network interface which is used for communication with the FFELIX Bus service. Default value is `ZSYS_INTERFACE`, in which case the actual network name will be taken from `ZSYS_INTERFACE` environment variable.
- **FelixBusTimeout** - time to wait (in milliseconds) for a link ID resolution in the FelixBus.

SwRodRob Class

As it was explained earlier the objects of this classes are referenced by a **SwRodApplication** instance via a **SwRodModule** object. Each instance of the **SwRodRob** class provides configuration for a specific ROB or in another words defines how to produce ATLAS event fragments for a given slice of the detector. This class has the following parameters:

- **Id** - ROB identifier that must be unique across all SW ROD and ROS systems. This ID is used by HLT for requesting the corresponding ROB fragments.
- **FragmentBuilder** - a pointer to an instance of a class that inherits **SwRodFragmentBuilder** interface. Such an object provides configuration to the given data assembling algorithm, which builds ROB fragments from incoming data packets.
- **Consumers** - a list of objects implementing **SwRodFragmentConsumer** interface. Consumers from this list will get all fragments for this ROB immediately after they are produced.
- **Contains** - this relationship is inherited from the **ResourceSetAND** class and is used to reference an instance of the **SwRodDataChannel** class. This relationship provides the sole link between the detector and TDAQ specific portions of the SW ROD application configuration.

Note: For compatibility with the legacy read-out system configuration **Contains** is a multi-value relationship that potentially could reference more than one **Resource** object. However for a valid SW ROD configuration this relationship must point to exactly one unique **SwRodDataChannel** instance.

SwRodModule Class

An object of this class provides input data configuration for a given set of **SwRodRob** objects, which are referenced via its **Contains** relationship. This class has the following parameters:

- **WorkersNumber** - number of **DataInput** objects created for this module. Each input object will use a dedicated thread for receiving input data.
- **CPU** - a string that defines affinity for the data receiving threads. This string contains CPU numbers separated by commas and may optionally include ranges. For example: 0,5,7,9-11. If this string is empty then no affinity will be assigned.
- **L1AHandler** - if this relationship is not empty then the referenced object will be used to create an private receiver of L1 Accept data to be used exclusively by the ROB's that belong to this **SwRodModule**. If this relationship is empty then these ROB's will use the global L1 Accept receiver that is created from the object referenced by the **L1AHandler** relationship of the **SwRodConfiguration** instance. If that relationship is empty as well then the fragment building algorithms for the ROB's belonging to this model will operate in data-driven mode.
- **Contains** - this relationship is inherited from the **ResourceSetAND** class and is used to reference objects of the **SwRodRob** class. The fragment builders of the corresponding ROB's will share the same input objects that are created with respect to the configuration provided by this **SwRodModule** class instance.

Customizing SW ROD application

SW ROD declares a number of abstract interfaces which define interactions between its internal components:

- **DataInput** interface can be used for reading input data from different sources, e.g. from FELIX
- **ROBFragmentBuilder** interface can be used for implementing assembling of ROB fragments from the data packets received via **DataInput** interface
- **ROBFragmentConsumer** interface can be used for implementing post-processing of fully assembled ROB fragments

Default implementations of these interfaces are provided by the **libswrod_core_impl.so** library and can be used out of the box. SW ROD also provides a way of integrating any number of custom implementations of these interfaces into the standard SW ROD application.

Providing a Custom Interface Implementation

The following example shows how a custom **ROBFragmentConsumer** can be implemented.

```
class ROBFragmentCounter : public swrod::ROBFragmentConsumer {
public:
    ROBFragmentCounter(const boost::property_tree::ptree & config, const
        swrod::Core & core)
        : m_counter(0),
          m_ROB_id(-1) {
        m_output_frequency = config.get<uint32_t>("OutputFrequency");
        if (config.count("RobConfig")) {
            m_ROB_id = config.get<uint32_t>("RobConfig.Id");
        }
    }

    void insertROBFragment(const std::shared_ptr<swrod::ROBFragment> & fragment) override
    {
        if ((++m_counter % m_output_frequency) == 0) {
            std::cout << m_counter << " fragments have been built for ROB " << m_ROB_id << std::endl;
        };
        forwardROBFragment(fragment);
    }

    void runStarted(const RunParams & run_params) {
        m_counter = 0;
    }
private:
```

```
uint32_t m_output_frequency;
uint32_t m_ROB_id;
uint64_t m_counter;
};
```

This class implements a simple counter of ROB fragments. It can be used at the level of an individual ROB as well as at the level of a whole SW ROD application. Each time a new ROB fragment is produced it is passed to the instance of the **ROBFragmentCounter** class via the *insertROBFragment()* function. In this example the implementation of this function increments the fragment counter and then forwards the given ROB fragment to the other consumers by calling the *forwardROBFragment()* function. In addition every **m_output_frequency** fragments the function prints the fragment counter counter to the standard output. The value of the **m_output_frequency** parameter is taken from the OKS configuration. The next section explains how that was done.

Configuring Custom Interface Implementation

The **ROBFragmentCounter** class constructor gets a reference to the **ptree** object that represents parameters taken from the corresponding OKS class. For a new type of consumer a new OKS class that inherits from the **SwRodFragmentConsumer** has to be created. TO do this one should use the following procedure:

- run the OKS schema editor and create a new OKS schema file
- add include of the **daq/schema/swrod.schema.xml** OKS schema file into the new file
- create a new class **ROBFragmentCounter** inheriting it from the **SwRodFragmentConsumer** class
- add a new attribute called **OuputFrequency** to the new class
- save the new schema file

The new schema file has to be included by the SW ROD configuration. After that one can create a new instance of the **ROBFragmentCounter** class and add it either to a **SwRodRob** or to a **SwRodConfiguration** class instance depending on whether counting has to be done at the level of an individual ROB or for a whole SW ROD application.

Note: **ROBFragmentCounter** constructor implementation uses the "RobConfig" configuration object that is obtained by calling *config.get_child("RobConfig")* function on the given configuration object. Note that the "RobConfig" is available only if consumer configuration object was linked to an instance of the **SwRodRob** class via its *Contains* relationship. In this case the *_"RobConfig"_* parameter will contain the corresponding **SwRodRob** object configuration. If the consumer was attached to the **SwRodConfiguration** object, which means that it has to be used for all ROB of the given SW ROD application, the *_"RobConfig"_* configuration parameter will not be set and an attempt to call the *boost::property_tree::ptree::get_child("RobConfig")* function will yield an exception.

Registering Custom Interface Implementation with the SW ROD

Finally the SW ROD has to be made aware of the new interface implementation in order to be able to use it at run time. To achieve this the corresponding interface implementation class has to be registered with the **swrod::Core** singleton as shown in the following example.

```
using namespace swrod;

namespace {
    Factory<ROBFragmentConsumer>::Registrator __reg_custom plugin_
    (
        "ROBFragmentCounter",
        [] (const boost::property_tree::ptree& config, const Core& core) {
            return std::make_shared<ROBFragmentCounter>(config, core);
        });
}
```

This code creates a new factory object that will be used for producing new instances of the **ROBFragmentCounter** class. The factory will be associated with the "ROBFragmentCounter" name. This name will be used in the OKS configuration as described in the next section. Note that the base class of the new component (**ROBFragmentConsumer**) must be used as template parameter of the **Factory** class.

This code has to be compiled and linked together with the **ROBFragmentConsumer** class implementation into a shared library that can be dynamically loaded by the SW ROD application. Let's assume that such a library is called **libswrod_custom_test.so**. To make this library known to the SW ROD application a new instance of the **SwRodPluginLib** class has to be created in the corresponding OKS configuration and the new shared library name has to be set to its **LibraryName** attribute. One can either use a full path-name of the shared library or use a short file-name and put the library location to the **LD_LIBRARY_PATH** environment variable. Finally the new instance of the **SwRodPluginLib** class has to be linked with the **SwRodConfiguration** object via the **Plugins** relationship.

Testing SW ROD Custom Processing library

A custom implementation of the **DataInput** interface can be used to validate detector specific custom processing plugins. This chapter explains how that can be done.

Implementing internal data generator for SW ROD

The simplest way of providing custom input to SW ROD is to implement internal data generator that produces data with desired formatting in memory of the SW ROD application. The **swrod** package contains an example of such generator in **test/core/InternalDataGenerator.h(cpp)** files. One can customize internal data generation by declaring a new class that inherits **swrod::test::InternalDataGenerator** and overrides the **generatePacket(InputLinkId link, uint32_t l1id, uint16_t bcid)** virtual function. This function is called for every new packet to be produced and has to make a new packet and pass it to the **dataReceived(InputLinkId link, const uint8_t * data, uint32_t size, uint8_t status)** function. The following example shows how this can be done.

```
void MyDataGenerator::generatePacket(InputLinkId link, uint32_t l1id, uint16_t bcid) {
    // For efficiency m_packet memory block had been preallocated in the constructor
    // Here we just calculate the size of the new packet.
    // It must not exceed the size of the m_packet memory block
    uint32_t new_packet_size = ...;

    // set TTC values to the appropriate places of the new packet, for example
    *((uint32_t*) (m_packet + 2)) = l1id;
    *((uint16_t*) (m_packet + 6)) = bcid;

    dataReceived(link, m_packet, new_packet_size, 0);
}
```

Note that if a custom implementation produces packets of fixed size it can calculate packet size only once in the constructor. A custom implementation that needs to generate packets of varying size has to calculate a new packet size each time a new packet is produced. Finally the new packet has to be passed to the **dataReceived()** function that in turn will pass it to the SW ROD fragment builder.

The **swrod::test::InternalDataGenerator** class provides another virtual function that is named **beforeStart()**. This function is called each time a new run is about to be started and can be used to reset internal counters of the custom data generator.

The new data generator has to be advertised to the SW ROD by creating and registering a new object factory as shown by the following example.

```
namespace {
    Factory<DataInput>::Registrar __reg__(
        "MyDataGenerator",
        [] (const boost::property_tree::ptree& config, const Core& ) {
            return std::make_shared<MyDataGenerator>(config);
        });
}
```

Finally the new generator class has to be compiled and linked to a new shared library and the library name has to be set to the **LibraryName** attribute of the new instance of the **SwRodPluginLib** object created in OKS configuration. This OKS object has to be linked with the **SwRodConfiguration** via the **Plugins** relationship. This will make the new input method implementation known to the SW ROD application.

In order to use the new generator a new instance of the **SwRodInternalDataGenerator** class has to be created and its **Type** attribute has to be set to the "MyDataGenerator" string, that was used for registering the generator class with the SW ROD plugins factory. Finally this object has to be linked with the **SwRodConfiguration** or **SwRodModule** via the **InputMethod** relationship.

Note: An internal data generator has to be used in conjunction with **InternalL1AGenerator** class provided by the **swrod** package. The **InternalL1AGenerator** produces L1A packets which are passed to the SW ROD fragment builder and are used as seeds for data packets generation guaranteeing coherence of the generated TTC and data packets.

Using Felix Emulator application

A SW ROD application can be tested by getting input data via network like this is done for real data taking. This can be achieved by using the **FelixEmulator** application that can send data produced by internal data generator via NetIO protocol to a given SW ROD application. This can be done by creating a new **FelixEmulator Segment** object and associating it with the SW ROD application that has to be tested. The **swrod** package contains an example of such a segment in the **data/FelixEmulatorSegment.data.xml** file. Here are some important high-lights of the **FelixEmulator** configuration:

- **FelixEmulator** is an instance of the **SwRodApplication** class that uses a special **DataInput** interface implementation provided by the **libswrod_felix_emulator.so** library. This is achieved by linking the **FelixEmulator-Implementation** object located in the **daq/sw/swrod-common.data.xml** file with the **SwRodConfiguration** object that is used the **FelixEmulator** application.
- **FelixEmulator** object has to be linked with all instances of the **SwRodModule** class that are used by the SW ROD application, for which the emulated input has to be produced.
- **FelixEmulator** application has to use internal data generator that can produce data with the format expected by the SW ROD to be tested. See the previous chapter for an explanation of how to implement a custom input generator.
- **FelixEmulator** object has to be linked with a **Variable** object that must provide a value of **ZSYS_INTERFACE** environment variable that is appropriate for the machine where the **FelixEmulator** application will be running.
- **FelixEmulator** object has to be linked with a **Variable** object that must provide an initial port number to be used by the NetIO protocol for publishing data. The name of this environment variable must be set to **SWROD_FELIX_EMULATOR_PORT**.

