VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Minh Nguyen

# IEC 61850 Intelligent Electronic Device Configuration Tool

Technology and Communication

2020

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

## ABSTRACT

| | |
|---|---|
| Author | Minh Nguyen |
| Title | IEC 61850 Intelligent Electronic Device Configuration Tool |
| Year | 2020 |
| Language | English |
| Pages | 40 |
| Name of Supervisor | Jukka Matila |

The aim of this work was to create a Configured Intelligent Electronic Device Description (CID) editor supporting both IEC 61850 edition 1 and edition 2, based on a working version of an edition 1 CID editor. The main technologies used to deliver the work were C++ programming language and Qt framework. This work was done for Arcteq Ltd.

IEC 61850 is an internationally used standard in substation automation. It specifies a representation format, System Configuration description Language (SCL), for the configuration of Intelligent Electronic Devices (IEDs). A CID file is a type of SCL file, containing necessary information for an IED to configure itself.

The final implementation completely fulfilled the requirements for an edition 2 compliant IED configuration tool. Nonetheless, there were additional features that can be implemented to facilitate the use of the tool.

Keywords IEC 61850, IED configuration tool, substation automation

# CONTENTS

## LIST OF FIGURES AND TABLES

## TERMS AND DEFINITIONS

**Device**

Mechanism or piece of equipment designed to serve a purpose or perform a function, for example, breaker, relay, or substation computer

**IDE**

Integrated Development Environment

**Intelligent Electronic Device**
**IED**

Any device incorporating one or more processors with the capability of receiving or sending data/control from or to an external source (for example, electronic multifunction meters, digital relays, controllers)

**IEC**

International Electrotechnical Commission

**Relay**

Digital protective relay - a computer-based system with software-based protection algorithms for the detection of electrical faults

**Substation**

Electrical substations - the interface between the distribution grid and transmission systems

**STL**

Standard library (C++)

# 1 INTRODUCTION

The work in this thesis was done for Arcteq Ltd. The company designs, manufactures, sells, and supports protection relays and arc flash protection systems for electrical utility, power generation, industrial, offshore, marine, institutional, and commercial users /1/. Many of Arcteq's products support a wide range of communication protocols including, but not limited to, Modbus, IEC 60870-5-103/101/104, IEC 61850 and DNP3.

Arcteq's protection relays were compliant with IEC 61850 edition 1 and the company has already got an IED configuration tool for edition 1. Since edition 2 of the standard was evaluated to be mature and can be deployed, an IED configuration tool capable of handling both edition 1 and edition 2 was needed. The objective of this work is to fulfil that need.

# 2 IEC 61850 STANDARD

## 2.1 Overview

IEC 61850 is an important international standard for substation automation that has a very significant impact on how electric power systems are designed and built. IEC 61850 is a part of the IEC's Technical Committee 57 architecture for electric power systems /2/. Being the first standardized communication protocol in substation automation systems, one of the its main objectives is supporting interoperations among substation automation devices of different vendors. The standard is based on Ethernet communication, ensuring high communication bandwidth and speed while reliability is ensured by utilizing redundancy network protocols such as Parallel Redundancy Protocol (PRP).

The standard defines a unified information model with a naming hierarchy and specific data structures to be used in any compliant device in order to reduce the cost in engineering time and to improve maintenance tasks. Vendors are requested to identify the same concepts with the same names, and they must use a common format to build their information. The hierarchical information model contains the following levels: IED, logical device, logical node, data object, data attribute, basic type. The relationships of the levels are described in Figure 1 /3/.

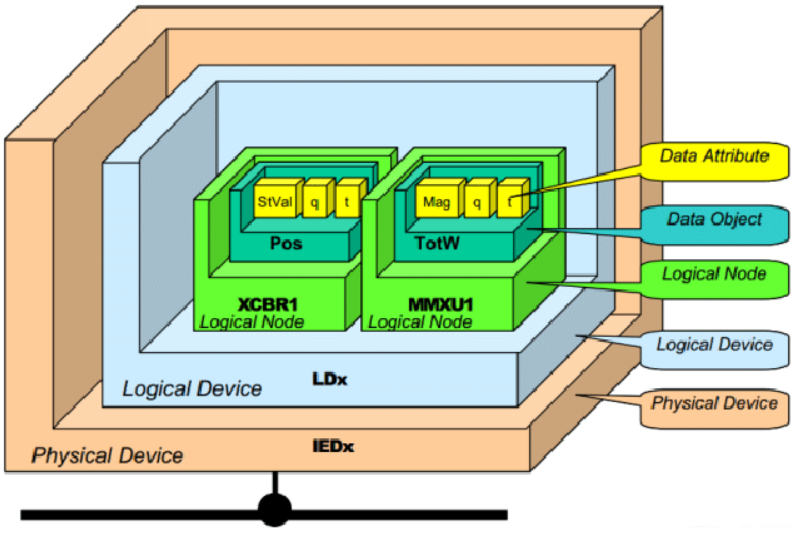


**Figure 1.** Data modelling in IEC 61850

The logical devices are logical containers used to organize the information of an IED, splitting it into different categories. The logical nodes are the functions or the components that automate the system. There are functions related to control, protection, measurement, supervision and more. They are identified by four letters; the first one indicates their category. The table of all categories are defined in IEC 61850-7-4. Each logical node includes a set of mandatory and optional data objects to fulfil their tasks. The data objects can represent status information, measurements, set points, controllable points, or descriptive information. Each logical node includes a table in the standard with its related data indicating their name and type. The vendors must select the ones that they can provide and publish in their IEC 61850 model. /4/

## 2.2 Generic Object-Oriented Substation Events Messaging

A part of IEC 61850's services is Generic Object-Oriented Substation Events (GOOSE) messaging. It is a mechanism for fast transmission of substation events, such as commands and alarms. A single GOOSE message sent by an IED can be received and used by several receivers for fast messaging /5/. The service works on publisher-subscriber mechanism and has a higher priority than normal messages. GOOSE messages are retransmitted at increasing intervals, thus making sure the messages are received properly.

## 2.3 System Configuration Description Language

System Configuration description Language (SCL) is a representation format specified by IEC 61850 to exchange configuration data between different devices. There are six types of SCL files for different purposes of SCL data exchange, four of which are defined in edition 1. The SCL files types are:

- IED Capacity Description (ICD).
- System Specification Description (SSD).
- Substation Configuration Description (SCD).
- Configured IED Description (CID).
- Instantiated IED Description (IID), which is defined in edition 2.

- System Exchange Description (SED), which is defined in edition 2.

The usage of each file types is shown in Figure 2.

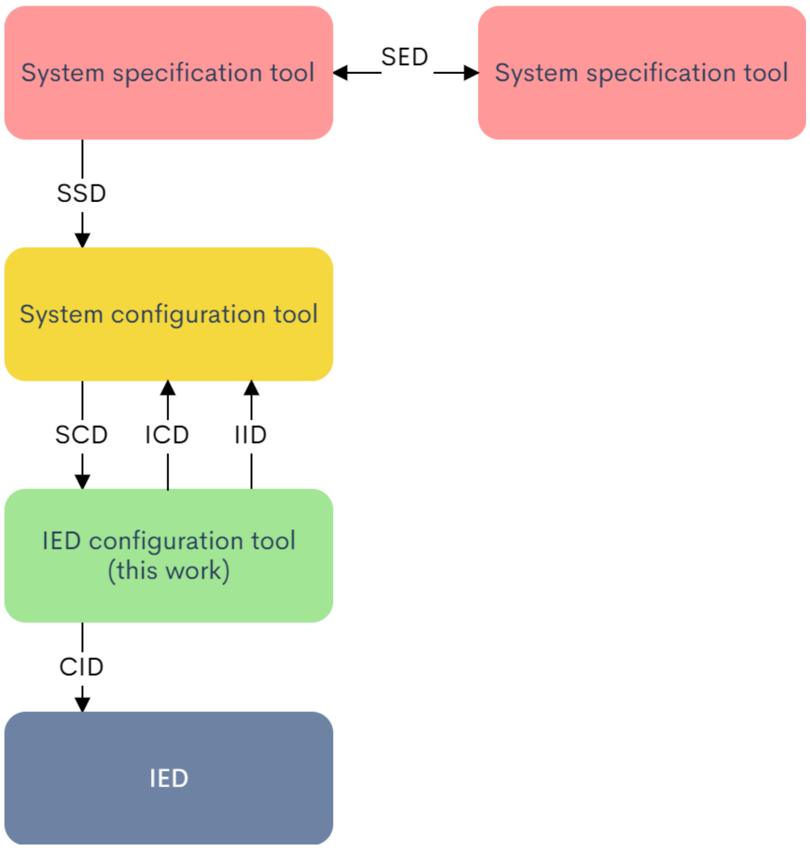

System specification tool
SED
System specification tool
SSD
System configuration tool
SCD
ICD
IID
IED configuration tool
(this work)
CID
IED

**Figure 2.** SCL file types' usages

The configuration process is done through multiple steps. First, ICD files shall be provided by the substation's vendor. It is a template containing the description of an IED type capabilities and not containing the communication configuration. After that, using the ICD files, an SCD file is created using the system configuration tool. The SCD file, containing all substations' description and communication configuration, can then be import to the IED configuration tool to extract the information specific to one substation and store those in a CID file. Finally, the CID file may be sent from the IED configuration tool to an IED to configure it.

Instead of creating an SCD file, it is also possible to use an ICD file to produce a CID file, which is what Arcteq's IED configuration tool has been doing in practice.

The IID file becomes handy when there is a change to be made in one configured substation in the SCD file. While it can be done using the ICD file, the user will have to re-configure all the communication section. An IID file, on the other hand, is a CID file with needed modifications, which can then be imported to the system configuration tool to modify the SCD file.

The SSD and SED files are not relevant to an IED configuration tool. An SSD file describes the single line diagram of the substation and the required logical nodes. An SED file describes the interfaces of one project to be used by the other project, and at reimport the additionally engineered interface connections between the projects. /7/

# 3 CONFIGURED IED DESCRIPTION FILE

## 3.1 Extensible Markup Language (XML)

A CID file is a type of SCL file, whose format is based on XML version 1. Therefore, it is necessary to understand an XML file structure.

