The Compact Muon Solenoid Experiment
TriDAS Trigger and Data Acquisition

RU Builder
User Manual
Version rubuilder_G_V01_12_00
August 21, 2008

Authors: R. Mommsen, S. Murray
CI identifier EVB_D_18306

Fermilab

Revision History

Date	Version	Description	Author
July 11, 2003	0.1	Document creation	S. Murray
July 31, 2003	1.0	Finalization of version 1.0	J. Gutleber
May 24, 2004	2.0	Updated for version 2.0 of the EVB, now referred to as the RU builder	S. Murray
November 1, 2004	2.1	Updated for version 2.1 of the RU builder	S. Murray
May 2, 2005	3.0	Updated for version 3.0 of the RU builder	S. Murray
July 4, 2005	3.2	Updated for version 3.2 of the RU builder	S. Murray
August 16, 2005	3.3	Updated for version 3.3 of the RU builder	S. Murray
August 26, 2005	3.4	Updated for version 3.4 of the RU builder	S. Murray
October 5, 2005	3.5	Updated for version 3.5 of the RU builder	S. Murray
October 27, 2005	3.6	Updated for version 3.6 of the RU builder	S. Murray
January 30, 2006	3.7.1	Updated for version 3.7.1 of the RU builder	S. Murray
March 29, 2006	3.8	Updated for version 3.8 of the RU builder	S. Murray
May 8, 2006	3.9	Updated for version 3.9 of the RU builder	S. Murray
September 18, 2006	3.9.8	Updated for version 3.9.8 of the RU builder	S. Murray
August 7, 2007	RUBUILDER_G_V1_3_0	Updated for the latest version of the RU builder	S. Murray
October 11, 2007	rubuilder_G_V01_06_00	Updated for the latest version of the RU builder. Added a description of the two ways to send triggers to the EVM. Added a simple description of all the RU-builder I2O messages. Added a description of the two RU- builder threading-models. Added more information about the contents of the RU builder application web-pages.	S. Murray
August 21, 2008	rubuilder_G_V01_12_00	Updated naming of eventNumber* monitoring parameters and document the fault tolerant behavior of the RU.	R. Mommsen

CI Record

Field	Description
CI Identifier	EVB_D_18306
Description	Describes the RU builder for an integrator who will put the builder into a full DAQ system, and for a DAQ operator who needs to understand how the RU builder works in order to efficiently diagnose problems.
Point of Contact	R. Mommsen (Remigius.Mommsen@cern.ch) S. Murray (Steven.Murray@cern.ch)
Physical Location	http://cms-ru-builder.web.cern.ch/cms-ru-builder/RUBUILDER_G_V1_6_0.doc

Table of Contents

List of Figures

List of Tables

1 Introduction

The RU builder is a distributed XDAQ application that is part of a larger system called the event builder (EVB). The CMS data acquisition group is presently developing the EVB as described in the TriDAS TDR [1]. This document explains how to obtain, build and configure version rubuilder_G_V01_06_00 of the RU builder.

1.1 Document purpose and scope

This document has two goals. The first is to enable the reader to integrate the RU builder into a "running system", and the second is to help the reader understand how the RU builder works so that they may be able to diagnose any problems which may occur. A "running" system is composed of the RU builder itself, a trigger source, one or more event data sources, one or more data sinks and a run-control system. The RU builder cannot run without the components just listed. This document describes how to obtain, build and configure the RU builder. This document does not describe the other components of a "running system", such as run control software or how to setup a trigger or event data source. Although this document does describe some of the internal workings of the RU builder, the reader is referred to the comments within the source code as the authoritative guide on the subject. The code has been structured and commented so that it can be easily read and understood. It is recommended to use doxygen to generate documentation from the code, as compatible comment tags have been used. If the reader is not familiar with doxygen, then they are referred to its website:

http://www.doxygen.org

Doxygen documentation for this particular version of the RU builder can be browsed on-line at the official RU builder web site:
http://cms-ru-builder.web.cern.ch/cms-ru-builder/rubuilder_G_V01_06_00_doxygen/html/index.htm

It must be emphasized that the RU builder is still under development and subject to change. No description of the RU builder given in this document can be relied upon to be valid beyond this release.

1.2 Intended readership

This document has been written with two audiences in mind, system integrators and DAQ operators. A system integrator is someone that needs to integrate the RU builder into a data acquisition system (DAQ). It is assumed that the DAQ system is based on the XDAQ framework. If the reader is not familiar with this framework, then they are referred to the XDAQ website: http://xdaq.web.cern.ch/xdaq. A DAQ operator should know how the RU builder works so that they can efficiently diagnose problems that may occur during a DAQ shift.

1.3 References

[1] The CMS collaboration, The Trigger and Data Acquisition project, Volume II, Data Acquisition & High-Level Trigger. CERN/LHCC 2002-26, ISBN 92-9083-111-4

1.4 Definitions, Acronyms and Abbreviations

BU	Builder Unit	**I2O**	Intelligent Input/Output
CVS	Concurrent Versioning System	**RCMS**	CMS Run-control system
DAQ	Data Acquisition system	**RU**	Readout Unit
EVB	Event builder	**RUI**	Readout Unit Input
EVM	Event Manager	**TA**	Trigger Adapter
FED	Front End Driver	**TDR**	Technical Design Report
FSTN	Finite State Transition Network	**TriDAS**	Trigger and Data Acquisition
FU	Filter Unit	**XDAQ**	Cross platform data acquisition toolkit

1.5 RU-builder application class names

The following table shows the names of the RU builder applications and their corresponding C++ class names. The C++ class names are used when making XDAQ configurations for the RU builder. They are also used on the web-pages of the running RU-builder applications.

Application name		C++ class name
Acronym	**Full name**	
BU	Builder unit	rubuilder::bu::Application
EVM	Event manager	rubuilder::evm::Application
FU	Filter unit	rubuilder::fu::Application
RU	Readout unit	rubuilder::ru::Application
RUI	Readout-unit input	rubuilder::rui::Application
TA	Trigger adapter	rubuilder::ta::Application

Table 1 RU builder application class names

2 RU builder overview

2.1 How the RU builder connects to the components around it

The RU builder is a component of a larger system called the event builder (EVB). The EVB is a distributed application that reads out event fragments from one set of nodes and assembles them into entire events in another set of nodes. Two scenarios dictate how the RU builder connects to the components around it. In the first scenario, the RU builder is connected to the rest of the EVB. This is how the RU builder will be connected during official CMS data taking runs. In the second scenario the RU builder is connected within a local DAQ or test-beam system. The difference between the two scenarios is how the triggers are sent to the EVM. Triggers are sent to the EVM via the RUI/EVM interface when the RU builder is connected to the rest of the EVB. Triggers are sent to the EVM via the TA/EVM interface when the RU builder is used in a local DAQ or test-beam system. Figure 1 depicts the first scenario and figure 2 the second.

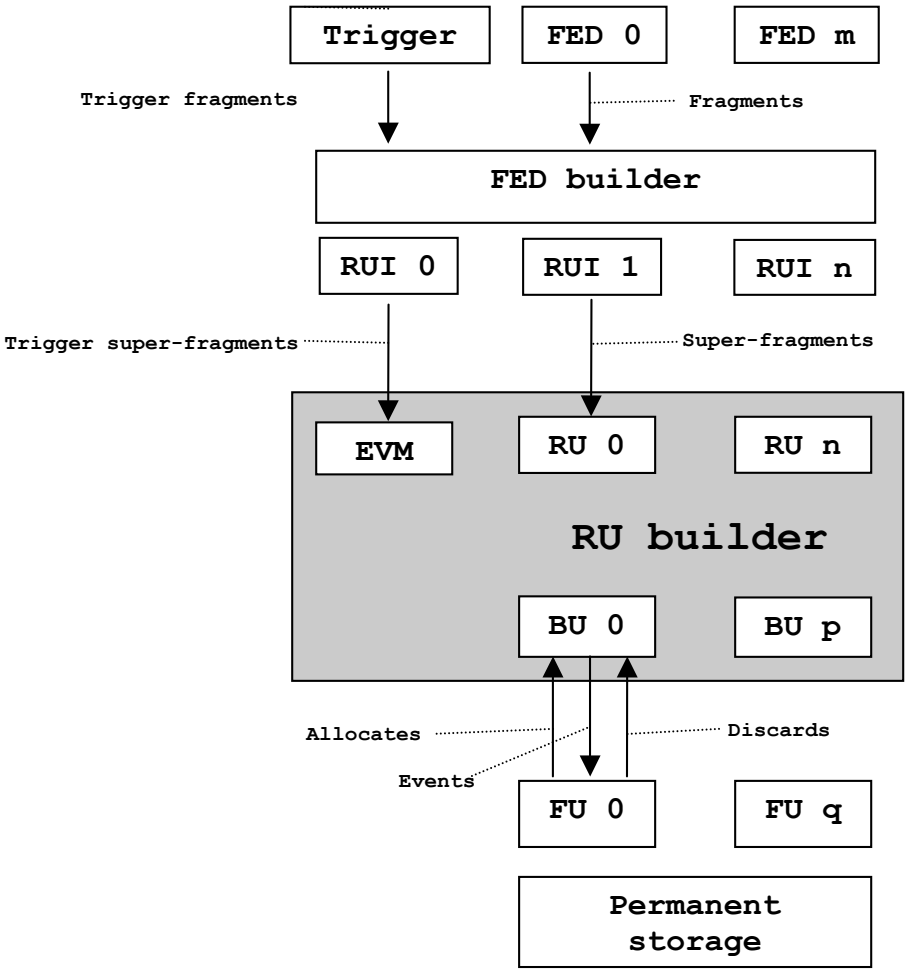

Figure 1 RU-builder connected to the rest of the EVB

The external interfaces of the RU builder assume that triggers are given to the EVM in the same order as their corresponding event data is given to the RUs.

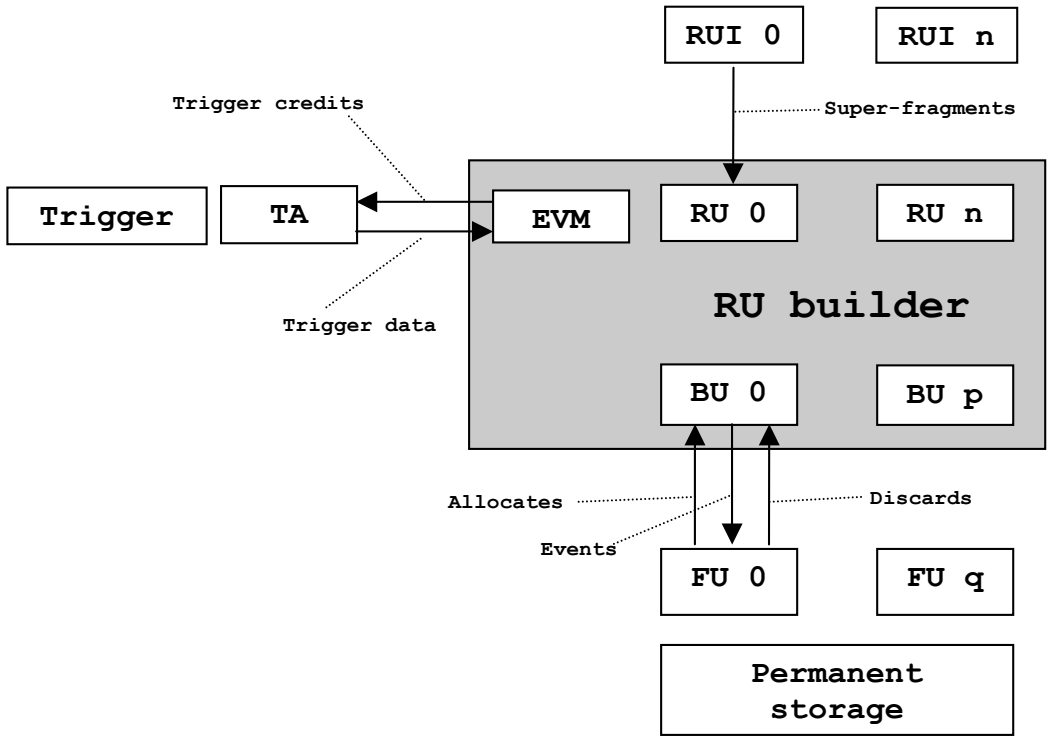

Figure 2 RU-builder connections within a local-DAQ or test-beam system

The external interfaces of the RU builder assume that triggers are given to the EVM in the same order as their corresponding event data is given to the RUs.