2 Hierarchical Index

2.1 Class Hierarchy

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

swrod::BadConfigurationException	17
swrod::Component	18
swrod::CustomProcessor	24
swrod::DataInput	28
swrod::DataInputHandler	31
swrod::DataInputHandlerBase	33
swrod::ROBFragmentWorkerBase	58
swrod::L1AInputHandler	41
swrod::ROBFragmentBuilder	46
swrod::ROBFragmentBuilderBase	47

swrod::ROBFragmentConsumer	51
swrod::ROBFragmentConsumerBase	53
swrod::Core	19
swrod::CustomProcessingFramework	23
swrod::ROBFragment::DataBlock	25
swrod::Exception	38
swrod::Factory< Product >	38
swrod::helper::FragmentCollator	39
swrod::GetParameterException	40
swrod::ROBFragmentBuilderBase::hash_compare	40
swrod::GBTChunk::Header	40
swrod::IsUnique< T >	41
swrod::IsUnique< L1AInputHandler >	41
swrod::L1AInfo	41
swrod::ROBFragmentBuilderBase::L1ID	43
swrod::DataInputHandlerBase::Link	43
swrod::Factory< Product >::Registrator	43
swrod::ROBFragment	45
swrod::ROBFragmentProvider	57
swrod::ROBFragmentBuilder	46
swrod::ROBFragmentConsumer	51

3 Class Index

3.1 Class List

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

swrod::BadConfigurationException	17
swrod::Component	18
swrod::Core	19
swrod::CustomProcessingFramework	23
swrod::CustomProcessor	24
swrod::ROBFragment::DataBlock	25
swrod::DataInput	28

swrod::DataInputHandler	31
swrod::DataInputHandlerBase	33
swrod::Exception	38
swrod::Factory< Product >	38
swrod::helper::FragmentCollator	39
Helper class to collate ROBfragments belonging to the same L1 ID	
swrod::GetParameterException	40
swrod::ROBFragmentBuilderBase::hash_compare	40
swrod::GBTChunk::Header	40
swrod::IsUnique< T >	41
swrod::IsUnique< L1AInputHandler >	41
swrod::L1AInfo	41
swrod::L1AInputHandler	41
swrod::ROBFragmentBuilderBase::L1ID	43
swrod::DataInputHandlerBase::Link	43
swrod::Factory< Product >::Registrator	43
swrod::ROBFragment	45
swrod::ROBFragmentBuilder	46
swrod::ROBFragmentBuilderBase	47
swrod::ROBFragmentConsumer	51
swrod::ROBFragmentConsumerBase	53
swrod::ROBFragmentProvider	57
swrod::ROBFragmentWorkerBase	58

4 Class Documentation

4.1 swrod::BadConfigurationException Class Reference

```
#include <exceptions.h>
```

4.1.1 Detailed Description

This issue is used to report problems with SW ROD configuration

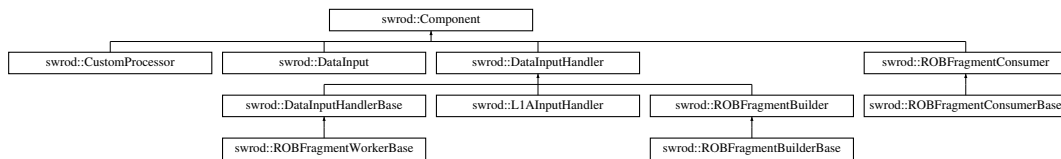
The documentation for this class was generated from the following file:

- swrod/exceptions.h

4.2 swrod::Component Class Reference

```
#include <Component.h>
```

Inheritance diagram for swrod::Component:



Public Member Functions

- virtual const std::string & [getName](#) () const =0
- virtual ISInfo * [getStatistics](#) ()
- virtual void [runStarted](#) (const RunParams &run_params)
- virtual void [runStopped](#) ()
- virtual void [userCommand](#) (const daq::rc::UserCmd &cmd)

4.2.1 Detailed Description

This class defines a basic interface for SW ROD a internal component.

4.2.2 Member Function Documentation

4.2.2.1 virtual const std::string& swrod::Component::getName () const [pure virtual]

A subclass shall override this function to return a string ID of the current object. This ID has to be unique in the scope of the current SW ROD application if [getStatistics\(\)](#) function of the current object returns non-null pointer to an IS information object. This will guarantee that this IS object can be published correctly. For objects that don't provide statistics to be published to IS this ID is not required to be unique.

Returns

The object ID

Implemented in [swrod::ROBFragmentConsumerBase](#), [swrod::DataInputHandlerBase](#), [swrod::ROBFragmentBuilderBase](#), [swrod::DataInput](#), and [swrod::CustomProcessor](#).

4.2.2.2 virtual ISInfo* swrod::Component::getStatistics () [inline],[virtual]

Returns monitoring statistics for this component. Default implementation returns null pointer.

Returns

A pointer to the IS statistics object.

Reimplemented in [swrod::ROBFragmentBuilderBase](#).

4.2.2.3 virtual void swrod::Component::runStarted (const RunParams & run_params) [inline],[virtual]

Called to inform this component that a new run is about to be started.

Parameters

<i>in</i>	<i>run_params</i>	The new run parameters.
-----------	-------------------	-------------------------

Reimplemented in [swrod::DataInputHandlerBase](#), [swrod::ROBFragmentBuilderBase](#), and [swrod::ROBFragmentConsumerBase](#).

4.2.2.4 virtual void swrod::Component::runStopped () [inline],[virtual]

Called to inform this component that the current run is being stopped.

Reimplemented in [swrod::DataInputHandlerBase](#), [swrod::ROBFragmentBuilderBase](#), and [swrod::ROBFragmentConsumerBase](#).

4.2.2.5 virtual void swrod::Component::userCommand (const daq::rc::UserCmd & cmd) [inline],[virtual]

Called to pass a custom Run Control command to this component.

Parameters

<i>in</i>	<i>cmd</i>	user defined Run Control command.
-----------	------------	-----------------------------------

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

- swrod/Component.h

4.3 swrod::Core Class Reference

```
#include <Core.h>
```

Public Types

- typedef std::vector< std::shared_ptr< [ROBFragmentBuilder](#) > > **FragmentBuilders**
- typedef std::vector< std::shared_ptr< [ROBFragmentConsumer](#) > > **FragmentConsumers**

Public Member Functions

- [Core](#) (const boost::property_tree::ptree &config)
- void [connectToFelix](#) ()
- void [disconnectFromFelix](#) ()
- void [disableInputLinks](#) (std::vector< InputLinkId > &link_ids)
- void [disableROBs](#) (std::vector< uint32_t > &ROB_ids)
- void [enableInputLinks](#) (std::vector< InputLinkId > &link_ids, uint32_t lastL1ID)
- void [enableROBs](#) (std::vector< uint32_t > &ROB_ids, uint32_t lastL1ID)
- const FragmentBuilders & [getBuilders](#) () const
- const boost::property_tree::ptree & [getConfiguration](#) () const
- const FragmentConsumers & [getConsumers](#) () const
- const std::vector< std::shared_ptr< [DataInput](#) > > & [getInputs](#) (uint32_t ROB_id) const
- const [CustomProcessingFramework](#) & [getProcessingFramework](#) () const
- std::vector< ISInfo * > [getStatistics](#) ()
- void [runStarted](#) (const RunParams &run_params)
- void [runStopped](#) ()
- void [resynchAfterRestart](#) (uint32_t lastL1ID)
- void [userCommand](#) (const daq::rc::UserCmd &cmd)

4.3.1 Detailed Description

This is a top level class of a SW ROD application. A SW ROD application shall create exactly one instance of this class with a desired configuration.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 `swrod::Core::Core (const boost::property_tree::ptree & config) [explicit]`

Creates a new instance of SW ROD using the given configuration.

Parameters

<i>in</i>	<i>config</i>	SW ROD configuration.
-----------	---------------	-----------------------

4.3.3 Member Function Documentation

4.3.3.1 `void swrod::Core::connectToFelix ()`

The [Core](#) instance passes this request to all implementations of the `DataInputHanler` interface instructing them to connect to the read-out system. If any connection attempt fails this function will throw a [swrod::Exception](#) exception. This function is expected to be called from the Run Control CONNECT transition.

4.3.3.2 `void swrod::Core::disableInputLinks (std::vector< InputLinkId > & link_ids)`

This function demands the [Core](#) instance to stop receiving data packets from the given input links. The [Core](#) passes this request to all implementations of the `DataInputHanler` interface which typically have to unsubscribe from these input links. The [Core](#) also informs all ROB Fragment Consumers that the given set of links has been disabled. This function can be called at any moment and does not require putting the Trigger on hold.

Parameters

<i>in, out</i>	<i>link_ids</i>	A vector of input link IDs to be disabled. The function may modify this vector by removing IDs of the links, for which recovery attempt has failed.
----------------	-----------------	---

4.3.3.3 `void swrod::Core::disableROBs (std::vector< uint32_t > & ROB_ids)`

Demands the [Core](#) instance to stop producing data fragments for the given ROB(s). The [Core](#) instance passes this request to the corresponding ROB Fragment Builder components, which typically unsubscribe from all input links in order to stop getting data. The [Core](#) also informs all ROB Fragment Consumers that the given set of ROBs has been disabled. This function can be called at any moment and does not require putting the Trigger on hold.

Parameters

<i>in, out</i>	<i>ROB_ids</i>	IDs of the ROBs that have to be disabled. The function may modify this vector by removing IDs of the ROBs, which couldn't be disabled.
----------------	----------------	--

4.3.3.4 `void swrod::Core::disconnectFromFelix ()`

The [Core](#) instance passes this request to all implementations of the `DataInputHanler` interface instructing them to disconnect from the read-out system. This function should be called from the Run Control DISCONNECT transition.

4.3.3.5 `void swrod::Core::enableInputLinks (std::vector< InputLinkId > & link_ids, uint32_t lastL1ID)`

This function shall be used to request the [Core](#) instance to start getting data from the input links that had been previously disabled. The [Core](#) passes this request to all implementations of the `DataInputHanler` interface which typically have to re subscribe to the given input links. The [Core](#) also informs all ROB Fragment Consumers that the given set of links has been re enabled. The Trigger must be put on hold before this function is called and therefore

no new data shall be produced until this function returns. A failure to respect this condition may result in a fatal crash of the SW ROD application.

Parameters

in, out	<i>link_ids</i>	A vector of input link IDs to be re enabled. The function may modify this vector by removing IDs of the links, for which recovery attempt has failed.
in	<i>lastL1ID</i>	The last L1ID that has been produced by the Trigger before it was put on hold.

4.3.3.6 void swrod::Core::enableROBs (std::vector< uint32_t > & *ROB_ids*, uint32_t *lastL1ID*)

Demands the [Core](#) instance to restart data fragments production for the given ROBs. The [Core](#) passes this request to the corresponding ROB Fragment Builders, which typically re subscribe to all input links in order to start getting data. The [Core](#) also informs all ROB Fragment Consumers that the given set of ROBs has been enabled. The Trigger must be put on hold before this function is called and therefore no new data shall be produced until this function returns. A failure to respect this condition may result in a fatal crash of the SW ROD application.

Parameters

in, out	<i>ROB_ids</i>	A vector of ROB IDs to be re enabled. The function can modify this vector by removing IDs of the ROBs, for which an attempt of recovery fails.
in	<i>lastL1ID</i>	The last L1ID that has been produced by the Trigger before it was put on hold.

4.3.3.7 const FragmentBuilders& swrod::Core::getBuilders () const [inline]

Returns a reference to the vector containing all ROB Fragment Builders for the current configuration.