An XML file consists of nodes. For the purpose of this document, three types of nodes are of concern: element nodes, attribute nodes, and text nodes. An element node may have attributes (attribute nodes) and a context (text node). For example, the following line can be an XML node.

```
<Duck canFly=”true”>Quack</Duck>
```

It is an element node named “Duck” with an attribute named “canFly” having value “true”. Its context is a text node with value “Quack”. An XML node may also have a parent node (a node that contains this node) and child nodes (nodes that this node contains). The nodes that share one same parent are called siblings. One special case is the top-level element node called root node. It must have no parent and no sibling.

An XML document with correct syntax is called well-formed. A validated XML document is both well-formed and valid. An XML document can be validated using a Document Type Definition (DTD) or an XML schema, both defining the structure and legal elements and attributes of an XML document /6/. Since the IEC 61850 standard uses XML schemas, an example and explanation of an XML schema is described below.

```
<xs:element name=”colour” type=”xs:string”/>
<xs:element name=”material” type=”xs:string”/>
<xs:attribute name=”theme” type=”xs:string” use=”required”/>
<xs:element name=”chair”>
     <xs:complexType>
          <xs:sequence>
               <xs:element ref=”colour”/>
               <xs:element ref=”material”/>
          <xs:sequence>
     </xs:complexType>
     <xs:attribute ref=”theme” use=”required”/>
</xs:element>
```

The schema above is interpreted line-by-line as followed:

- Define element “colour” of type string
- Define element “material” of type string
- Define attribute “theme” of type string
- Defining element “chair”
- Element “chair” contains a complex type
- The complex type is a sequence of two elements “colour” and “material” as described above
- Element “chair” contains attribute “theme” as described above

Based on the structure defined by the schema, the following XML content is well-formed and valid:

```
<chair theme=”Royal”>
     <colour>Yellow</colour>
     <material>Gold</material>
</chair>
```

## 3.2 CID File Structure and Functionalities

A CID file shall contain a root node named “SCL”, which may consist of the following child nodes: header, substation, communication, IED and data type templates. The descriptions of the elements are listed in Table 1.

**Table 1.** SCL child nodes description

| **Element** | **Description** |
|---|---|
| Header | Contains the identification of a CID file and its version |
| Substation | Describes a substation’s functional structure related to the configured IED |
| Communication | Describes which access points of an IED are connected to a subnetwork |
| IED | Describes an IED’s configuration |
| Data type templates | Contains templates of the data of elements |

Functions in a substation equipment are represented by logical node elements in the IED node. A logical node may have the attributes stated in Table 2.

**Table 2.** Attributes of a logical node

| **Attribute** | **Description** |
|---|---|
| prefix | The logical node’s prefix |
| lnClass | The logical node class defined by the standard |
| lnType | The logical node’s type |
| inst | The instance number of this node |
| desc | Description text |

The combination of prefix, lnClass, and inst attributes must be unique for every logical node. There shall be a LNodeType element in the data type templates that defines the content of each logical node. The logical node type template is built from data object elements, each of which has a DOType element defining its content. A data object type template is built from data attribute elements, which can either have a basic type, be an enumeration, or a structure of a DAType. The DAType is built from BDA elements, defining the structure elements, which again can be BDA elements or have a base type such as a DA /7/. An example of the structure is shown in Figure **3**.

```
<?xml version='1.0' encoding='UTF-8'?>
<SCL xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns="http://www.iec.ch/61850/2003/SCL">
  <Header id="" version="3" />
  <Communication>…</Communication>
  <IED name="AQx2xx"
       desc="Generic IEC61850 IED"
       configVersion="1.0"
       manufacturer="Arcteq Relays Ltd"
       type="IEC61850 IED">
    <Services>…</Services>
    <AccessPoint name="AP1">
      <Server timeout="30">
        <Authentication/>
        <LDevice desc="" inst="Relay">
          <LN lnClass="LNCL" inst="" lnType="LNCL_1">
            <DataSet name="DataSetURCB1">…</DataSet>
            <ReportControl rptID="URCB101"
                           datSet="DataSetURCB1"
                           name="URCB101"
                           confRev="1"
                           bufTime="1000"
                           intgPd="0">…</ReportControl>
            <DataSet name="DataSetGOOSE1">…</DataSet>
            <GSEControl datSet="1"
                        name="GSE_CB1"
                        confRev="1"
                        appID="0"
                        type="GOOSE"/>
            <DOI name="Loc">
              <DAI name="stVal" valKind="Set">…</DAI>
              <DAI name="q">…</DAI>
              <DAI name="t">…</DAI>
            </DOI>
          </LN0>
        </LDevice>
      </Server>
    </AccessPoint>
  </IED>
  <DataTypeTemplates>
    <LNodeType lnClass="LNCL" id="LNCL_1">
      <DO name="Loc" type="SPS_0"/>
    </LNodeType>
    <DOType desc="Single point status" id="SPS_0" cdc="SPS">
      <DA name="stVal" fc="ST" bType="BOOLEAN" dchg="true"/>
      <DA name="q" fc="ST" qchg="true" bType="Quality"/>
      <DA name="t" fc="ST" bType="Timestamp"/>
    </DOType>
  </DataTypeTemplates>
</SCL>
```

**Figure 3.** Example CID file

Some of the nodes in Figure 3 are collapsed to shorten the content. The logical node and the logical node template are connected by the same values in the logical node's lnType attribute and the template's id attribute. Each instantiated data object (DOI) is based on the data object in the template. The similar logic is used for data objects and instantiated data attributes (DAIs).

Apart from DOIs, any logical node may also contain dataset nodes and report control nodes. Dataset nodes are made up of one or more functionally constraint data attributes (FCDA). They are used to access many attributes with only one request. An FCDA may have the attributes stated in Table 3.

**Table 3.** Attributes of a functionally constraint data attribute

| **Attribute** | **Description** |
|---|---|
| ldInst | Name of the logical device |
| prefix | Prefix of the logical node |
| lnClass | Class of the logical node |
| lnInst | Instance number of the logical node |
| doName | Name of the data object |
| daName | Name of the data attribute |
| fc | Functional constraint |

Report control nodes are used to configured reporting of events from the server without the client having to explicitly request for it. A report control node is always associated with a dataset node in the same logical node.

There is always a special logical node in the IED node, called logical node zero (LN0). It is used for functions related to the logical device. It may contain GSE control nodes to store the configuration of GOOSE messaging.

# 4 REQUIREMENT SPECIFICATION

This chapter states the changes required from the edition 1 IED configuration tool. From this point forward, the edition 1 IED configuration tool provided by Arcteq will be regarded as the old tool, as opposed to the new tool – the result of this work.

## 4.1 Add Description to Dataset Editor

In the old tool, the interface for the users to edit the dataset shows only the prefixes and types of the logical nodes, making it difficult for the users to use. The new tool shall also show the description of the logical node.

## 4.2 Add and Remove Logical Nodes Manually

In the old tool, logical nodes are added and removed automatically. It is done based on whether the logical node has any data that is in a dataset. The new tool shall support adding and removing logical nodes manually.

## 4.3 Change Prefix Attributes of Logical Nodes

The users shall be able to change the value of the prefix attribute of a logical node. The prefix values of the logical nodes shall be unique. The prefix value “GS” is reserved. The users shall not be allowed to change the prefix from and to that value. The old tool had no support for this feature.

## 4.4 Edit GOOSE Subscription Information

In the old tool, the users can import GOOSE dataset information, but are not able to edit the information. The new tool shall support both importing and editing of the information.

## 4.5 Export Logical Nodes Information

The users shall be able to generate documentation of the logical nodes and their data. The old tool had no support for this feature. The documentation should be a tab delimited text file. Each line of the file corresponds to a data object of a logical node. Each line shall contain (in the listing order):

- The logical node’s prefix, description, class, and instance number
- The data object’s name
- The functional constraint of the data attributes
- Data attributes, in the following format: “x$y$z.t”, where
    - “x” is the logical node’s prefix, class, and instance number
    - “y” is the functional constraint of the data attributes
    - “z” is the data object’s name
    - “t” is the data attribute’s name

## 4.6 Extract Data from ICD Files

In the old tool, the logical nodes and FCDAs that a CID file can have are stored in an XML file and a text file. Those data are now covered by ICD files. The new tool shall be able to extract the same information from the provided ICD files.

## 4.7 Set Maximum Client for Report Control Nodes

In edition 1, each report control node serves one client only. When the maximum number of clients is set in the old tool, that number of report control nodes are written to the CID file. In edition 2, the report control node has a child node with a “max” attribute to store the maximum number of clients. Therefore, only one report control node is added. The child node is described below.

```
<ReportControl …>
    …
    <RptEnabled max="1"/>
</ReportControl>
```

The new tool shall detect the current edition and generate the content accordingly.

## 4.8 Store GOOSE Subscription Information

In edition 1, when subscribing to a GOOSE dataset, the subscription information is not stored in the CID file but is handled elsewhere. In edition 2, the information shall be stored in the CID file. The new tool shall detect the current edition and act accordingly.

The information is stored as followed. When a user imports GOOSE information by passing in the publisher's CID file, logical nodes shall be added in the subscriber's CID file, having the following structure:

```
<LN …>
    <DOI desc="Single point status" name="Ind1">
        <DAI name="stVal">
            <Private type="SystemCorp_Generic">
                <SystemCorp_Generic:GenericPrivateObject …/>
            </Private>
        </DAI>
    </DOI>
</LN>
```

The attributes of the logical node are described in Table 4.

**Table 4.** Attributes of a GOOSE subscription logical node

| Attribute | Value |
|---|---|
| prefix | "GS" |
| lnClass | "GGIO" |
| lnType | "GGIO_0" |
| inst | Range from 1 to 64, no duplicate |
| desc | "Generic process I/O" |

The attributes of the general private object node are described in Table 5.

