

User Manual

Part Three

IED Model Extension

V 1.5

2015-07



Copyright: All rights reserved.

None of the information contained in this document may be reproduced or stored in a database or retrieval system or disclosed to others without written authorization by Fuhua Technologies Co. Ltd.

The information in this document is subject to change without prior notice and should not be construed as a commitment by Fuhua Technologies Co. Ltd. Fuhua does not assume responsibility for any errors, which may be in this document.

For more information please visit <http://iedmodeler.com> or contact us at info@fuhuatech.com.



1. Introduction.....	4
2. IED Model Extension.....	6
2.1 SCL Syntax Programming.....	7
2.2 Private Namespace Programming.....	14
2.2.1. How to define XML Schema for Private Namespace?	15
2.2.2. Import Private Namespace	18
2.2.3. Create Private Element.....	21
2.2.4. Create Private Attribute	22
2.2.5. Export SCL Model with Private Namespaces.....	25
2.3 Project Namespaces Management	30

1. Introduction

The IED Modeler/Designer is a comprehensive IED oriented SCL modelling tool for IEC 61850/61400 and companion standards. It has been designed to keep pace with the evolution of standards.

Model Designer, as one of our core products of tool suite is designed to address all the features which are required by different type of users, including IEC 61850 Standard developers, IED vendors, stack vendors, researchers, engineers, utilities companies and people who are interested in applying Model Driven Architecture in system design. With feedbacks from a diversity of users, we keep on developing new features to pill the “pain” confronted when applying IEC 61850/61400 into production. It was born on a mission to help enjoy life in your professional career!

The user manual is composed of four parts, but they are independent and not in sequence.

✓ Part One: IED Model Design

This part is the fundamental. It covers details about installation, introduction to GUI layout, system settings, project management, create/import/modify/export ICD/CID/IID, extract CID from SCD, and generation of TEMPLATE ICD etc. The **Intelligent Creation** features to save you a lot of time and energy in building error-free IED model, such as customizing data types, creation of LNs of different LNodeTypes, creation of DataSet and Control Blocks, initializing DOI values and configuring Communication parameters etc. All those jobs are very convenient to fulfil with help of user friendly Wizards. In addition it also introduces how to batch edit Attribute Values and Element Values using external tools like **Excel**; The Search Utilities is designed to be helpful in searching project items. It uses Fuzzy search algorithm, allowing context search of documentation for each Element and Attribute.

✓ Part Two: Domain Design

This part will introduce you a very comfortable and much higher level of Domain design by utilizing UML technologies. Like other UML tools in the market, Domain design is based on Diagrams. With this tool on hand, you will never feel like asking Domain support from anyone any more. Bet you urgently require Edition 2.0 package for Wind Power, you can home-brew it within one-two hours or even faster if you are an IEC member who is in charge of designing this domain. Within this part Data Type Diagram and Domain Diagram will be introduced, which are much like Class Diagram and Component Diagram in UML, but much more powerful and convenient.

✓ Part Three: IED Model Extension

This part will introduce how to embed private model information into your SCL without breaking any rules defined by IEC 61850-6. Knowing that SCL is lack of PLC logic equations and internal mappings and many others so on. It also doesn't address non-IEC 61850 and vendor-specific

IED Model Extension



parameters configuration and that all those information are essential to run an application. To do this, we have to turn to model extension according to IEC 61850-6. Model Designer can give you a cutting-edge, vendor-independent, flexible and programmable way to accomplish that.

✓ Part Four: IED Model Validation

This part will introduce Schema Check, Integrity Check and Semantic Check against rules defined by Standards. Schema Check is the most popular feature used nowadays and users tend to believe that if Schema Check is passed, their SCL is error-free. But experiences tell us it sounds too good to be true. Why? To fully tell you the reason, I have to write a book about XML Schema defined by W3C. But that is out of the scope of this manual, here I can only tell the reason is because Schema Check are blind to many types of errors in SCL dynamic structures and semantic constraints.

Give you one of the most typical example which I guess you are unaware of:

```
<FCDA lnClass="MMXU" fc="MX" daName="PhV.phsA.cVal.mag.i" lnInst="1" ldInst="LDPQ"/>
```

```
<FCDA lnClass="MMXU" fc="MX" daName="A.phsA.cVal.mag.i" lnInst="1" ldInst="LDPQ"/>
```

The Schema Check is OK about the two DataSet entries above. But they are actually incorrect according to IEC 61850-6. Because PhV.phsA is DO.SDO, not DA. So they should be corrected to

```
<FCDA lnClass="MMXU" fc="MX" doName=" PhV.phsA" daName="cVal.mag.i" lnInst="1" ldInst="LDPQ"/>
```

```
<FCDA lnClass="MMXU" fc="MX" doName="A.phsA" daName="cVal.mag.i" lnInst="1" ldInst="LDPQ"/>
```

To our surprise is MOST of the stack suppliers “accept” this type of errors. In other words, most of the IEDs in the market are running against the rule defined by IEC 61850-6. This is also the reason why IED Model Validation deserves an independent part of user manual.

The good news is that Model Designer offers Integrity Check which can detect errors which are blind to Schema Check.