Returns

A vector of [ROBFragmentBuilder](#) interface implementations.

4.3.3.8 const boost::property_tree::ptree& swrod::Core::getConfiguration () const [inline]

Returns a reference to the configuration object that was passed to the constructor of this [Core](#) instance.

Returns

The original configuration object that was used for creating this [Core](#) instance.

4.3.3.9 const FragmentConsumers& swrod::Core::getConsumers () const [inline]

Returns a reference to the vector containing all ROB Fragment Consumers for the current configuration.

Returns

A vector of [ROBFragmentConsumer](#) interface implementations.

4.3.3.10 const std::vector<std::shared_ptr<DataInput> >& swrod::Core::getInputs (uint32_t *ROB_id*) const

Returns input objects to be used for receiving input data for the given ROB.

Parameters

in	<i>ROB_id</i>	
----	---------------	--

Returns

A vector of input objects that are used for receiving input data for the given ROB.

Exceptions

<i>BadConfigurationException</i>	if no input objects are found for the given ROB ID
--	--

4.3.3.11 `const CustomProcessingFramework& swrod::Core::getProcessingFramework () const [inline]`

Returns a reference to the [`CustomProcessingFramework`](#) object.

Returns

A reference to the [`CustomProcessingFramework`](#) object.

4.3.3.12 `std::vector<ISInfo*> swrod::Core::getStatistics ()`

Returns a vector of pointers to monitoring statistics objects for all SW ROD internal components (ROB Fragment Builders and Fragment Consumers). It is guaranteed that this vector will contain no null pointers.

Returns

A vector of monitoring statistics objects.

4.3.3.13 `void swrod::Core::resynchAfterRestart (uint32_t lastL1ID)`

This function shall be used as part of TTC Restart procedure to re synch SW ROD Fragment Builders after the last restart of the SW ROD application. The Trigger must be put on hold before this function is called and therefore no new data shall be produced until this function returns. A failure to respect this condition may result in a fatal crash of the SW ROD application.

Parameters

<i>in</i>	<i>lastL1ID</i>	The last L1ID that has been produced when the Trigger was put on hold.
-----------	-----------------	--

4.3.3.14 `void swrod::Core::runStarted (const RunParams & run_params)`

This function has to be called when a new run is about to be started. It passes the new run parameters to all internal components.

Parameters

<i>in</i>	<i>run_params</i>	The new run parameters.
-----------	-------------------	-------------------------

4.3.3.15 `void swrod::Core::runStopped ()`

This function has to be called when the current run is being stopped. It passes this notification to all internal components.

4.3.3.16 `void swrod::Core::userCommand (const daq::rc::UserCmd & cmd)`

This function passes a given Run Control user command to all internal components (ROB Fragment Builders and Fragment Consumers).

Parameters

<i>in</i>	<i>cmd</i>	The Run Control user command.
-----------	------------	-------------------------------

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

- `swrod/Core.h`

4.4 swrod::CustomProcessingFramework Class Reference

```
#include <CustomProcessingFramework.h>
```

Public Types

- typedef std::tuple< uint32_t, uint32_t, uint16_t > (* **TriggerInfoExtractor**)(const uint8_t *, uint32_t)
- typedef std::optional< bool > (* **DataIntegrityChecker**)(const uint8_t *, uint32_t)

Public Member Functions

- [CustomProcessingFramework](#) (const boost::property_tree::ptree &config)
- TriggerInfoExtractor [getTriggerInfoExtractor](#) (uint32_t ROB_id) const
- DataIntegrityChecker [getDataIntegrityChecker](#) (uint32_t ROB_id) const noexcept
- std::unique_ptr< [CustomProcessor](#) > [createCustomProcessor](#) (uint32_t ROB_id, const boost::property_tree::ptree &config) const

4.4.1 Detailed Description

This class manages custom detector procedures provided by detector plugins.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 swrod::CustomProcessingFramework::CustomProcessingFramework (const boost::property_tree::ptree & *config*)
[explicit]

Creates a new instance of custom processing framework for the given configuration. This constructor loads all the plugins that are defined in the given configuration and resolves the custom functions using the names provided by the plugins configuration.

Parameters

<i>in</i>	<i>config</i>	SW ROD configuration.
-----------	---------------	-----------------------

4.4.3 Member Function Documentation

4.4.3.1 std::unique_ptr<CustomProcessor> swrod::CustomProcessingFramework::createCustomProcessor (uint32_t *ROB_id*, const boost::property_tree::ptree & *config*) const

Creates a new instance of the custom processing procedure for the given ROB. This procedure will be applied to all data fragments produced for the given ROB. Note that for scalability reasons multiple instances of custom processor can be created for the same ROB.

Parameters

<i>in</i>	<i>ROB_id</i>	The ID of the ROB for which a new custom processor has to be created.
<i>in</i>	<i>config</i>	Configuration to be passed to the new custom processor constructor.

Returns

A new custom processor instance.

Exceptions

<i>swrod::BadConfiguration-Exception</i>	This exception is thrown if no custom processor factory had been previously registered with the given ROB ID.
--	---

4.4.3.2 DataIntegrityChecker swrod::CustomProcessingFramework::getDataIntegrityChecker (uint32_t ROB_id) const [noexcept]

Returns a pointer to the data integrity check function for the given ROB. This is an optional function that can be implemented for a specific type of input data to be used to check a data packet of this type for corruption. If no data integrity check function is provided by the detector plugin, this function returns a pointer to the default integrity checker that always returns undefined result.

Parameters

in	<i>ROB_id</i>	The ID of the ROB for which the data integrity checking function has to be returned
----	---------------	---

Returns

Data integrity check function for the given ROB.

4.4.3.3 TriggerInfoExtractor swrod::CustomProcessingFramework::getTriggerInfoExtractor (uint32_t ROB_id) const

Returns a pointer to the trigger information extraction function for the given ROB. Such a function shall be implemented for each input data type and shall be capable of extracting L1ID and BCID values from a data packet of that type.

Parameters

in	<i>ROB_id</i>	The ID of the ROB for which the trigger information extraction function has to be returned
----	---------------	--

Returns

Trigger information extraction function for the given ROB.

Exceptions

<i>swrod::BadConfiguration-Exception</i>	This exception is thrown if no trigger information extraction function had been previously registered with the given ROB ID.
--	--

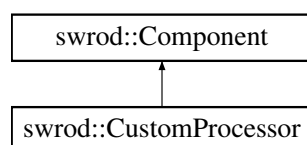
The documentation for this class was generated from the following file:

- swrod/CustomProcessingFramework.h

4.5 swrod::CustomProcessor Class Reference

```
#include <CustomProcessor.h>
```

Inheritance diagram for swrod::CustomProcessor:



Public Member Functions

- const std::string & [getName](#) () const override
- virtual void [linkDisabled](#) (const InputLinkId &link_id)
- virtual void [linkEnabled](#) (const InputLinkId &link_id)
- virtual void [processROBFragment](#) (ROBFragment &fragment)=0

4.5.1 Detailed Description

This is an abstract interface for detector specific processing of fully built ROB fragments.

4.5.2 Member Function Documentation

4.5.2.1 const std::string& swrod::CustomProcessor::getName () const [inline],[override],[virtual]

A subclass shall override this function to return a string ID of the current object. This ID has to be unique in the scope of the current SW ROD application if [getStatistics\(\)](#) function of the current object returns non-null pointer to an IS information object. This will guarantee that this IS object can be published correctly. For objects that don't provide statistics to be published to IS this ID is not required to be unique.

Returns

The object ID

Implements [swrod::Component](#).

4.5.2.2 virtual void swrod::CustomProcessor::linkDisabled (const InputLinkId &link_id) [inline],[virtual]

Called when an input link has been disabled.

Parameters

in	<i>link_id</i>	The ID of the input link that has been disabled.
----	----------------	--

4.5.2.3 virtual void swrod::CustomProcessor::linkEnabled (const InputLinkId &link_id) [inline],[virtual]

Called when an input link has been re enabled.

Parameters

in	<i>link_id</i>	The ID of the input link that has been re enabled.
----	----------------	--

4.5.2.4 virtual void swrod::CustomProcessor::processROBFragment (ROBFragment &fragment) [pure virtual]

This function is used to pass a fully built ROB fragment to this processor. An implementation of this method can modify any non-constant attribute of the [ROBFragment](#) object.

Parameters

in, out	<i>fragment</i>	ROB fragment to be processed.
---------	-----------------	-------------------------------

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

- swrod/CustomProcessor.h

4.6 swrod::ROBFragment::DataBlock Class Reference

```
#include <ROBFragment.h>
```

Public Member Functions

- `DataBlock` (`uint32_t memory_block[]=nullptr`, `uint32_t memory_size=0`, `uint32_t data_size=0`, `const std::function< void(uint32_t[])> &deleter=[](uint32_t d[]){delete[] d;}`)
- `bool append` (`GBTChunk::Header &header`, `const void *data`, `uint16_t size`)
- `uint32_t append` (`const void *data`, `uint32_t size`)
- `uint32_t * dataBegin` () `const`
- `uint32_t * dataEnd` () `const`
- `uint32_t dataSize` () `const`
- `uint32_t freeSpaceSize` () `const`
- `uint32_t memorySize` () `const`
- `void reset` ()

4.6.1 Detailed Description

This class implements memory management for ROB fragment data and provides API for data manipulation.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 `swrod::ROBFragment::DataBlock::DataBlock (uint32_t memory_block[] = nullptr, uint32_t memory_size = 0, uint32_t data_size = 0, const std::function< void(uint32_t[])> & deleter = [] (uint32_t d[]){delete[] d; }) [inline]`

Creates a new object that owns the given memory block. When this object is destroyed it deletes the memory block using the custom deleter given to this constructor.

Parameters

<i>memory_block</i>	A pointer to the memory block that will be owned by the new object.
<i>memory_size</i>	A size of the memory block in 4-byte words.
<i>data_size</i>	A size of the data that is already present in the given memory block in 4-byte words.
<i>deleter</i>	A function that will be used to delete the memory block. It will be called by the destructor of the new object.

4.6.3 Member Function Documentation

4.6.3.1 `bool swrod::ROBFragment::DataBlock::append (GBTChunk::Header & header, const void * data, uint16_t size) [inline]`

Copies the given header and data chunk to the internal buffer of the data block if the buffer has enough free space. Otherwise does nothing.

Parameters

<i>header</i>	chunk header to be copied to the buffer in front of the data chunk.
<i>data</i>	data chunk to be copied to the data block's internal buffer
<i>size</i>	number of bytes to copy

Returns

true if there was enough space in the buffer to accommodate the given header and data chunk, false otherwise. If false is returned the data block's internal buffer remains in the same state as it was before the function invocation.

4.6.3.2 `uint32_t swrod::ROBFragment::DataBlock::append (const void * data, uint32_t size) [inline]`

Copies the given data chunk to the data block internal buffer. If N is the amount of free space in the buffer and N is smaller than the given data chunk size then only the first N bytes of the data chunk will be copied.

Parameters

<i>data</i>	data chunk to be copied to the data block buffer
<i>size</i>	number of bytes to copy

Returns

the number of copied bytes

4.6.3.3 uint32_t* swrod::ROBFragment::DataBlock::dataBegin () const [inline]

Returns a pointer to the beginning of the memory block.

Returns

pointer to the beginning of data

4.6.3.4 uint32_t* swrod::ROBFragment::DataBlock::dataEnd () const [inline]

Returns a pointer to the end of data that is present in the memory block. This pointer points to the first byte after the last one used by the data. If no data is present the function returns the same pointer as [dataBegin\(\)](#)

Returns

end of data pointer

4.6.3.5 uint32_t swrod::ROBFragment::DataBlock::dataSize () const [inline]

Returns size of the data that is present in the memory block in 4-byte words. This size is equal to [dataEnd\(\)](#) - [dataBegin\(\)](#).