**Table 5.** Attributes of a GOOSE subscription private node

| Attribute | Description |
|---|---|
| xmlns | Defines the namespace for the prefix. The value is always “http://www.systemcorp.com.au/61850/SCL/Generic” |
| Field1 | A unique application identification stored in the Communication element of the publisher’s CID file. |
| Field2 | The configuration revision stored in the GSE control element |
| Field3 | The index of the data in the dataset, starting from 0 |
| Field4 | “1” if the next subscribed data is the quality indicator of this data, “0” otherwise. |
| Field5 | Indicates the data type. “0” for Boolean values, “1” for integers, “2” for unsigned integers, “3” for floating point values. |

One logical node shall be added for each data imported. However, if the next subscribed data is the quality indicator of the previous one, only one logical node is added. In that case, the data object node will have another data attribute:

```
<DAI name="q">
    <Private type="SystemCorp_Generic">
        <SystemCorp_Generic:GenericPrivateObject …/>
    </Private>
</DAI>
```

The general private object node of this data attribute shares the same attributes described in Table 5.

Additionally, an external reference node is to be added for each imported data. Those nodes shall be children of the input node in the logical node zero like described below:

```
<LN0 …>
    <Inputs>
        <ExtRef …/>
        …
    </Inputs>
    …
</LN0>
```

The attributes of the external reference nodes are described in Table 6.

**Table 6.** Attributes of an external reference node

| **Attribute** | **Description** |
|---|---|
| iedName | Name of the publisher’s IED |
| ldInst | Name of the publisher’s logical device |
| prefix | Prefix of the publisher’s logical node |
| lnClass | Class of the publisher’s logical node |
| lnInst | Instance number of the publisher’s logical node |
| doName | Name of the publisher’s data object |
| daName | Name of the publisher’s data attribute |
| intAddr | Reference to the data attribute of the added logical node in the subscriber’s CID file |

The internal address attribute shall be in the form “a/b/c/d/e”, where:

- “a” is the subscriber’s logical device instance
- “b” is the prefix, class, and instance number of the added logical node
- “c” is the functional constraint of the added data attribute
- “d” is the added data object name
- “e” is the added data attribute name

# 5 CODE BASE ASSESSMENT

Despite being a tool in released state, the old tool still suffered from minor undiscovered faults. After one week of faults fixing and refactoring, the code base was evaluated to be inconsistent, error-prone, and able to benefit from a complete revamp.

The class diagram of the old tool is shown in Figure 4. The class `IEC61850Dialog` is the main dialog. The class kept lists of XML nodes, tree items, attributes, attribute values, nodes' tag names, nodes' (context) values. Those lists were always "synchronized" so that the nth element in one list is related to the nth elements in other lists. This design created considerable amount of maintenance work to keep the lists in the right order. It was also time-inefficient when finding and removing nodes. Another aspect worth considering is handling the CID file. In the old tool, the CID file was read from and written to inside the main class, lengthening the source file significantly. In addition to the design problems, there was an inconvenience related to the Qt framework. The class `QDomNodeList` is designed without iterators, making it impossible to utilize any standard library (STL) algorithm. The mentioned problems shall be fixed before adding any new feature to the tool.

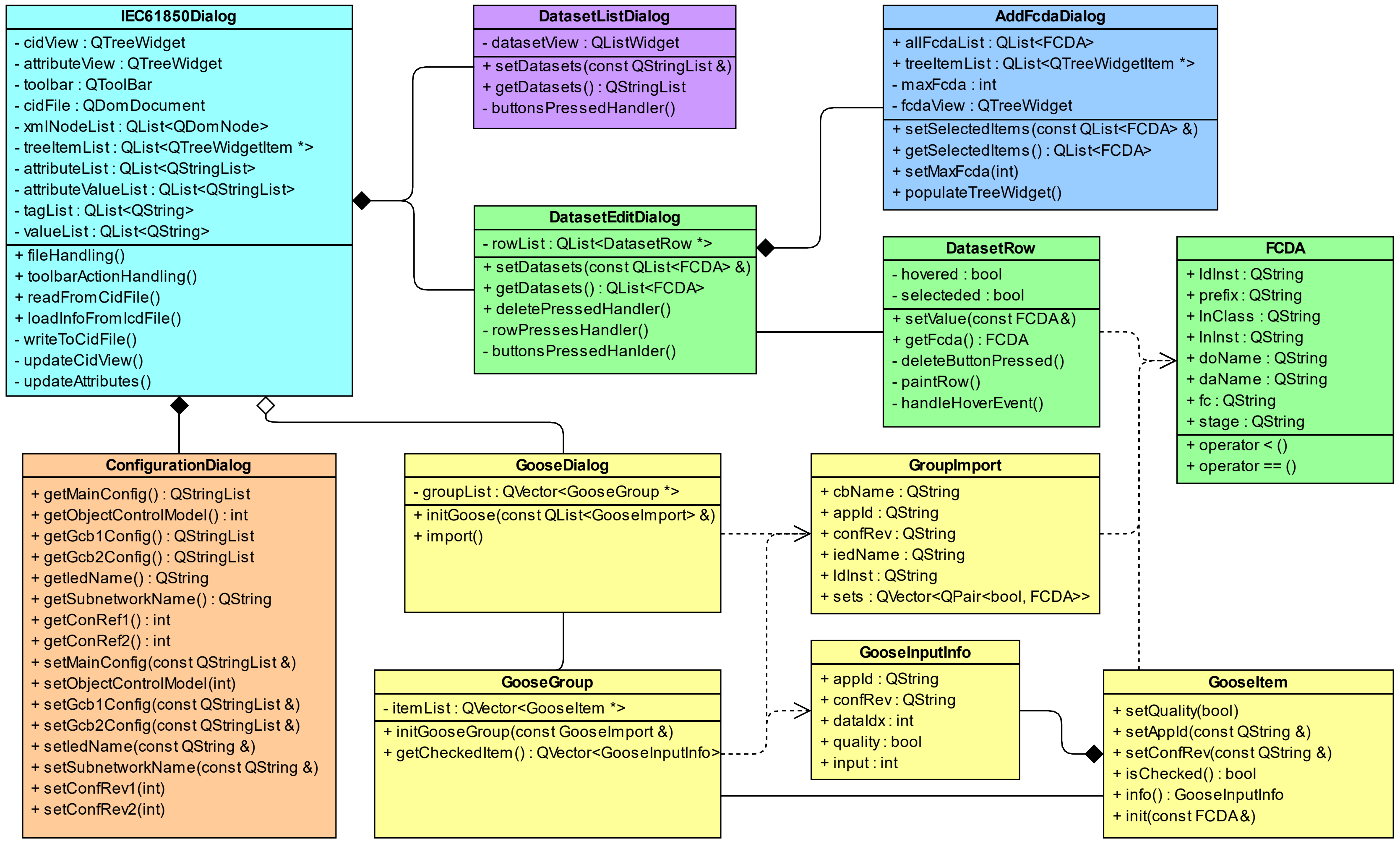

IEC61850Dialog
- cidView : QTreeWidget
- attributeView : QTreeWidget
- toolbar : QToolBar
- cidFile : QDomDocument
- xmlNodeList : QList<QDomNode>
- treeItemList : QList<QTreeWidgetItem *>
- attributeList : QList<QStringList>
- attributeValueList : QList<QStringList>
- tagList : QList<QString>
- valueList : QList<QString>
+ fileHandling()
+ toolbarActionHandling()
+ readFromCidFile()
+ loadInfoFromIcdFile()
- writeToCidFile()
- updateCidView()
- updateAttributes()
DatasetListDialog
- datasetView : QListWidget
+ setDatasets(const QStringList &)
+ getDatasets() : QStringList
- buttonsPressedHandler()
AddFcdaDialog
+ allFcdaList : QList<FCDA>
+ treeItemList : QList<QTreeWidgetItem *>
- maxFcda : int
- fcdaView : QTreeWidget
+ setSelectedItems(const QList<FCDA> &)
+ getSelectedItems() : QList<FCDA>
+ setMaxFcda(int)
+ populateTreeWidget()
DatasetEditDialog
- rowList : QList<DatasetRow *>
+ setDatasets(const QList<FCDA> &)
+ getDatasets() : QList<FCDA>
+ deletePressedHandler()
- rowPressesHandler()
- buttonsPressedHanlder()
DatasetRow
- hovered : bool
- selecteded : bool
+ setValue(const FCDA&)
+ getFcda() : FCDA
- deleteButtonPressed()
- paintRow()
- handleHoverEvent()
FCDA
+ ldInst : QString
+ prefix : QString
+ lnClass : QString
+ lnInst : QString
+ doName : QString
+ daName : QString
+ fc : QString
+ stage : QString
+ operator < ()
+ operator == ()
ConfigurationDialog
+ getMainConfig() : QStringList
+ getObjectControlModel() : int
+ getGcb1Config() : QStringList
+ getGcb2Config() : QStringList
+ getIedName() : QString
+ getSubnetworkName() : QString
+ getConRef1() : int
+ getConRef2() : int
+ setMainConfig(const QStringList &)
+ setObjectControlModel(int)
+ setGcb1Config(const QStringList &)
+ setGcb2Config(const QStringList &)
+ setIedName(const QString &)
+ setSubnetworkName(const QString &)
+ setConfRev1(int)
+ setConfRev2(int)
GooseDialog
- groupList : QVector<GooseGroup *>
+ initGoose(const QList<GooseImport> &)
+ import()
GroupImport
+ cbName : QString
+ appId : QString
+ confRev : QString
+ iedName : QString
+ ldInst : QString
+ sets : QVector<QPair<bool, FCDA>>
GooseGroup
- itemList : QVector<GooseItem *>
+ initGooseGroup(const GooseImport &)
+ getCheckedItem() : QVector<GooseInputInfo>
GooseInputInfo
+ appId : QString
+ confRev : QString
+ dataIdx : int
+ quality : bool
+ input : int
GooseItem
+ setQuality(bool)
+ setAppId(const QString &)
+ setConfRev(const QString &)
+ isChecked() : bool
+ info() : GooseInputInfo
+ init(const FCDA&)

**Figure 4.** Old tool class diagram

# 6 IMPLEMENTATION

This chapter covers the design of the program and the methods used to fulfil the requirements above. The old tool was developed using C++ and Qt framework version 5.9.0. Hence, the new tool was developed using the same tools.

## 6.1 Refactor

In order to give more contexts to datum, especially the frequently used XML nodes, more classes were made as shown in Figure 5. Most of the classes were for storing data and providing comparison, conversion, and sorting functionalities.

