|  |  |  |
| --- | --- | --- |
| **FRIB_rgb** | **Facility for Rare Isotope Beams** U.S. Department of Energy Office of Science | Michigan State University East Lansing, MI 48824-1321 • Ph: (517) 355-9672 • Fax: (517) 353-5967 www.frib.msu.edu | MSU-Wordmark-PMS-567.eps |

CCDB - API Manual

Version 0.9

|  |  |  |  |
| --- | --- | --- | --- |
|  | Name | Signature | Date |
| Prepared By | Vasu Vuppala |  |  |
| Reviewed By |  |  |  |
| Approved By |  |  |  |

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Description |
| 1.0 | 6-Sep-2016 | Vasu V | Initial |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1 Introduction 5](#_Toc343496118)

[1.1 Purpose 5](#_Toc343496119)

[1.2 Scope 5](#_Toc343496120)

[1.3 Definitions, Acronyms, and Abbreviations 5](#_Toc343496121)

[1.4 References 6](#_Toc343496122)

[1.5 Overview 6](#_Toc343496123)

[2 Installation 6](#_Toc343496124)

[2.1 Service URL 6](#_Toc343496125)

[2.2 Entity, Attribute, Qualifier, and Result 7](#_Toc343496126)

[3 REST Interface 7](#_Toc343496127)

[3.1 URL Construction 7](#_Toc343496128)

[3.2 Response 8](#_Toc343496129)

[3.3 Example 8](#_Toc343496130)

[4 Java Interface 9](#_Toc343496131)

[4.1 Build and Execute 9](#_Toc343496132)

[4.2 Example 9](#_Toc343496133)

[4.3 Accessing Data from MATLAB 10](#_Toc343496134)

[5 Python Interface 11](#_Toc343496135)

[5.1 Example 11](#_Toc343496136)

[6 Exception Handling 12](#_Toc343496137)

[7 Support 12](#_Toc343496138)

[Appendix A – Entities, Qualifiers, Results 13](#_Toc343496139)

[Appendix B – Entity Classes 14](#_Toc343496140)

List of Figures and Tables

[Figure 1 REST Examples 8](#_Toc343496141)

[Figure 2 Results in XML 8](#_Toc343496142)

[Figure 3 Results in JSON 8](#_Toc343496143)

[Figure 4 Java Example 10](#_Toc343496144)

[Figure 5 Matlab Example 11](#_Toc343496145)

[Figure 6 Python Example 12](#_Toc343496146)

[Table 1 Definition, Acronyms, and Abbreviations 5](#_Toc343496147)

# Introduction

*Open EPICS Configuration Module* (or simply *Configuration Module*) is software for managing the configuration of an accelerator facility. The configuration of an accelerator facility includes information about its components, their properties, relationships among the components, their layout, measurements, etc.

*Configuration Module* consists of the following:

* A database to store the accelerator facility’s configuration data
* A Graphical User Interface (GUI) to the data
* Services to access the data.
  + RESTful Service
  + EPICS V4 service (under development).
* Module API:
  + REST Interface
  + Java Interface
  + Python Interface

## Purpose

The objective of this document is to describe the methods of programmatically accessing the configuration data through *Configuration Module’s* API.

## Scope

This document provides an overview of the API.

## Definitions, Acronyms, and Abbreviations

Table Definition, Acronyms, and Abbreviations

|  |  |
| --- | --- |
| Item | Description |
| EPICS | Experimental Physics and Industrial Control System |
| ERD | Entity-Relationship Diagram |
| FRIB | Facility for Rare Isotope Beam |
| IOC | Input/Output Controller |
| IRMIS | Integrated Relational Model of Installed Systems |
| PV | Process Variable. Identifier for a data item in EPICS. |
| RDBMS | Relational Database Management System |
| REST | Representational State Transfer |
| UML | Unified Modeling Language |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
|  |  |
|  |  |

## References

1. Jersey: An open source JAX-RS implementation, <http://jersey.java.net/>
2. Open EPICS Web Site, <http://openepics.sourceforge.net>
3. Configuration Module: Installation Manual, Open EPICS Web Site
4. Configuration Module: Schema Design Manual, Open EPICS Web Site

## Overview

The next section describes the prerequisites for using the API. The subsequent sections describe the REST, Java, and Python interfaces.