Returns

data size

4.6.3.6 uint32_t swrod::ROBFragment::DataBlock::freeSpaceSize () const [inline]

Returns size of the free space in the memory block in 4-byte words. This size is equal to [memorySize\(\)](#) - [dataSize\(\)](#)

Returns

free space size

4.6.3.7 uint32_t swrod::ROBFragment::DataBlock::memorySize () const [inline]

Returns size of the memory block in bytes.

Returns

memory block size

4.6.3.8 void swrod::ROBFragment::DataBlock::reset () [inline]

Sets data size to zero.

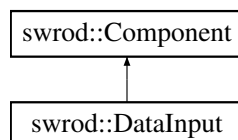
The documentation for this class was generated from the following file:

- swrod/ROBFragment.h

4.7 swrod::DataInput Class Reference

```
#include <DataInput.h>
```

Inheritance diagram for swrod::DataInput:



Public Types

- typedef detail::DataCallback **DataCallback**
- typedef std::function< void(void)> **Action**

Public Member Functions

- const std::string & [getName](#) () const override
- void [executeSynchronousAction](#) (const Action &action)
- void [subscribe](#) (const InputLinkId &link, const DataCallback &callback)
- void [unsubscribe](#) (const InputLinkId &link)

Protected Member Functions

- virtual void [actionsPending](#) ()=0
- virtual void [subscribeToFelix](#) (const InputLinkId &link_id)=0
- virtual void [unsubscribeFromFelix](#) (const InputLinkId &link_id)=0
- virtual uint32_t [translateStatus](#) (uint8_t status) const noexcept
- void [dataReceived](#) (InputLinkId link, const uint8_t *data, uint32_t size, uint8_t status)
- void [executeActions](#) ()

4.7.1 Detailed Description

This is an abstract class that defines an interface for receiving input data by SW ROD. An implementation of this interface has to pass all data it receives to the subscribers by calling the protected [DataInput::dataReceived\(\)](#) function. An implementation of this interface must always call this function from a single thread or serialise the calls if they are done from different threads. In addition an implementation of this interface has to be able to execute a given action in a synchronous way with respect to the data handling threads. This should be done by overriding the [DataInput::actionsPending\(\)](#) virtual function, that is called when a new action is requested to be executed. When this happens the implementation of this function shall be accountable for execution of the [DataInput::executeActions\(\)](#) function from the same thread that is used for the [DataInput::dataReceived\(\)](#) function invocations.

4.7.2 Member Function Documentation

4.7.2.1 virtual void swrod::DataInput::actionsPending () [protected], [pure virtual]

This function is called when there are actions scheduled for execution in the context of the data receiving thread. In response to this call an implementation of the [DataInput](#) interface has to call [DataInput::executeActions\(\)](#) function as soon as possible in the context of the data receiving thread.

4.7.2.2 void swrod::DataInput::dataReceived (InputLinkId *link*, const uint8_t * *data*, uint32_t *size*, uint8_t *status*)
[inline], [protected]

An implementation of the [DataInput](#) interface shall provide input data by calling this function.

Parameters

<i>link</i>	input link ID for which the data has been received
<i>data</i>	pointer to the beginning of the data
<i>size</i>	data size
<i>status</i>	error status that is given by the data source

4.7.2.3 void swrod::DataInput::executeActions () [protected]

This function executes all pending actions. If there are no actions pending it does nothing. It shall be called by an implementation of [DataInput](#) interface in the context of the data receiving thread in response to a [DataInput::actionsPending\(\)](#) function invocation.

4.7.2.4 void swrod::DataInput::executeSynchronousAction (const Action & action)

This function can be used to execute a given action in the context of the data handling thread. This is necessary to avoid using locks in the data handling thread that may otherwise result in significant performance penalty.

Parameters

<i>action</i>	The action to be executed in the context of the data handling thread.
---------------	---

4.7.2.5 const std::string& swrod::DataInput::getName () const [inline],[override],[virtual]

A subclass shall override this function to return a string ID of the current object. This ID has to be unique in the scope of the current SW ROD application if [getStatistics\(\)](#) function of the current object returns non-null pointer to an IS information object. This will guarantee that this IS object can be published correctly. For objects that don't provide statistics to be published to IS this ID is not required to be unique.

Returns

The object ID

Implements [swrod::Component](#).

4.7.2.6 void swrod::DataInput::subscribe (const InputLinkId & link, const DataCallback & callback)

Associates the given callback function with the given link and calls [DataInput::subscribeToFelix\(\)](#). If a callback had been already set for the given link the function will throw [swrod::SubscriptionException](#) exception.

Parameters

<i>in</i>	<i>link</i>	The ID of the input link.
<i>in</i>	<i>callback</i>	This function will be called for any data received from the given input link.

Exceptions

<i>swrod::SubscriptionException</i>	This exception is thrown if a callback is already registered with the given link.
-------------------------------------	---

4.7.2.7 virtual void swrod::DataInput::subscribeToFelix (const InputLinkId & link_id) [protected],[pure virtual]

An implementation of the [DataInput](#) interface shall override this function. This function is called by the [DataInput::subscribe\(\)](#) and is responsible for establishing subscription for the given input link.

Parameters

in	<i>link_id</i>	The ID of the input link to be subscribed.
----	----------------	--

4.7.2.8 `virtual uint32_t swrod::DataInput::translateStatus (uint8_t status) const` `[inline], [protected], [virtual], [noexcept]`

This function can be overridden to translate communication transport error status to the corresponding ROB fragment header error status. Default implementation returns zero.

Parameters

<i>status</i>	- communication transport status
---------------	----------------------------------

Returns

ROB fragment header status

4.7.2.9 `void swrod::DataInput::unsubscribe (const InputLinkId & link)`

Removes the callback that had been previously set for the given input link. If no callback is associated with the given input link the function immediately returns, otherwise calls the [DataInput::unsubscribeFromFelix\(\)](#) function.

Parameters

in	<i>link</i>	The ID of the input link.
----	-------------	---------------------------

4.7.2.10 `virtual void swrod::DataInput::unsubscribeFromFelix (const InputLinkId & link_id)` `[protected], [pure virtual]`

An implementation of the [DataInput](#) interface shall override this function. This function is called by the [DataInput::unsubscribe\(\)](#) and is responsible for removing subscription that had been previously established for the given input link.

Parameters

in	<i>link_id</i>	The ID of the input link to be unsubscribed.
----	----------------	--

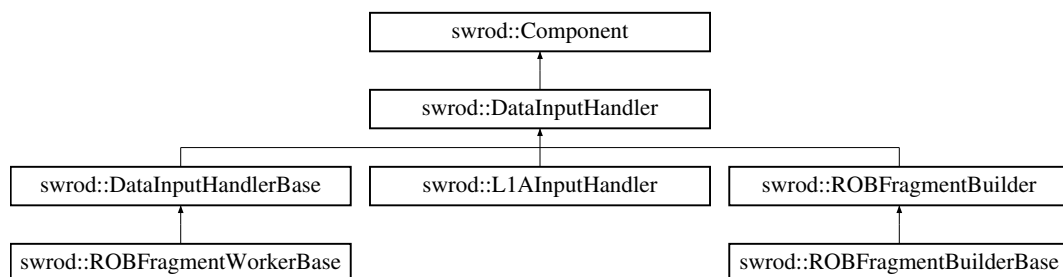
The documentation for this class was generated from the following file:

- swrod/DataInput.h

4.8 swrod::DataInputHandler Class Reference

```
#include <DataInputHandler.h>
```

Inheritance diagram for swrod::DataInputHandler:



Public Member Functions

- virtual void [disable](#) ()=0
- virtual void [disableLinks](#) (std::vector< InputLinkId > &link_ids)=0
- virtual void [enable](#) (uint32_t lastL1ID, uint64_t triggersNumber)=0
- virtual void [enableLinks](#) (std::vector< InputLinkId > &link_ids, uint32_t lastL1ID, uint64_t triggersNumber)=0
- virtual void [resynchAfterRestart](#) (uint32_t lastL1ID)=0
- virtual void [subscribeToFelix](#) ()=0
- virtual void [unsubscribeFromFelix](#) ()=0

4.8.1 Detailed Description

This is an abstract class that defines interface for data handling components of the SW ROD.

4.8.2 Member Function Documentation

4.8.2.1 virtual void swrod::DataInputHandler::disable () [pure virtual]

This function can be used to stop getting data from all input links managed by this object. A typical implementation of this function should unsubscribe from the input links.

Implemented in [swrod::DataInputHandlerBase](#), and [swrod::ROBFragmentBuilderBase](#).

4.8.2.2 virtual void swrod::DataInputHandler::disableLinks (std::vector< InputLinkId > & link_ids) [pure virtual]

This function is used to inform the data handler that it shall ignore data from the given input links. A typical implementation of this function should unsubscribe from the given links.

Parameters

<i>in, out</i>	<i>link_ids</i>	The IDs of the input links to be disabled. If an attempt to disable a link from the given vector succeeds an implementation of this function shall remove the corresponding ID from the vector.
----------------	-----------------	---

Implemented in [swrod::DataInputHandlerBase](#), and [swrod::ROBFragmentBuilderBase](#).

4.8.2.3 virtual void swrod::DataInputHandler::enable (uint32_t lastL1ID, uint64_t triggersNumber) [pure virtual]

This function can be called to re enable all input links handled by this object, which had been previously disabled via [DataInputHandler::disable\(\)](#) function. It is assumed that the Trigger is on hold and no input data may be coming in when this function is executed.

Parameters

<i>lastL1ID</i>	Last L1ID generated when the Trigger had been put on hold.
<i>triggersNumber</i>	A total number of triggers received since the last restart of the SW ROD application

Implemented in [swrod::ROBFragmentBuilderBase](#), and [swrod::DataInputHandlerBase](#).

4.8.2.4 virtual void swrod::DataInputHandler::enableLinks (std::vector< InputLinkId > & link_ids, uint32_t lastL1ID, uint64_t triggersNumber) [pure virtual]

This function is used to inform the data handler that it shall resume getting data from the given input links. A typical implementation of this function should re subscribe to the given links. It is assumed that the Trigger is put on hold and no input data can be produced when this function is executed. A failure to respect this condition may result in a fatal crash of the SW ROD application.

Parameters

<i>in, out</i>	<i>link_ids</i>	A vector of IDs of the input links to be re enabled. If an attempt to enable a link from this vector succeeds an implementation of this function shall remove the corresponding ID from the vector.
<i>in</i>	<i>lastL1ID</i>	The last L1ID that has been produced before the Trigger has been put on hold.
	<i>triggersNumber</i>	A total number of triggers produced since the last restart of the SW ROD application

Implemented in [swrod::DataInputHandlerBase](#), and [swrod::ROBFragmentBuilderBase](#).

4.8.2.5 virtual void swrod::DataInputHandler::resynchAfterRestart (uint32_t *lastL1ID*) [pure virtual]

This function shall be used as part of TTC Restart procedure to re synchronise SW ROD data handlers after restart. The Trigger must be put on hold when this function is called to guarantee that no data shall arrive to the SW ROD until this function returns. A failure to respect this condition may result in a fatal crash of the SW ROD application.

Parameters

<i>in</i>	<i>lastL1ID</i>	The last L1ID that has been produced before the Trigger has been put on hold.
-----------	-----------------	---

Implemented in [swrod::DataInputHandlerBase](#), and [swrod::ROBFragmentBuilderBase](#).

4.8.2.6 virtual void swrod::DataInputHandler::subscribeToFelix () [pure virtual]

An implementation of this function shall subscribe to the input links, which were assigned to this data handler with respect to the current configuration. This function should be called during the Run Control CONNECT transition.

Implemented in [swrod::DataInputHandlerBase](#), and [swrod::ROBFragmentBuilderBase](#).

4.8.2.7 virtual void swrod::DataInputHandler::unsubscribeFromFelix () [pure virtual]

An implementation of this function shall unsubscribe from all input data links. This function should be called during the Run Control DISCONNECT transition.

Implemented in [swrod::DataInputHandlerBase](#), and [swrod::ROBFragmentBuilderBase](#).