2.2 What the EVB does

For each event the EVB:

- Reads out the trigger data. This trigger data will become the first super-fragment of the event.
- Reads out the fragments of the event from the detector front-end drivers (FEDs).
- Builds the fragments into RU super-fragments using the FED builder.
- Builds the whole event using the RU builder. The whole event is the trigger super-fragment plus the set of RU super-fragments.
- Decides whether or not the event is interesting for physics using the filter units (FUs).
- Sends the event to permanent storage if it is interesting for physics, or discards it if it is not.

2.3 The RU-builder applications

The RU builder consists of a single event manager (EVM), one or more readout units (RUs) and one or more builder units (BUs). The EVM is responsible for controlling the flow of data through the RU builder. The RUs are responsible for buffering super-fragments until they are requested by the BUs. The BUs are responsible for building and buffering events until they are requested by the filter units (FUs).

The trigger adapter (TA), readout unit inputs (RUIs) and filter units (FUs) are external to the RU builder. The TA is used within local DAQ and test-beam systems to interface the DAQ trigger to the EVM. The RUIs are responsible for pushing super-fragment data from the FED builder into the RUs. The FUs are responsible for selecting interesting events for permanent storage.

3 RU-builder application FIFOs

The RU-builder applications use FIFOs to keep track of requests, trigger data and event data. Knowledge of these FIFOs is required in order to correctly configure the RU builder. This chapter is divided into three sections, one for the BU, one for the EVM and one for the RU. Each section gives a brief description of the application's behavior and how its FIFOs are used.

3.1 BU FIFOs

A BU is responsible for building events. An event is composed of one trigger super-fragment and N RU super-fragments, where N is the number of RUs. To understand the internal FIFOs of a BU, it is necessary to know its dynamic behavior. Figure 3 shows the internal FIFOs of a BU. With free capacity available, a BU requests the EVM to allocate it an event (**step 1**). The EVM confirms the allocation by sending the BU the event ID and trigger data of an event (**step 2**). This trigger data is the first super-fragment of the event. The BU now requests the RUs to send it the rest of the event's super-fragments (**step 3**). The BU builds the super-fragments it receives from the RUs (**step 4**) into a whole event within its resource table (**step 5**). FUs can ask a BU to allocate them events (**step 6**). A BU services a FU request by sending the FU a whole event (**step 7**). When a FU has finished with an event, it tells the BU to discard it (**step 8**).

The `eventIdFIFO`, `blockFIFO`, `requestFIFOs` and `discardFIFO` store incoming message until they can be processed. The `fullResourceFIFO` stores which in memory events are built.

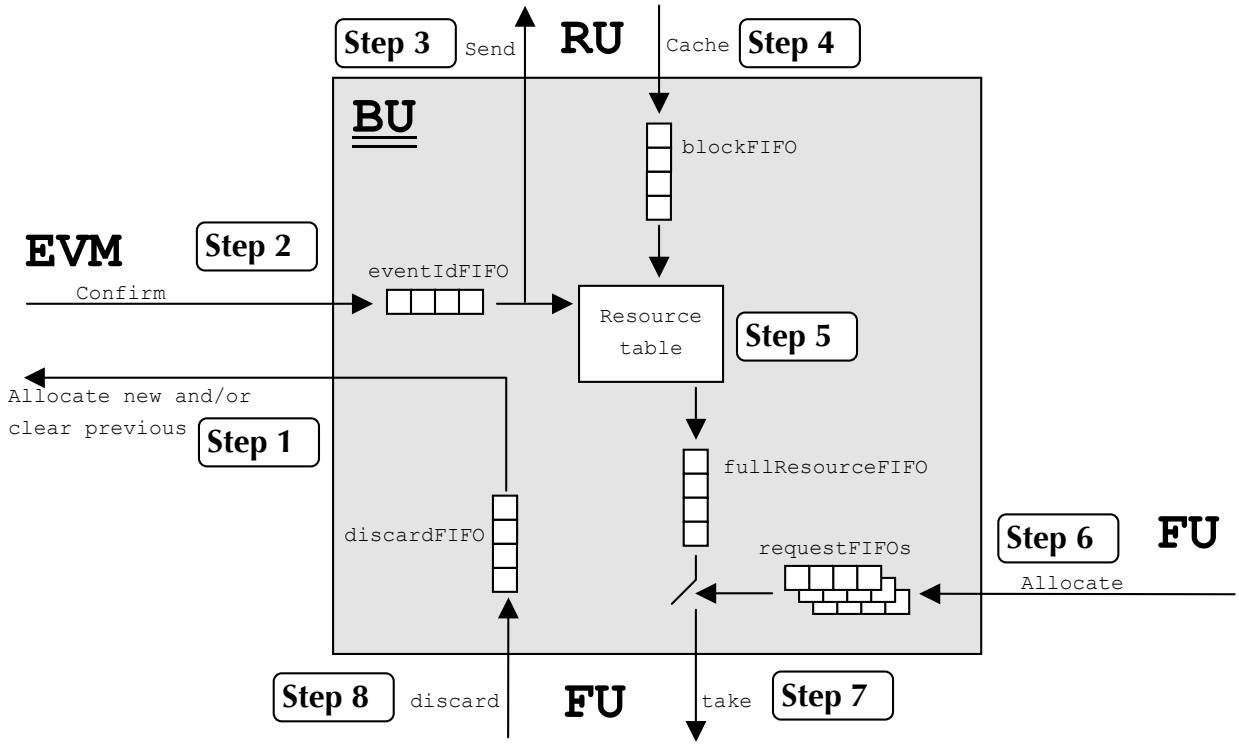

Figure 3 BU FIFOs

3.2 EVM FIFOs

The EVM is responsible for controlling the flow of event data through the RU builder. To understand the internal FIFOs of the EVM, it is necessary to know its dynamic behavior. As explained in "Section 2.1 How the RU builder connects to the components around it", triggers are sent to the EVM using the RUI/EVM interface when the RU builder is connected to the rest of the EVB, and via the TA/EVM interface when connected within a local DAQ or test-beam system. This section is divided into two sub-sections, tone describing the EVM FIFOs when the triggers are sent via the RUI/EVM interface, and the other describing the EVM FIFOs when triggers are sent via the TA/EVM interface.

3.2.1 EVM FIFOs when triggers are sent via the RUI/EVM interface

Figure 4 shows the internal FIFOs of the EVM. The RUI sends the EVM the trigger data of an event (**step 1**). The EVM pairs the trigger data with a free event ID (**step 2**). The EVM also requests the RUs to readout the event's data (**step 3**). A BU with the ability to build an event will ask the EVM to allocate it an event (**step 4**). Within such a request, a BU will normally give back the ID of an event to be cleared. For each cleared event ID, the EVM makes the ID a free event ID (**step 5**). The EVM confirms the allocation of an event by sending the requesting BU the event ID and trigger data of the allocated event (**step 6**).

The `triggerFIFO`, `clearedEventIdFIFO` and `requestFIFO` store incoming messages until they can be processed. The `pairFIFO` keeps track of the "event ID / trigger data" pairs that have yet to be sent to requesting BUs. The `freeEventIdFIFO` stores the ids of free events.

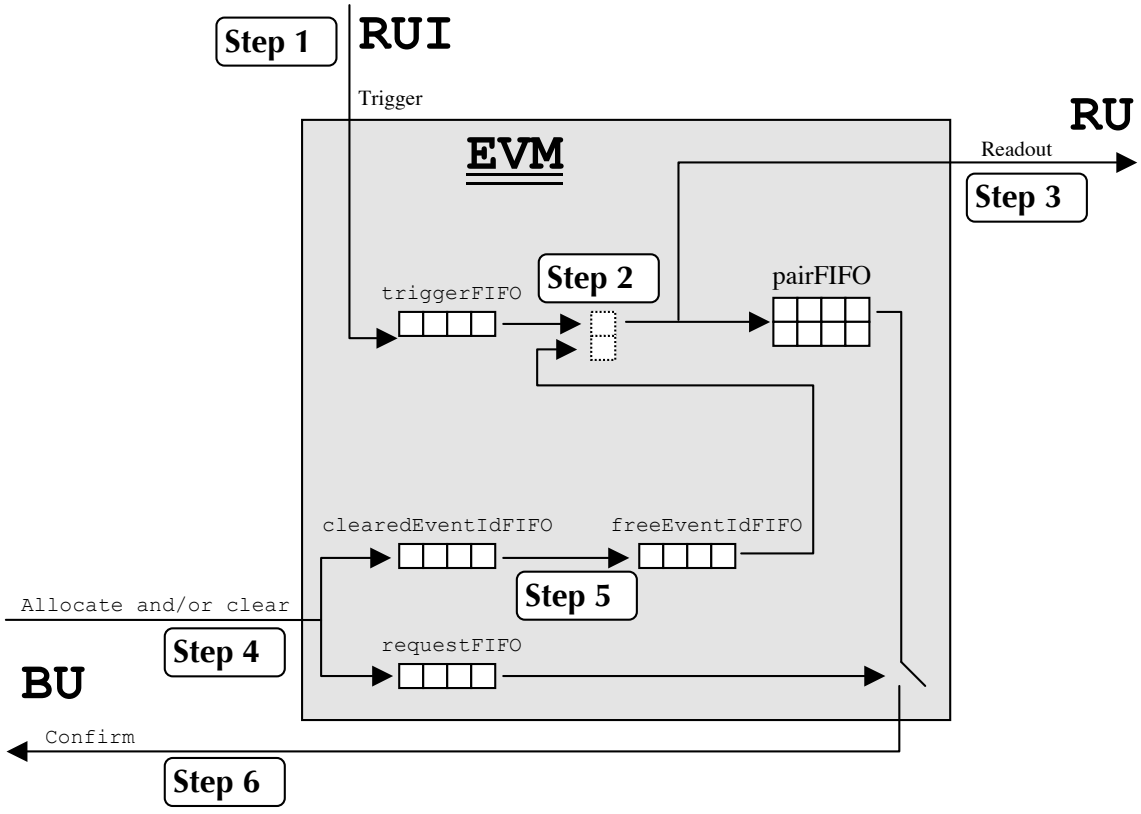

Figure 4 EVM FIFOs when trigger sent via RUI/EVM interface

3.2.2 EVM FIFOs when triggers are sent via the TA/EVM interface

Figure 5 shows the internal FIFOs of the EVM. The EVM tells the TA the capacity of the RU builder by sending it trigger credits (**step 1**). One trigger credit represents the ability to build one event. Given a credit, the TA sends the EVM the trigger data of an event (**step 2**). The EVM pairs the trigger data with a free event ID (**step 3**). The EVM also requests the RUs to readout the event's data (**step 4**). A BU with the ability to build an event will ask the EVM to allocate it an event (**step 5**). Within such a request, a BU will normally give back the ID of an event to be cleared. For each cleared event ID, the EVM sends a trigger credit to the TA and makes the ID a free event ID (**step 6**). The EVM confirms the allocation of an event by sending the requesting BU the event ID and trigger data of the allocated event (**step 7**).

The `triggerFIFO`, `clearedEventIdFIFO` and `requestFIFO` store incoming messages until they can be processed. The `pairFIFO` keeps track of the "event ID / trigger data" pairs that have yet to be sent to requesting BUs. The `freeEventIdFIFO` stores the ids of free events for which trigger credits have been sent to the TA.