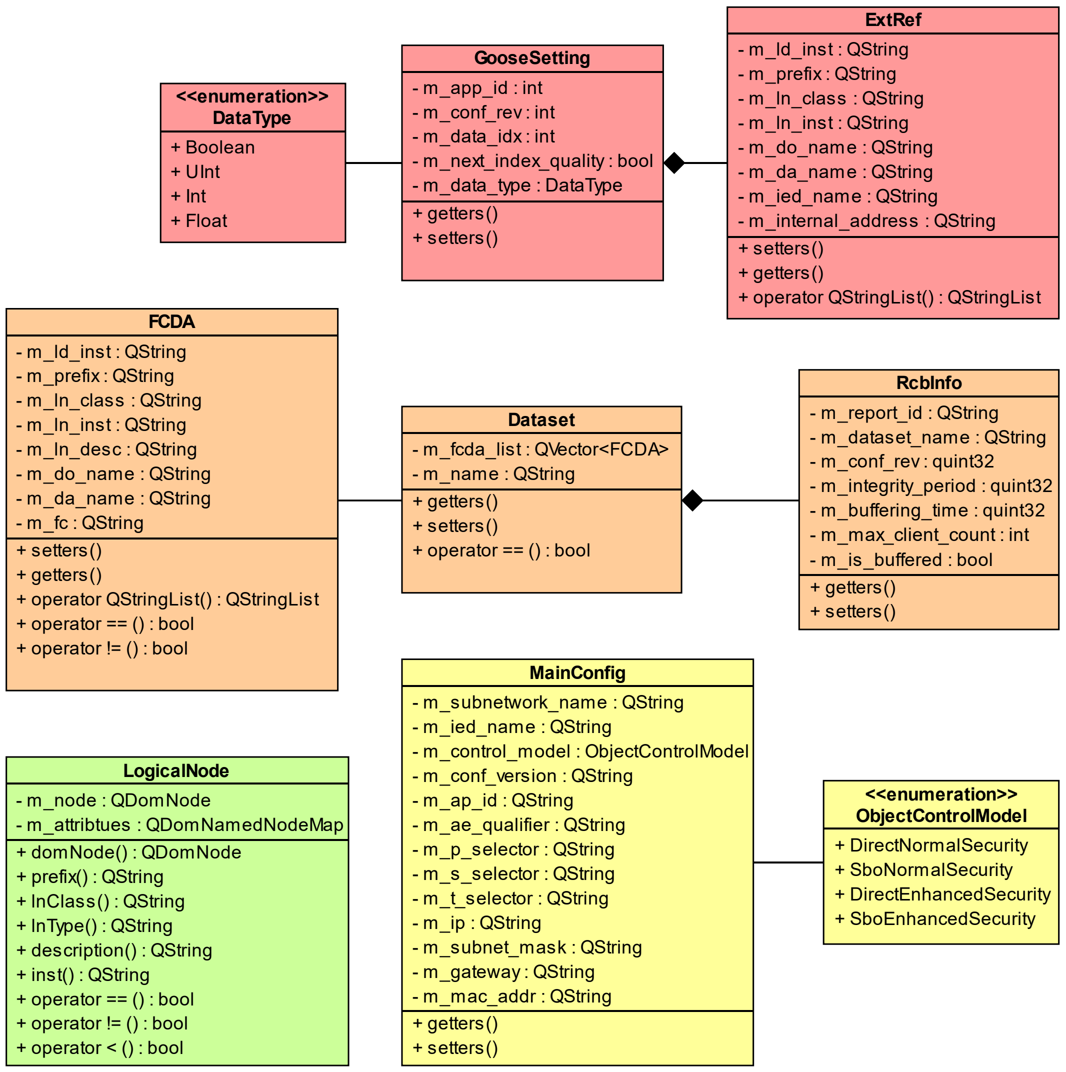


**Figure 5.** Classes representing frequently used XML node

To overcome the lack of iterators in `QDomNodeList`, a new class inheriting from `QVector<QDomNode>`, `DomNodeList`, was introduced. Since the class inherits from `QVector`, iterator functionalities are also inherited. A constructor taking `QDomNodeList` as a parameter was also provided to facilitate the conversion between the classes. It was implemented simply by copying the `QDomNode` objects one-by-one to the new class. This operation will be slow if the object count is great enough. This drawback should be considered when the class is used.

Next, three more classes were created – `CidFile`, `CidReader` and `CidWriter`, separating the code interacting with the CID file from the main dialog. The classes relationship is shown in Figure 6.

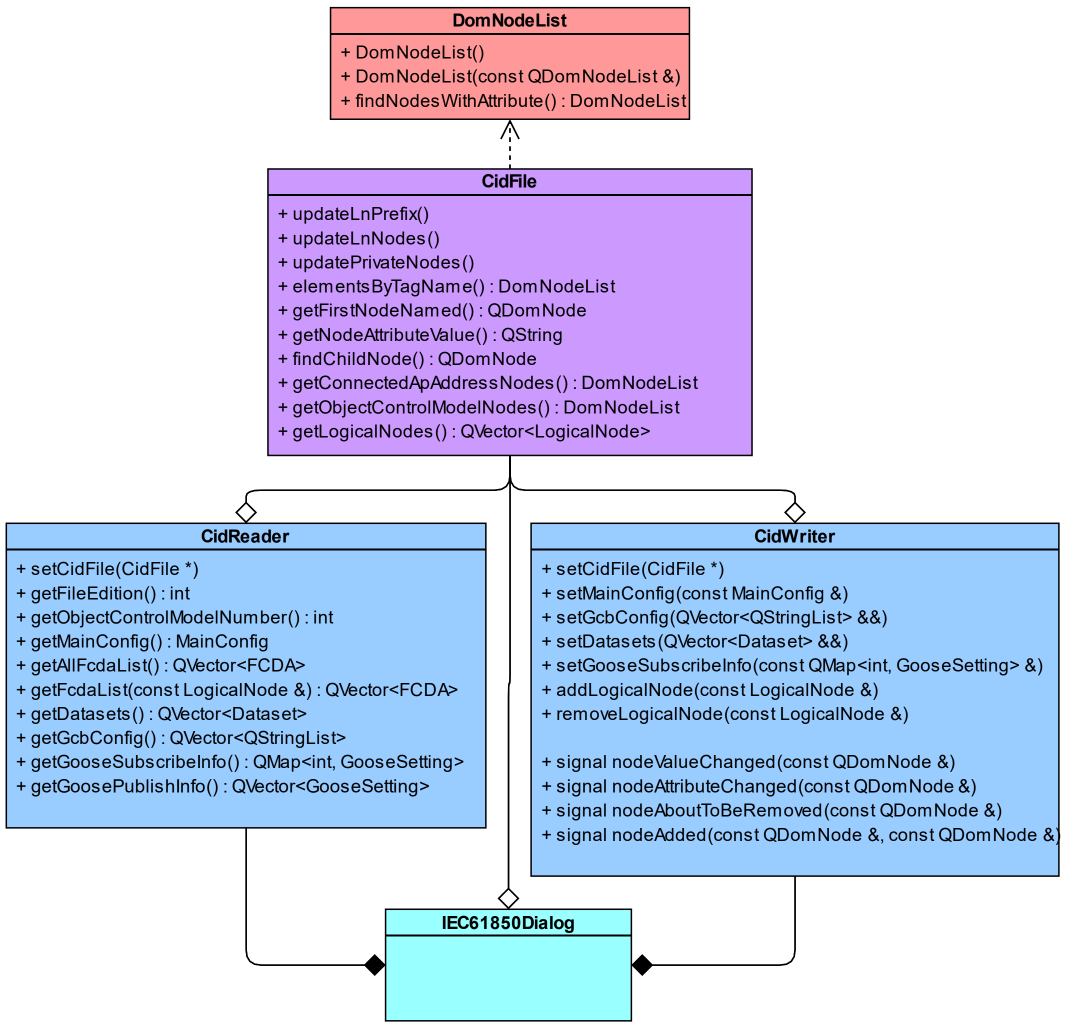

DomNodeList
+ DomNodeList()
+ DomNodeList(const QDomNodeList &)
+ findNodesWithAttribute() : DomNodeList
CidFile
+ updateLnPrefix()
+ updateLnNodes()
+ updatePrivateNodes()
+ elementsByTagName() : DomNodeList
+ getFirstNodeNamed() : QDomNode
+ getNodeAttributeValue() : QString
+ findChildNode() : QDomNode
+ getConnectedApAddressNodes() : DomNodeList
+ getObjectControlModelNodes() : DomNodeList
+ getLogicalNodes() : QVector<LogicalNode>
CidReader
+ setCidFile(CidFile *)
+ getFileEdition() : int
+ getObjectControlModelNumber() : int
+ getMainConfig() : MainConfig
+ getAllFcdaList() : QVector<FCDA>
+ getFcdaList(const LogicalNode &) : QVector<FCDA>
+ getDatasets() : QVector<Dataset>
+ getGcbConfig() : QVector<QStringList>
+ getGooseSubscribeInfo() : QMap<int, GooseSetting>
+ getGoosePublishInfo() : QVector<GooseSetting>
CidWriter
+ setCidFile(CidFile *)
+ setMainConfig(const MainConfig &)
+ setGcbConfig(QVector<QStringList> &&)
+ setDatasets(QVector<Dataset> &&)
+ setGooseSubscribeInfo(const QMap<int, GooseSetting> &)
+ addLogicalNode(const LogicalNode &)
+ removeLogicalNode(const LogicalNode &)
+ signal nodeValueChanged(const QDomNode &)
+ signal nodeAttributeChanged(const QDomNode &)
+ signal nodeAboutToBeRemoved(const QDomNode &)
+ signal nodeAdded(const QDomNode &, const QDomNode &)
IEC61850Dialog

**Figure 6.** Classes interacting with CID file

`CidFile` extends from `QDomDocument`, providing convenient functions to acquire attributes or nodes related to the CID file. The class was designed to only return “basic” types – `QDomNode`, `DomNodeList` and `LogicalNode`. The functions extracting information from `QDomNode` objects into data classes were provided in the class `CidReader`. Those functions implicitly guarantee not to alter the CID file in anyway. On the other hand, almost every function in the `CidWriter` class alters the CID file. The class `CidWriter` also provides signals to notify about nodes being added, removed, or changed in attributes or context value to facilitate updating item views.