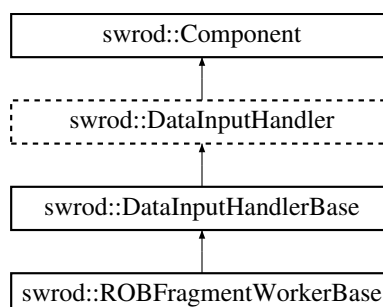
The documentation for this class was generated from the following file:

- [swrod/DataInputHandler.h](#)

4.9 swrod::DataInputHandlerBase Class Reference

```
#include <DataInputHandlerBase.h>
```

Inheritance diagram for [swrod::DataInputHandlerBase](#):



Classes

- struct [Link](#)

Public Types

- typedef std::vector< [Link](#) > **InputLinks**

Public Member Functions

- [DataInputHandlerBase](#) (const std::string &id, const InputLinkVector &links, const boost::property_tree::ptree &config, const std::shared_ptr< [DataInput](#) > &input, detail::InputCallback callback)
- const std::string & [getName](#) () const override
- const InputLinks & [getLinks](#) () const
- void [disable](#) () override
- void [disableLinks](#) (std::vector< InputLinkId > &link_ids) override
- void [enable](#) (uint32_t lastL1ID, uint64_t triggersNumber) override
- void [enableLinks](#) (std::vector< InputLinkId > &link_ids, uint32_t lastL1ID, uint64_t triggersNumber) override
- void [resynchAfterRestart](#) (uint32_t lastL1ID) override
- void [runStarted](#) (const RunParams &run_params) override
- void [runStopped](#) () override
- void [subscribeToFelix](#) () override
- void [unsubscribeFromFelix](#) () override
- virtual void [disabled](#) ()
- virtual void [enabled](#) (uint64_t triggersCounter)
- virtual void [linkDisabled](#) (const [Link](#) &link)
- virtual void [linkEnabled](#) (const [Link](#) &link)

Protected Attributes

- const std::string **m_id**
- bool **m_running** = false
- uint32_t **m_links_number**
- InputLinks **m_links**
- detail::InputCallback **m_callback**
- std::shared_ptr< [DataInput](#) > **m_input**

4.9.1 Detailed Description

This class provides a basic reusable implementation of the InputDataHandler interface. Deriving from the InputDataHandlerBase is a recommended way of implementing this interface.

4.9.2 Constructor & Destructor Documentation

- 4.9.2.1 `swrod::DataInputHandlerBase::DataInputHandlerBase (const std::string &id, const InputLinkVector &links, const boost::property_tree::ptree &config, const std::shared_ptr< DataInput > &input, detail::InputCallback callback)`

Creates a new input data handler with the given configuration.

Parameters

in	<i>id</i>	Id to be used for this data handler
in	<i>links</i>	Input links which have to be handled by this object
in	<i>config</i>	Contains configuration parameters for the new data handler.

in	<i>input</i>	the object to be used for data subscription
in	<i>callback</i>	a function to be called when a new data arrive

4.9.3 Member Function Documentation

4.9.3.1 void swrod::DataInputHandlerBase::disable () [override],[virtual]

This function can be used to stop getting data from all input links managed by this object. In order to achieve this it unsubscribes from all input links.

Implements [swrod::DataInputHandler](#).

4.9.3.2 virtual void swrod::DataInputHandlerBase::disabled () [inline],[virtual]

Is called as part of the ROB fragment builder disabling procedure to notify the derived class that this data handler has just been disabled.

4.9.3.3 void swrod::DataInputHandlerBase::disableLinks (std::vector< InputLinkId > & link_ids) [override],[virtual]

This function checks the given links and unsubscribes from those that are managed by this data handler. The IDs of the links that were successfully disabled are removed from the input vector.

Parameters

<i>link_ids[in,out]</i>	IDs of the input links which have to be disabled
-------------------------	--

Implements [swrod::DataInputHandler](#).

4.9.3.4 void swrod::DataInputHandlerBase::enable (uint32_t lastL1ID, uint64_t triggersNumber) [override],[virtual]

This function can be called to re enable all input links handled by this object. It re subscribes to all input links, which were not individually disabled by using [DataInputHandler::disableLinks\(\)](#) function. It is assumed that the Trigger is on hold and no input data may be coming in when this function is executed. A failure to respect this condition may result in a fatal crash.

Parameters

in	<i>lastL1ID</i>	Last L1ID generated before the Trigger had been put on hold.
in	<i>triggersNumber</i>	A total number of triggers received since the last restart of the SW ROD application

Implements [swrod::DataInputHandler](#).

4.9.3.5 virtual void swrod::DataInputHandlerBase::enabled (uint64_t triggersCounter) [inline],[virtual]

Is called as part of the ROB fragment builder enabling procedure to notify the derived class that this data handler has just been re enabled.

Parameters

in	<i>triggersCounter</i>	A total number of triggers received by the SW ROD application since the last restart. This number can be used to synchronise this data handler with the TTC input stream.
----	------------------------	---

4.9.3.6 void swrod::DataInputHandlerBase::enableLinks (std::vector< InputLinkId > & link_ids, uint32_t lastL1ID, uint64_t triggersNumber) [override],[virtual]

This function checks the given links and tries to re enable those that are handled by this object. The IDs of the links that were successfully re enabled are removed from the input vector. This way when the function returns, the vector of links IDs contains only IDs of the links which are either not handled by this worker or failed to be re enabled. It is

assumed that the Trigger is on hold and no input data may be coming in when this function is executed. A failure to respect this condition may result in a fatal crash.

Parameters

in, out	<i>link_ids</i>	IDs of the input links which have to be re enabled
in	<i>lastL1ID</i>	Last L1ID generated when the Trigger had been put on hold.
in	<i>triggersNumber</i>	A total number of triggers produced since the last restart of this SW ROD application

Implements [swrod::DataInputHandler](#).

4.9.3.7 `const InputLinks& swrod::DataInputHandlerBase::getLinks () const [inline]`

Returns a reference to the vector of input links handled by this object.

Returns

a vector of input links

4.9.3.8 `const std::string& swrod::DataInputHandlerBase::getName () const [inline],[override],[virtual]`

Returns unique object ID.

Returns

A unique object ID.

Implements [swrod::Component](#).

4.9.3.9 `virtual void swrod::DataInputHandlerBase::linkDisabled (const Link & link) [inline],[virtual]`

Is called as part of the input links disabling procedure to notify the derived class that the given input link has just been disabled.

Parameters

in	<i>link</i>	A reference to the object that contains description of the just disabled input link.
----	-------------	--

4.9.3.10 `virtual void swrod::DataInputHandlerBase::linkEnabled (const Link & link) [inline],[virtual]`

Is called as part of the input links re enabling procedure to notify the derived class that the given input link has just been re enabled.

Parameters

in	<i>link</i>	A reference to the object that contains description of the just re enabled input link
----	-------------	---

4.9.3.11 `void swrod::DataInputHandlerBase::resynchAfterRestart (uint32_t lastL1ID) [override],[virtual]`

This function can be called after restarting the SW ROD application to re synchronise the new SW ROD instance with the TTC system. It is assumed that the Trigger is on hold and no input data may be produced when this function is executed. A failure to respect this condition may result in a fatal crash.

Parameters

in	<i>lastL1ID</i>	Last L1ID generated when the trigger had been put on hold.
----	-----------------	--

Implements [swrod::DataInputHandler](#).

4.9.3.12 void [swrod::DataInputHandlerBase::runStarted](#) (const RunParams & *run_params*) [override],[virtual]

Is called to inform this object about starting a new run.

Parameters

in	<i>run_params</i>	New run parameters
----	-------------------	--------------------

Reimplemented from [swrod::Component](#).

4.9.3.13 void [swrod::DataInputHandlerBase::runStopped](#) () [override],[virtual]

Is called to inform this object about the end of the current run.

Reimplemented from [swrod::Component](#).

4.9.3.14 void [swrod::DataInputHandlerBase::subscribeToFelix](#) () [override],[virtual]

Subscribes to all input links handled by this object.

Implements [swrod::DataInputHandler](#).

4.9.3.15 void [swrod::DataInputHandlerBase::unsubscribeFromFelix](#) () [override],[virtual]

Unsubscribes from all input links handled by this object.

Implements [swrod::DataInputHandler](#).

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

- [swrod/DataInputHandlerBase.h](#)

4.10 swrod::Exception Class Reference

```
#include <exceptions.h>
```

4.10.1 Detailed Description

This is a base class for any SW ROD exception.

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

- [swrod/exceptions.h](#)

4.11 swrod::Factory< Product > Class Template Reference

```
#include <Factory.h>
```

Classes

- struct [Registrator](#)

Public Types

- typedef std::conditional
< [IsUnique](#)< Product >::value,
std::unique_ptr< Product >
, std::shared_ptr< Product >
>::type **ReturnType**
- typedef std::function
< ReturnType(const
boost::property_tree::ptree
&, const [Core](#) &)> **Creator**

Public Member Functions

- ReturnType **create** (const std::string &type, const boost::property_tree::ptree &config, const [Core](#) &core)
- void **registerCreator** (const std::string &type, const Creator &creator)
- void **unregisterCreator** (const std::string &type)

Static Public Member Functions

- static [Factory](#) & **instance** ()

4.11.1 Detailed Description

```
template<class Product>class swrod::Factory< Product >
```

This is a helper class that can be used to register a factory of a custom implementation of a SW ROD component interface. This can be done by declaring a global instance of the [Factory::Registrar](#) class in one of the compilation modules.

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

- swrod/Factory.h

4.12 swrod::helper::FragmentCollator Class Reference

Helper class to collate ROBfragments belonging to the same L1 ID.

```
#include <FragmentCollator.h>
```

Public Member Functions

- void **expected** (unsigned int exp)
- unsigned int **expected** () const
- void **maxIndex** (unsigned int size)
- void **close** ()
- void **reset** ()
- bool **insert** (std::shared_ptr< [ROBFragment](#) > frag)
- void **clear** (unsigned int L1Id)
- std::vector< unsigned int > **clear** (std::vector< unsigned int > &L1IdVec)
- void **enable** (unsigned int robId)
- void **disable** (unsigned int robId)
- std::vector< unsigned int > **latestL1** () const
- unsigned int **lowestL1** () const

- const std::vector
 < std::shared_ptr< ROBFragment > > & **get** (unsigned int l1Id)
- std::vector< std::shared_ptr
 < ROBFragment > > **get** (unsigned int l1Id, std::set< unsigned int > *robs)
- std::vector< std::shared_ptr
 < ROBFragment > > **take** (unsigned int l1Id)
- unsigned int **size** () const
- void **dump** () const

4.12.1 Detailed Description

Helper class to collate ROBfragments belonging to the same L1 ID.

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

- swrod/FragmentCollator.h

4.13 swrod::GetParameterException Class Reference

```
#include <exceptions.h>
```

4.13.1 Detailed Description

This issue is used to report problems with SW ROD configuration parameters

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

- swrod/exceptions.h

4.14 swrod::ROBFragmentBuilderBase::hash_compare Struct Reference

Static Public Member Functions

- static size_t **hash** (const uint64_t &a)
- static bool **equal** (const uint64_t &a, const uint64_t &b)

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

- swrod/ROBFragmentBuilderBase.h

4.15 swrod::GBTChunk::Header Struct Reference

```
#include <GBTChunk.h>
```

Public Attributes

- uint16_t **m_size**
- uint8_t **m_felix_status**
- uint8_t **m_swrod_status**
- uint32_t **m_link_id**

4.15.1 Detailed Description

This class defines a structure of the header of a data chunk within data payload of a ROB fragments produced by the GBTModeBuilder algorithm.

