



**Detailed Specification of the
FAIR Accelerator Control System Component
„FEC Software Framework“**

Document Name
F-DS-C-01e

Date yyyy-mm-dd
2012-08-16

Abstract

This document is the Detailed Specification of the accelerator control system component “FEC Software Framework”. This work package is part of the “Core Control System Software Packages” work package and covers the PSP code 2.14.10.1.2.



Table of Contents

List of Tables.....	2
1. Purpose and Classification of the Document	3
1.1. Responsibilities	3
1.2. Classifications of Requirements	3
2. Scope of the Technical System	4
2.1. System Overview	4
2.2. Limits of the System and Environment	4
2.2.1. Limits	4
2.2.2. Interfaces	4
2.2.3. Environment	5
2.3. Basis of Concept	5
2.3.1. Functional Requirements	5
2.3.2. Non-functional Requirements	7
2.3.3. General Constraints	7
2.3.4. Architectural Principles	7
3. Technical Specifications	8
3.1. Client Interface	8
3.1.1. Get data	8
3.1.2. Set data	8
3.1.3. Subscription	8
3.1.4. Transactional Settings	9
3.2. FEC Software Functionality	9
3.2.1. Data Multiplexing	9
3.2.2. Data Integrity	9
3.2.3. Data Persistency	9
3.2.4. Real-Time Capability	9
3.2.5. Multithreading	10
3.2.6. Flexible Client Software for Testing	10
3.3. Interfaces to Control System Components	10
3.3.1. FAIR Timing System	10
3.3.2. FAIR Logging System	10
3.3.3. FAIR Alarm System	10
3.3.4. FAIR Post Mortem System	11
3.4. Open Source	11
4. Quality Assurance, Tests and Acceptance	11
4.1. Development Methodology	11
4.2. Quality Assurance System of the Supplier	11
4.3. FAT	11
4.4. SAT	12
5. Documentation	12
6. Warranty	12
7. Scope of Delivery	12
I. Attached Documents	13
II. Related Documentation	13
III. Document Information	13
III.1. Document History	13

List of Tables

Table 1: Framework Requirements	5
Table 2: Front-end Software Requirements	7

1. Purpose and Classification of the Document

The purpose of this document is to specify the Accelerator Control System component "FEC Software Framework" for FAIR (PSP code 2.14.10.1.2).

This document is the most detailed type of document in the hierarchy of Control System specifications.

Whenever regulations and requirements are specified in the General Specifications, Technical Guidelines or Common Specifications of the Control System they are only referenced in this document. The related documents are listed in Appendix II.

No legal or contractual conditions are treated in this document. All related information is given in the General Specifications for FAIR II.

1.1. Responsibilities

The responsibilities with respect to changes and modifications of the present document are entirely in the hands of the Controls Department of the GSI Helmholtz Centre for Heavy Ion Research GmbH (GSI) Darmstadt.

For initial information please contact the administration of the Controls Department.

Further information on the organigram, names of responsible persons and task leaders, as well as the agreed document release and approval procedure is summarized in the organizational note 'Controls Project for FAIR'.

1.2. Classifications of Requirements

The following definitions of requirement classifications are being used throughout the document:

- **"Must"** or **"shall"** or **"is required to"** is used to indicate mandatory requirements, strictly to be followed in order to conform to the standard and from which no deviation is permitted.
- **"Must not"** or **"shall not"** mean that the definition is an absolute prohibition of the specification.
- **"Should"** or **"is recommended"** is used to indicate that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others or that a certain course of action is preferred but not required.
- **"Should not"** or **"is not recommended"** mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighted before implementing any behavior described with this label.
- **"May"**, which is equivalent to **"is permitted"**, is used to indicate a course of action permissible within the limits of the standard.

2. Scope of the Technical System

2.1. System Overview

The purpose of the FEC Software Framework is to provide a uniform, consistent, and easy way to develop, deploy, and instantiate all the front-end software which is used to control equipment of the FAIR accelerators.

The framework provides a graphical user interface that helps to interactively design the logical device representation, internal data, and real-time software actions for a device class [4]. It generates C++ code frames for real-time and server actions which must be completed by the developer and builds class libraries.

The framework helps in defining deployment units, in configuring the scheduling policy, and in building the binaries.

It also helps in adding device instances, binding logical events to real-time actions, and defining initial values and names for devices.

Software designed and generated with the FEC Software Framework usually runs on FAIR front-end controllers.

2.2. Limits of the System and Environment

2.2.1. Limits

The FEC Software Framework does not cover the communication between the FEC and the hardware device. This is done by the FEC device class developer by extending the framework generated code with equipment specific implementation.