The three CID file handling classes should remove the need of keeping lists of attributes, tags, and context values in the main class. However, the XML nodes list and tree items list were still necessary to update the underlying nodes when the tree items’ values change and vice versa. Hence, to make the searching operation more time efficient, `QHash` was used. It is a template class providing a hash-table-based dictionary. It is implemented so that if the key lookup operation is used multiple times, the average complexity is O(1) /8/. One requirement of using `QHash` is to provide a global function called `qHash()` for the key type, which must return different values for different keys. The Qt framework already provided this function for many of its classes but `QDomNode`, which is reasonable since there can be two identical nodes in an XML file. However, given the structure of a CID file, it is not possible for two nodes to share the same parent and have identical attributes. In that case, it is straight-forward to create a unique string for each node by combining the node’s absolute path from the root node with its attributes and attributes’ values. Notice that the hash cannot be changed after the key is added to the container. Therefore, it should be updated whenever the node object has its attributes’ values changed, or the attributes that can be changed should not be added to the hash string at all. In this work, the latter option was chosen since there were not many changing attributes and removing those would not affect the unicity of the nodes.

## 6.2 Implement New Features

### 6.2.1 Row-focused Table View

In the tool, the data was shown as rows in table objects. In order to maintain the uniformity of the look and behavior of the tables, a class structure was introduced, as shown in Figure 7.

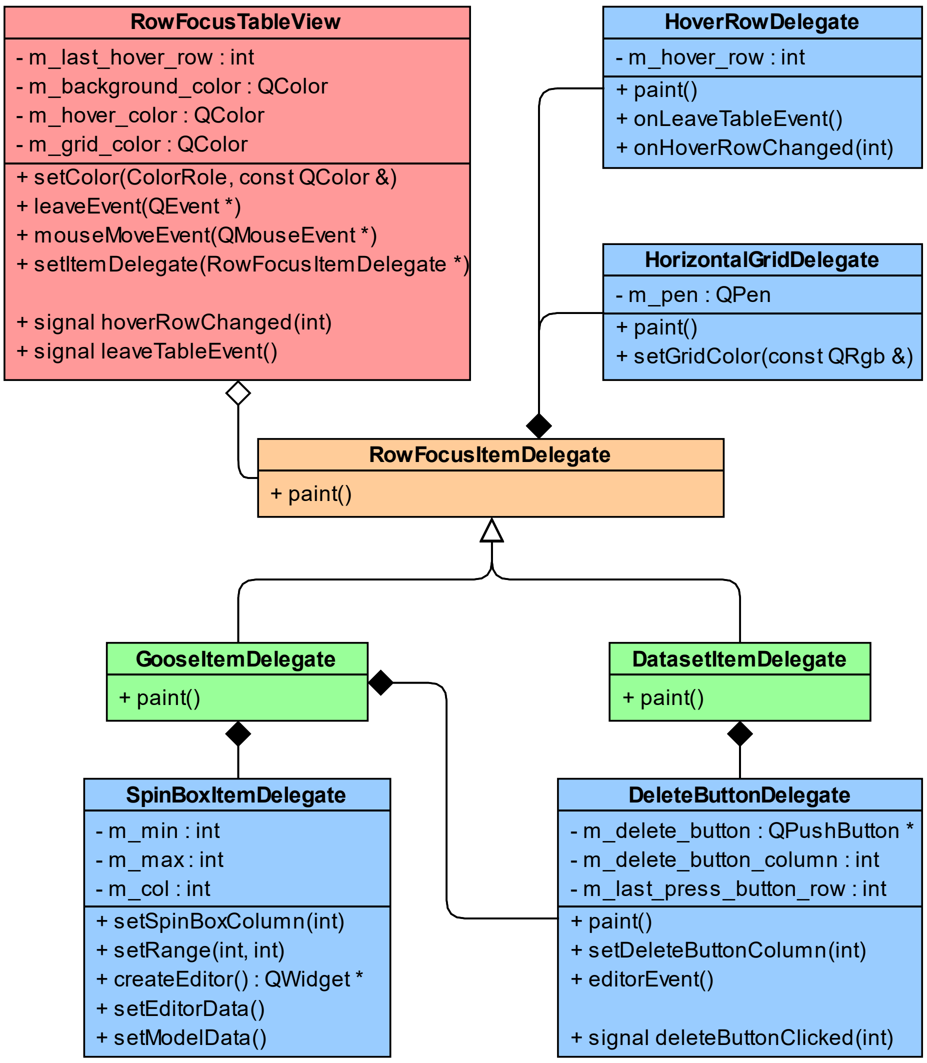


**Figure 7.** Item delegate classes

The item delegate's functionalities were separated into elementary classes: `SpinBoxItemDelegate`, `DeleteButtonDelegate`, `HoverRowDelegate` and `HorizontalGridDelegate`, all of which inherits from `QStyledItemDelegate`.

The idea was that if any delegate class need a combination of functionalities, it can just have the necessary elementary classes as its members.

The class `RowFocusTableView` provides an implementation of a table view that only has horizontal grid and has row-highlighting behavior on mouse hover. To achieve those, it must be paired with a specific delegate, `RowFocusItemDelegate`, through the function:

- `setItemDelegate(RowFocusItemDelegate *)`

connecting the mouse tracking signals from the table view to the handling function in the delegate. As a result, the following functions inheriting from `QTableView` were rendered obsolete and were marked deleted:

- `setItemDelegate(QAbstractItemDelegate *)`
- `setItemDelegateForColumn(int, QAbstractItemDelegate *)`
- `setItemDelegateForRow(int, QAbstractItemDelegate *)`

### 6.2.2 Extract Data from ICD Files

Since the class `CidFile` already provided a convenient function `elementsByTagName()`, the logical nodes can effortlessly be acquired with the tags LN and LN0 (for the logical node zero node). The FCDAs' data (described in Table 3), on the other hand, are stored in different nodes. The ldInst attribute is stored in the inst attribute of the LDevice node (parent of all logical nodes). The attributes prefix, lnClass and lnInst are stored in the logical node containing the data attribute. The doName attribute can be found from a LNodeType node, whose id attribute matches the lnType attribute of the logical node. Similarly, the attributes daName and fc can be found from a DOType node.

To facilitate finding the type nodes, the LNodeType, DOType and DAType nodes were stored as vectors in the class `CidFile` right after the document was loaded. The vectors were sorted so that a node can be found faster using `std::lower_bound()`. The functions to extract the FCDAs' data were implemented in the `CidReader` class as shown in Figure 8.

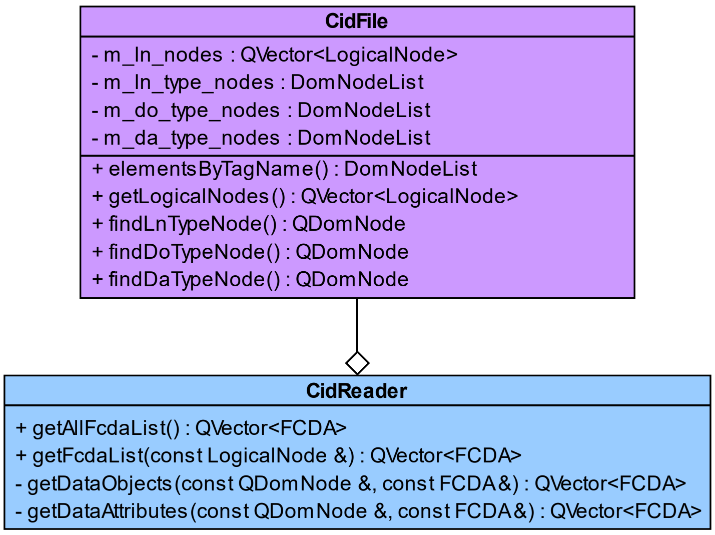

CidFile
- m_ln_nodes : QVector<LogicalNode>
- m_ln_type_nodes : DomNodeList
- m_do_type_nodes : DomNodeList
- m_da_type_nodes : DomNodeList
+ elementsByTagName() : DomNodeList
+ getLogicalNodes() : QVector<LogicalNode>
+ findLnTypeNode() : QDomNode
+ findDoTypeNode() : QDomNode
+ findDaTypeNode() : QDomNode
CidReader
+ getAllFcdaList() : QVector<FCDA>
+ getFcdaList(const LogicalNode &) : QVector<FCDA>
- getDataObjects(const QDomNode &, const FCDA&) : QVector<FCDA>
- getDataAttributes(const QDomNode &, const FCDA&) : QVector<FCDA>

**Figure 8.** Functions extracting FCDAs' attribute

The function `getAllFcdaList()` is just a convenient function to get all logical nodes and call `getFcdaList()` on each of the nodes. The function `getFcdaList()` shall be used to return all FCDAs inside a specific logical node. It is done by creating a template FCDA for each data attribute found in the logical node and filling in the attributes it can acquire from the logical node and the LDevice node. Then, the template FCDA is passed to `getDataObjects()` together with the LNodeType node to get the attribute doName. The `getDataObjects()` function operates in the same way and subsequently calls `getDataAttributes()` to fill the attributes daName and fc.

### 6.2.3 Add Description to Dataset Editor

When extracting FCDAs from the opening CID file, the descriptions of the logical nodes are also collected and stored in the `FCDA` object as shown in Figure 5 (although not needed in the actual FCDA nodes). This will allow the descriptions of the logical nodes to be shown in the editor as required.

### 6.2.4 Edit Logical Nodes

There were two requirements concerning logical nodes: adding/removing and changeable prefix attribute. To support adding and removing logical nodes manually, a dialog represented by the class `EditLogicalNodeDialog` was made. It simply contains two vectors of `LogicalNode` objects, a `QStandardItemModel` object as an item model connecting to a `RowFocusTableView` as the item view and interfaces to interact with the two vectors. The data for the vector of all logical nodes are taken from one of the two ICD files provided by Arcteq, depending on the edition of the editing CID file. The vector is sorted in order to quickly find the selected node.