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

- swrod/GBTChunk.h

4.16 swrod::IsUnique< T > Struct Template Reference

Static Public Attributes

- static constexpr bool **value** = false

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

- swrod/Factory.h

4.17 swrod::IsUnique< L1InputHandler > Struct Template Reference

Static Public Attributes

- static constexpr bool **value** = true

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

- swrod/Factory.h

4.18 swrod::L1AInfo Struct Reference

```
#include <L1AInfo.h>
```

Public Member Functions

- **L1AInfo** (uint16_t bcid, uint16_t trigger_type, uint32_t l1id, uint64_t index)
- bool **isECR** () const noexcept

Public Attributes

- const uint16_t **m_bcid**
- const uint16_t **m_trigger_type**
- const uint32_t **m_l1id**
- const uint64_t **m_index**

4.18.1 Detailed Description

Contains information from a L1 Accept message.

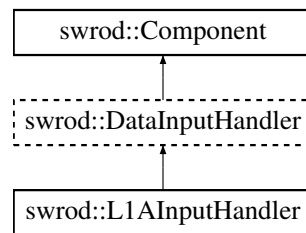
The documentation for this struct was generated from the following file:

- swrod/L1AInfo.h

4.19 swrod::L1AInputHandler Class Reference

```
#include <L1AInputHandler.h>
```

Inheritance diagram for swrod::L1AInputHandler:



Public Member Functions

- [L1AInputHandler](#) (const boost::property_tree::ptree &config, const [Core](#) &core)
- virtual void [subscribe](#) (const std::shared_ptr< [ROBFragmentBuilder](#) > &builder)

Protected Member Functions

- void **notify** (const [L1AInfo](#) &l1a_info) const

Protected Attributes

- std::vector< std::shared_ptr< [ROBFragmentBuilder](#) > > **m_builders**
- std::shared_ptr< [DataInput](#) > **m_data_input**

4.19.1 Detailed Description

This class defines an interface and provides a default implementation of a L1 Accept data handler.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 swrod::L1AInputHandler::L1AInputHandler (const boost::property_tree::ptree & config, const Core & core)

Creates a new L1 Accept handler using the given configuration.

Parameters

<i>config</i>	A new object configuration.
<i>core</i>	a reference to swrod::Core object

4.19.3 Member Function Documentation

4.19.3.1 virtual void swrod::L1AInputHandler::subscribe (const std::shared_ptr< [ROBFragmentBuilder](#) > & builder) [inline], [virtual]

This function can be used to register a [ROBFragmentBuilder](#) instance to receive notifications about L1 Accept data that this object receives. The L1A data will be passed to the fragment builder via the [ROBFragmentBuilder::l1a-Received\(\)](#) function.

Parameters

<i>builder</i>	fragment builder that has to be notified about L1 Accept received data
----------------	--

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

- swrod/L1AInputHandler.h

4.20 swrod::ROBFragmentBuilderBase::L1ID Class Reference

Public Member Functions

- **L1ID** (uint32_t value=c_invalid_value)
- bool **invalidOrEqual** (uint32_t l1id) const
- void **set** (uint64_t counter, uint32_t l1id)
- uint64_t **getCounter** () const
- uint32_t **getValue** () const

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

- swrod/ROBFragmentBuilderBase.h

4.21 swrod::DataInputHandlerBase::Link Struct Reference

```
#include <DataInputHandlerBase.h>
```

Public Member Functions

- **Link** (const InputLinkId &id, const DetectorLinkId &det_id)
- void **reset** ()

Public Attributes

- const InputLinkId **m_fid**
- const DetectorLinkId **m_resource_id**
- bool **m_enabled**
- uint16_t **m_last_bcid**
- uint32_t **m_expected_l1id**
- uint64_t **m_packets_counter**
- uint64_t **m_packets_missed**
- uint64_t **m_packets_corrupted**
- uint64_t **m_packets_dropped**

4.21.1 Detailed Description

This class contains control and monitoring counters for an input link. This information is used for input data book-keeping, error detection and monitoring.

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

- swrod/DataInputHandlerBase.h

4.22 swrod::Factory< Product >::Registrator Struct Reference

Public Member Functions

- [Registrator](#) (const std::string &type, const Creator &creator)

4.22.1 Constructor & Destructor Documentation

4.22.1.1 `template<class Product > swrod::Factory< Product >::Registrator (const std::string & type, const Creator & creator) [inline]`

Use this constructor to declare an instance that registers a new factory.

Parameters

<i>type</i>	The type name of the new factory.
<i>creator</i>	The factory procedure that can be used to create an instance of a given class.

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

- swrod/Factory.h

4.23 swrod::ROBFragment Class Reference

```
#include <ROBFragment.h>
```

Classes

- class [DataBlock](#)

Public Member Functions

- [ROBFragment](#) (uint32_t source_id, uint32_t l1id, uint16_t bcid, uint32_t trigger_type, uint32_t rob_status, uint16_t missed_packets, uint16_t corrupt_packets, std::vector< [DataBlock](#) > &&data, bool rod_header_present=false)
- [ROBFragment](#) (uint32_t source_id, uint32_t l1id, uint32_t status_word)
- std::vector< std::pair< uint8_t *, uint32_t > > [serialize](#) (uint32_t run_number) const

Public Attributes

- const uint32_t **m_source_id**
- const uint32_t **m_l1id**
- const uint16_t **m_bcid**
- const uint32_t **m_trigger_type**
- const uint16_t **m_missed_packets**
- const uint16_t **m_corrupted_packets**
- const bool **m_rod_header_present**
- uint32_t **m_detector_type**
- uint16_t **m_rod_minor_version**
- bool **m_status_front**
- std::vector< uint32_t > **m_status_words**
- std::vector< [DataBlock](#) > **m_data**

4.23.1 Detailed Description

This class contains information for a fully built ROB fragment.

4.23.2 Constructor & Destructor Documentation

4.23.2.1 `swrod::ROBFragment::ROBFragment (uint32_t source_id, uint32_t l1id, uint16_t bcid, uint32_t trigger_type, uint32_t rob_status, uint16_t missed_packets, uint16_t corrupt_packets, std::vector< DataBlock > && data, bool rod_header_present = false)`

Constructs a new ROB data fragment with the given data and parameters.

Parameters

<i>source_id</i>	The ROB ID.
<i>l1id</i>	The extended L1 ID.
<i>bcid</i>	The bunch crossing ID.
<i>trigger_type</i>	The trigger type.
<i>rob_status</i>	ROB header status word.
<i>missed_packets</i>	The number of missed packets in this fragment.
<i>corrupt_packets</i>	The number of corrupt packets in this fragment.
<i>data</i>	The data payload of this fragment.

4.23.2.2 `swrod::ROBFragment::ROBFragment (uint32_t source_id, uint32_t l1id, uint32_t status_word)`

Constructs a new empty ROB data fragment with the given parameters.

Parameters

<i>source_id</i>	The ROB ID.
<i>l1id</i>	The extended L1 ID.
<i>status_word</i>	Usually contains the code of the error that prevents proper building of this event.

4.23.3 Member Function Documentation

4.23.3.1 `std::vector<std::pair<uint8_t*, uint32_t> > swrod::ROBFragment::serialize (uint32_t run_number) const`

This function puts the content of the fragment into a number of memory chunks that can be used to send it over network or to write it to a file.

Parameters

<i>run_number</i>	The current run number.
-------------------	-------------------------

Returns

The event represented as a number of memory blocks.

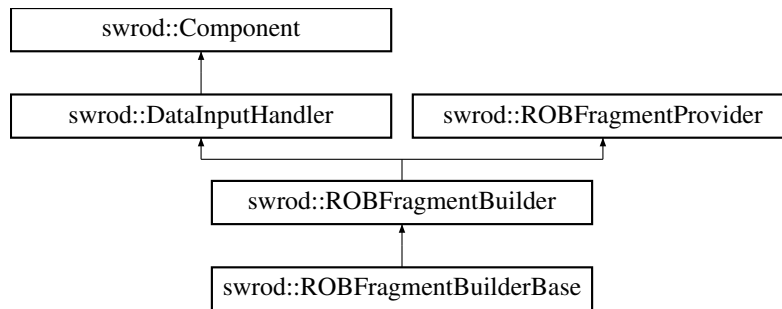
The documentation for this class was generated from the following file:

- swrod/ROBFragment.h

4.24 swrod::ROBFragmentBuilder Class Reference

```
#include <ROBFragmentBuilder.h>
```

Inheritance diagram for swrod::ROBFragmentBuilder:



Public Member Functions

- virtual uint32_t [getROBId](#) () const =0
- virtual void [l1aReceived](#) (const [L1AInfo](#) &l1a)=0

Additional Inherited Members

4.24.1 Detailed Description

An abstract class that defines interface for a ROB Fragment Builder implementation. It is not recommended to use this class directly, but to derive a ROB Fragment Builder implementation class from the [ROBFragmentBuilderBase](#) class.

4.24.2 Member Function Documentation

4.24.2.1 virtual uint32_t swrod::ROBFragmentBuilder::getROBId () const [pure virtual]

Returns ID of the ROB whose fragments are built by this object.

Returns

The ID of the ROB.

Implemented in [swrod::ROBFragmentBuilderBase](#).

4.24.2.2 virtual void swrod::ROBFragmentBuilder::l1aReceived (const [L1AInfo](#) & l1a) [pure virtual]

Is called when a new L1Accept message is received.

Parameters

in	<i>l1a</i>	A new L1Accept message.
----	------------	-------------------------

Implemented in [swrod::ROBFragmentBuilderBase](#).

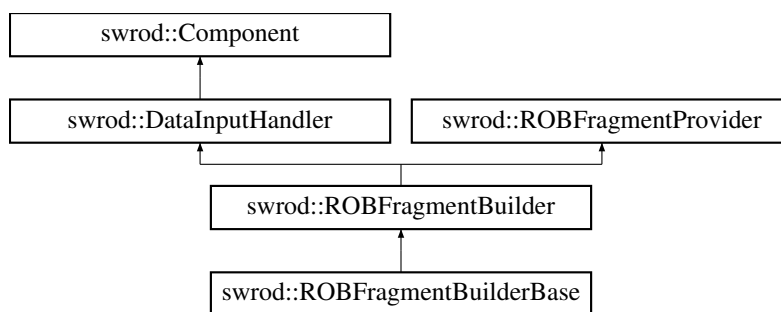
The documentation for this class was generated from the following file:

- swrod/ROBFragmentBuilder.h

4.25 swrod::ROBFragmentBuilderBase Class Reference

```
#include <ROBFragmentBuilderBase.h>
```

Inheritance diagram for swrod::ROBFragmentBuilderBase:



Classes

- struct [hash_compare](#)
- class [L1ID](#)

Public Types

- typedef std::function
< [DataInputHandlerBase](#) *(const
InputLinkVector &, const
std::shared_ptr< [DataInput](#) > &)> **WorkerFactory**
- typedef std::function< void(const
[L1AInfo](#) &)> **L1AReceiver**

Public Member Functions

- **ROBFragmentBuilderBase** (const boost::property_tree::ptree &robConfig, const [Core](#) &core, const L1A-Receiver &l1a_receiver, const WorkerFactory &factory, bool header_present=false)
- const std::string & [getName](#) () const override
- uint32_t [getROBId](#) () const final
- void [l1aReceived](#) (const [L1AInfo](#) &l1a) override
- void [disableLinks](#) (std::vector< InputLinkId > &link_ids) override
- void [disable](#) () override
- void [enableLinks](#) (std::vector< InputLinkId > &link_ids, uint32_t lastL1ID, uint64_t triggersNumber) override
- void [enable](#) (uint32_t lastL1ID, uint64_t triggersNumber) override
- void [resynchAfterRestart](#) (uint32_t lastL1ID)
- void [runStarted](#) (const RunParams &run_params) override
- void [runStopped](#) () override
- void [subscribeToFelix](#) () override
- void [unsubscribeFromFelix](#) () override
- ISInfo * [getStatistics](#) () override

Protected Types

- typedef
tbb::concurrent_hash_map
< uint64_t, detail::DataHolder,
[hash_compare](#) > **FragmentAssembler**
- typedef
tbb::concurrent_bounded_queue
< uint64_t > **ReadyQueue**
- typedef std::function< void(FragmentAssembler::accessor &)> **FragmentReadyCallback**

Protected Member Functions

- void **transmitter** (const bool &active)
- void **synchronize** (uint32_t lastL1ID, const L1ID &data)
- void **addToReadyQueue** (FragmentAssembler::accessor &a)
- void **fragmentReady** (FragmentAssembler::accessor &a)
- void **submitFragment** (FragmentAssembler::accessor &a)

Protected Attributes

- const uint32_t **m_ROB_id**
- const std::string **m_unique_id**
- const bool **m_flush_buffer**
- const bool **m_ROD_header_present**
- const uint32_t **m_l1a_wait_timeout**
- const uint32_t **m_resynch_timeout**
- uint64_t **m_buffer_size** = 0
- int64_t **m_queue_size** = 0
- bool **m_running**
- bool **m_enabled**
- std::mutex **m_built_mutex**
- L1ID **m_last_built_L1ID**
- L1ID **m_last_received_L1ID**
- ROBStatistics **m_statistics**
- std::chrono::time_point
< std::chrono::system_clock > **m_start_of_run**
- std::chrono::time_point
< std::chrono::system_clock > **m_last_update**
- L1AReceiver **m_l1a_receiver**
- FragmentAssembler **m_fragment_assembler**
- ReadyQueue **m_ready**
- detail::ThreadPool **m_transmitters**
- std::vector< std::unique_ptr
< DataInputHandlerBase > > **m_workers**
- FragmentReadyCallback **m_fragment_ready_callback**

4.25.1 Detailed Description

This is an abstract class that provides a reusable basic implementation of the [ROBFragmentBuilderInterface](#) interface. Deriving custom class from the [ROBFragmentBuilderInterface](#) is a recommended way of implementing a ROB fragment building algorithm.