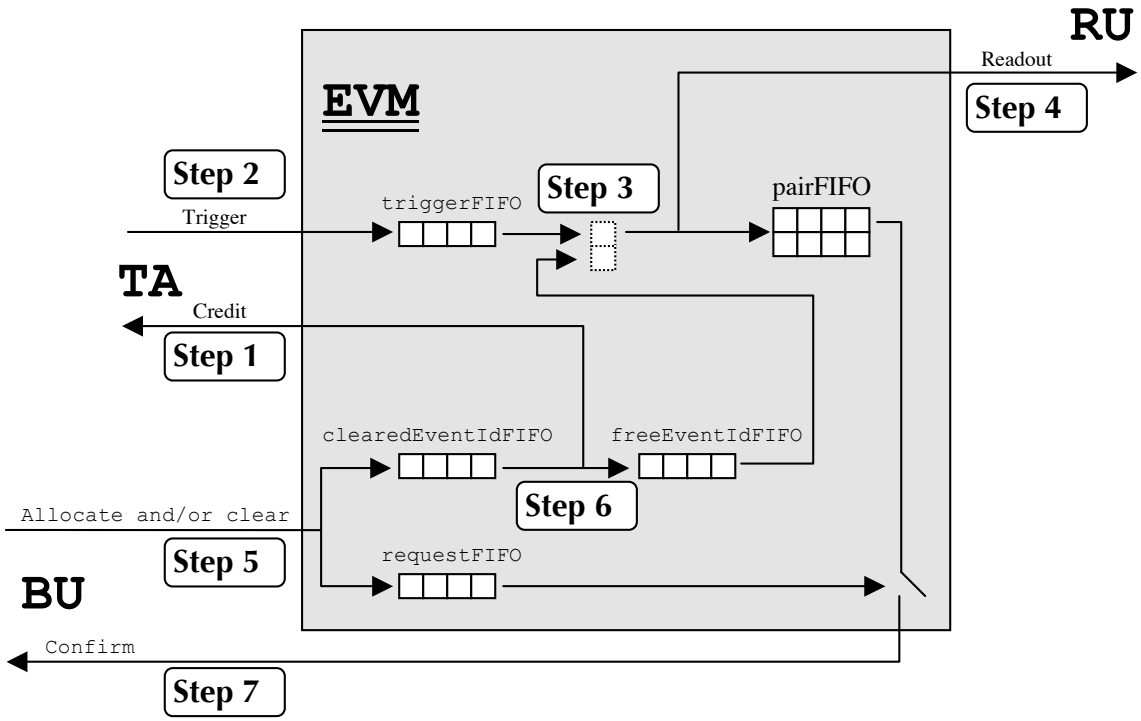

Figure 5 EVM FIFOs

3.3 RU FIFOs

A RU is responsible for buffering super-fragments until they are request by the BUs. To understand the internal FIFOs of a RU it is necessary to know its dynamic behavior. Figure 6 shows the internal FIFOs of a RU. The EVM sends a RU an "event ID / trigger event number" pair when it asks the RU to readout the corresponding event's data (**step 1**). In parallel, the RUI informs the RU of event data that is ready to be processed (**step 2**). A RU places each super-fragment for which it has received a pair into the fragment lookup table (**step 3**). BUs ask RUs to send them the super-fragments of the events they are building (**step 4**). A RU services a BU request by retrieving the super-fragment from its fragment lookup table and asking the BU to cache the super-fragment (**step 5**).

All of the internal FIFOs of a RU, that is to say the `pairFIFO`, `blockFIFO` and `requestFIFOs`, store incoming messages until they can be processed.

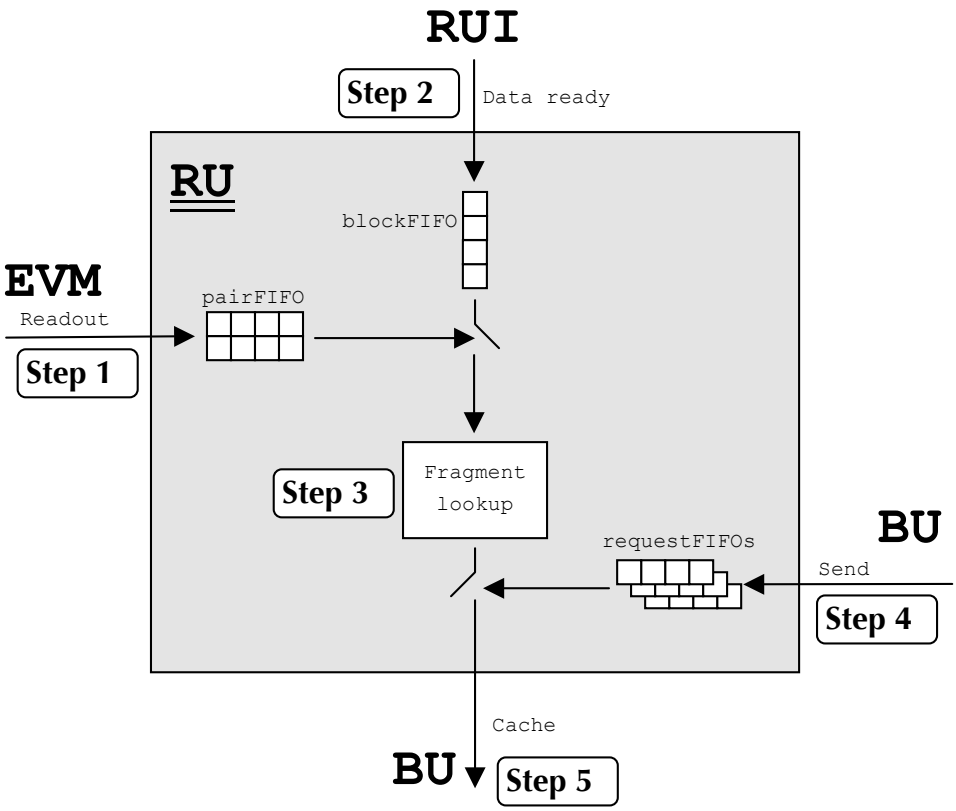

Figure 6 RU FIFOs

The RU can be set into a fault tolerant mode in which it continues to operate if the RUI stops sending super-fragments. In that case, the RU will go into the 'TimedOutTolerating' state and serve empty super-fragments for each BU request and ignore the EVM. If data is again made available by the RUI, the RU synchronizes the `pairFIFO` and `blockFIFO` by removing messages if earlier event numbers. Once the FIFOs are aligned, the RU goes back to the normal processing ('Enabled' state).

4 RU-builder I2O interface

This chapter is divided into two sections. The first gives an overview of the RU builder I2O interface which shows which I2O messages are sent between which applications. This overview will help the reader understand the message counters displayed on the web-pages of the running applications. The second section of this chapter goes into the details of the external I2O interfaces of RU builder. This will help the reader integrate the RU builder with the DAQ components around it.

4.1 Overview of the RU-builder I2O interface

All the I2O messages of the EVB, including the internal and external messages of the RU builder, are defined in the package:

`TriDAS/daq/interface`

The I2O function codes of all the RU builder I2O messages are given in the file:

`TriDAS/daq/interface/shared/include/i2oXFunctionCodes.h`

The C structures that define the I2O messages are in the file:

`TriDAS/daq/interface/evb/include/i2oEVBMsgs.h`

The I2O interface of the RU builder is subject to change. The description of the interface provided by this document cannot be relied upon to be valid beyond this release.

As explained in "Section 2.1 How the RU builder connects to the components around it", triggers are sent to the EVM using the RUI/EVM interface when the RU builder is connected to the rest of the EVB, and via the TA/EVM interface when connected within a local DAQ or test-beam system. Figures 7 and 8 show the RU builder I2O messages in these two scenarios.

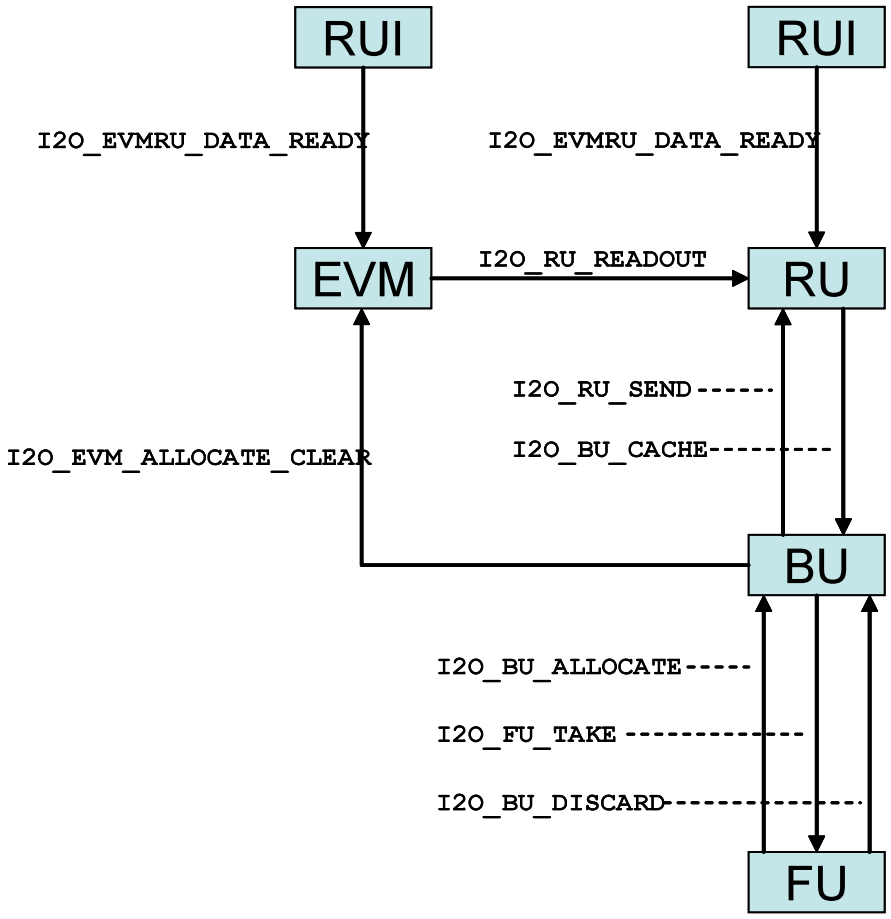

Figure 7 RU-builder I2O-messages when connected to the rest of the EVB

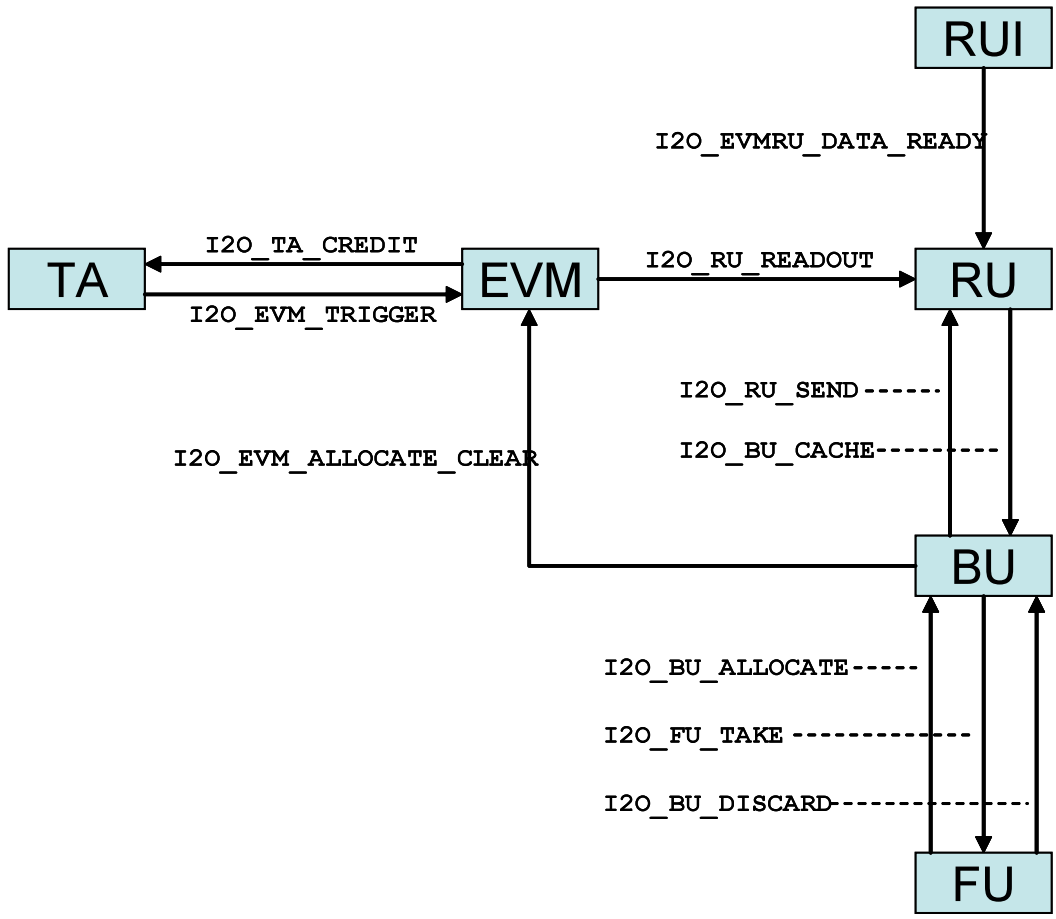

Figure 8 RU-builder I2O-messages when connected within a local-DAQ or test-beam system

4.2 The external I2O interfaces of the RU builder

Figure 9 shows the external I2O interfaces of the RU builder: the TA/EVM interface, the RUI/RU interface and the BU/FU interface. This chapter is divided into three sub-sections, one for each interface.