It also does not cover the mechanisms to communicate via network.

2.2.2. Interfaces

Front-end software generated by the FEC Software Framework may have interfaces to the following FAIR software components:

- Timing System via the timing receiver
- Logging System
- Alarm System
- Post Mortem System
- Databases
- Name Service to resolve device names
- The middleware layer for network communication
- Hardware drivers for equipment access
- Message library

These interfaces are supported by the FEC Software Framework.

2.2.3. Environment

No special environment is required or has to be considered.

2.3. Basis of Concept

2.3.1. Functional Requirements

Framework Requirements

The framework as a tool must fulfill the following requirements:

Number	Description of Requirement
FSF_010	The framework must present a GUI to enable the user to design device class software, to generate and deploy binaries, to instantiate device objects, and to test device classes.
FSF_020	The framework must generate C++ code frames to be supplemented by the device class developer.
FSF_030	The framework must be able to generate binaries for X86 and X86_64 platforms under the Linux operating system.
FSF_040	The GUI must enable the user to configure the scheduling attributes of the class, and to build executable class-binaries.
FSF_050	The GUI must enable the user to add device instances, bind logical to physical timing events, define initial values for a device object, and to define device names.
FSF_060	The GUI must enable the user to start a binary and to launch the navigator tool for test purposes.
FSF_070	The framework must support unit tests.
FSF_080	The framework must be able to simulate timing events of the FAIR timing system.
FSF_090	The FEC Software Framework must be based on a commonly used integrated development environment like Eclipse.
FSF_100	The FEC Software Framework must provide a tool to upgrade existing device class software to the latest framework version.

Table 1: Framework Requirements

Generated Software Requirements

Software generated by the framework (and supplemented by the developer of a device class) must fulfill the following requirements:

Number	Description of Requirement
FSF_200	Each device must be represented as an instance in a distributed

Document Title: Detailed Specification FEC Software Framework

	system.
FSF_210	The device must be accessible using the device name.
FSF_220	Properties must be described in a way to allow generic access for applications. Property descriptions should be stored in a database.
FSF_230	Property data must be accessible via get and/or set methods.
FSF_240	Property get and set methods must offer blocking access, non-blocking access with no response or with callback, and subscription with callback.
FSF_250	The property set method must offer transactional settings to allow transactions over many properties/devices.
FSF_260	Properties must offer partial get and set methods to transfer only a subset of data instead of the full property data.
FSF_270	Applications must be able to subscribe to all relevant properties. The framework must provide the possibility to subscribe to value changes. This service must be configurable by the application.
FSF_280	The generated software must supervise subscriptions to assure that they still exist.
FSF_290	Beam-dependent set and actual values must be multiplexed.
FSF_300	Set values must be kept in synchronized double buffers to assure data integrity.
FSF_310	It must be possible to persist set values so that they are present even after device restart.
FSF_320	Actual values must be kept in synchronized double buffers or in a ring buffer. The designer must be able to choose one alternative.
FSF_330	Actual values must be stored with (multiplexing) context (time stamp, timing event stamp, cycle ID, ...)
FSF_340	Therapy-related operation, where each cycle has different settings, must be possible.
FSF_350	Real-time actions must be triggered by logical events.
FSF_355	It must be possible to assign names to logical events and to reference them within the device class implementation.
FSF_360	Logical events must be assigned to: <ul style="list-style-type: none"> • timing events • timing events plus delay • external hardware events (HW trigger input, interlock, equipment interrupts, etc.) • software events • periodical timer events
FSF_370	The event (trigger) context (e.g. timestamp or timing event) must be readable.
FSF_380	Software must have multithreading capability.

Table 2: Front-end Software Requirements

2.3.2. Non-functional Requirements

The framework and software generated by the framework must fulfill the following non-functional requirements:

Number	Description of Requirement
FSF_600	All parts of the FEC Software Framework source code must be available under a free software license.
FSF_620	Software, especially framework generated code, must be easily extendable to adapt to future requirements.

Table 3: Non-functional Requirements

2.3.3. General Constraints

Decision was taken in 2008 that main parts of the front-end software framework will be realized using a software product from CERN called FESA (**F**ront-end **S**oftware **A**rchitecture). Since FESA is a modular framework that is easily adaptable, it will be the core component for the front-end software framework. For detailed information on the FESA system see [8].

For covering the needs of a full front-end software framework, additional functionality may be needed, which is not or not yet covered by the FESA framework as it is now. This has to be taken into account for the design and implementation phases.

For front-end software the FESA Software Design Guideline [5] applies.

Any usage of means for localization of accelerator components must comply with the accelerator's control system localization guidelines.

