**OpenIEC61850 User Guide**

* unnumbered toc  
  {:toc}

**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.

**Distribution**

After extracting the distribution tar file, the OpenIEC61850 library can be found in the folder *build/libs-all*.

**Dependencies**

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

* *jasn1* - the ASN.1 BER encoding/decoding library by beanit, license: Apache 2.0, [https://www.beanit.com](https://www.beanit.com/)
* *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](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](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/>

**Console & GUI Applications**

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

**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.

**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.

**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.

**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.

**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 is 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.

**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.