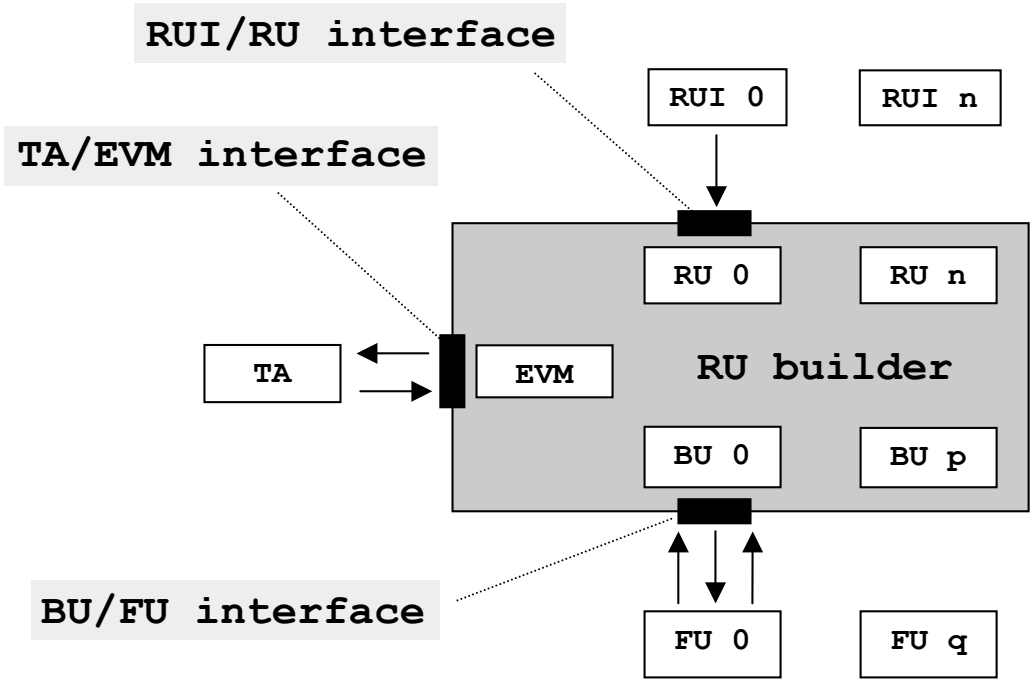

Figure 9 External I2O interfaces of the RU builder

4.2.1 TA/EVM interface

The TA/EVM interface specifies how:

- The EVM gives the TA trigger credits
- The TA gives the EVM trigger data

Figure 10 is a sequence diagram describing the protocol between the EVM and the TA.

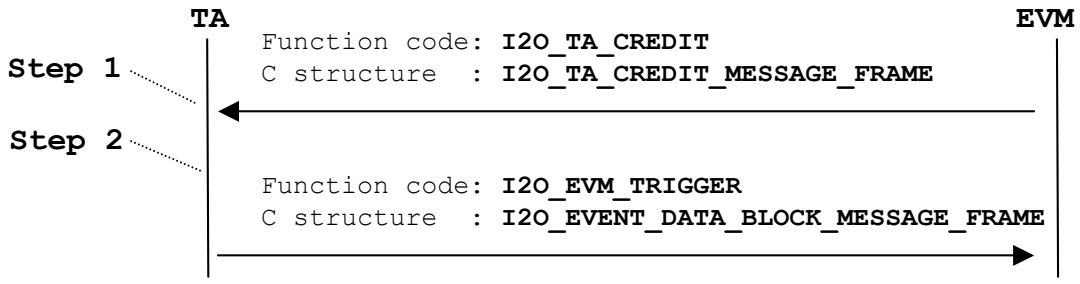

Figure 10 TA/EVM interface sequence diagram

The EVM communicates with the TA using a credit-based mechanism. The EVM tells the TA the current capacity of the RU builder by sending the TA a trigger credit count (**step 1**). One trigger credit represents the RU builder's ability to build one event. The TA should only send the EVM trigger data for as many events as the EVM has given the TA credits (**step 2**). The TA is responsible for getting / receiving trigger data from the trigger and for providing backpressure to the trigger as necessary.

The **I2O_TA_CREDIT_MESSAGE_FRAME** C structure is as follows:

```
typedef struct _I2O_TA_CREDIT_MESSAGE_FRAME
{
I2O_PRIVATE_MESSAGE_FRAME PvtMessageFrame;
U32 nbCredits;
} I2O_TA_CREDIT_MESSAGE_FRAME, *PI2O_TA_CREDIT_MESSAGE_FRAME;
```

The EVM must fill **nbCredits**.

The **I2O_EVENT_DATA_BLOCK_MESSAGE_FRAME** C structure is as follows:

```
typedef struct I2O_EVENT_DATA_BLOCK_MESSAGE_FRAME
{
I2O_PRIVATE_MESSAGE_FRAME PvtMessageFrame;
U32 eventNumber;
U32 nbBlocksInSuperFragment;
U32 blockNb;
U32 eventId;
U32 buResourceId;
U32 fuTransactionId;
U32 nbSuperFragmentsInEvent;
U32 superFragmentNb;
U32 padding;
} I2O_EVENT_DATA_BLOCK_MESSAGE_FRAME, *PI2O_EVENT_DATA_BLOCK_MESSAGE_FRAME;
```

The TA must fill:

- **eventNumber**
- **nbBlocksInSuperFragment**
- **blockNb**

Thee RU builder only supports single block trigger data. Therefore the TA must always set `nbBlocksInSuperFragment` to 1 and `blockNb` to 0

4.2.2 RU/RUI interface

The RU/RUI interface specifies how a RUI passes super-fragments to a RU. Figure 11 is a sequence diagram describing the protocol between the RUI and the RU.

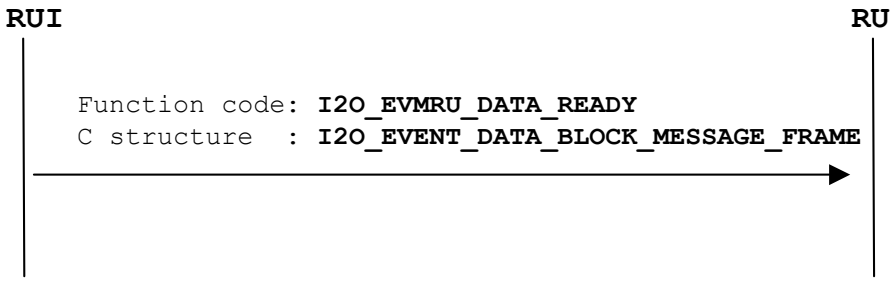

Figure 11 RU/RUI interface sequence diagram

A super-fragment is composed of one or more **I2O_EVENT_DATA_BLOCK_MESSAGE_FRAME**s. The **I2O_EVENT_DATA_BLOCK_MESSAGE_FRAME** C structure is as follows:

```
typedef struct I2O_EVENT_DATA_BLOCK_MESSAGE_FRAME
{
I2O_PRIVATE_MESSAGE_FRAME PvtMessageFrame;
U32 eventNumber;
U32 nbBlocksInSuperFragment;
U32 blockNb;
U32 eventId;
U32 buResourceId;
U32 fuTransactionId;
U32 nbSuperFragmentsInEvent;
U32 superFragmentNb;
U32 padding;
} I2O_EVENT_DATA_BLOCK_MESSAGE_FRAME, *PI2O_EVENT_DATA_BLOCK_MESSAGE_FRAME;
```

The RUI must fill:

- **eventNumber**
- **nbBlocksInSuperFragment**
- **blockNb**

The **nbBlocksInSuperFragment** field gives the number of blocks the super-fragment is composed of. The **blockNb** field indicates the block's position within the super-fragment. Blocks are numbered from **0** to **nbBlocksInSuperFragment – 1**.

4.2.3 BU/FU interface

The BU/FU interface specifies how:

- A FU requests events from a BU
- A BU sends an event to a FU
- A FU tells a BU to discard an event

Figure 12 is a sequence diagram describing the protocol between a BU and a FU.

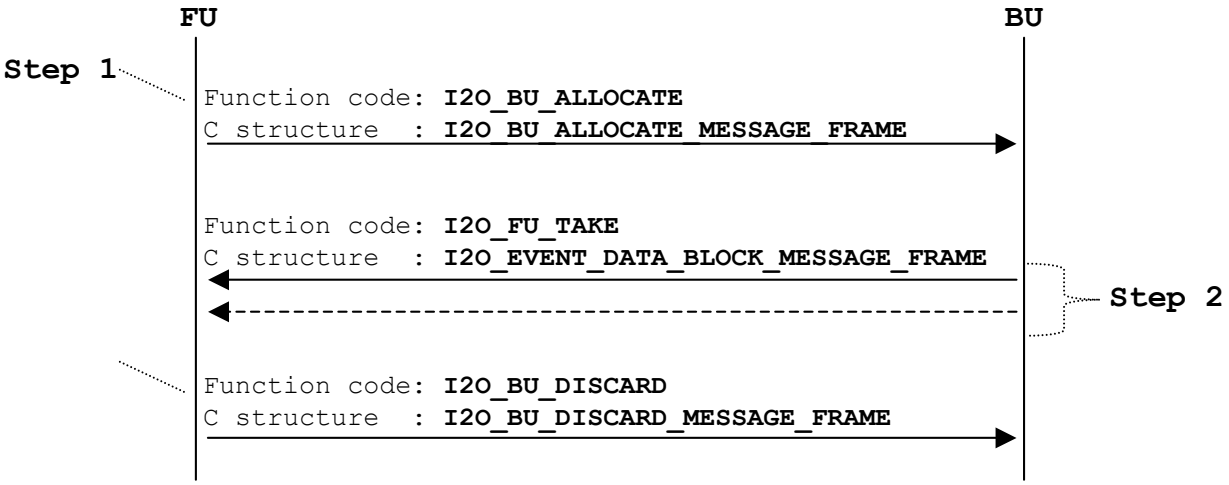

Figure 12 BU/FU interface sequence diagram

A FU requests a BU to allocate it one or more events (**step 1**). In response, the BU asks the FU to take the requested event data as a set of event data blocks (**step 2**). When a FU has finished processing one or more events, it tells the BU to discard them (**step 3**).

The BU/FU interface of this version of the RU builder does not support partial events. Partial events may be supported in a future version.

The **I2O_BU_ALLOCATE_MESSAGE_FRAME** C structure and its companion **BU_ALLOCATE** C structure are as follows:

```
typedef struct _BU_ALLOCATE
{
U32 fuTransactionId;
U32 fset;
} BU_ALLOCATE, *PBU_ALLOCATE;
typedef struct _I2O_BU_ALLOCATE_MESSAGE_FRAME {
I2O_PRIVATE_MESSAGE_FRAME PvtMessageFrame;
    U32                     n;
    BU_ALLOCATE             allocate[1];
} I2O_BU_ALLOCATE_MESSAGE_FRAME, *PI2O_BU_ALLOCATE_MESSAGE_FRAME;
```

The FU must fill:

n
allocate[]

The **n** field specifies the number of events the FU is requesting. The **allocate** field is an array of FU transaction ids and fragment sets. For each event a FU requests, the FU fills in the **fuTransactionId** field and the **fset** field of a **BU_ALLOCATE** C structure and puts it in the **allocate** array. The **fuTransactionId** field is a transaction ID that a FU can use to match its requests with the events it receives. A BU treats the **fuTransactionId** field as being opaque, in other words it is not interpreted. A BU will send back a copy of the **fuTransactionId** field in each of the **I2O_EVENT_DATA_BLOCK_MESSAGE_FRAME**s that make up the requested event. The **fset** field is a fragment set identifier. Fragment sets are a way to describe partial events. The **fset** field is ignored by the BU in this version of the RU builder, because this version does not support partial events.