Any user or rights management must be designed and developed in accordance with the central rights management within the control system.

All GUIs must comply with the GUI Guidelines [6].

FEC Software must use consistent (error) messages provided by a central message library.

2.3.4. Architectural Principles

The Software Architecture Guideline for the Control System [7] fully applies.

3. Technical Specifications

The FEC Software Framework has to provide common services which can be utilized by the software developers.

3.1. Client Interface

Applications need access to the data stored in the front-end. The common interface which provides this access is the middleware.

Necessary access methods are get, set and subscribe as well as transactional settings. Accordingly, the FEC Software Framework must provide an implementation to support these services in the front-end software.

3.1.1. Get data

The FEC Software Framework provides a method to transport data of a property from the device class software at the front-end via the middleware to the requesting application.

Data may be requested synchronously, meaning that the calling method at the application side will block until an answer arrives.

Data may be requested asynchronously, meaning that the calling method at the application side will *not* block but return immediately. The application must provide a callback method with the get method that receives the data.

For multi-data properties the application may request subsets of the data in a granular way.

3.1.2. Set data

The FEC Software Framework provides a method to transport data of a property from the application via the common middleware to the device class software at the front-end.

Data may be set synchronously, meaning that the calling method at the application side will block until an acknowledge arrives.

Data may be set asynchronously, meaning that the calling method at the application side will *not* block but return immediately. The application may request no acknowledge at all. If an acknowledge is requested, the application must provide a callback method with the set method that receives the acknowledge.

For multi-data properties the application may set subsets of the data in a granular way.

3.1.3. Subscription

The FEC Software Framework provides a method to subscribe to property data, meaning that data is sent from the device class software at the front-end to the subscribed applications each time an event occurs.

Applications may subscribe to and unsubscribe from property data.

Document Title: Detailed Specification FEC Software Framework

Subscriptions must specify a callback method that receives the published data.

Different options must be offered to clients to further specify its subscription (device name, property name, context information, filter options, notify only when data has changed, etc.)

For multi-data properties the application may request subsets of the property data in a granular way.

All connections between clients and the FEC software must have the ability to indicate, whenever a connection got lost, and to automatically re-connect as soon as the connection got re-established.

3.1.4. Transactional Settings

The FEC Software Framework must provide a way to commit new set values on many front-ends simultaneously. The source of the commit event is configurable.

3.2. FEC Software Functionality

3.2.1. Data Multiplexing

Multiplexing means that data of a single property is stored multiply in the front-end for use with different beams. The FEC Software Framework must provide means for the device class software developer, to flag if a data value is multiplexed or not, and to manage multiplexed data in a proper way.

Therapy operation based on the so-called multi-parameter beam mechanism requires to store up to a few hundred different data sets (set or actual values) of a property for a single beam for one irradiation session. The FEC Software Framework must provide means to manage multi-parameter beams properly.

3.2.2. Data Integrity

The FEC Software Framework provides a way to get or set consistent data at any time. Even if a real-time thread writes data to the memory and interrupts data readout of a lower priority thread, the received data of the low priority thread must not be corrupted in any way.

This data integrity must be achieved by the use of a synchronized double buffering mechanism.

The FEC Software Framework synchronizes double buffered data in a way that no priority inversion can occur between the different threads.

3.2.3. Data Persistency

The FEC Software Framework provides a service, which allows front-end software to persist the last set values, and directly set the according data entries accordingly after a restart. This service must be easily configurable per data entry.

3.2.4. Real-Time Capability

Real-time capability means that a system can handle a specific task within a deterministic time interval. Systems based on real-time patched Linux can only provide soft real-time capabilities. There is a statistical probability on not finishing

the task in time which is not considered as an error. This probability must be very small.

A longtime test on a FEC (SCU, real-time patched Linux, proper hardware configuration) with heavy load results in an average latency of 50 microseconds and a maximum latency of 278 microseconds.

The FEC Software Framework allows the software developer to run tasks of his code in a soft real-time thread, and to manage its priority in a proper way.

3.2.5. Multithreading

The FEC Software Framework supports the possibility to run several threads in parallel. A software developer must be able to define in which of these threads actions are executed. It must be possible to manage all threads and the corresponding actions in a transparent and easy way.

3.2.6. Flexible Client Software for Testing

All front-end software produced by usage of the FEC Software Framework must be testable in a convenient way. A client which generically can perform get and set commands as well as subscriptions must be provided within the FEC Software Framework.

3.3. Interfaces to Control System Components

3.3.1. FAIR Timing System

The FEC Software Framework must provide an easy way to make use of the FAIR Timing System [9], which will be available at most of the front-end controllers. The developer must have the possibility to directly configure which timing event triggers which thread and which user code is executed by this thread.