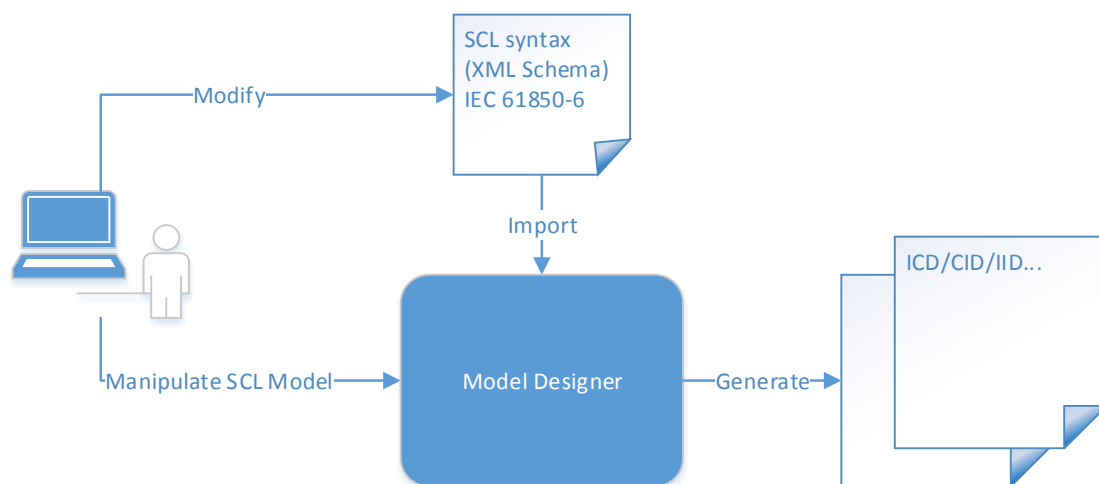
Semantic Check is a feature we will introduce in the future.

2. IED Model Extension

One of the most outstanding features that Model Designer supports is Model Extension. Model Extension is vendor independent, based on XML Schema. It has the ability to compile XML Schema data and produce Meta data which are used by Model Designer to create ICD/CID/IID.

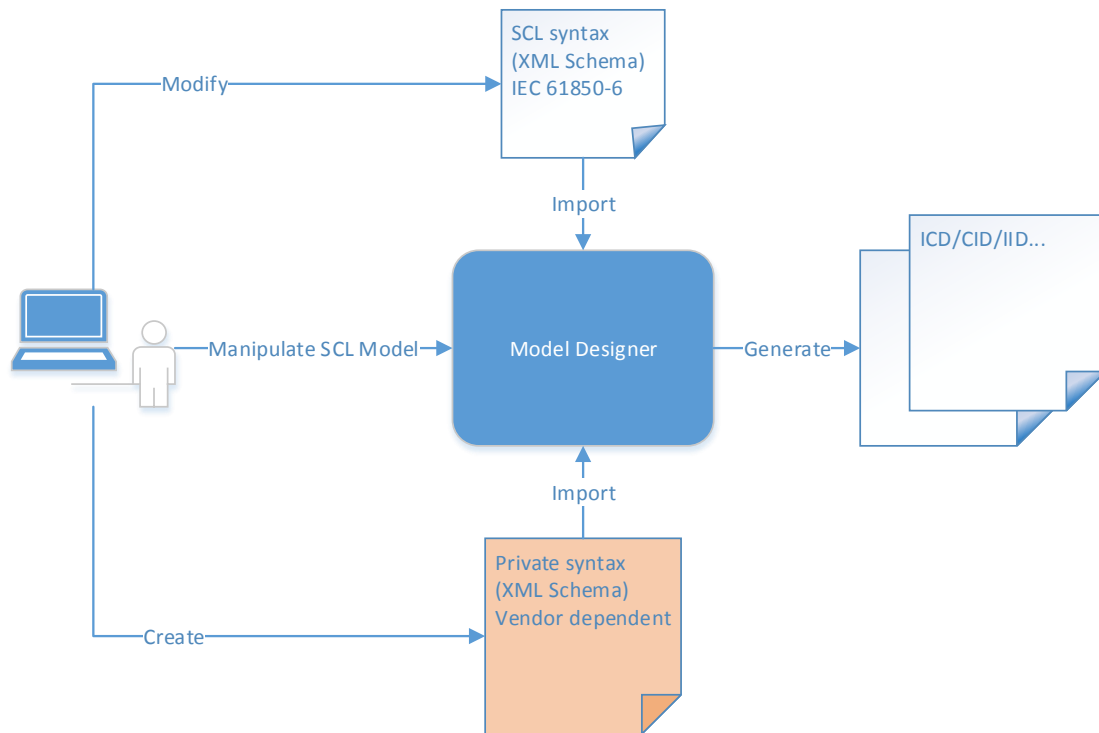
Because XML is considered a middle computer language; it has constructs and comprehensive data type system, so it is possible to program in XML. Model Designer is a tool which supports program written in XML Schema:

✓ SCL Syntax Programming



SCL Syntax is defined in XML Schema specification by IEC 61850-6, but there are a lot of variants used by different vendors throughout the world. Model Designer loads SCL Syntax dynamically and manipulate ICD/CID/IID accordingly. There is no need to update Model Designer program, if the new released version of SCL Syntax is backward compatible. As the picture described above, users can modify SCL syntax and then import it to the tool. The output ICD/CID/IID generated will be compliant to the SCL Syntax imported.

✓ Private Namespace Programming



You can develop Private syntax of your own to extend your SCL model according to rules defined by IEC 61850-6. We know a lot of companies making use of `scl:Private` elements to hold private model data for different application purposes, but that is not the only way to extend model. Model Designer understands Private Syntax defined XML Schema. Once the Private Syntax imported, it will help you to assemble extra model data according to the Private Syntax defined and without breaking the SCL Syntax.