The **I2O_EVENT_DATA_BLOCK_MESSAGE_FRAME** C structure is as follows:

```
typedef struct I2O_EVENT_DATA_BLOCK_MESSAGE_FRAME
{
I2O_PRIVATE_MESSAGE_FRAME PvtMessageFrame;
U32 eventNumber;
U32 nbBlocksInSuperFragment;
U32 blockNb;
U32 eventId;
U32 buResourceId;
U32 fuTransactionId;
U32 nbSuperFragmentsInEvent;
U32 superFragmentNb;
U32 padding;
} I2O_EVENT_DATA_BLOCK_MESSAGE_FRAME, *PI2O_EVENT_DATA_BLOCK_MESSAGE_FRAME;
```

The FU should only read:

- **nbSuperFragmentsInEvent**
- **superFragmentNb**
- **nbBlocksInSuperFragment**
- **blockNb**
- **buResourceId**
- **fuTransactionId**

An event is composed of **I2O_EVENT_DATA_BLOCK_FRAME**s. The **nbSuperFragmentsInEvent**, **superFragmentNb**, **nbBlocksInSuperFragment**, **blockNb** fields are used to identify the position of an event data block within an event. An event is composed of one trigger super-fragment plus N RU super-fragments, where N is the number of RUs. Therefore the **nbSuperFragmentsInEvent** field is set to the number of RUs plus 1. The **superFragmentNb** field is numbered from **0** to **nbSuperFragmentInEvent – 1**. The **blockNb** field is numbered from **0** to **nbBlocksInSuperFragment – 1**.

The **buResourceId** field is an opaque handle that a FU should use to identify events/resources to be discarded. The **fuTransactionId** field is the FU transaction ID of the FU request that caused the BU to reply with the current event.

The **I2O_BU_DISCARD** C structure is as follows:

```
typedef struct _I2O_BU_DISCARD_MESSAGE_FRAME {
I2O_PRIVATE_MESSAGE_FRAME PvtMessageFrame;
   U32                    n;
   U32                    buResourceId[1];
} I2O_BU_DISCARD_MESSAGE_FRAME, *PI2O_BU_DISCARD_MESSAGE_FRAME;
```

The FU must fill:

- **n**
- **buResourceId[]**

The **n** field specifies the number of events/resources to be discarded. The **buResourceId** field is an array of the ids of the BU resources to be discarded.

5 RU builder threading-models

The BU, EVM and RU applications have two threading models, event-driven and self-driven. Knowledge of these threading-models is required in order to correctly configure the RU builder . Only one threading model can be active during the lifetime of an application. The threading model used is chosen on the first ever configure of an application. The following exported parameters allow the user of the RU builder applications to choose which threading model is used:

```
xsd:boolean rubuilder::bu::application::selfDriven
xsd:boolean rubuilder::evm::application::selfDriven
xsd:boolean rubuilder::ru::application::selfDriven
```

A value of true indicates the self-driven threading-model whereas a value of false indicates event-driven. The default threading model is event-driven.

A RU builder application contains message FIFOs to store incoming messages until they can be processed. In both the event-driven and thread-driven models the underlying peer transport thread(s) push incoming messages onto these FIFOs. Which thread(s) pop messages off these FIFOs depends on the threading model being used.

In the case of the event-driven threading-model, the peer transport thread which pushed a message onto a message FIFO is also the same thread which pops and processes messages of the message FIFOs.

In the case of the self-driven threading-model, a separate worker thread pops and processes messages from the message FIFOs. A counting semaphore is used to synchronize the worker thread with the peer transport thread(s).

To date, the event-driven threading-model has been tested the most and was successfully used in the "CMS Magnet Test and Cosmic Challenge" (MTCC) of 2006 and in the global runs of 2007. The self-driven threading-model has been kept as a debugging aid, as the occupancy of the message FIFOs in this threading-model give a clear indication of whether or not an application is keeping up with its workload.

6 Application state machines

6.1 Commonalities of the application finite state machines

The finite state machines of the BUs, EVM and RUs have commonalities. Figure 13 shows the finite state transition network (FSTN) which all three types of application follow. There are four common behaviors. Firstly, all RU builder applications read and act upon configuration parameters when they receive a Configure SOAP message. Secondly, all RU builder applications only participate in event building when they are enabled. Thirdly, all RU builder applications throw away their internal data and any incoming I2O message frames when they are halted. Fourthly all RU builder applications go to the Failed state when they encounter a fatal error. A RU builder application cannot be recovered from the Failed state.

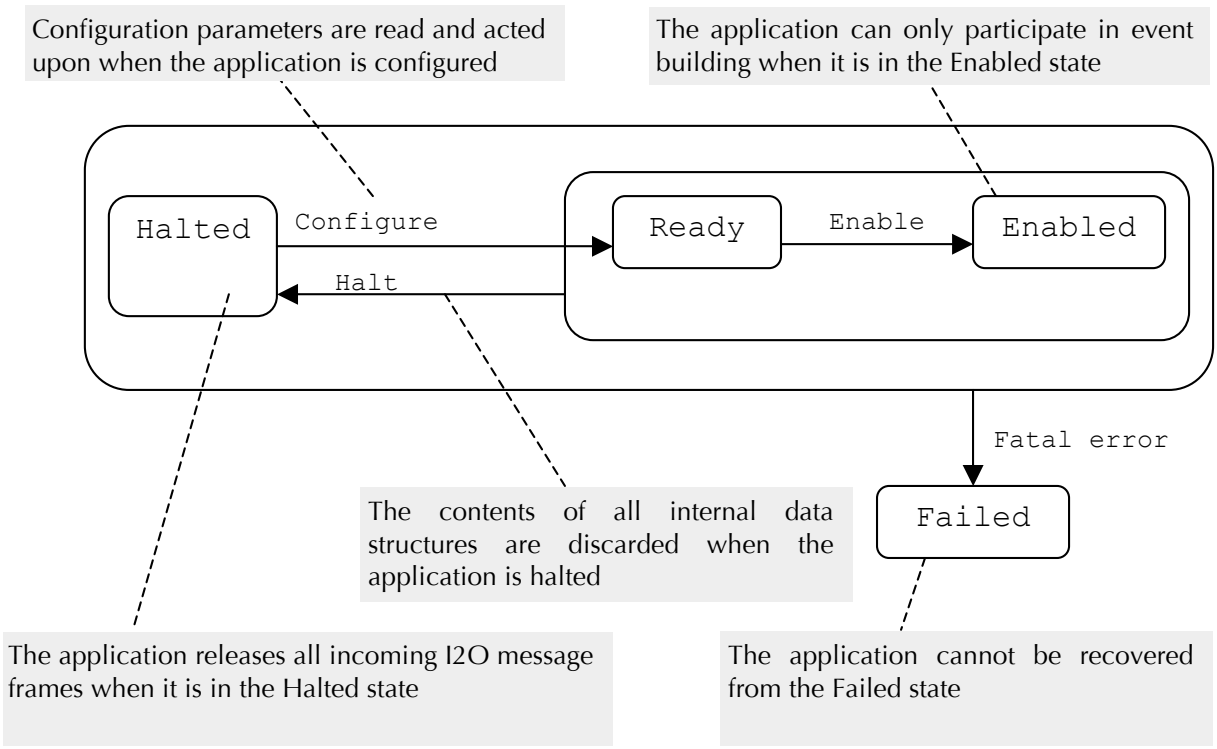

Figure 13 FSTN of a RU builder application

6.2 BU, EVM and RU finite state machines

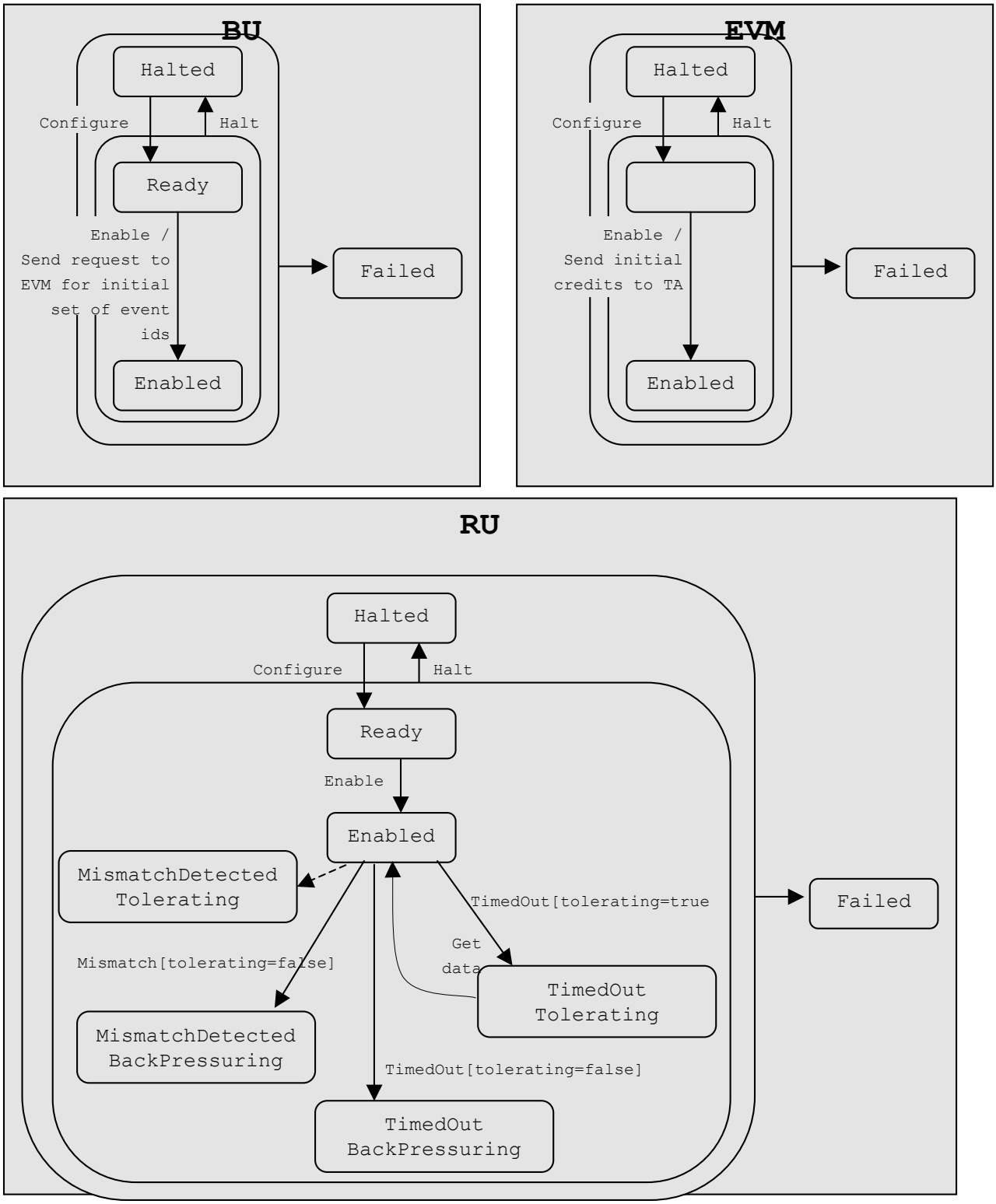

Figure 14 BU, EVM and RU FSTNs

The FSTNs specific to each type of RU builder application are shown in figure 10.

7 Starting the RU builder

The RU builder is part of a larger system called the event builder (EVB). Besides run-control, the RU builder communicates with a TA, one or more RUIs and one of more FUs. The RU builder cannot be started at any arbitrary moment in time; its start-up must be synchronized with that of the TA, RUIs and FUs. The RU builder and the EVB components it interacts with are XDAQ applications, and as such depend on one or more peer transports to communicate with each other. These peer transports must be up and running before these applications try to communicate with each other. To start the RU builder and the components it interacts with, run-control should do the following in order:

1. Start the necessary peer transports so that the RU builder and its surrounding applications can communicate
2. Start the TA if it is present, so that is can service credits from the EVM
3. Start the RU builder so that it can receive super-fragments from the RUIs
4. Start the RUIs and FUs so they can start pushing in super-fragments and extracting events respectively

The RU builder is a distributed application whose nodes (BUs, EVM and RUs) need to be started up in a specific order. To put the RU builder into the state where it will build events, run-control should do the following in order:

1. Send Configure to all of the RU builder applications
2. Send Enable to all of the RUs
3. Send Enable to the EVM
4. Send Enable to the BUs

State changes are synchronous. There is no need to poll the state of an application after a request to change state.

The RUs have to be enabled first because they have to be ready to receive “event number” / “event ID” pairs from the EVM. The EVM can start sending these pairs immediately after it has been enable.

The EVM has to be enabled before the BUs so that it is ready to service their requests for event ids. BUs can start requesting event ids as soon as they are enabled. Enabling the EVM causes it to send an initial trigger credit count to the TA if it is present. The number of initial credits is equal to the total number of event ids in the RU builder. As soon as trigger data arrives at the EVM, the EVM sends “event number” / “event ID” pairs to the RUs. As explained in the previous paragraph, this is why the RUs have to be enabled before the EVM.