For testing purposes, the developer must have the possibility to simulate the Timing System by using a software implementation which must be provided by the FEC Software Framework.

3.3.2. FAIR Logging System

The developer has the possibility to easily make use of the FAIR logging system [10] in order to log any kind of equipment messages (debug, info, error, etc.).

Error messages which are generated by the FEC Software Framework itself should be logged by this system as well.

3.3.3. FAIR Alarm System

The FEC Software Framework must provide the possibility to configure a field to be monitored and specify conditions and limits under which alarms shall be generated. The corresponding configuration is part of the device property model and must be changeable at runtime. Alarms are managed by the FAIR alarm system [2].

3.3.4. FAIR Post Mortem System

For some front-ends it will be necessary to always keep the last data samples by using a variable ring buffer. This front-end data can be collected by a central post mortem server which is part of the FAIR post mortem system [3].

The FEC Software Framework has to provide an easy way to use a ring buffer system, and to transfer the collected data to the central post mortem server.

3.4. Open Source

The FEC Framework is based on the existing CERN's Front End Software Architecture Framework (FESA) as described in 0.

Being part of the GSI-CERN collaboration the development and distribution must be provided under a free software license. A free software license is a software license which grants recipients extensive rights to modify and redistribute, which would otherwise be prohibited by copyright law.

4. Quality Assurance, Tests and Acceptance

The system to be built must adhere to the guidelines and recommendations for software developments in the FAIR accelerator control system context, as referenced in the FAIR Common Specification F-CS-C-01e (Common Specification Accelerator Control System). The supplier of the work package must identify the relevant standards and recommendations before start of the development. Details must be fixed as part of the technical design concept in the initialization phase.

4.1. Development Methodology

The FEC Software Framework shall be developed in an iterative and incremental methodology.

Each iteration cycle must result in a running system which can be evaluated and tested at FAIR site. The first iteration, which has to be available as early as possible, must concentrate on the most critical functionality. In successive iterations, the system is enhanced by adding features until the desired total functionality is reached.

In the initialization phase, the technical design concept and the plan for the iterations must be developed, and must be approved by the FAIR contracting body. At end of each iteration cycle the achieved status of the system will be evaluated and the iteration plan will be adjusted. Each iteration cycle must be approved by the FAIR contracting body before it can be started.

4.2. Quality Assurance System of the Supplier

Generally, the software specific measures of Quality Assurance described in Common Specification "Accelerator Control System" [1] fully applies.

4.3. FAT

The Common Specification "Accelerator Control System" [1] fully applies.

4.4. SAT

The Common Specification "Accelerator Control System" [1] fully applies.

5. Documentation

The Common Specification "Accelerator Control System" [1] fully applies.

6. Warranty

The conditions and warranty period specified in the Contract applies.

7. Scope of Delivery

The following components must be delivered with the FEC Software Framework:

- Source Code
- Build Environment
- Installation
- Documentation
- Test Environment
- User Training
- User Support

I. Attached Documents

List of abbreviations for controls (Abbreviations_Controls.pdf).

II. Related Documentation

- [1] F-CS-C-01e, FAIR Common Specification "Accelerator Control System"
- [2] F-DS-C-09e, FAIR Detailed Specification "Alarm System"
- [3] F-DS-C-13e, FAIR Detailed Specification "Post Mortem System"
- [4] F-DS-C-15e, FAIR Detailed Specification "FEC Device Classes"
- [5] F-DG-C-01e, "FESA Software Design Guideline"
- [6] F-DG-C-02e, "GUI Guideline"
- [7] F-DG-C-03e, "Software Architecture Guideline"
- [8] A. Guerrero et al, "CERN Front-End Software Architecture for Accelerator Controls", ICALEPCS'03, Gyeongju, Korea, WE612
- [9] F-DS-C-05e, FAIR Detailed Specification "General machine timing system"
- [10] F-DS-C-10e, FAIR Detailed Specification "Diagnostic logging system"

III. Document Information

III.1. Document History

Version	Date	Description	Author	Review / Approval
0.1	20. Jan. 2012	Draft version	A. Schwinn, L. Hechler	
0.2	31. Jan. 2012	2 nd draft version	L. Hechler, A. Schwinn	
0.5	28. Feb. 2012	After CCT review	L. Hechler, A. Schwinn	
0.6	01. Mar. 2012	Restructured	A. Schwinn, L. Hechler	
1.0	06. Mar. 2012	Final version	A. Schwinn, L. Hechler	CCT
3.0	16. Aug. 2012	Incorporated FAIR review comments	CCT	