OpenIEC61850 User Guide

Fraunhofer Institute for Solar Energy Systems ISE

openmuc.org

Table of Contents

1. Intro

OpenIEC61850 is a library implementing the IEC 61850 standard based on the MMS mapping for client and server communication. It is licensed under the Apache 2.0 license. OpenIEC61850 includes a console client and server as well as a gui client.

1.1. Distribution

After extracting the distribution tar file the OpenIEC61850 library can be found in the build/libs-all folder. For license information check the *license* directory in the distribution.

1.1.1. Dependencies

Besides the OpenIEC61850 library the folder *build/libs-all/* contains the following external libraries:

- **jasn1** - a library from the jASN1 project doing BER encoding/decoding, Copyright 2011-17 Fraunhofer ISE, Author: Stefan Feuerhahn, License: LGPLv2.1 or later, http://www.openmuc.org
- **slf4j-api** - a popular logging API. It is only needed if openiec61850 is used to implement a server. The client part does not log anything. License: MIT, http://www.slf4j.org
- **logback-core/logback-classic** - an actual logger implementation of the slf4-api. It is used by the console server application to output log information. It can be replaced by a logger of your choice that supports the slf4j API. Like slf4j it is only needed for server implementations. License: EPLv1.0 and LGPLv2.1, http://logback.qos.ch
- **jcalendar** - a calendar library needed by the client GUI. You don’t need this dependency if you don’t use the client gui. © 1999-2011 Kai Toedter, License: LGPLv3, http://toedter.com/jcalendar/

1.2. Console & GUI Applications

You can execute the console client and server through the scripts found in the folder *run-scripts*. Executing the scripts without any parameters will print help information to the screen. Note that under Unix/Linux you need root privilages if you want the server to listen on any port lower than 1000.

Instead of running the applications from the terminal you can create Eclipse project files as explained in our FAQs and run them from within Eclipse.

1.3. OSI Stack

The OpenIEC61850 library includes an OSI stack implementation as it is needed by the IEC 61850 MMS mapping. The API of the OSI stack and the OSI transport layers are made public so that they can be used by other projects.

- **josistack** - implements the Application Control Service Element (ACSE) protocol as defined by ISO 8650 or ITU X.217/X.227, the lower ISO Presentation Layer as defined by ISO 8823/ITU X226, and the ISO Session Layer as defined by 8327/ITU X.225.

- **jositransport** - implements RFC 1006 and the OSI Transport Service Layer.

2. Using OpenIEC61850

The easiest way to learn how OpenIEC61850 works is by running and analyzing the console client and server applications. You might want to look at the source code of the console applications to get an understanding of how they work. They can be used as a basis for you to code your individual client or server applications.

An IEC 61850 device that is to be controlled or monitored is called an IEC 61850 server. An IEC 61850 server normally listens on port 102 for incoming connection requests by IEC 61850 clients.

2.1. Client

If you want to connect to an IEC 61850 server, you should first create an instance of ClientSap (SAP = Service Access Point) and configure it to your needs. Then you build up the association to the server using the associate() method.

2.2. Server

First get a List of ServerSaps using the method ServerSap.getSapsFromSclFile(). This method reads in the SAP from the given ICD file. Take the ServerSap you want to run and configure it to your needs (e.g. set the port to listen on). The ServerSap includes the complete device model defined in the ICD file. Retrieve a copy of it using the method getModelCopy(). Tell the ServerSap to start to listen on the configured port using startListening(). This is a non-blocking function.

2.3. Data Model

An IEC 61850 server contains a treelike data model that contains at its leafs the data (integers, boolean, strings etc) that can be accessed by clients. Clients can also retrieve the whole data model from the server.

The upper most model node ist called "server". In OpenIEC61850 it is an object of type ServerModel. The server node contains 1..n logical devices (LD). A logical device may contain 1..n logical nodes (LN). A logical node may contain 1..n data objects. In OpenIEC61850 the logical nodes do not contain complete data objects but instead contain so called functionally constraint data objects (FCDO). An FCDO is a data object that is split up by functional constraint. An FCDO can contain a combination of other FCDOs, arrays, constructed data attributes and/or basic data attributes.

All nodes of the server model in OpenIEC61850 are of one of the following seven types:

- ServerModel
- LogicalDevice
- LogicalNode
- FcDataObject

- Array
- ConstructedDataAttribute
- BasicDataAttribute

They all implement the ModelNode interface. The nodes FcDataObject, Array, ConstructedDataAttribute and BasicDataAttribute also implement the interface called FcModelNode because they are considered functionally constraint data in the standard. Many of the services of IEC 61850 can only be applied to functionally constraint data (e.g. GetDataValues and SetDataValues).