Enabling a BU causes it to send its initial request for event ids to the EVM. The number of initial event ids requested is equal to the maximum number of event ids the BU is allowed to acquire at any single moment in time.

8 Stopping the RU builder

The current version of RU builder foresees two ways of stopping:

- Stop the trigger and event data entering the RU builder
- Halt all of the RU builder applications

When stopping the trigger and event data entering the RU builder, it is useful to know when the RU builder is flushed. This can be found out by reading the following tri-state exported parameter of the EVM:

`xsd:unsignedInt rubuilder::evm::Application::ruBuilderIsFlushed`

The possible values of this parameter are:

0 : FALSE
1 : TRUE
2 :UNDEFINED, e.g. The EVM is in the Halted state.

The RU builder is said to be flushed when it is empty and the data rate is 0. The sample period of the rate calculation is defined by the value of the following exported parameter:

`rubuilder::evm::Application::monitoringSleepSec`

Halting a RU builder application causes it to discard (destroy) all the data in its internal data structures and to release all incoming I2O messages.

9 Exported configuration parameters

Configuration parameters need to be set before an application is sent a Configure SOAP message. Table 2 lists the exported control parameters of each type of RU builder application. The type and default value of each parameter is given.

Application	Parameter name	XData type	Value
BU, EVM & RU	nbEvtIdsInBuilder	UnsignedInteger32	4096
BU, EVM & RU	ageMessages	Boolean	true
BU, EVM & RU	msgAgeLimitDtMSec	UnsignedInteger32	1000
BU, EVM & RU	exitOnFail	Boolean	false
BU	blockFIFOCapacity	UnsignedInteger32	16384
BU	discardFIFOCapacity	UnsignedInteger32	65536
BU	I2O_EVM_ALLOCATE_CLEAR_Packing	UnsignedInteger32	8
BU	maxEvtsUnderConstruction	UnsignedInteger32	64
BU	requestFIFOCapacity	UnsignedInteger32	1024
BU	I2O_RU_SEND_Packing	UnsignedInteger32	8
EVM	sendCreditsWithDispatchFrame	Boolean	false
EVM	triggerFIFOCapacity	UnsignedInteger32	4096
EVM	I2O_RU_READOUT_Packing	UnsignedInteger32	8
EVM	I2O_TA_CREDIT_Packing	UnsignedInteger32	8
RU	blockFIFOCapacity	UnsignedInteger32	16384
Legend BU = rubuilder::bu::Application EVM = rubuiilder::evm::Application RU = rubuilder::ru::Application			

Table 2 Exported configuration parameters

The default values are set when the RU builder application is instantiated. The default values have been chosen with the goal of covering the majority of use-cases for the RU builder. A user should rarely need to diverge from these default values.

The following assumptions were made when calculating the default values of the RU builder's configuration parameters:

- A RU builder is composed of 64 BUs and 64 RUs.
- A RU has 64MB of physical memory for caching super-fragments
- An event is 1MB
- An event is made up of 64 super-fragments (1 per RU) of equal size; therefore the size of a super-fragment is 16KB.
- The block size (size of an I2O message frame used to transport event data) is 4KB
- The RUIs only give as many events to the RUs as the TA gives triggers to the EVM
- The maximum number of FUs per BU is 64
- A FU will never have more than 1024 outstanding requests for events
- Fast control messages are sent if they are older than 1 second
- The packing factor of fast control messages is 8

The need to know the total number of event ids in the RU builder is common to all three types of RU builder applications.

- The total number of event ids in the RU builder is a function of RU memory. Assuming each RU has 64MB of memory for buffering super-fragments and that the size of an event is 1 MB:

  `rubuilder::bu::Application::nbEvtIdsInBuilder`,
  `rubuilder::evm::Application::nbEvtIdsInBuilder`,
  `rubuilder::ru::Application::nbEvtIdsInBuilder`

= sum of the memory of all RUs / size of an event
= (64 × 64MB) / 1MB
= 4096

The BUs, and EVM send fast control messages.

- Fast control messages are sent if they are older than 1 second

  `rubuilder::bu::Application::ageMessages`,
  `rubuilder::evm::ApplicationageMessages`
  = true
  `rubuilder::bu::Application::msgAgeLimitDtMSec`
  `rubuilder::evm::Application::msgAgeLimitDtMSec`
  = 1000

It has been assumed that all events have the fixed size of 1MB. If the RU builder is to build events of varying sizes then the appropriate safety factor needs to be taken into account when calculating the number of event ids in the RU builder.

The default values of the BU control parameters were calculated as follows:

- To prevent a BU from monopolizing event ids, each BU has a maximum number of event ids it can acquire at any moment in time. Assuming all BUs are equal, each BU is allowed to acquire:

  rubuilder::bu::Application::maxEvtsUnderConstruction
  = **nbEvtIdsInBuilder** / number of BUs
  = 4096 / 64
  = 64

- The blockFIFO of a BU (see figure 2 in section 3.2) is responsible for buffering incoming event data. In the worst case this FIFO would have to buffer the blocks of all outstanding requests for event data:

  rubuilder::bu::Application::blockFIFOCapacity
  = **maxEvtIdsUnderConstruction** × size of an event / block size
= 64 × 1 MB / 4 KB
= 64 ×256
= 16384

- A BU has a single FIFO called discardFIFO for FU discard messages, and one FIFO per FU called requestFIFO for FU request messages. Knowing that a BU can service a maximum of 64 FUs and assuming that a single FU will never have more than 1024 outstanding requests for events:

  rubuilder::bu::Application::discardFIFOCapacity
  = 1024 × maximum number of FUs
  = 1024 × 64
  = 65536

  rubuilder::bu::Application::requestFIFOCapacity
  = 1024

The default values of the RU exported parameters were calculated as follows:

- The blockFIFO of the RU is responsible for buffering incoming super-fragment data. Assuming a RUI only gives as many super-fragments to a RU as the TA gives triggers to the EVM, then in the worst case the blockFIFO must hold the blocks of as many super-fragments as there are event ids in the RU builder:

  rubuilder::ru::Application::blockFIFOCapacity
  = **nbEvtIdsInBuilder** × (size of a super-fragment / block size)
  = 4096 × (16K / 4KB)
  = 16384

The parameters of the form **_Packing** should not normally be modified as they have only been tested with the default value of 8. However the user may modify them if they are experiencing performance problems with the RU builder.

10RU builder application web-pages

The BU, EVM and RU applications each have default and debug web-pages. This chapter describes the parameters displayed on these pages. This chapter has been written to be used as a reference by DAQ operators.

This chapter is divided into three sections, one for each type of application. Each section contains a set of tables which describe the parameters displayed on the default and debug web-pages of the corresponding application type.

10.1 BU web-pages

Parameter	Description
throughput (bytes sec-1)	The event throughput in bytes per second.
average (bytes)	The average size of an event.
rate (events sec-1)	The number of events per second.
rms (bytes)	The RMS of the event size.

Table 3 BU – Default web-page "Data flow through node" parameters

Parameter	Description
fragmentSetsUrl	If useFragmentSet is set to true then this parameter must be set to the URL of the fragment sets file. The BU will use this file to determine which RUs are participating in event building.
useFragmentSet	Indicates whether the BU is using a fragments sets file or the value of ruInstances to determine which RUs are participating in event building. If set to true, the BU will use the fragments sets file specified by fragmentSetsUrl, else the BU will use the contents of ruInstances.
fragmentSetId	The BU can use a fragment sets file to determine which RUs are participating in event building. If useFragmentSet is set to true and fragmentSetsUrl specifies a valid location of a valid fragment sets file, then this parameter specifies which fragment set within the file is to be used.
Index	BU indices are internal to the RU builder application. They are used to facilitate efficient lookups in the EVM and RUs. A negative value is interpreted as meaning this parameter has not been set. If this parameter is not set when the BU is configured, then the BU will use its instance number.
evmInstance	The BU uses this parameter to determine the instance number of the EVM which manages it. If this parameter is not set, then the BU will use the first EVM it finds in the configuration given to its XDAQ executive. A negative value is interpreted as meaning this parameter has not been set.
nbEvtIdsInBuilder	The total number of event IDs in the RU builder.
maxEvtsUnderConstruction	The maximum number of events the BU can build concurrently.
blockFIFOCapacity	The capacity of the event data block FIFO.

discardFIFOCapacity	The capacity of the discard FIFO

Table 4 BU – Default web-page "Standard configuration" parameters

Parameter	Description
runNumber	The current run number
nbEventsInBU	The occupancy of the BU as the number of events currently in the BU. This number is calculated by subtracting the number of events sent to the FU or dropped from the total number of events built.
deltaT	The duration of the last delta t in seconds with respect to monitoring events that pass through the BU.
deltaN	The umber of events that passed through the BU in the last deltaT.
deltaSumOfSquares	The sum of the squares with respect to the sizes of the events (in bytes) that passed through the BU in the last deltaT.
deltaSumOfSizes	The sum of the event sizes (in bytes) that passed through the BU in the last deltaT.
stateName	The current state of the application
lastEventNumberFromEVM	The last event number received from the EVM
lastEventNumberFromRUs	The last event number received from the RUs
lastEventNumberToFUs	The last event number sent to the FUs.
nbEvtsBuilt	The number of events built since the BU was last enabled.
nbEventsDropped	The number of events dropped by the BU.
I2O_BU_ALLOCATE_Payload	The payload in bytes transferred by I2O_BU_ALLOCATE messages. See I2O_BU_ALLOCATE_LogicalCount.
I2O_BU_ALLOCATE_LogicalCount	The number of events requested by the FUs. FUs request events from the BUs using I2O_BU_ALLOCATE messages.
I2O_BU_ALLOCATE_I2oCount	The I2O message frame count for I2O_BU_ALLOCATE messages. . See I2O_BU_ALLOCATE_LogicalCount.
I2O_BU_DISCARD_Payload	The payload in bytes transferred by I2O_BU_DISCARD messages. . See I2O_BU_DISCARD_LogicalCount.
I2O_BU_DISCARD_LogicalCount	The number of events discarded by the FUs. FUs tells the BUs to using I2O_BU_DISCARD messages.
I2O_BU_DISCARD_I2oCount	The I2O message frame count for I2O_BU_DISCARD messages. See I2O_BU_DISCARD_LogicalCount.
I2O_FU_TAKE_Payload	The payload in bytes transferred by I2O_FU_TAKE messages. See I2O_FU_TAKE_LogicalCount.
I2O_FU_TAKE_LogicalCount	The number of events sent to the FUs. BUs send event data to the FUs using I2O_FU_TAKE messages.
I2O_FU_TAKE_I2oCount	The I2O message frame count for I2O_FU_TAKE messages. See I2O_FU_TAKE_LogicalCount.
I2O_EVM_ALLOCATE_CLEAR_Payload	The payload in bytes transferred by I2O_EVM_ALLOCATE_CLEAR messages. See I2O_EVM_ALLOCATE_CLEAR_LogicalCount.
I2O_EVM_ALLOCATE_CLEAR_LogicalCount	The number of event IDs that have been requested plus the number of event IDs that have been recycled/cleared. BUs request new event IDs from the EVM and recycle/clear old event IDs using I2O_EVM_ALLOCATE_CLEAR messages.

I2O_EVM_ALLOCATE_CLEAR_I2oCount	The I2O message frame count for I2O_EVM_ALLOCATE_CLEAR messages. See I2O_EVM_ALLOCATE_CLEAR_LogicalCount.
I2O_BU_CONFIRM_Payload	The payload in bytes transferred by I2O_BU_CONFIRM messages. See I2O_BU_CONFIRM_LogicalCount.
I2O_BU_CONFIRM_LogicalCount	The number of event IDs received from the EVM. The EVM sends event IDs to the BU using I2O_BU_CONFIRM_LogicalCount messages
I2O_BU_CONFIRM_I2oCount	The I2O message frame count for I2O_BU_CONFIRM messages. See I2O_BU_CONFIRM_LogicalCount.
I2O_RU_SEND_Payload	The payload in bytes transferred by I2O_RU_SEND messages. See I2O_RU_SEND_LogicalCount.
I2O_RU_SEND_LogicalCount	The number of super-fragments requested from the RUs. BUs request super-fragments from the BUs using I2O_RU_SEND messages.
I2O_RU_SEND_I2oCount	The I2O message frame count for I2O_RU_SEND messages. See I2O_RU_SEND_LogicalCount .
I2O_BU_CACHE_Payload	The payload in bytes transferred by I2O_BU_CACHE messages. See I2O_BU_CACHE_LogicalCount.
I2O_BU_CACHE_LogicalCount	The number of super-fragments received from the RUs. RUs send BUs super-fragment data using I2O_BU_CACHE messages.
I2O_BU_CACHE_I2oCount	The I2O message frame count for I2O_BU_CACHE messages. See I2O_BU_CACHE_LogicalCount.