2.1 SCL Syntax Programming

This Section is for advanced users who have a complete understanding of the XML Schema defined by W3C – the SCL Schema as specified in IEC 61850-6 is defined in XML Schema.

To date there are two major editions of SCL Schema, namely edition 1.0 and edition 2.0. In practice vulnerabilities or open issues have resulted in many variants of editions derived from these two major schemas.

It is possible that some IEC 61850 users may need to modify the SCL Schema to address these types of issues. Any modification to SCL Schema affects the structure, syntax and semantics of the SCL model. Most tools in the current marketplace cannot provide a “transparent” implementation of SCL



Schema Programmable.

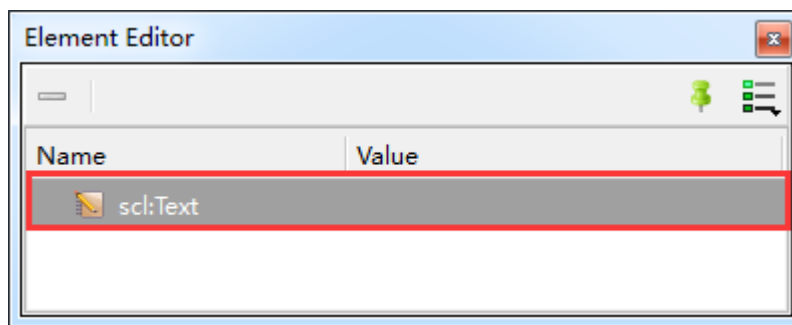
SCL Schema Programmable means that changes/modifications made to SCL Schema automatically program the behavior of the tool.

Using scl:Text as an example:

In SCL_BaseTypes.xsd it is defined as:

```
<xs:complexType name="tText" mixed="true">  
  <xs:complexContent mixed="true">  
    <xs:extension base="tAnyContentFromOtherNamespace">  
      <xs:attribute name="source" type="xs:anyURI" use="optional"/>  
    </xs:extension>  
  </xs:complexContent>  
</xs:complexType>
```

From the definition we know that any Element instance of tText doesn't have any Text Node. A user cannot assign any text value to it.



The scl:Text is grayed, not allow to edit.

If you change the tText definition as:

```
<xs:complexType name="tText" mixed="true">  
  <xs:simpleContent>
```

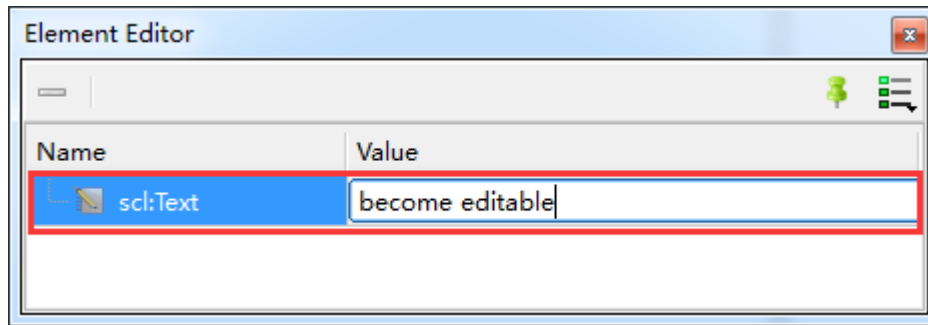


```
<xs:extension base="xs:normalizedString"/>
```

```
</xs:simpleContent>
```

```
</xs:complexType>
```

The tool automatically reconfigures to make the field editable.

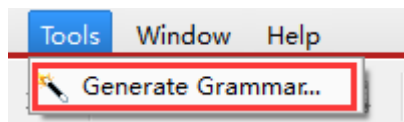


Now scl:Text is no longer grey and is editable.