# Installation

To be able to use the Configuration Module’s API, you must first install the Configuration Module database and services. This is described in the Configuration Module Installation Manual. If you intend to use the Java or Python interfaces, you must download the corresponding libraries. The documentation and the libraries are available on the Configuration Module website [2]. The website also has sample programs and the entire source code for the module.

## Service URL

All the interfaces (REST, Java, Python etc) require URL to the REST service. This base URL is dependent on the installation [], and is determined during installation. If you had installed the Configuration Module yourself, you would know the URL. Otherwise, you will need to check with the installation team about it. To form the URL, you need to know the following:

* If http or https protocol is being used
* The name (or IP address) of the application or web server
* The port number

Section 3.1 describes the format of the URL in detail. Some examples of the URL are: <https://service.frib.msu.edu/conf/rs/v0>, <http://qa01.nscl.msu.edu:8080/conf/rs/v0>

## Entity, Attribute, Qualifier, and Result

All the interfaces (REST, Java, Python etc), are based off the concept of *entity*. An entity[[1]](#footnote-1) is a thing that exists within an accelerator facility and can be identified. *Component*, *signal*, and *component type* are some examples of entity. Entities have *attributes* that describe it. For example, *name*, *serial-number*, *type* are attributes of the *component* entity. A *qualifier* is a *key,value* pair that specifies a set of entity instances. For example, *‘type=QH’* is a *qualifier,* and when applied to *component* specifies all quadrupoles[[2]](#footnote-2). The entity names and associated qualifiers are listed in *Appendix A – Entities, Qualifiers, Results*.

Retrieval operation on entities returns a list of entity instances whose structures are listed in *Appendix B – Entity Classes*.

# REST Interface

## URL Construction

The base URL for GET is as follows:

protocol://serverName[:Port]/serviceName/***rs***/version/resource?parameters

* Protocol: http or https (preferred)
* ServerName: Name of server hosting *Configuration Module*. Example: services.frib.msu.edu
* Port: Port number on the server. It is optional.
* ServiceName: Name of the service. For *Configuration Module,* it is ‘conf’. Example: https:// services.frib.msu.edu/conf
* **rs**: The literal ‘rs’ which stands for ‘REST Service’.
* Version: Version number of the API. Example: v0, v1
* Resource: Name of the entity. Examples: component, comptype, property
* Parameters: Qualifiers to the resource. Examples: type=qh, id=AC2CLEN

The URL for other operations (POST, PUT, and DELETE) is similar and will be covered in the subsequent versions of this document because these operations are not supported in the current version of *Configuration Module*.

## Response

Both XML and JSON formats are supported by REST API. The tag names in XML response and the key names in JSON response match the class structures in *Appendix B – Entity Classes*

## Example

Figure 1 shows examples of REST GET requests using *curl*. The first command requests for all dipoles (type=dv). The XML response for the request is shown in Figure 2. The third *curl* command requests for properties of the component named *LS1\_CA01:GV\_D1124*. The response in JSON is listed in Figure 3.



Figure REST Examples



Figure Results in XML



Figure Results in JSON

# Java Interface

The Java API is a very thin wrapper around the REST API. It is initiated by instantiating an object of the class org.openepics.conf.api.PAF (PAF stands for Particle Accelerator Facility). PAF must be instantiated with the base URL to the REST service. For each entity, there is a unique method in PAF, and it follows a naming convention. The method is named by capitalizing the first letter of the entity name and prefixing it with ‘get’. For example, the method for ‘component’ will be ‘*getComponent’*, for ‘property’ it will be ‘*getProperty’* and so on. The qualifiers are passed onto the Java API methods as a map (javax.ws.rs.core.MultiValuedMap<String,String>). Results from the method have structures as described in *Appendix B – Entity Classes.* So *getProperty* method returns a list of *UAProperty* objects (List<UAProperty>).

The following code snippet illustrates the API usage. It retrieves all the quadrupoles[[3]](#footnote-3) (device type: QH) from the database. First, a PAF object is instantiated with the base URL to the REST service. The resource qualifiers are passed in a map (*params*). The results will be a list of components (UAComponent). The qualifier named “*type*” and its value are added to *params.* The components are retrieved by invoking the *getComponent* method.



## Build and Execute

To use the Java API, you must download Configuration Module’s API jar file (named as conf-japi-*version*.jar), and include it in your CLASSPATH. The Java API depends on Jersey [1] client API which can be downloaded from *jersey.java.net*. The method of building and executing the program will depend on your development environment. Samples of compile and run scripts are included in the conf-samples directory of the source code.