Table 5 BU – Default web-page "Standard monitoring" parameters

Parameter	Description
rcmsStateListener	The class name and instance number of the RCMS state listener.
selfDriven	The threading-model used by the application. A value of true indicates self-driven, whereas a value of false indicates event-driven. See chapter 6 "RU builder threading-models" for more information.
workLoopName	The name of the worker thread when the application is using the self-driven threading-model. See chapter 6 "RU builder threading-models" for more information.
ageMessages	Specifies whether or not control messages should be aged, i.e. whether or not a control message should be sent if it is too old, as opposed to only when it is full.
msgAgeLimitDtMSec	How old a message can be in milliseconds before it will be sent if ageMessages is set to true.
monitoringSleepSec	Number of seconds the monitoring thread should sleep between monitoring information updates.
dropEventData	If the value is set to true then the BU will drop an event when it has finished building it.
I2O_RU_SEND_Packing	The packing factor of I2O_RU_SEND messages.
I2O_EVM_ALLOCATE_CLEAR_Packing	The packing factor of I2O_EVM_ALLOCATE_CLEAR messages.
monitoringFilename	The name of the file to which monitoring information should be written.
ruInstances	The instance numbers of the RUs that will participate in event building. If this parameter is not set, then it assumed that all RUs will participate in event building. This parameter is ignored if useFragment sets is set to true.
oldMessageSenderSleepUSec	The number of micro seconds between checks for old messages.
dumpTriggersToLogger	Specifies whether or not triggers should be dumped to the logger. This is a debugging aid. Dumping triggers to the logger can have a serious impact on the performance of the BU.

Table 6 BU – Debug web-page "Debug configuration" parameters

Parameter	Description
foundRcmsStateListener	Specifies whether or not the application has found the RCMS state listener.
eventIdFIFOElements	The number of elements in event ID FIFO.
blockFIFOElements	The number of elements in the block FIFO.
fullResourceFIFOElements	The number of elements in the "full resource" FIFO.
discardFIFOElements	The number of elements in the discard FIFO.
requestFIFOsElements	The number of elements in the request FIFO.

Table 7 BU – Debug web-page "Debug monitoring" parameters

10.2 EVM web-pages

Parameter	Description
throughput (bytes sec-1)	The trigger data throughput in bytes per second.
average (bytes)	The average size of a trigger's data.
rate (events sec-1)	The number of triggers per second.
rms (bytes)	The RMS of the trigger data size.

Table 8 EVM – Default web-page "Data flow through node" parameters

Parameter	Description
fragmentSetsUrl	If useFragmentSet is set to true then this parameter must be set to the URL of the fragment sets file. The EVM will use this file to determine which RUs are participating in event building.
useFragmentSet	Indicates whether the EVM is using a fragments sets file or the value of ruInstances to determine which RUs are participating in event building. If set to true, the EVM will use the fragments sets file specified by fragmentSetsUrl, else the EVM will use the contents of ruInstances.
fragmentSetId	The EVM can use a fragment sets file to determine which RUs are participating in event building. If useFragmentSet is set to true and fragmentSetsUrl specifies a valid location of a valid fragment sets file, then this parameter specifies which fragment set within the file is to be used.
nbEvtIdsInBuilder	The total number of event IDs in the RU builder.
triggerFIFOCapacity	The capacity of the trigger FIFO.
taClass	The class name of the TA.
taInstance	The instance number of the TA.

Table 9 EVM – Default web-page "Standard configuration" parameters

Parameter	Description
runNumber	the current run number.
nbTriggersInEVM	The occupancy of the EVM as the number of triggers currently in the EVM.
deltaT	The duration of the last delta t in seconds with respect to monitoring triggers that pass through the EVM.
deltaN	The number of triggers that passed through the EVM in the last deltaT.
deltaSumOfSquares	The sum of the squares with respect to the sizes of the triggers that passed through the EVM in the last deltaT.
deltaSumOfSizes	The sum of the trigger sizes that passed through the EVM in the last deltaT.
stateName	The current state of the application.
foundTA	Specifies whether or not the EVM has found the TA. The EVM will look for a TA when it is configured.
ruBuilderIsFlushed	Tri-state variable, specifying whether or not the RU builder is flushed: 0 : FALSE 1 : TRUE 2 : UNDEFINED, e.g. The EVM is in the Halted state. The RU builder is said to be flushed when it is empty and the data rate is 0. The sample period of the rate calculation is defined by the value of monitoringSleepSec.
nbTriggers	The number of triggers that have entered the EVM
nbEvtsBuilt	The number of events built by the RU builder.
lastEventNumberFromTrigger	The last event number that the EVM received from the trigger.
lastEventNumberToRUs	The last event number sent to the RUs.
lastEventNumberToBUs	The last event number sent to the BUs.
I2O_TA_CREDIT_Payload	The payload in bytes transferred by I2O_TA_CREDIT messages. See I2O_TA_CREDIT_LogicalCount.
I2O_TA_CREDIT_LogicalCount	The number of trigger credits sent to the TA. The EVM sends the TA trigger credits using I2O_TA_CREDIT messages.
I2O_TA_CREDIT_I2oCount	The I2O message frame count for I2O_TA_CREDIT messages. See I2O_TA_CREDIT_LogicalCount.
I2O_EVM_TRIGGER_Payload	The payload in bytes transferred by I2O_EVM_TRIGGER messages. See I2O_EVM_TRIGGER_LogicalCount.
I2O_EVM_TRIGGER_LogicalCount	The number of triggers received from the TA. The TA sends the EVM triggers using I2O_EVM_TRIGGER messages.
I2O_EVM_TRIGGER_I2oCount	The I2O message frame count for I2O_EVM_TRIGGER messages. See I2O_EVM_TRIGGER_LogicalCount.
I2O_EVMRU_DATA_READY_Payload	The payload in bytes transferred by I2O_EVMRU_DATA_READY messages. See I2O_EVMRU_DATA_READY_LogicalCount.
I2O_EVMRU_DATA_READY_LogicalCount	The number of triggers received from the RUI. The RUI sends the EVM trigger data using I2O_EVMRU_DATA_READY messages.

I2O_EVMRU_DATA_READY_I2oCount	The I2O message frame count for I2O_EVMRU_DATA_READY messages. See I2O_EVMRU_DATA_READY_LogicalCount.
I2O_RU_READOUT_Payload	The payload in bytes transferred by I2O_RU_READOUT messages. See I2O_RU_READOUT_LogicalCount.
I2O_RU_READOUT_LogicalCount	The number of "event number" / "event ID" pairs received from the EVM. The EVM sends the RUs "event number" / "Event ID" pairs using I2O_RU_READOUT messages.
I2O_RU_READOUT_I2oCount	The I2O message frame count for messages I2O_RU_READOUT. See I2O_RU_READOUT_LogicalCount.
I2O_EVM_ALLOCATE_CLEAR_Payload	The payload in bytes transferred by I2O_EVM_ALLOCATE_CLEAR messages. See I2O_EVM_ALLOCATE_CLEAR_LogicalCount.
I2O_EVM_ALLOCATE_CLEAR_LogicalCount	The number of requests for new event IDs plus the number of old event IDs to be recycled/cleared received from the BUs. The BUs request new event IDs from the EVM and ask the EVM to recycle/clear old ones using I2O_EVM_ALLOCATE messages.
I2O_EVM_ALLOCATE_CLEAR_I2oCount	The I2O message frame count for I2O_EVM_ALLOCATE_CLEAR messages. See I2O_EVM_ALLOCATE_CLEAR_LogicialCount.
I2O_BU_CONFIRM_Payload	The payload in bytes transferred by I2O_BU_CONFIRM messages. See I2O_BU_CONFIRM_LogicalCount.
I2O_BU_CONFIRM_LogicalCount	The number of event IDs sent to the BUs. The EVM sends event IDs to the BU using I2O_BU_CONFIRM messages.
I2O_BU_CONFIRM_I2oCount	The I2O message frame count for I2O_BU_CONFIRM messages. See I2O_BU_CONFIRM_LogicalCount.

Table 10 EVM – Default web-page "Standard monitoring" parameters

Parameter	Description
rcmsStateListener	The class name and instance number of the RCMS state listener.
selfDriven	The threading-model used by the application. A value of true indicates self-driven, whereas a value of false indicates event-driven. See chapter 6 "RU builder threading-models" for more information.
workLoopName	The name of the worker thread when the application is using the self-driven threading-model. See chapter 6 "RU builder threading-models" for more information.
ageMessages	Specifies whether or not control messages should be aged, i.e. whether or not a control message should be sent if it is too old, as opposed to only when it is full.
msgAgeLimitDtMSec	How old a message can be in milliseconds before it will be sent if ageMessages is set to true.
monitoringSleepSec	The number of seconds the monitoring thread should sleep between monitoring information updates.
generateDummyTriggers	Specifies whether or not the EVM should generate dummy triggers.
fedPayloadSize	When generateDummyTriggers is set to true, this parameter specifies the size of the FED payload of dummy triggers in bytes. The FED payload must be a multiple of 8 bytes.
dummyTriggerSourceId	When generateDummyTriggers is set to true, this parameter specifies the trigger source ID that is to be put into each dummy trigger.
I2O_TA_CREDIT_Packing	The packing factor of I2O_TA_CREDIT messages.
I2O_RU_READOUT_Packing	The packing factor of I2O_RU_READOUT messages.
ruInstances	The instance numbers of the RUs that will participate in event building. If this parameter is not set, then it assumed that all RUs will participate in event building. This parameter is ignored if useFragment sets is set to true.
oldMessageSenderSleepUSec	The number of micro seconds between checks for old messages.
dumpTriggersToLogger	Specifies whether or not triggers should be dumped to the logger. This is a debugging aid. Dumping triggers to the logger can have a serious impact on the performance of the EVM.

Table 11 EVM – Debug web-page "Debug configuration" parameters

Parameter	Description
foundRcmsStateListener	Specifies whether or not the application has found the RCMS state listener.
eventIdGaugeName	The name of the gauge responsible for measuring the number of event ids (resources) in use.
eventIdGaugeValue	The current value of the gauge responsible for measuring the number of event ids (resources) in use.
dummyEventNumber	When generateDummyTriggers is set to true, this parameter specifies the event number of the next dummy trigger.
nbCreditsToBeSent	The number of outstanding trigger credits to be sent to the TA.
clearedEventIdFIFOElements	The number of elements in the "cleared event-ID" FIFO.
freeEventIdFIFOElements	The number of elements in the "free event-ID" FIFO.
triggerFIFOElements	The number of elements in the trigger FIFO.
pairFIFOElements	The number of elements in the pair FIFO.
requestFIFOElements	The number of elements in the request FIFO.

1 **Table 12 EVM – Debug web-page "Debug monitoring" parameters**

2

10.3 RU web-pages

Parameter	Description
throughput (bytes sec-1)	The super-fragment throughput in bytes per second.
average (bytes)	The average size of a super-fragment.
rate (events sec-1)	The number of super-fragments per second.
rms (bytes)	The RMS of super-fragment size.

Table 13 RU – Default web-page "Data flow through node" parameters

Parameter	Description
nbEvtIdsInBuilder	The total number of event IDs in the RU builder.
blockFIFOCapacity	The capacity of the block FIFO.