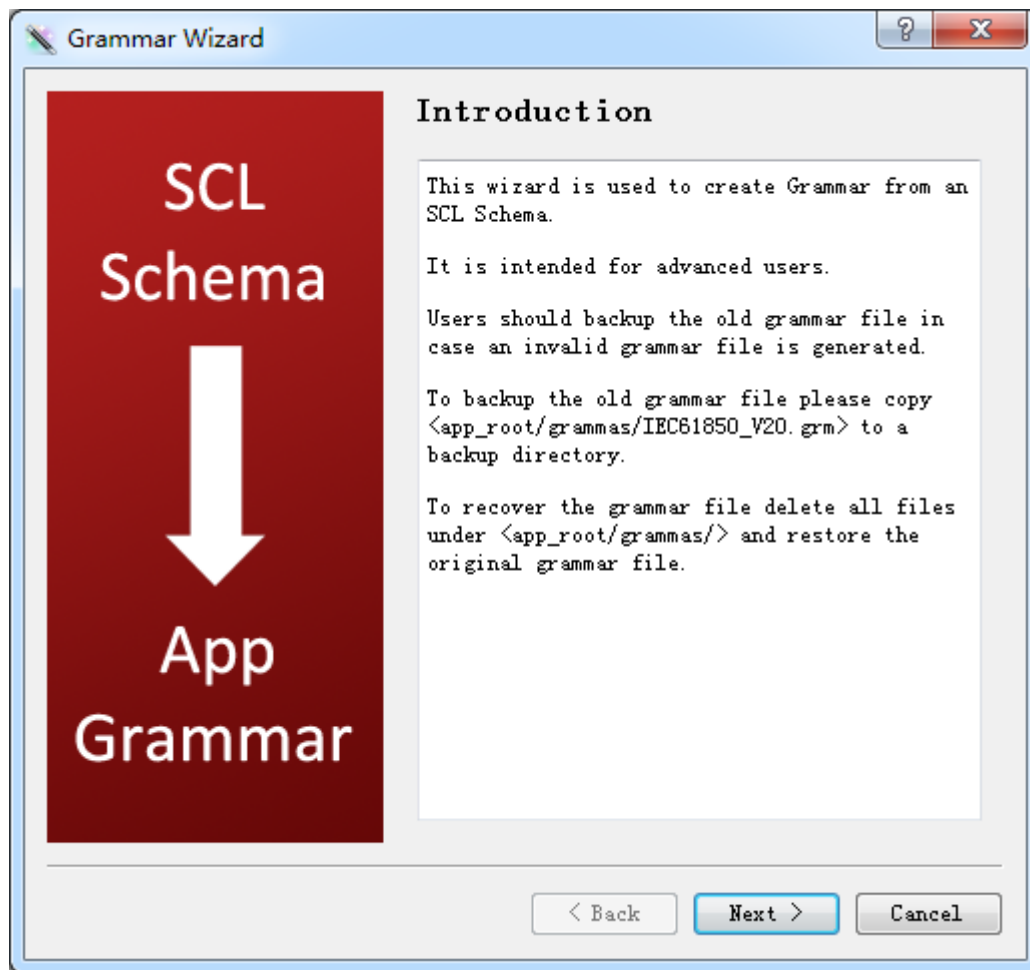
The example illustrates how a user can control the tool behavior by programming the SCL Schema (without requiring modifications to the source code of the tool). In this way the tool can easily adapt and keep pace with the evolution of IEC 61850. A detailed knowledge of XML Schema is required to use the tool in this way and this is outside the scope of this documentation. This is a powerful feature that sets the tool apart from other tools in this space.

The tool loads a grammar file that is generated from the SCL Schema and other collected information. This grammar file is internal flexible metadata that instructs the tool on how to behave/respond to events issued by users/system etc.

To generate a grammar file click “**Generate Grammar...**” from **Tools** menu:

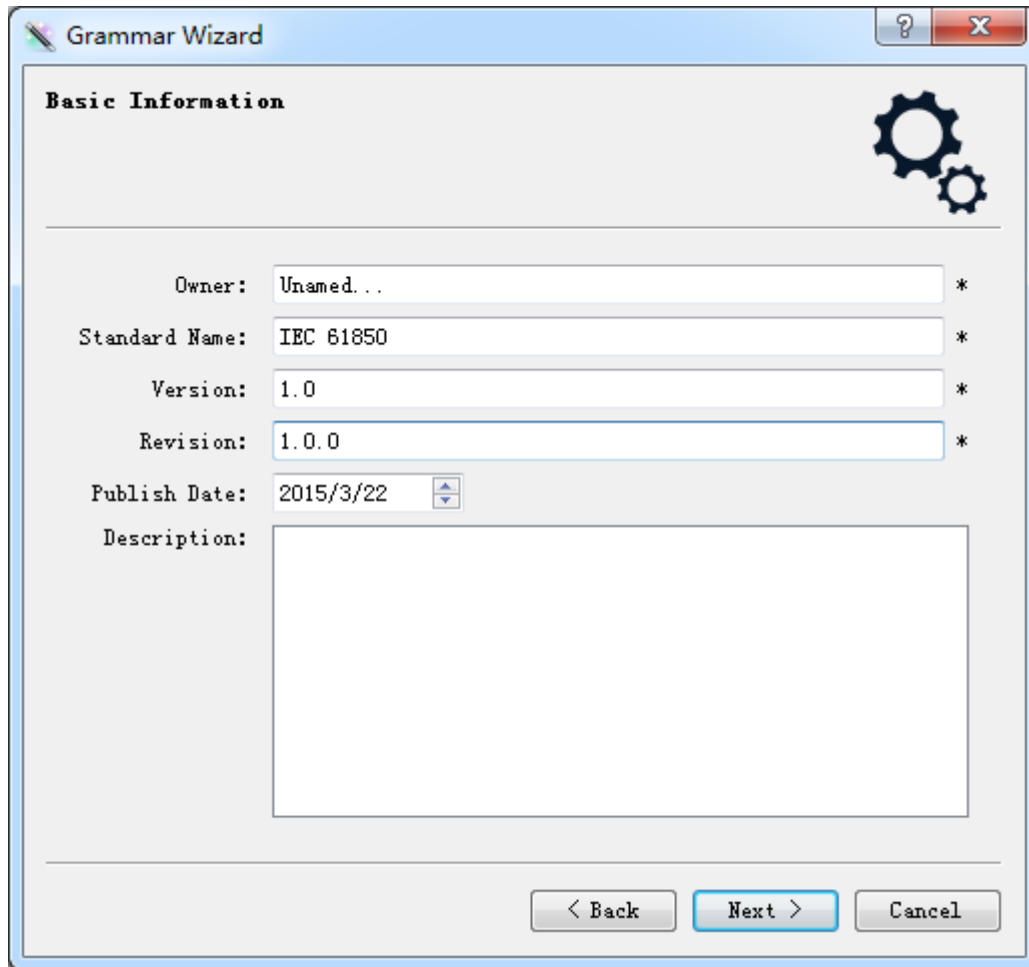


➤ A wizard will show up



Note: backup grammar is important for beginners

➤ Click Next



Grammar Wizard

Basic Information

Owner: *

Standard Name: *

Version: *

Revision: *

Publish Date: *

Description:

< Back Next > Cancel

Owner: who generate the grammar?