## Example

Figure 4 shows a sample Java program that utilizes Configuration Module’s Java API. The imports are hidden for brevity. It reads a component type specification from the user, retrieves all the components of that type, and prints their details.



Figure Java Example

## Accessing Data from MATLAB

Due to MATLAB’s strong integration with Java, accessing *Configuration Module API* from MATLAB is very similar to the way it is done in Java. Figure 5 shows a MATLAB script that retrieves and displays the properties of a component.



Figure Matlab Example

# Python Interface

The Python interface is implemented as a native Python module (it is not based on the Java interface). However, the interface has been kept similar to the Java interface. It is also a very thin layer over the REST interface. Access to REST interface is initiated by instantiating PAF class which requires the base REST service URL. The PAF class is in *confpy* module in org.openepics.conf.pyapi package. PAF’s methods are named after the entities. The first letter of the entity name is capitalized and prefixed with ‘get’ to form the corresponding method name. For example, the resource ‘signal’ is accessed through the *getSignal* method. Qualifiers are specified using Python dictionary. Each method takes a dictionary as an argument. Each method returns a list of objects that correspond to the entity (this association is described in *Appendix B – Entity Classes*). For example, getComptype returns a list of UAComponentType objects.

## Example

Figure 6 illustrates a sample Python script to display information about components of a given type.



Figure Python Example

# Exception Handling

Exceptions from the interfaces are not standardized and still being worked upon. In the current version:

* The REST interface does not propagates exceptions to HTTP errors. It responds with an error message.
* The Java and Python APIs do not propagate errors from service.
* Misspelt qualifier (parameter) names are silently ignored. For example, if a parameter named “xxx” used to qualify the component resource, it is ignored.
* The values of qualifiers are not validated.

# Support

To report errors, corrections, and suggestions please visit the Configuration Module web site [2].

# Appendix A – Entities, Qualifiers, Results

The following table lists the entities and their atribute.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity** | **Qualifier** | **Value** | **Description** | **Returns** |
| component | containedin | Name of a component | Retrieves all components that are contained in the given component | List of UAComponent |
| name | Name of a component | Retrieves components with names that match the given name |
| type | Component type Id | Retrieves all components of the given type |
| comptype | Id | Component type id | Retrieves component type with the given id | List of UAComponentType |
| property | cname | Name of a component | Retrieves all properties of the given component | List of UAProperty |
| Id | Property id | Retrieves property with the given id |
| signal | cname | Name of a component | Retrieves signals of the given component | List of UASignal |
| Id | Signal id | Retrieves signal with the given id |

# Appendix B – Entity Classes

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Attribute** | **Type** | **Description** |
| UAComponent |  |  | Represents a component |
|  | comment | String | Comment about the component |
| componentTyepId | String | Component type |
|  |  |  |
| description | String | Component description |
| isAbstract | Boolean | True – it is not a physical component i.e. there is no corresponding real-world physical object. False – it is a physical component. |
| name | string | Name of the component |
| properties | List<UAProperty> | Properties of the component |
| signalList | List<UASignal> | Signals of the component |
|  |  |  |  |
| UAComponentType |  |  | Represents a component type |
|  | componentTypeId | String | Unique identifier |
| description | String | Description |
|  |  |  |  |
| UAProperty |  |  | Represents a component property |
|  | propertyId | String | Unique identifer |
| description | String | Description |
| units | String | Units |
| dataType | String | Type of data (for property value). For a list of data types see […] |
| value | String | Value of the property |
|  |  |  |  |
| UASignal |  |  | Represents a control signal |
|  | alias | String | Alternative name for the signal |
| type | String | Type of signal. See […] for all the types |
| signalName | String | Name of the signal |
| description | Sting | Description |
| componentName | String | Name of the component to which the signal belongs |
| serverName | String | Name of the IOC (component) that serves the signal |

1. These entities are from a user’s perspective; they are higher-level structures and may not correspond to the entities or tables in the database schema. [↑](#footnote-ref-1)
2. ‘QH’ is used here only for illustrative purposes. It is dependent on the facility’s naming convention. Configuration Module is not tied to any naming convention; it works with any qualifier value that is present in its database. [↑](#footnote-ref-2)
3. Again, component types and their meanings are based on the local lab’s naming convention. We are using QH for quadrupoles only for illustrative purposes. [↑](#footnote-ref-3)