When programming a client you get a copy of the server model either through ClientAssociation.retrieveModel() or ClientAssociation.getModelFromSclFile(). When programming a server you get a copy of the server model through the ServerSap.getModelCopy() function.

You can then navigate through the model using several functions:

- ServerModel.findModelNode(ObjectReference objectReference, Fc fc) will search for the subnode with the given reference and functional constraint.
- ModelNode.getChild(String name, Fc fc) will return the child node with the given name and functional constraint.
- ModelNode.getBasicDataAttributes() will return a list of all leaf nodes (basic data attributes) of the model node.

3. Modifying and Compiling OpenIEC61850

We use the Gradle build automation tool. The distribution contains a fully functional gradle build file (*build.gradle*). Thus if you changed code and want to rebuild a library you can do it easily with Gradle. Also if you want to import our software into Eclipse you can easily create Eclipse project files using Gradle. Just follow the instructions on our FAQ site.

4. IEC 61850 Concepts

4.1. Data Sets

Data sets (DS) form a group/set of data. Data sets can be used to read or write several data objects/data attributes at once using a single request/response message exchange. The service used for this are GetDataSetValues and SetDataSetValues. Besides the data set services the reporting, logging, GOOSE and sampled value services also use the data set concept.

The data attributes or data objects that are part of a data set are called the members of a data set. Only functionally constraint data may be a member of a data set.

The IEC 61850 standard defines five ASCI services related to data sets:

- **GetDataSetValues** to get the value of all members of the data set.

- **SetDataSetValues** to set the value of all members of the data set.
- **CreateDataSet** to create a new data set dynamically.
- **DeleteDataSet** to delete a data set that was created.
- **GetDataSetDirectory** to get the list of all existing data sets in the server

Two types of data sets exist:

- **Persistent data sets** - they have a reference of the format LDName/LNName.DataSetName . They are visible to all clients. Persistent data sets can be preconfigured in the SCL file. In this case they cannot be deleted. Persistent data sets can also be dynamicall created by clients. In this case they may be deleted again later. Dynamically created data sets will be automatically deleted once the server stops.
- **Non-persistent data sets** - they have a reference of the format @datasetname . They are only visible to the client that created it through the CreateDataSet service. These data sets only exist as long as the association is open.

4.2. Reporting

Reporting allows a server to send data based on events and without explict request by the client. What data is sent and the events that cause reports are configured through so called report control blocks (RCB).

The standard distinguishes between two types of reporting: buffered reporting and unbuffered reporting. With buffered reporting reports are buffered by the server in case a connection to the client is interrupted. This way reports can be sent after the client has connected again. Buffered reporting is configured through buffered report control blocks (BRCB). Unbuffered reporting is configured through unbuffered report control blocks (URCB).

RCBs are located within logical nodes of the server’s data model (i.e. the server’s ICD/SCL file). They are fixed and cannot be deleted or added at runtime. An RCB is blocked once a client has registered to receive reports (by enabling it). That means a single RCB cannot send reports to several clients in parallel.

RCBs are always associated with a specific data set that must be located in the same logical node as the RCB. A subset of the data set’s members will be reported if any of the configured events occures. The associated data set of the RCB can be dynamically changed by a client at run-time.

RCBs are located in logical nodes in the server’s model tree. Therefor the reference of an RCB is of the form LogicalDeviceName/LogicalNodeName.RCBName . Even though RCBs can only reference data sets located in the same logical node as the RCB itself, they can still monitor data from other logical nodes because the data set’s members can be located in other logical nodes.

The following events can cause a server to send a report:

1. The client issues a general interrogation.
2. A data attribute that is part of the associated data set changes or is updated.
3. Periodic sending of reports has been enabled by setting the integrity period of an RCB.

Whether one of these events really causes a report to be sent is configured through the RCB.

A report control block contains the following fields:

- Configuration fields. They can only be written if the RCB has not been enabled nor reserved by another client:
  - **RptID** - The report ID identifies the RCB. If not set, it equals the RCB reference. The RptID will be sent with every report so that the client can identify the RCB responsible for the report.
  - **DatSet** - The reference of the data set associated with this RCB. It must be located in the same logical node as the RCB. Members of this data set will be reported whenever the configured events occure. The data set reference as it is set in an RCB must contain a dollar sign instead of a dot to separate the logical node from the data set name, e.g.: *LDevice1/LNode$DataSetName*
  - **OptFlds** - A bitstring where each bit indicates whether an optional field is included in the reports caused by this RCB. The following optional fields exist: sequence-number, report- timestamp, reason-for-inclusion, data-set-name, data-reference, buffer-overflow, entryID, segmentation, and conf-revision. For URCBs the values of buffer-overflow and entryID are ignored.
  - **BufTm** - In case of an event that causes a report the server will wait for *BufTm* ms for other events. All data that is to be reported because of events in this time span is sent in a single report.
  - **TrgOps** - Specifies which events will trigger reports. Possible events are:
    - data change (dchg)
    - quality change (qchg)
    - data update (dupd)
    - integrity - if enabled the server will send periodic integrity reports to the client. Integrity reports will contain the current values of all members of the referenced data set.
    - general interrogation (GI).
  - **IntgPd** - The integrity period specifies an interval in ms for the periodic sending of intergrity reports.
  - **PurgeBuf** (only part of BRCBs not URCBs) -
  - **EntryID** (only part of BRCBs not URCBs) -
- Information fields: can only be read:
  - **SqNum** - the current sequence number of the RCB. After sending a report the RCB will increment the sequence number by one.
  - **ConfRev** - The configuration revision is a counter representing the number of times the referenced data set was changed.
  - **Owner** - Shall be equal to the IP of the client that reserved the RCB. Shall be NULL (i.e. the empty string) if the RCB is not reserved.
  - **TimeOfEntry** (only part of BRCBs not URCBs)
- Functional fields: The following fields of the RCB can be read or written to execute functions:

- **Resv** (only part of URCBs not BRCBs) - Before doing anything with an RCB a client should reserve it by setting this field to true. If the setting was successful the RCB will be reserved exclusively for this client. Enabling an RCB that has not been reserved implicitly reserves it.
- **ResvTms** (only part of BRCBs not URCBs) - This attribute of BRCBs is optional. If not present the control block is reserved by simply enabling it. A ResvTms value of -1 indicates that the control block was reserved by configuration for a certain set of clients. A value of 0 indicates that the BRCB is not reserved. A client can reserve the control block by writing a value largen than 0. The value represents the number of seconds that the reservation shall be maintained after the association was closed or interrupted.
- **RptEna** - By setting this variable to true reporting will be enabled. Note that changing/writing and configuration fields of the RCB will fail as long as reporting is enabled.
- **GI** - General Interrogation. Setting this parameter to true will initiate a report to be sent to the client. This report will contain all the data of the associated data set. The GI parameter will be reset to false automatically by the server once the report has been sent.

A report always contains the following information fields:

- **RptID** - The report ID identifies the RCB that has caused the generation of the report. It equals the RptID field of the RCB.
- **OptFlds** - It is equal to the OptFlds field of the corresponding RCB.
- **SqNum** (optional, included if OptFlds.SequenceNumber is true) - The sequence number of the report. It is equal to the SqNum field of the corresponding RCB by the time it was sent.
- **TimeOfEntry** (optional, included if OptFlds.report-timestamp is true) - specifies the time when the Entry ID was created.
- **DatSet** (optional, included if OptFlds.dataset-name is true) - The reference of the data set whose data is sent in this report.
- **BufOvfl** (optional, only included if it is a buffered report and OptFlds.buffer-overflow is true) - Indicates the server lost report entries due to a buffer overflow. Only applies to buffered reporting.
- **EntryID** (optional, only included if it is a buffered report and OptFlds.entryID is true) - Entry identification ToDo
- **ConfRev** (optional, only included if OptFlds.conf-rev is true) - It is equal to the ConfRev field of the corresponding RCB.
- **SubSqNum** (optional, included if OptFlds.segmentation is true) - In case that a long report does not fit into one message, a single report shall be divided into subreports. Each subreport shall have the same sequence number and a unique SubSqNum.
- **MoreSegmentsFollow** (optional, included if OptFlds.segmentation is true) - indicates that more subreports with the same SqNum follow.
- **Inclusion Bitstring** - A bit string whose length equals the number of data members of the referenced data set. Each bit indicates whether the corresponding data member is included in the report or not.

- **Data-references** (optional, included if OptFlds.data-reference is true) - The references of the reported data set members.
- **Values** - The values of the reported data set members.
- **Reason Codes** (optional, included if OptFlds.reason-for-inclusion is true) - The reason codes indicate for each reported value, the reason why it was reported. Possible reasons are data change, quality change, data update, integrity, general interrogation, and application trigger. Several reasons can be true for a single reported value.

A client that wants to receive reports from a certain RCB would usually first attempt to reserve the RCB. If successful he would then configure the RCB as he wishes. Finally he would enable reporting.

5. Terminology

- **BRCB** - Buffered Report Control Block
- **DS** - Data Set
- **LD** - Logical Device
- **LN** - Logical Node
- **SAP** - Service Access Point
- **URCB** - Unbuffered Report Control Block

6. Authors

Developers:

- Stefan Feuerhahn

Former developers:

- Claus Brunzema
- Bertram Lückehe
- Chau Do
- Tobias Weidelt
- Michael Zillgith