To determine which nodes are chosen, each logical node in the ICD file must match exactly one logical node in the opening CID file. Since the prefix attributes of logical nodes are editable (see requirement 4.3), the unicity of the combination of prefix, lnClass, and inst attributes defined by the standard cannot be used. Instead, the new tool uses the combination of lnClass, lnType, inst, and desc attributes, which is unique in Arcteq's ICD file. Any future changes that affect the unicity of the used combination will require changes in the implementation of this feature.

After the dialog is accepted, there are two vectors of logical nodes: one contains the newly selected nodes obtained through the dialog, another one contains the previously selected nodes in the CID file. The two vectors are then compared using `std::set_difference()` to obtain the nodes to be added and the nodes to be deleted. Note that the function only works with sorted ranges.

Implementing the editable prefix attribute feature was quite straight-forward. Since the specification was already detailed to logical nodes, the program was hard coded to set the value items of those attributes to be editable. The validation of the inputted prefix is done after the data is changed in the item model. If the inputted prefix already exists or is the same with the reserved prefix, it is reset to the value before changing. The old value is stored beforehand by implementing an item model class inheriting from `QStandardItemModel`. The function `setData()` was overridden to emit a `dataAboutToChange()` signal before setting the data.

Since the FCDAs for the datasets are stored inside logical nodes, when the logical nodes are changed, the following lists may have to be updated:

- List of available FCDAs to be added to datasets
- Lists of already added FCDAs in datasets

The functions `updatePrefix()`, `addFcda()` and `removeFcda()` from the `DatasetListDialog` class were made available to the main dialog as shown in Figure 9. Those functions can be used to update the FCDAs in the dataset list stored in the `DatasetListDialog` object and the available FCDA list stored in the `AddFcdaDialog` object.

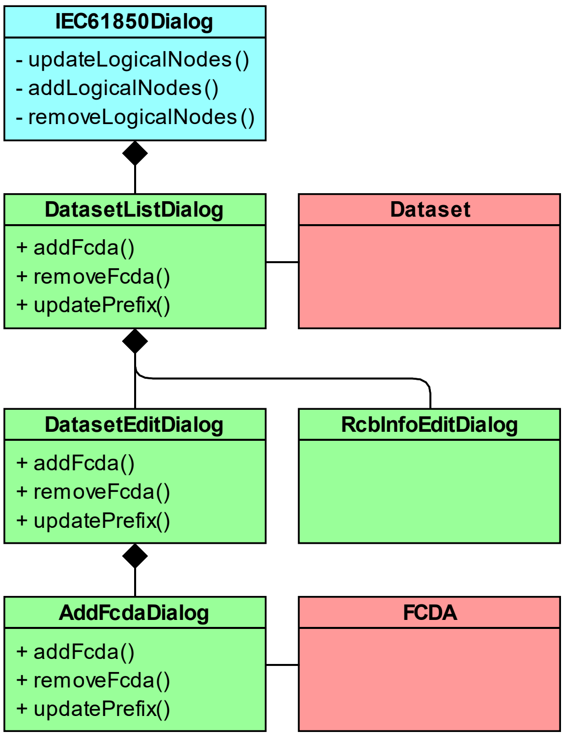


**Figure 9.** Editing datasets classes

### 6.2.5 Set Maximum Client for Report Control Nodes

The private function `createReportControlNodes()` in the class `CidWriter` returns a `DomNodeList` object, with different contents depending on the edition of the opening CID file. The edition information is acquired from the `CidReader` object by the main dialog and then given to the `CidWriter` object.

### 6.2.6 Export Logical Nodes Information

Since all the information required to be exported can be found in the class `FCDA`, the vector of all `FCDA` objects – the “master” vector – were used as a starting point. The vector was sorted by the combination of prefix, lnClass, lnInst, and doName attributes in the listing order to group all data attributes under the same data object together. The function `std::unique_copy()` was then used on the sorted vector to get a new vector containing one FCDA from each data object – the “unique” vector.

From there, the work was straight-forward. For each element in the “unique” vector, find all elements in the “master” vector that are in the same data object and create one entry in the output file. To accelerate the search, `std::lower_bound()` and `std::upper_bound()` were used and the matched elements were erased right after the information extraction was done.

### 6.2.7 Edit GOOSE Subscription Information

To support editing GOOSE subscription information, two more classes were introduced: `GooseDialog` and `GoosePublisherDialog`. The class `GooseDialog` was designed to show the GOOSE configuration in the opening CID file. It holds a map connecting GOOSE input numbers (integers range from 1 to 64) to objects of type `GooseSetting`. The GOOSE configuration to be imported from another CID file (the publisher) should be shown in an object of type `GoosePublisherDialog`. Both classes contain a `RowFocusTableView` object as the item view, paired with an item delegate of type `GooseItemDelegate` (described in Figure 7), allowing the user to delete any row using the button placed at the end of each row and change the GOOSE input number using a spin box widget. The interfaces and relationship of the classes are shown in Figure 10.

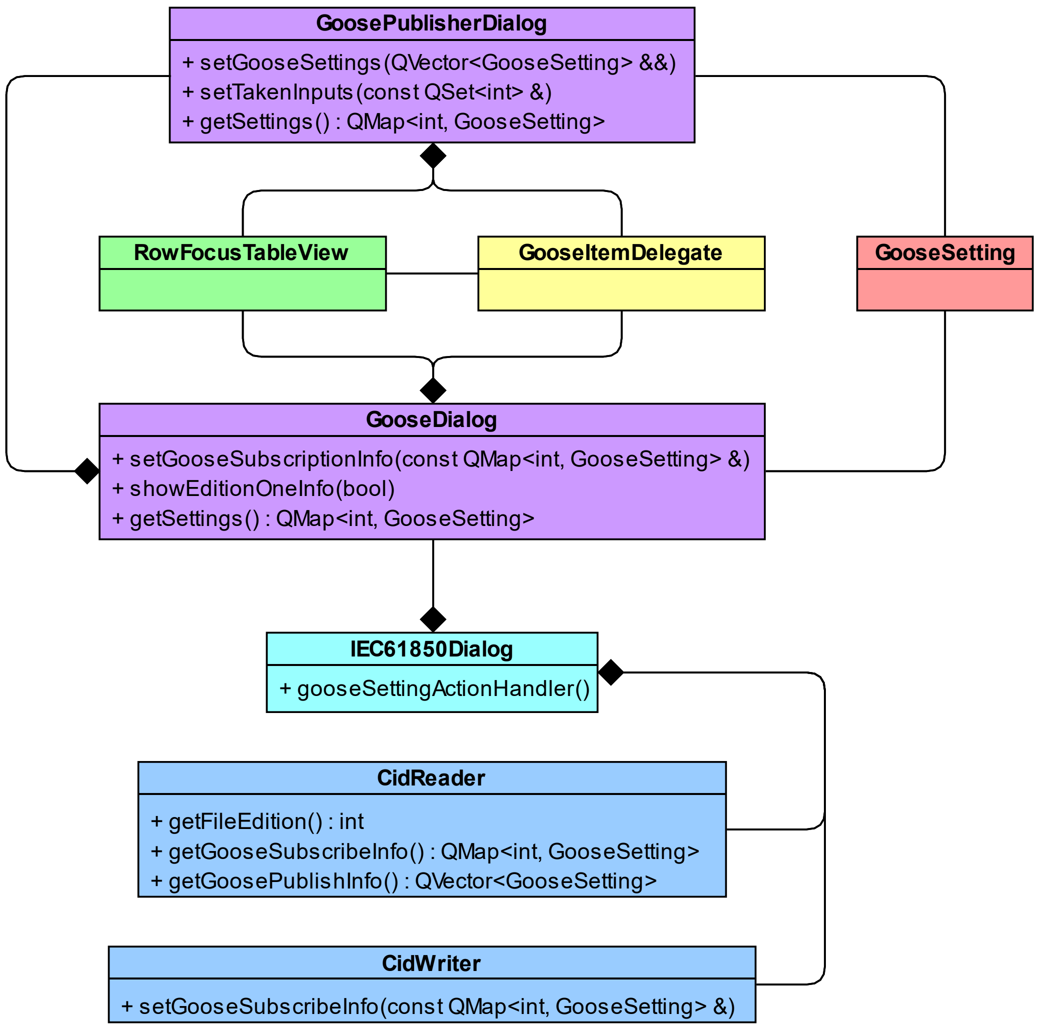

GoosePublisherDialog
+ setGooseSettings(QVector<GooseSetting> &&)
+ setTakenInputs(const QSet<int> &)
+ getSettings() : QMap<int, GooseSetting>
RowFocusTableView
GooseItemDelegate
GooseSetting
GooseDialog
+ setGooseSubscriptionInfo(const QMap<int, GooseSetting> &)
+ showEditionOneInfo(bool)
+ getSettings() : QMap<int, GooseSetting>
IEC61850Dialog
+ gooseSettingActionHandler()
CidReader
+ getFileEdition() : int
+ getGooseSubscribeInfo() : QMap<int, GooseSetting>
+ getGoosePublishInfo() : QVector<GooseSetting>
CidWriter
+ setGooseSubscribeInfo(const QMap<int, GooseSetting> &)