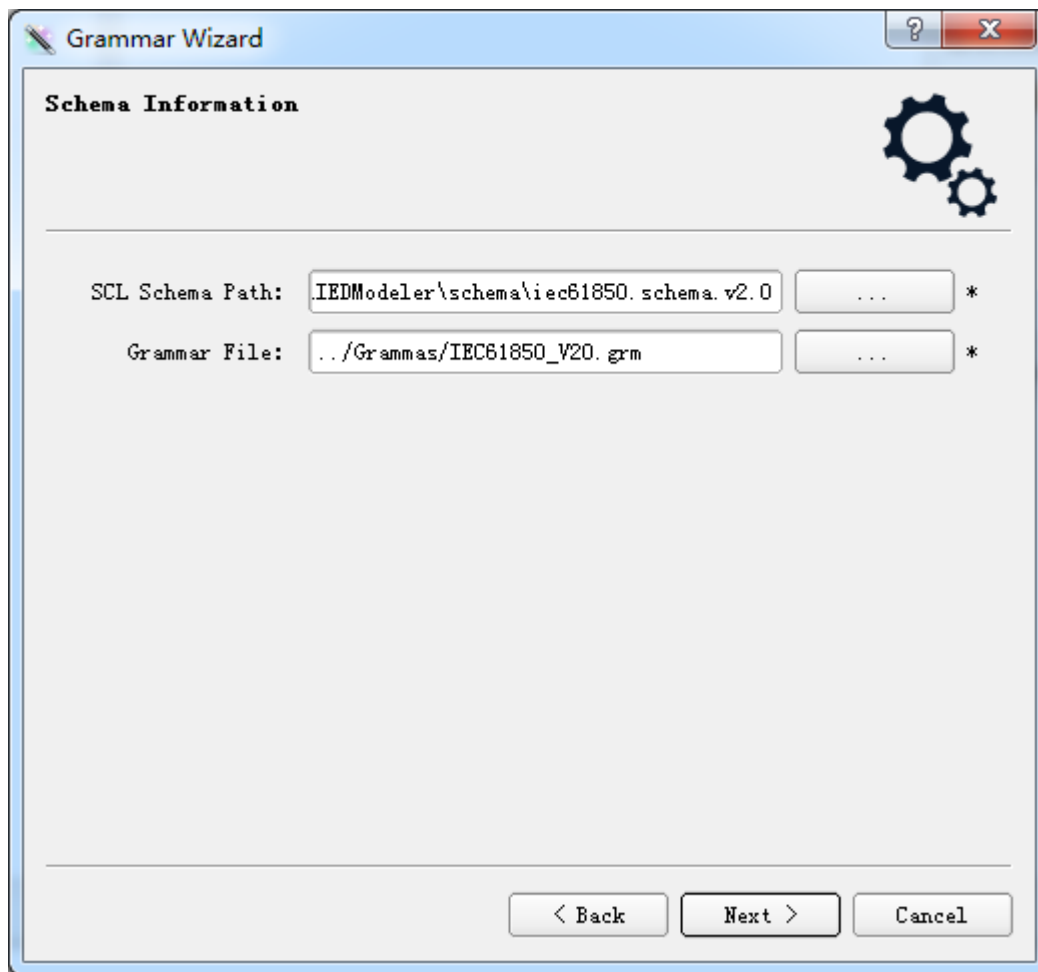
Standard Name: What standard the grammar is targeted to?

Version/Revision: Version control options

Publish Date: The date to publish the grammar

Description: To document the grammar

➤ Click Next



SCL Schema Path: The root directory of SCL Schema

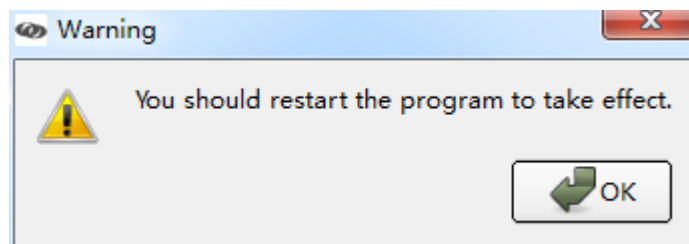
Grammar File: Where the grammar file should be generated?

Note: If you generate the grammar file in a directory other than “../Grammas”, remember to place it under “../Grammas” to take effect.