4.25.2 Member Function Documentation

4.25.2.1 void swrod::ROBFragmentBuilderInterface::disable () [override],[virtual]

Should be used to disable event building. This function unsubscribes from all input links.

Implements [swrod::DataInputHandler](#).

4.25.2.2 void swrod::ROBFragmentBuilderInterface::disableLinks (std::vector< InputLinkId > & link_ids) [override],[virtual]

Should be used to inform the fragment builder that it shall not use data from the given input links for event building. This function unsubscribes from the given input links.

Parameters

in	<i>link_ids</i>	The IDs of the disabled input links. If an attempt to disable a link from this vector succeeds this function removes the corresponding ID from the vector.
----	-----------------	--

Implements [swrod::DataInputHandler](#).

4.25.2.3 `void swrod::ROBFragmentBuilderBase::enable (uint32_t lastL1ID, uint64_t triggersNumber) [override], [virtual]`

Called to inform the fragment builder that it has to resume fragment building. This function re subscribes to all input links. This function assumes that the Trigger is on hold and no new data may be produced. A failure of respect this condition may result in a fatal crash.

Parameters

in	<i>lastL1ID</i>	The last L1ID that has been produced when the Trigger was put on hold.
in	<i>triggersNumber</i>	Number of triggers that has been produced since the last restart of the SW ROD application.

Exceptions

swrod::Exception	If an attempt to re enable this fragment builder fails (e.g. subscription to one or several input link failed) the function will throw swrod::Exception exception.
----------------------------------	--

Implements [swrod::DataInputHandler](#).

4.25.2.4 `void swrod::ROBFragmentBuilderBase::enableLinks (std::vector< InputLinkId > & link_ids, uint32_t lastL1ID, uint64_t triggersNumber) [override], [virtual]`

This function is used to inform the fragment builder that it shall resume using data from the given input links for event building. The IDs of the links that were successfully re enabled are removed from the input vector. This way when the function returns the vector of links IDs contains only IDs of the links which are either not handled by this worker or failed to be re enabled. It is assumed that the Trigger is on hold and no input data may be coming in when this function is executed. A failure to respect this condition may result in a fatal crash.

Parameters

in, out	<i>link_ids</i>	A vector of IDs of the input links to be enabled. If an attempt to enable a link from this vector succeeds the function removes the corresponding ID from the vector.
in	<i>lastL1ID</i>	The last L1ID that has been produced when the Trigger had been put on hold.
in	<i>triggersNumber</i>	Number of triggers that has been produced when the Trigger was put on hold.

Implements [swrod::DataInputHandler](#).

4.25.2.5 `const std::string& swrod::ROBFragmentBuilderBase::getName () const [inline], [override], [virtual]`

Returns unique ID for this object. This ID will be used to publish the fragment builder's statistics to IS.

Returns

The object's unique ID.

Implements [swrod::Component](#).

4.25.2.6 `uint32_t swrod::ROBFragmentBuilderBase::getROBId () const [inline], [final], [virtual]`

Returns ID of the ROB whose fragments are built by this object.

Returns

The ID of the ROB.

Implements [swrod::ROBFragmentBuilder](#).

4.25.2.7 ISInfo* swrod::ROBFragmentBuilderBase::getStatistics () [override],[virtual]

Calculates average rates since the last call to this function and updates the statistics object with the new rates before returning.

Returns

A pointer to the IS statistics object.

Reimplemented from [swrod::Component](#).

4.25.2.8 void swrod::ROBFragmentBuilderBase::l1aReceived (const L1AInfo & l1a) [inline],[override],[virtual]

This function is called by the [L1AInputHandler](#) implementation to inform this builder about a new L1A data.

Parameters

<i>l1a</i>	L1 Accept information.
------------	------------------------

Implements [swrod::ROBFragmentBuilder](#).

4.25.2.9 void swrod::ROBFragmentBuilderBase::resynchAfterRestart (uint32_t lastL1ID) [virtual]

This function shall be used as part of TTC Restart procedure to synchronise this fragment builder with the TTC system after the restart of the SW ROD application. The Trigger must be put on hold when this function is executed and therefore the fragment builder shall not received any data until this function returns.

Parameters

in	<i>lastL1ID</i>	The last L1ID that has been produced when the Trigger was put on hold.
----	-----------------	--

Implements [swrod::DataInputHandler](#).

4.25.2.10 void swrod::ROBFragmentBuilderBase::runStarted (const RunParams & run_params) [override],[virtual]

Called when a new data taking session is started.

Parameters

in	<i>run_params</i>	The new run parameters.
----	-------------------	-------------------------

Reimplemented from [swrod::Component](#).

4.25.2.11 void swrod::ROBFragmentBuilderBase::runStopped () [override],[virtual]

Called when the current data taking session is terminated.

Reimplemented from [swrod::Component](#).

4.25.2.12 void swrod::ROBFragmentBuilderBase::subscribeToFelix () [override],[virtual]

This function subscribes to all input links. This function is called during the Run Control CONNECT transition.

Implements [swrod::DataInputHandler](#).

4.25.2.13 void swrod::ROBFragmentBuilderBase::unsubscribeFromFelix () [override],[virtual]

This function cancels all previously established input links subscriptions. This function is called during the Run Control DISCONNECT transition.

Implements [swrod::DataInputHandler](#).

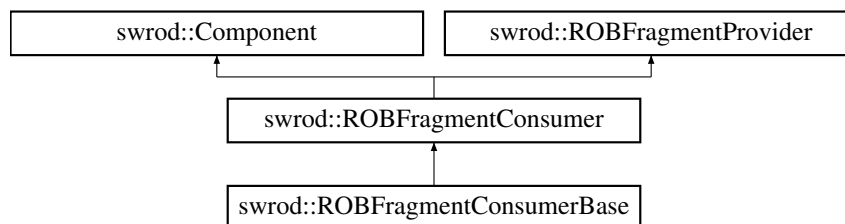
The documentation for this class was generated from the following file:

- swrod/ROBFragmentBuilderBase.h

4.26 swrod::ROBFragmentConsumer Class Reference

```
#include <ROBFragmentConsumer.h>
```

Inheritance diagram for swrod::ROBFragmentConsumer:



Public Member Functions

- virtual void [insertROBFragment](#) (const std::shared_ptr< [ROBFragment](#) > &fragment)=0
- virtual void [linkDisabled](#) (const InputLinkId &link_id)
- virtual void [linkEnabled](#) (const InputLinkId &link_id)
- virtual void [ROBDisabled](#) (uint32_t ROB_id)
- virtual void [ROBEnabled](#) (uint32_t ROB_id)

Additional Inherited Members

4.26.1 Detailed Description

An abstract class that defines an interface for a ROB fragment consumer implementation.

It is not recommended to use this class directly, but to derive a ROB Fragment Consumer implementation from the [ROBFragmentConsumerBase](#) class.

4.26.2 Member Function Documentation

4.26.2.1 virtual void swrod::ROBFragmentConsumer::insertROBFragment (const std::shared_ptr< [ROBFragment](#) > &fragment) [pure virtual]

This function is used to pass fully built ROB fragments to this consumer. A consumer has to process the given fragment as fast as possible and pass it to the next consumer in the chain using the [ROBFragmentProvider::forwardROBFragment\(\)](#) function. It is crucially important for the overall SW ROD performance to return from this function as fast as possible as otherwise this may create back pressure to the caller of this function (e.g. the previous consumer in the chain) and may decrease the overall event processing rate of the SW ROD application. It is strongly recommended not to inherit the [ROBFragmentConsumer](#) class directly, but to derive a custom consumer implementation from the [ROBFragmentConsumerBase](#) class. That class provides configurable implementation of the [ROBFragmentConsumer](#) interface, which is simple yet efficient.

Parameters

<i>in</i>	<i>fragment</i>	ROB fragment to be processed.
-----------	-----------------	-------------------------------

Implemented in [swrod::ROBFragmentConsumerBase](#).

4.26.2.2 `virtual void swrod::ROBFragmentConsumer::linkDisabled (const InputLinkId & link_id) [inline], [virtual]`

Called to inform this consumer that the given input link has been disabled.

Parameters

<i>in</i>	<i>link_id</i>	The link that has been disabled.
-----------	----------------	----------------------------------

4.26.2.3 `virtual void swrod::ROBFragmentConsumer::linkEnabled (const InputLinkId & link_id) [inline], [virtual]`

Called to inform this consumer that the given input link has been re enabled.

Parameters

<i>in</i>	<i>link_id</i>	The link that has been enabled.
-----------	----------------	---------------------------------

4.26.2.4 `virtual void swrod::ROBFragmentConsumer::ROBDisabled (uint32_t ROB_id) [inline], [virtual]`

Called to inform this consumer object that the given ROB has been disabled.

Parameters

<i>ROB_id</i>	The ID of the isabled ROB.
---------------	----------------------------

4.26.2.5 `virtual void swrod::ROBFragmentConsumer::ROBEnabled (uint32_t ROB_id) [inline], [virtual]`

Called to inform this consumer object that the given ROB has been re enabled.