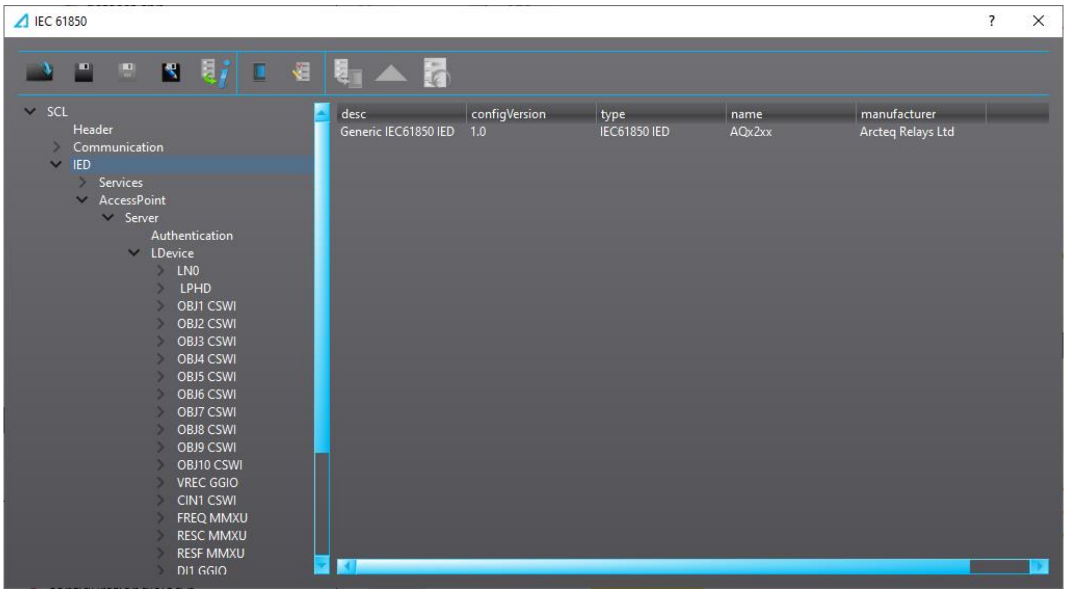
**Figure 10.** Editing GOOSE configuration classes

When a CID file is opened, the subscription information is processed and displayed in the `GooseDialog` object. The user will be given a warning when changing the GOOSE input number to an existing one, and the input number will be changed back to the previous value. A file dialog will be shown when the import button is pressed. After the publisher's CID file is chosen, the publishing datum are displayed in the `GoosePublisherDialog` object for editing before being added. The program assigns an input number to each entry automatically by finding the smallest input number that has not been taken. If all input number have been taken, the input number will be -1, and will not be added to the opening CID file when the `GoosePublisherDialog` is accepted.

# 7 RESULT

## 7.1 Main Dialog

Figure 11 and 9 show the main dialog of the old tool and the new tool.

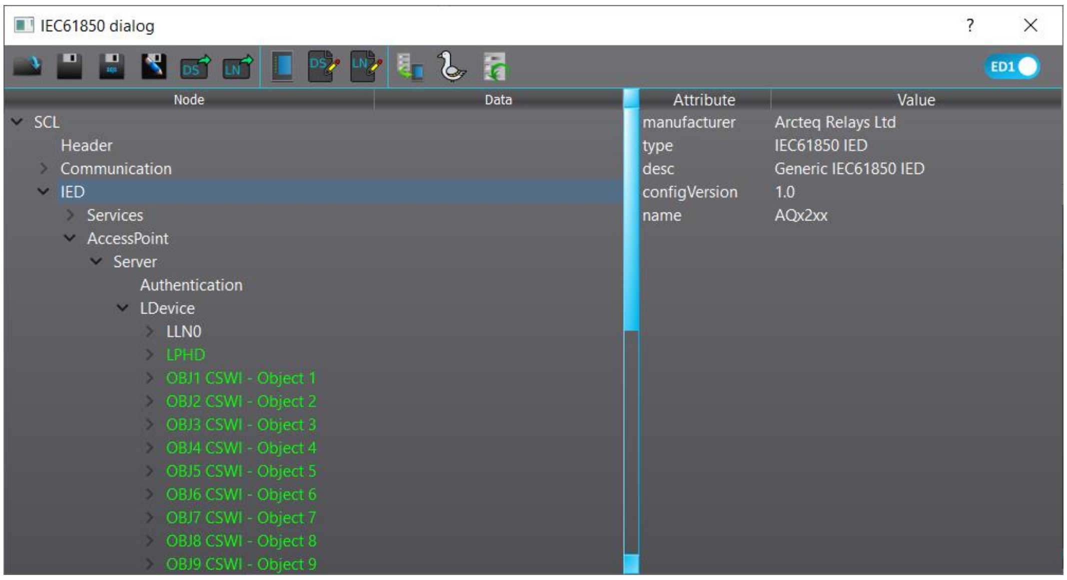


IEC 61850

```
SCL
  Header
  Communication
  IED
    Services
    AccessPoint
      Server
        Authentication
        LDevice
          LN0
          LPHD
          OBJ1 CSWI
          OBJ2 CSWI
          OBJ3 CSWI
          OBJ4 CSWI
          OBJ5 CSWI
          OBJ6 CSWI
          OBJ7 CSWI
          OBJ8 CSWI
          OBJ9 CSWI
          OBJ10 CSWI
          VREC GGIO
          CIN1 CSWI
          FREQ MMXU
          RESC MMXU
          RESF MMXU
          DI1 GGIO
```

| desc | configVersion | type | name | manufacturer |
|---|---|---|---|---|
| Generic IEC61850 IED | 1.0 | IEC61850 IED | AQx2xx | Arcteq Relays Ltd |

**Figure 11.** Old tool main dialog



IEC61850 dialog

ED1

| Node | Data |
|---|---|

```
SCL
  Header
  Communication
  IED
    Services
    AccessPoint
      Server
        Authentication
        LDevice
          LLN0
          LPHD
          OBJ1 CSWI - Object 1
          OBJ2 CSWI - Object 2
          OBJ3 CSWI - Object 3
          OBJ4 CSWI - Object 4
          OBJ5 CSWI - Object 5
          OBJ6 CSWI - Object 6
          OBJ7 CSWI - Object 7
          OBJ8 CSWI - Object 8
          OBJ9 CSWI - Object 9
```

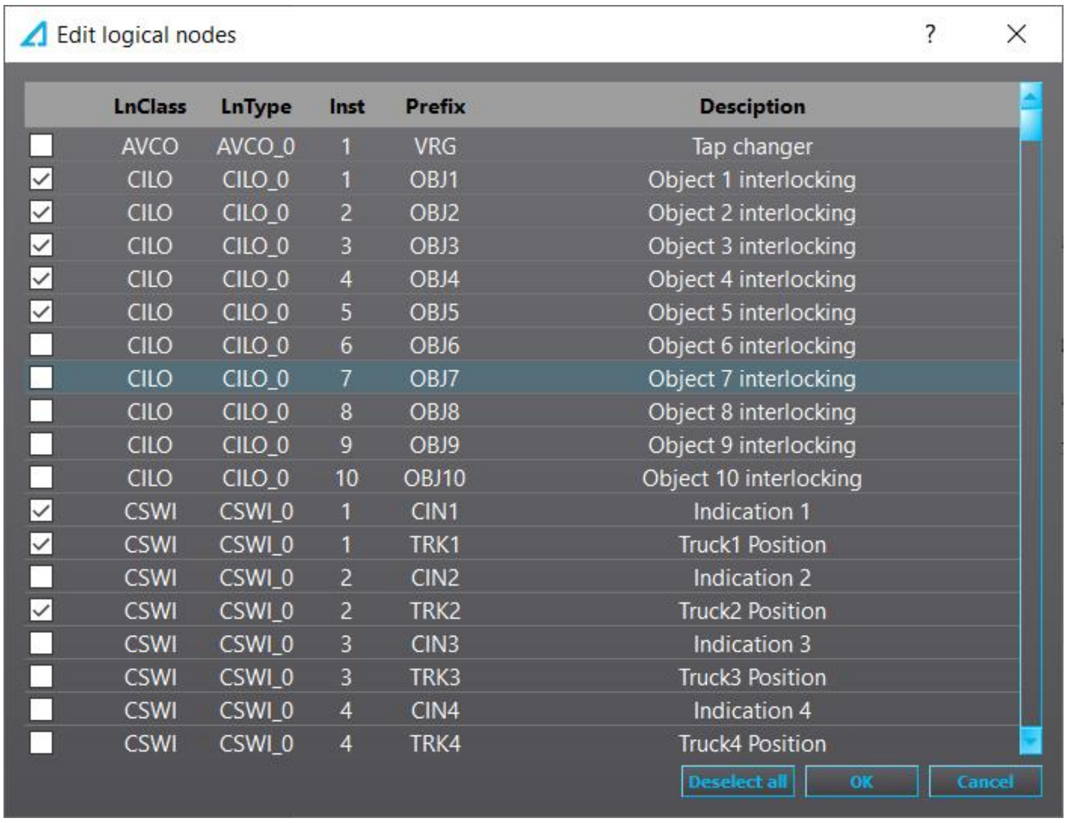
| Attribute | Value |
|---|---|
| manufacturer | Arcteq Relays Ltd |
| type | IEC61850 IED |
| desc | Generic IEC61850 IED |
| configVersion | 1.0 |
| name | AQx2xx |

**Figure 12.** New tool main dialog

The main visible differences are three more buttons in the tool bar, the attribute view, and the logical nodes' texts. The attribute view in the new tool displays the attributes vertically instead of horizontally. The logical nodes' texts are more descriptive, and coloured green since they contain editable attributes. The added buttons are for exporting logical nodes information, editing logical nodes, and indicating edition. The edition indicating button currently does not respond to presses from the user since the edition switching feature has not been supported.

## 7.2 Edit Logical Node Dialog

Figure 13 shows the dialog for adding/removing logical nodes.