➤ Click Next

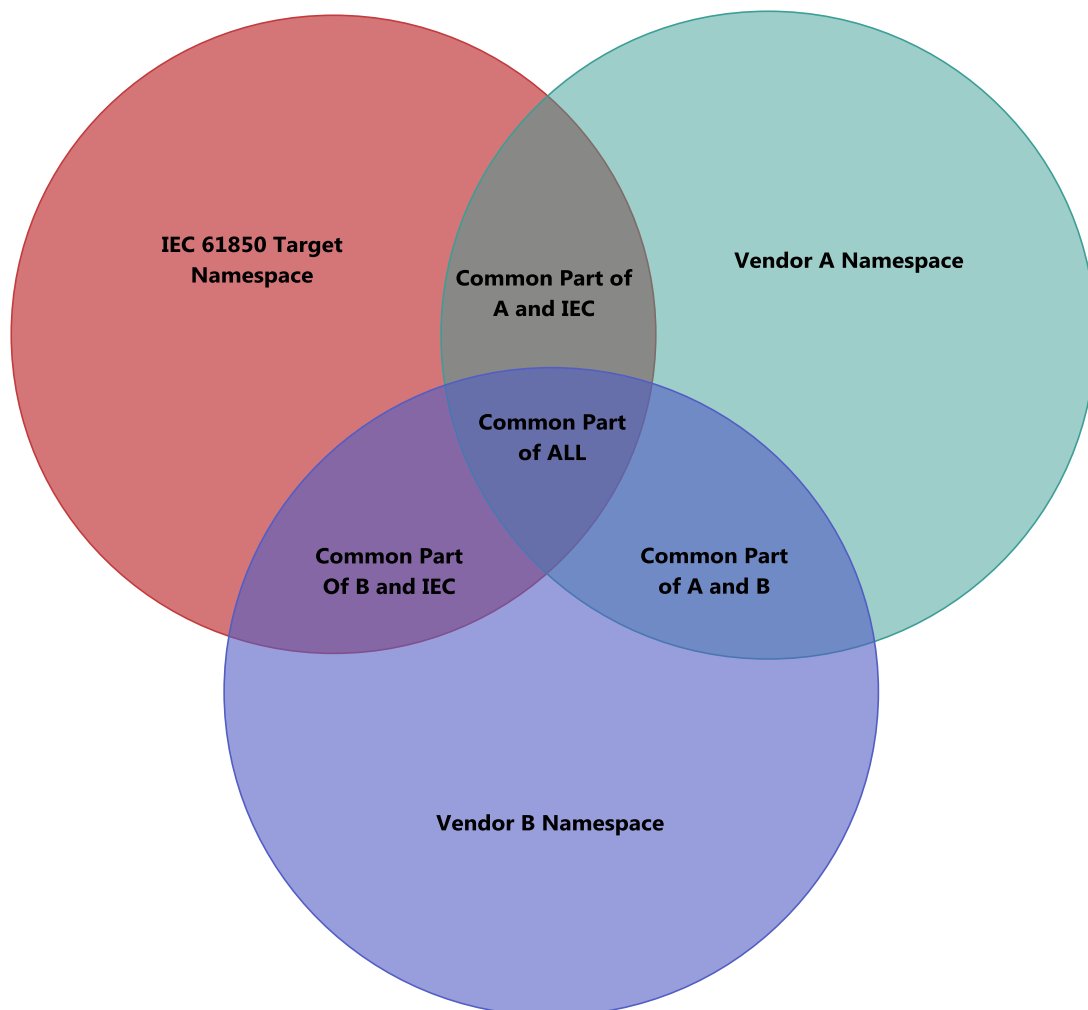


➤ Click Finish



Restart the program to take effect.

2.2 Private Namespace Programming



The diagram above is a practical illustration of the namespaces problem.

The diagram contains 3 namespaces: IEC 61850 (IEC), Vendor A and Vendor B. The Vendor A and Vendor B spaces claim IEC 61850 compatibility of a certain percentage because they have information in common with IEC 61850. However both of them keep large numbers of elements private for:

- Compatibility with other local standards (legacy issue)
- Implementation of new functions that are not yet addressed by IEC 61850
- Hiding the implementation details of their own IED(s)

➤ Other vendor proprietary reasons

A case where all 3 namespaces perfectly overlapped (100% overlapped) would indicate complete standardization. Real world implementations are currently far from perfect because these gaps exist between vendors and IEC 61850. Our challenge is to deliver a tool designed to accommodate current IEC 61850 and existing vendor implementations. Part of our solution to this problem is Private Namespaces Manipulation based on XML Schema.

2.2.1. How to define XML Schema for Private Namespace?

This is an advanced issue and users should have a good command of XML Schema to proceed. XML Schema knowledge is required and outside the scope of this documentation. The following is a simple example of how to map IEC 60870-5-104 to IEC 61850 as an introduction of the feature.

2.2.1.1. Overview of Example

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema      xmlns:IEC_60870_5_104="http://www.iec.ch/61850-80-1/2007/SCL"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.iec.ch/61850-80-1/2007/SCL"
  elementFormDefault="qualified" attributeFormDefault="unqualified">

  <xs:element name="GlobalAddress104">

    <xs:annotation>

      <xs:documentation>Data Point address</xs:documentation>

    </xs:annotation>

    <xs:complexType>

      <xs:attribute name="casdu" type="xs:integer" use="required"/>

      <xs:attribute name="ioa" type="xs:integer" use="required"/>

      <xs:attribute name="ti" type="xs:integer" use="required"/>

    </xs:complexType>

  </xs:element>

</xs:schema>
```

```
</xs:complexType>

</xs:element>

<xs:attributeGroup name="agVendor">

    <xs:attribute name="attA" type="xs:boolean"/>

    <xs:attribute name="attB" type="xs:integer"/>

    <xs:attribute name="attC" type="xs:float"/>

    <xs:attribute name="attD" type="xs:normalizedString"/>

</xs:attributeGroup>

</xs:schema>
```

Note: The example is present in the *AppRoot/examples* directory.

The XML text above defines Private Namespace (<http://www.iec.ch/61850-80-1/2007/SCL>) with a prefix named **IEC_60870_5_104**.

Currently the tool only recognizes global (top most) definitions of **Element** and **AttributeGroup**.

The example above has one global **Element** definition (**GlobalAddress104**) and one global **AttributeGroup** definition (**agVendor**).

2.2.1.2. AttributeGroup