Parameters

<i>ROB_id</i>	The ID of the re enabled ROB.
---------------	-------------------------------

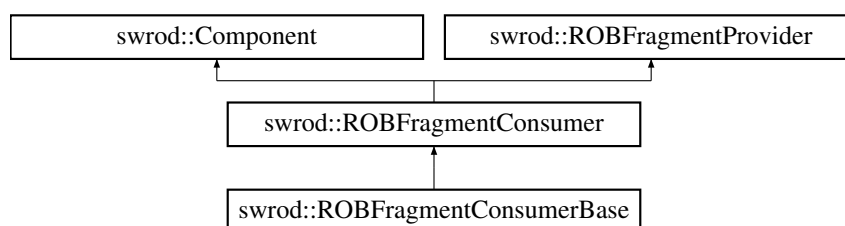
The documentation for this class was generated from the following file:

- swrod/ROBFragmentConsumer.h

4.27 swrod::ROBFragmentConsumerBase Class Reference

```
#include <ROBFragmentConsumerBase.h>
```

Inheritance diagram for swrod::ROBFragmentConsumerBase:



Public Types

- enum [ForwardingPolicy](#) { [ForwardingPolicy::None](#), [ForwardingPolicy::Immediate](#), [ForwardingPolicy::AfterProcessing](#) }
- enum [QueuingPolicy](#) { [QueuingPolicy::Drop](#), [QueuingPolicy::Wait](#) }
- typedef std::function< void(const std::shared_ptr< [ROBFragment](#) > &)> **UserFunction**

Public Member Functions

- [ROBFragmentConsumerBase](#) (const UserFunction &function=[](const std::shared_ptr< [ROBFragment](#) > &){}, [QueuingPolicy](#) queuing_policy=[QueuingPolicy::Wait](#), [ForwardingPolicy](#) forwarding_policy=[ForwardingPolicy::Immediate](#), uint16_t workers_number=1, int32_t queue_size=-1, const std::string &name="Consumer")
- [ROBFragmentConsumerBase](#) (const boost::property_tree::ptree &config, const UserFunction &function=[](const std::shared_ptr< [ROBFragment](#) > &){}, [QueuingPolicy](#) queuing_policy=[QueuingPolicy::Wait](#), [ForwardingPolicy](#) forwarding_policy=[ForwardingPolicy::Immediate](#), const std::string &name="Consumer")
- [ROBFragmentConsumerBase](#) (const boost::property_tree::ptree &config, const std::vector< UserFunction > &functions, [QueuingPolicy](#) queuing_policy=[QueuingPolicy::Wait](#), [ForwardingPolicy](#) forwarding_policy=[ForwardingPolicy::Immediate](#), const std::string &name="Consumer")
- const std::string & [getName](#) () const override
- void [insertROBFragment](#) (const std::shared_ptr< [ROBFragment](#) > &fragment) override
- void [runStarted](#) (const RunParams &run_params) override
- void [runStopped](#) () override

Protected Types

- typedef
tbb::concurrent_bounded_queue
< std::shared_ptr< [ROBFragment](#) > > **FragmentQueue**

Protected Member Functions

- void **run** (const UserFunction &user_function, const bool &active)

Protected Attributes

- const [ForwardingPolicy](#) **m_forwarding**
- const [QueuingPolicy](#) **m_queuing**
- const bool **m_flush_buffer**
- bool **m_running**
- [FragmentQueue](#) **m_fragments_queue**
- detail::ThreadPool **m_threads**

4.27.1 Detailed Description

This class provides an efficient configurable implementation of the [ROBFragmentConsumer](#) interface, that uses a given number of worker threads for data processing. Deriving a new class from the [ROBFragmentConsumerBase](#) is a recommended way to implement a custom ROB fragment consumer.

4.27.2 Member Enumeration Documentation

4.27.2.1 enum swrod::ROBFragmentConsumerBase::ForwardingPolicy [strong]

Defines a policy of passing ROB fragments to the next consumer.

Enumerator

None Does not pass incoming ROB fragments to the next consumer.

Immediate Pass incoming ROB fragment to the next consumer immediately upon receiving.

AfterProcessing Pass incoming ROB fragments after finishing local processing.

4.27.2.2 enum swrod::ROBFragmentConsumerBase::QueuingPolicy [strong]

Defines what to do with the incoming ROB fragments when the internal input queue is full.

Enumerator

Drop Drop ROB fragments if the input queue is full.

Wait Wait until a place becomes available in the queue.

4.27.3 Constructor & Destructor Documentation

4.27.3.1 swrod::ROBFragmentConsumerBase::ROBFragmentConsumerBase (const UserFunction & function = [] (const std::shared_ptr< ROBFragment > &) {}, QueuingPolicy queuing_policy = QueuingPolicy::Wait, ForwardingPolicy forwarding_policy = ForwardingPolicy::Immediate, uint16_t workers_number = 1, int32_t queue_size = -1, const std::string & name = "Consumer") [explicit]

Constructs a new ROB fragment consumer that will create the given number of worker threads that will execute the same function.

Parameters

in	<i>function</i>	Processing function to be executed.
in	<i>queuing_policy</i>	Fragment queueing policy.
in	<i>forwarding_policy</i>	Fragment forwarding policy.
in	<i>workers_number</i>	The number of worker threads.
in	<i>queue_size</i>	Size of the internal queue. Default value -1 means unlimited.
in	<i>name</i>	The ID of the new object. If the new consumer returns a non-null pointer to IS object from the getStatistics() function then this ID must be unique in the scope of the current SW ROD application.

4.27.3.2 swrod::ROBFragmentConsumerBase::ROBFragmentConsumerBase (const boost::property_tree::ptree & config, const UserFunction & function = [] (const std::shared_ptr< ROBFragment > &) {}, QueuingPolicy queuing_policy = QueuingPolicy::Wait, ForwardingPolicy forwarding_policy = ForwardingPolicy::Immediate, const std::string & name = "Consumer") [explicit]

Constructs a new ROB fragment consumer that will use the given number of worker threads to execute the given processing function.

Parameters

in	<i>config</i>	Configuration object that contains parameters taken from the SwRod-FragmentConsumer configuration class
in	<i>function</i>	The processing function to be executed.
in	<i>queuing_policy</i>	Fragment queueing policy.
in	<i>forwarding_policy</i>	Fragment forwarding policy.
in	<i>name</i>	The ID of the new object. If the new consumer returns a non-null pointer to IS object from the getStatistics() function then this ID must be unique in the scope of the current SW ROD application.

4.27.3.3 `swrod::ROBFragmentConsumerBase::ROBFragmentConsumerBase (const boost::property_tree::ptree & config, const std::vector< UserFunction > & functions, QueuingPolicy queuing_policy = QueuingPolicy::Wait, ForwardingPolicy forwarding_policy = ForwardingPolicy::Immediate, const std::string & name = "Consumer") [explicit]`

Constructs a new ROB fragment consumer creating a new worker thread for every processing function in the given functions vector.

Parameters

in	<i>config</i>	Configuration object that contains parameters taken from the SwRod-FragmentConsumer configuratiob class
in	<i>functions</i>	A vector of processing function to be executed. Each function will be executed in a dedicated worker thread.
in	<i>queuing_policy</i>	Fragment queueing policy, default is -1, which means unlimited.
in	<i>forwarding_policy</i>	Fragment forwarding policy.
in	<i>name</i>	The ID of the new object. If the new consumer returns a non-null pointer to IS object from the getStatistics() function then this ID must be unique in the scope of the current SW ROD application.

4.27.4 Member Function Documentation

4.27.4.1 `const std::string& swrod::ROBFragmentConsumerBase::getName () const [inline], [override], [virtual]`

Returns the name that was given to the object constructor. This name must be unique to support publishing of monitoring statistics to IS.

Returns

The object name.

Implements [swrod::Component](#).

4.27.4.2 `void swrod::ROBFragmentConsumerBase::insertROBFragment (const std::shared_ptr< ROBFragment > & fragment) [override], [virtual]`

This function puts the given ROB fragment into internal queue and returns. The actual processing is done by the worker threads, which are constantly polling this queue for new fragments.

Parameters

<i>fragment</i>	
-----------------	--

Implements [swrod::ROBFragmentConsumer](#).

4.27.4.3 void swrod::ROBFragmentConsumerBase::runStarted (const RunParams & *run_params*) [override],
[virtual]

Called when a new data taking session is started.

Parameters

in	<i>run_params</i>	The new run parameters.
----	-------------------	-------------------------

Reimplemented from [swrod::Component](#).

4.27.4.4 void [swrod::ROBFragmentConsumerBase::runStopped](#) () [override],[virtual]

Called when the current data taking session is terminated.

Reimplemented from [swrod::Component](#).

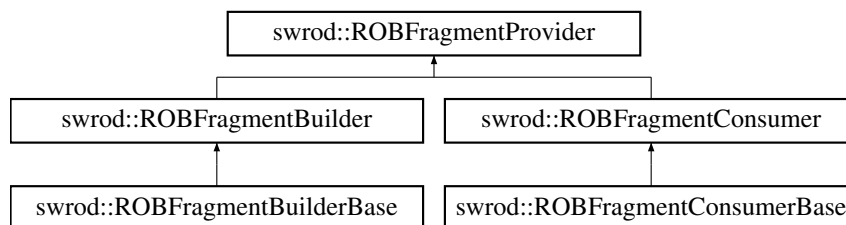
The documentation for this class was generated from the following file:

- [swrod/ROBFragmentConsumerBase.h](#)

4.28 swrod::ROBFragmentProvider Class Reference

```
#include <ROBFragmentProvider.h>
```

Inheritance diagram for [swrod::ROBFragmentProvider](#):



Public Member Functions

- virtual void [addConsumer](#) (const std::shared_ptr< [ROBFragmentConsumer](#) > &consumer)
- virtual void [forwardROBFragment](#) (const std::shared_ptr< [swrod::ROBFragment](#) > &fragment)

Protected Attributes

- std::shared_ptr
< [ROBFragmentConsumer](#) > **m_consumer**
- std::function< void(const
std::shared_ptr
< [swrod::ROBFragment](#) > &)> **m_function**

4.28.1 Detailed Description

This class defines an interface for a supplier of ROB fragments and provides default implementation of this interface. This implementation maintains a list of ROB fragment consumers by adding a new consumer to the end of the list.

4.28.2 Member Function Documentation

4.28.2.1 virtual void [swrod::ROBFragmentProvider::addConsumer](#) (const std::shared_ptr< [ROBFragmentConsumer](#) > &
consumer) [virtual]

Adds a new ROB Fragment Consumer to this provider. Default implementation adds the given consumer to the end of the consumers list.

Parameters

<i>in</i>	<i>consumer</i>	A new ROB fragment consumer.
-----------	-----------------	------------------------------

4.28.2.2 `virtual void swrod::ROBFragmentProvider::forwardROBFragment (const std::shared_ptr< swrod::ROBFragment > & fragment) [inline], [virtual]`

Default implementation passes the given ROB fragment to the first consumer in the list.

Parameters

<i>in</i>	<i>fragment</i>	ROB fragment to be passed to the consumers.
-----------	-----------------	---

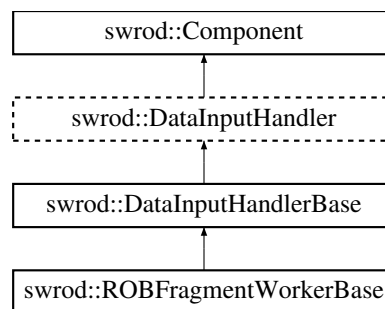
The documentation for this class was generated from the following file:

- swrod/ROBFragmentProvider.h

4.29 swrod::ROBFragmentWorkerBase Class Reference

```
#include <ROBFragmentWorkerBase.h>
```

Inheritance diagram for swrod::ROBFragmentWorkerBase:



Public Member Functions

- [ROBFragmentWorkerBase](#) (const InputLinkVector &links, const boost::property_tree::ptree &config, const [Core](#) &core, const std::shared_ptr< [DataInput](#) > &input, detail::InputCallback callback)

Protected Attributes

- const uint32_t **m_ROB_id**
- const uint32_t **m_max_message_size**
- CustomProcessingFramework::TriggerInfoExtractor **m_trigger_info_extractor**
- CustomProcessingFramework::DataIntegrityChecker **m_data_integrity_checker**

Additional Inherited Members

4.29.1 Detailed Description

This class provides a reusable basic implementation of a worker thread for a ROB fragment builder algorithm. Deriving custom class from the [ROBFragmentWorkerBase](#) is a recommended way of implementing a worker thread for a custom ROB fragment building algorithm.

4.29.2 Constructor & Destructor Documentation

4.29.2.1 `swrod::ROBFragmentWorkerBase::ROBFragmentWorkerBase (const InputLinkVector & links, const boost::property_tree::ptree & config, const Core & core, const std::shared_ptr< DataInput > & input, detail::InputCallback callback) [inline]`

Creates a new worker for the ROB fragment builder with the given configuration.

Parameters

<i>in</i>	<i>links</i>	Input links to be used for receiving data
<i>in</i>	<i>config</i>	Configuration object that contains parameters for the corresponding fragment builder (SwRodFragmentBuilder)
<i>in</i>	<i>core</i>	a reference to the swrod::Core object
<i>in</i>	<i>input</i>	the object to be used to subscribe for receiving data
<i>in</i>	<i>callback</i>	function to be called when new data arrive

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

- swrod/ROBFragmentWorkerBase.h