Edit logical nodes

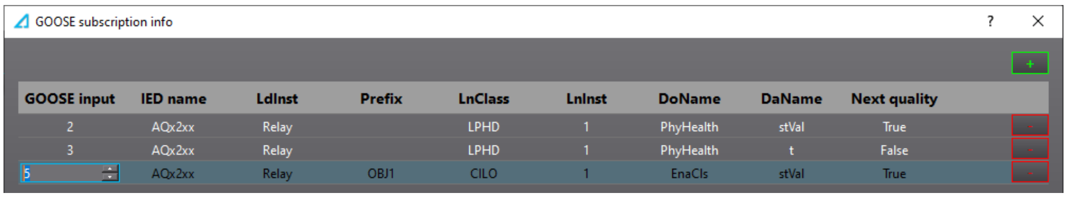
| | LnClass | LnType | Inst | Prefix | Desciption |
|---|---|---|---|---|---|
| ☐ | AVCO | AVCO_0 | 1 | VRG | Tap changer |
| ✓ | CILO | CILO_0 | 1 | OBJ1 | Object 1 interlocking |
| ✓ | CILO | CILO_0 | 2 | OBJ2 | Object 2 interlocking |
| ✓ | CILO | CILO_0 | 3 | OBJ3 | Object 3 interlocking |
| ✓ | CILO | CILO_0 | 4 | OBJ4 | Object 4 interlocking |
| ✓ | CILO | CILO_0 | 5 | OBJ5 | Object 5 interlocking |
| ☐ | CILO | CILO_0 | 6 | OBJ6 | Object 6 interlocking |
| ☐ | CILO | CILO_0 | 7 | OBJ7 | Object 7 interlocking |
| ☐ | CILO | CILO_0 | 8 | OBJ8 | Object 8 interlocking |
| ☐ | CILO | CILO_0 | 9 | OBJ9 | Object 9 interlocking |
| ☐ | CILO | CILO_0 | 10 | OBJ10 | Object 10 interlocking |
| ✓ | CSWI | CSWI_0 | 1 | CIN1 | Indication 1 |
| ✓ | CSWI | CSWI_0 | 1 | TRK1 | Truck1 Position |
| ☐ | CSWI | CSWI_0 | 2 | CIN2 | Indication 2 |
| ✓ | CSWI | CSWI_0 | 2 | TRK2 | Truck2 Position |
| ☐ | CSWI | CSWI_0 | 3 | CIN3 | Indication 3 |
| ☐ | CSWI | CSWI_0 | 3 | TRK3 | Truck3 Position |
| ☐ | CSWI | CSWI_0 | 4 | CIN4 | Indication 4 |
| ☐ | CSWI | CSWI_0 | 4 | TRK4 | Truck4 Position |

**Figure 13.** Edit logical nodes dialog

The class `RowFocusTableView` was used with the default item delegate (`RowFocusItemDelegate`) to present the data. The implementation worked as intended with no vertical grid and row highlighting on mouse hover.

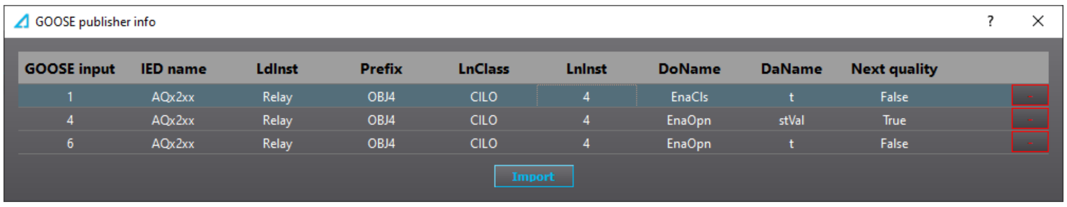
## 7.3 Edit GOOSE subscription dialog

The dialogs displaying GOOSE subscription and publishing information are shown in Figure 14 and Figure 15.



| GOOSE input | IED name | LdInst | Prefix | LnClass | LnInst | DoName | DaName | Next quality |
|---|---|---|---|---|---|---|---|---|
| 2 | AQx2xx | Relay | | LPHD | 1 | PhyHealth | stVal | True |
| 3 | AQx2xx | Relay | | LPHD | 1 | PhyHealth | t | False |
| 5 | AQx2xx | Relay | OBJ1 | CILO | 1 | EnaCls | stVal | True |

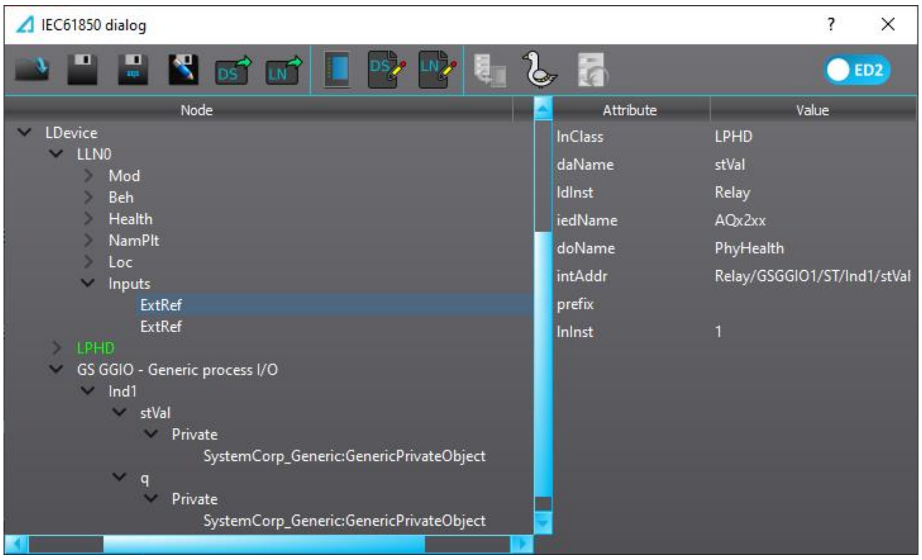
**Figure 14.** GOOSE subscription information dialog



| GOOSE input | IED name | LdInst | Prefix | LnClass | LnInst | DoName | DaName | Next quality |
|---|---|---|---|---|---|---|---|---|
| 1 | AQx2xx | Relay | OBJ4 | CILO | 4 | EnaCls | t | False |
| 4 | AQx2xx | Relay | OBJ4 | CILO | 4 | EnaOpn | stVal | True |
| 6 | AQx2xx | Relay | OBJ4 | CILO | 4 | EnaOpn | t | False |

**Figure 15.** GOOSE publishing information dialog

Since an item delegate of type `GooseItemDelegate` was used, the user can delete entries and edit GOOSE input numbers. The entries in the publishing dialog are assigned input numbers that have not been taken as expected. The resulting CID file structure after an entry is added to the subscription dialog is shown in Figure 16.



```
Node
LDevice
  LLN0
    Mod
    Beh
    Health
    NamPlt
    Loc
    Inputs
      ExtRef
      ExtRef
  LPHD
  GS GGIO - Generic process I/O
    Ind1
      stVal
        Private
          SystemCorp_Generic:GenericPrivateObject
      q
        Private
          SystemCorp_Generic:GenericPrivateObject
```

| Attribute | Value |
|---|---|
| lnClass | LPHD |
| daName | stVal |
| ldInst | Relay |
| iedName | AQx2xx |
| doName | PhyHealth |
| intAddr | Relay/GSGGIO1/ST/Ind1/stVal |
| prefix | |
| lnInst | 1 |

**Figure 16.** Resulting CID file after subscription

# 8 TESTING

## 8.1 Unit tests and integration tests

All unit tests and integration tests were done after a function or group of functions were implemented by observing outputs for different inputs. No testing framework was used.

## 8.2 System tests

After the work had been implemented, system testing was done by comparing the output file (CID file or text file) from the new tool with the desired outcome. The comparison was made for each of the following actions:

- Add, remove, and edit datasets
- Add, remove logical nodes
- Edit report control node attributes
- Edit logical nodes' prefixes
- Import, edit GOOSE configuration
- Export logical nodes' data

To facilitate the comparing process, a program called Oxygen XML Developer 22.0 developed by Syncro Soft SRL was used. The input and desired output files were provided by the customer. Additionally, the features inherited from the old tool were also tested to ensure their serviceability.

## 8.3 Acceptance tests

After all system tests have passed, the output CID files from the new tool were used to configure actual protection relays. This work was done by the customer.

# 9 DISCUSSION AND CONCLUSION

This chapter covers the assessment of the phases of the project, possible future developments of the work, and the conclusion made.

## 9.1 Project Assessment

The project was done over the time span of about 260 hours, of which:

- 80 hours spent on assessment and refactoring
- 140 hours spent on implementing new features
- 40 hours spent on testing and debugging

The time allocation was reasonable for a small-sized project. The final implementation of the configuration tool was fully functional and met the requirements stated by the customer. The tools used in the project were mainly an IDE for Qt project development and a private Git server provided by the customer. All the tests were done without utilizing any testing tool or framework, which was something to be improved.

## 9.2 Continuation

Future developments can be focusing to three aspects: improving the opening time, supporting edition switch, and utilizing logical nodes' contents to identity logical nodes.

The opening time for the largest CID file provided (the ICD file) was measured to be approximately 3 seconds. The time was used mostly in the process of getting all FCDA information and creating the tree items to display those data .While it is not too big a problem, lessening that time will increase the efficiency of the configuration tool.

Supporting edition switch would greatly facilitate the use of the configuration tool. The edition switch interface was already made, but since the feature was out of scope, it was never implemented.

The ability to identify a logical node by a private node inside of it was known very late in the testing phase, and the implementation was evaluated to be difficult. Therefore, it was not utilized in the final version of the tool. If it were in used, the identification of logical nodes would be based on a well-defined rule instead of an observed implicit rule of the CID file (see 6.2.4). In addition, more attributes of the logical nodes will be able to be changed.

## 9.3 Conclusion

The project presented all the needed works to upgrade an IED configuration tool from supporting only edition 1 to supporting both edition 1 and edition 2 of the IEC 61850 standard, covering code base refactoring, implementing new features and possible future developments. As long as the IEC 61850 standard is still in used, this project can serve as a reference for all vendors to work on similar problems.

# 10 REFERENCES

/1/ Arcteq. About company. Accessed 06.05.2020. https://arcteq.fi/company/

/2/ Mackiewicz, R. 2006. Overview of IEC 61850 and benefits. IEEE PES Power Systems Conference and Exposition October 29 – November 1, 2006. Montreal, Quebec.

/3/ IEC TR 61850-1:2013. IEC 61850 technical report. Edition 2.0. 2013. 73 pages.

/4/ International Society of Automation. IEC 61850 Power Communications Standard. Accessed 29.04.2020. https://www.automation.com/en-us/articles/2003-1/iec-61850-power-communications-standard-commercial

/5/ Energy software and testing. Introduction to IEC 61850 protocol. Accessed 06.05.2020. https://www.ensotest.com/iec-61850/introduction-to-iec-61850-protocol/

/6/ W3Schools. XML Schema. Accessed 28.04.2020. https://www.w3schools.com/xml/xml_schema.asp

/7/ IEC 61850-6:2009. IEC 61850. Edition 2.0. 2009. 436 pages.

/8/ Qt Documentation. Algorithmic complexity of containers. Accessed 27.4.2020. https://doc.qt.io/qt-5/containers.html#algorithmic-complexity