Table 14 RU – Default web-page "Standard configuration" parameters

Parameter	Description
runNumber	The current run number.
deltaT	The duration of the last delta t in seconds with respect to monitoring super-fragments that pass through the RU.
deltaN	Number of super-fragments that passed through the RU in the last deltaT.
deltaSumOfSquares	The sum of the squares with respect to the sizes of the super-fragments (in bytes) that passed through the RU in the last deltaT.
deltaSumOfSizes	The sum of the super-fragment sizes (in bytes) that passed through the RU in the last deltaT.
stateName	The current state of the application.
nbSuperFragmentsInRU	The occupancy of the RU as the number of super-fragments currently in the RU.
lastEventNumberFromEVM	The last event number received from the EVM.
lastEventNumberFromRUI	The last event number received from the RUI.
lastEventNumberToBUs	The last event number sent to the BUs.
nbSuperFragments	The number of super-fragments that have passed through the RU.
nbEmptySuperFragments	The number of empty super-fragments generated by the RU and sent to the BU.
I2O_EVMRU_DATA_READY_Payload	The payload in bytes transferred by I2O_EVMRU_DATA_READY messages. See I2O_EVMRU_DATA_READY_LogicalCount.
I2O_EVMRU_DATA_READY_LogicalCount	The number of super-fragments received from the RUI. RUIs send RUs super-fragment data using I2O_EVMRU_DATA_READY messages.
I2O_EVMRU_DATA_READY_I2oCount	The I2O message frame count for I2O_EVMRU_DATA_READY messages. See I2O_EVMRU_DATA_READY_LogicalCount.
I2O_RU_READOUT_Payload	The payload in bytes transferred by I2O_RU_READOUT

	messages. See I2O_RU_READOUT_LogicalCount.
I2O_RU_READOUT_LogicalCount	The number of "event number" / "event ID" pairs the RU has received from the EVM. The EVM sends "event number" / "event ID" pairs to the RUs using I2O_RU_READOUT messages.
I2O_RU_READOUT_I2oCount	The I2O message frame count for I2O_RU_READOUT messages. See I2O_RU_READOUT_LogicalCount.
I2O_RU_SEND_Payload_BUn	The payload in bytes transferred by I2O_RU_SEND messages. See I2O_RU_SEND_LogicalCount_BUn.
I2O_RU_SEND_LogicalCount_BUn	The number of requests for super-fragments received from a specific BU. BUs request super-fragments from RUs using I2O_RU_SEND messages.
I2O_RU_SEND_I2oCount_BUn	The I2O message frame count for I2O_RU_SEND messages. See I2O_RU_SEND_LogicalCount_BUn.
I2O_BU_CACHE_Payload	The payload in bytes transferred by I2O_BU_CACHE messages. See I2O_BU_CACHE_LogicalCount.
I2O_BU_CACHE_LogicalCount	The number of super-fragments sent to the BUs. RUs send super-fragment data to the BUs using I2O_BU_CACHE messages.
I2O_BU_CACHE_I2oCount	The I2O message frame count for I2O_BU_CACHE messages. See I2O_BU_CACHE_LogicalCount.

Table 15 RU – Default web-page "Standard monitoring" parameters

Parameter	Description
rcmsStateListener	The class name and instance number of the RCMS state listener.
selfDriven	The threading-model used by the application. A value of true indicates self-driven, whereas a value of false indicates event-driven. See chapter 6 "RU builder threading-models" for more information.
workLoopName	The name of the worker thread when the application is using the self-driven threading-model. See chapter 6 "RU builder threading-models" for more information.
monitoringSleepSec	The number of seconds the monitoring thread should sleep between monitoring information updates.
tolerateCSCFaults	Specifies whether or not the RU should tolerate CSC local DAQ faults.
dummyBlockSize	When generateDummySuperFragments is set to true, this parameter specifies the size in bytes of a dummy super-fragment data block.
generateDummySuperFragments	Specifies whether or not the RU should generate dummy super-fragments.
dummyFedPayloadSize	When generateDummySuperFragments is set to true, this parameter specifies the payload size in bytes of a FED fragment within a dummy super-fragment.
fedSourceIds	When generateDummySuperFragments is set to true, this parameter specifies the source ID of each fragment within a dummy super-fragment.
maxPairAgeMSec	The maximum age of an "event number" / "event ID" pair in milliseconds. A negative value means forever.
expiredPairDetectionSleepUSec	The number of microseconds the thread checking for an expired pair sleeps between checks.
tolerateFaults	Specifies whether or not the RU should service BUs with empty super-

	fragments when in the TimedOut state. Only relevant if maxPairAgeMSec is not negative.

Table 16 RU- Debug web-page "Debug configuration" parameters

Parameter	Description
measuredTimeOutSec	The seconds component of the time period that passed and triggered the finite state machine of the application to move to the TimedOutTolerating or TimedOutBackPressuring state.
measuredTimeOutUSec	The microseconds component of the time period that passed and triggered the finite state machine of the application to move to the TimedOutTolerating or TimedOutBackPressuring state.
foundRcmsStateListener	Specifies whether or not the application has found the RCMS state listener.
blockFIFOElements	The number of elements in the block FIFO.
pairFIFOElements	The number of elements in the pair FIFO.
requestFIFOsElements	The number of elements in the request FIFO.
nbSuperFragmentsReady	The number of super-fragments that are ready to be consumed by the BUs.
lowestBuInstance	The lowest BU instance number found in the XDAQ configuration.
highestBuInstance	The highest BU instance number found in the XDAQ configuration.
buInstances	The instance numbers of all the BUs found in the configuration.

Table 17 RU – Debug web-page "Debug monitoring" parameters

11 How to install the RU builder

Install the XDAQ coretools, powerpack and regular worksuite packages using the instructions at the following URL:

http://xdaqwiki.cern.ch/index.php/Main_Page

12 Example configuration file

The following example configuration file shows how to configure a 1x1 RU builder to run on a single XDAQ executive and service HTTP requests on port ***HOST:PORT*** :

```
<xc:Partition xmlns:soapenc= »http://schemas.xmlsoap.org/soap/encoding/ »
xmlns:xc= »http://xdaq.web.cern.ch/xdaq/xsd/2004/XMLConfiguration-30 »
xmlns:xsi= »http://www.w3.org/2001/XMLSchema-instance »>

<i2o:protocol xmlns:i2o="http://xdaq.web.cern.ch/xdaq/xsd/2004/I2OConfiguration-30">
<i2o:target class="rubuilder::evm::Application" instance="0" tid="23"/>
<i2o:target class="rubuilder::ru::Application"  instance="0" tid="24"/>
<i2o:target class="rubuilder::bu::Application"  instance="0" tid="25"/>
</i2o:protocol>

<xc:Context url="http://HOST:PORT">
<xc:Module>${XDAQ_ROOT}/lib/librubuilderutils.so</xc:Module>
<xc:Module>${XDAQ_ROOT}/lib/libxdaq2rc.so</xc:Module>
<xc:Application class="rubuilder::tester::Application" id="12" instance="0" network="local"/>
<xc:Module>${XDAQ_ROOT}/lib/librubuildertester.so</xc:Module>
<xc:Application class="rubuilder::evm::Application" id="13" instance="0" network="local"/>
<xc:Module>${XDAQ_ROOT}/lib/librubuilderevm.so</xc:Module>
<xc:Application class="rubuilder::ru::Application" id="14" instance="0" network="local"/>
<xc:Module>${XDAQ_ROOT}/lib/librubuilderru.so</xc:Module>
<xc:Application class="rubuilder::bu::Application" id="15" instance="0" network="local"/>
<xc:Module>${XDAQ_ROOT}/lib/librubuilderbu.so</xc:Module>
</xc:Context>

</xc:Partition>
```

Two import rules for the module tag

1. The module tags for librubuilderutils.so and libxdaq2rc.so must appear once within each context tag where there is a module tag for librubuilderbu.so, librubuilderevm.so and/or librubuilderru.so.
2. The module tags for librubuilderutils.so and libxdaq2rc.so must appear within the context tag before any of the module tags librubuilderbu.so, librubuilderevm.so or librubuilderru.so

13 RU builder self test

This section explains how to perform the self test of the RU builder. This test helps determine whether or not the RU builder has been successfully installed.

The self test consists of the rubuilder::tester::Application plus one EVM, one RU and one BU all running on the same XDAQ executive. The EVM is told to generate dummy triggers, the RU is told to generate dummy super-fragments and the BU is told to drop the events it builds The step-by-step instructions to run the self test are:

Step 1
Install the RU builder as described in chapter 11.

Step 2
Modify the configuration file of chapter 12 so that ***HOST:PORT*** refers to the port on which the XDAQ executive shall service HTTP requests.

Step 3
Setup the following shell environment variables:

```
export XDAQ_ROOT=/opt/xdaq
export XDAQ_DOCUMENT_ROOT=${XDAQ_ROOT}/htdocs
export PATH=${PATH}:${XDAQ_ROOT}/bin
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${XDAQ_ROOT}/lib
```

Step 4
Run the XDAQ executive:

```
xdaq.exe -h HOST -p PORT -e $XDAQ_ROOT/etc/default.profile -c CONFIGURATION_FILE.XML
```

Where ***HOST*** and ***PORT*** are the same as those given in step 2 and ***CONFIGURATION_FILE.XML*** is the file modified from chapter 12.

Step 4

Open a web browser and enter the following URL:

http://HOST:PORT

Where ***HOST*** and ***PORT*** are the same as those of steps 2 and 3.

You should now see the HyperDAQ page of the XDAQ executive you started in step 3. It should look similar to the following screenshot.

Figure 15 HyperDAQ web page for self test

Step 5

Go to the default web page of rubuilder::tester::Application by clicking on the "rubuilder tester" link. Your web browser should now display something similar to:

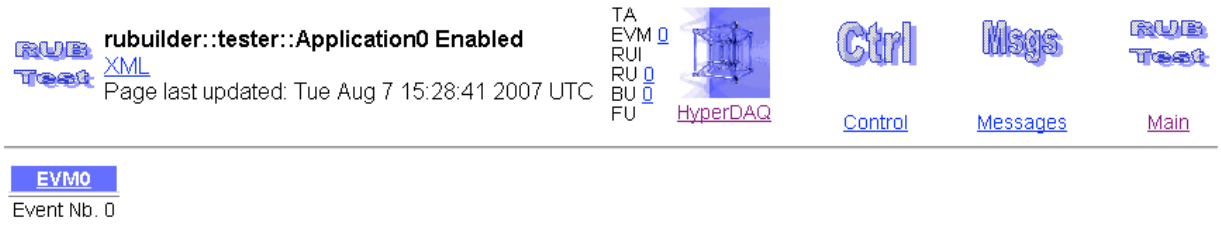

Figure 16 rubuilder::tester::Application web page

Step 6

Go to the control web page by clicking on the "Ctrl" icon. You should now see something like:

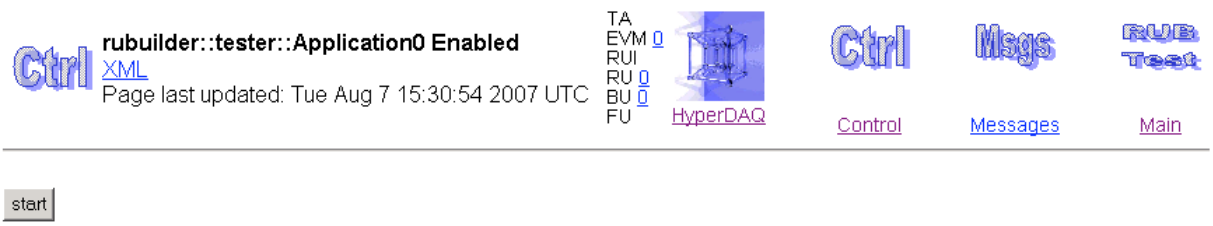

Figure 17 rubuilder::tester::Application control web page

Step 7

Start the self test by clicking the "start" button. The button should now be labeled "stop".

Step 8

Return to the main web page of the RU builder tester by clicking on the "RUB Test" icon. Clicking the refresh button of your browser should now show events being built. For example the "eventNb" variable of the EVM0 should increment.

Figure 18 Running RU builder

14 Configuration guidelines

This chapter summarizes and highlights the most important points with regards to configuring the RU builder.
The default values of the RU builder control parameters have been chosen for the majority of use-cases. The user should rarely need to diverge from these values.

The RU builder is dependent on the instance numbers of the BUs, EVM, FUs, RUs, and TA:

- RUs must be assigned instance numbers from 0 to the number of RUs – 1
- BUs must be assigned instance number from 0 to the number of BUs – 1
- The EVM must be assigned instance number 0
- The TA must be assigned instance number 0

The RU builder has the following configuration restrictions:

- A single BU can service a maximum of 64 FUs.
- The sum of the maximum number of event ids each BU can have at any moment in time:

  `rubuilder::bu::Application::maxEvtsUnderConstruction`

  must not exceed the total number of event ids in the RU builder:

  `rubuilder::bu::Application::nbEvtIdsInBuilder`
  `rubuilder::evm::Application::nbEvtIdsInBuilder`
  `rubuilder::ru::Application::nbEvtIdsInBuilder`

  If the total number of event ids is exceeded, then there is no guarantee that the EVM will be able to buffer BU requests for event ids, or that the RUs will be able to buffer BU requests for event data.
- The configuration parameters of a RU builder application must be set before it is configured.