```
<xs:attributeGroup name="agVendor">

    <xs:attribute name="attA" type="xs:boolean"/>

    <xs:attribute name="attB" type="xs:integer"/>

    <xs:attribute name="attC" type="xs:float"/>

    <xs:attribute name="attD" type="xs:normalizedString"/>

</xs:attributeGroup>
```

The XML text above defines four optional Attributes:

IED Model Extension



Name	Type	Condition
attA	xs:boolean	optional
attB	xs:integer	optional
attC	xs:float	optional
attD	xs:normalizedString	optional

Note: Not all the **Simple Types** defined by XML Schema are supported by the tool. The supported types are xs:boolean, xs:integer, xs:float, xs:normalizedString, any other types not listed will lead to undefined behavior.

2.2.1.3. Element

```

<xs:element name="GlobalAddress104">

    <xs:annotation>

        <xs:documentation>Data Point address</xs:documentation>

    </xs:annotation>

    <xs:complexType>

        <xs:attribute name="casdu" type="xs:integer" use="required"/>

        <xs:attribute name="ioa" type="xs:integer" use="required"/>

        <xs:attribute name="ti" type="xs:integer" use="required"/>

    </xs:complexType>

</xs:element>

```

The XML text above defines an Element GlobalAddress104, which has a annotation described as “Data Point address”. The element has three attributes as:

Name	Type	Condition
------	------	-----------

casdu	xs: integer	required
ioa	xs:integer	required
ti	xs: integer	required

The section below demonstrates how to make use of this XML Schema as **Model Extension**.

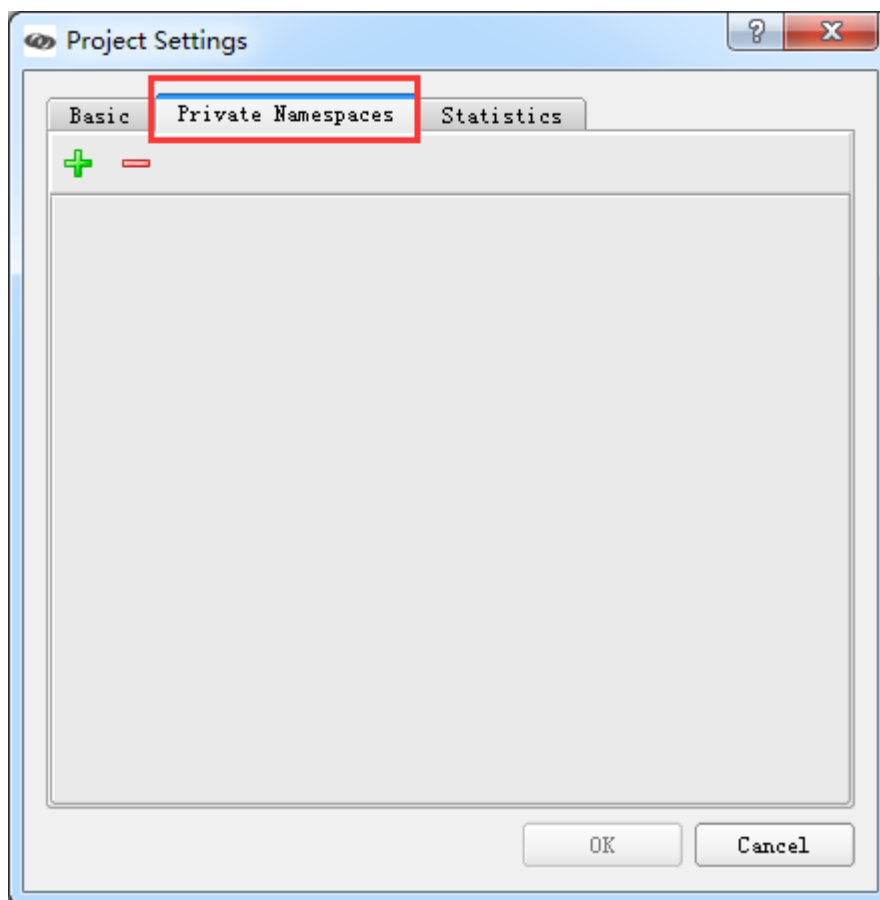
2.2.2. Import Private Namespace

- Click “**Project Settings**” button from the toolbar

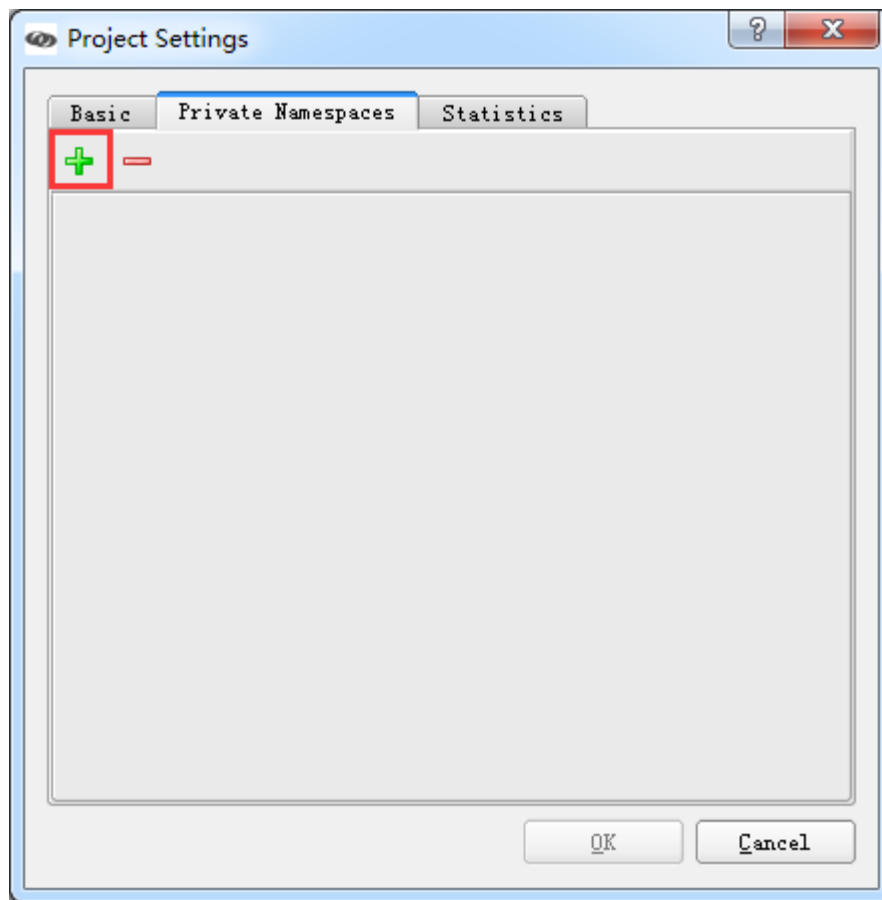


Note: if project isn't saved, the tool will ask you to save it first.

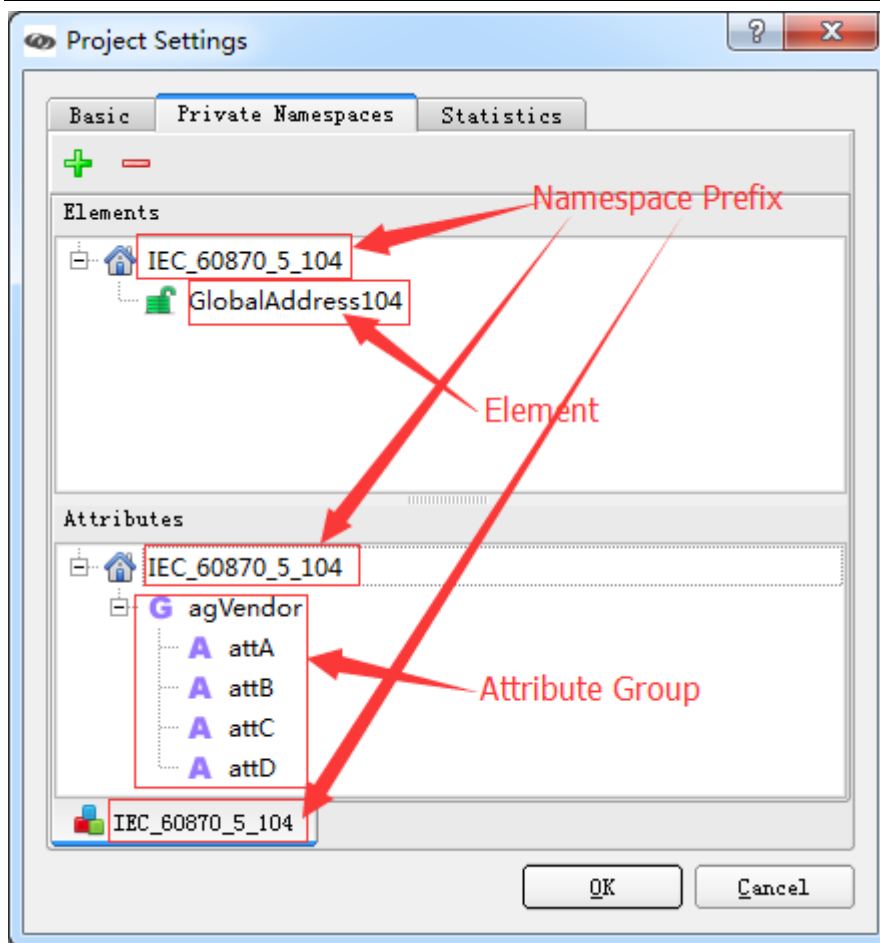
- Switch to **Private Namespaces** page



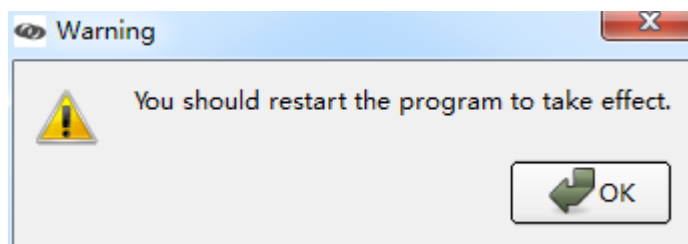
- Click Add button



- Browse and select IEC_60870_5_104.xsd file

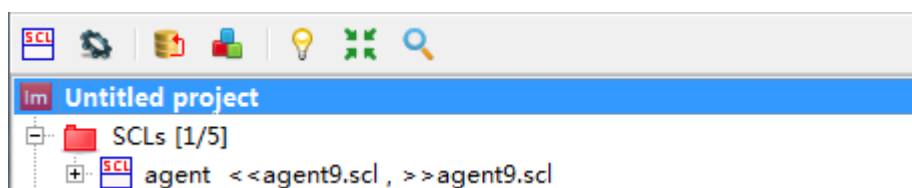


- Click OK and Restart the tool



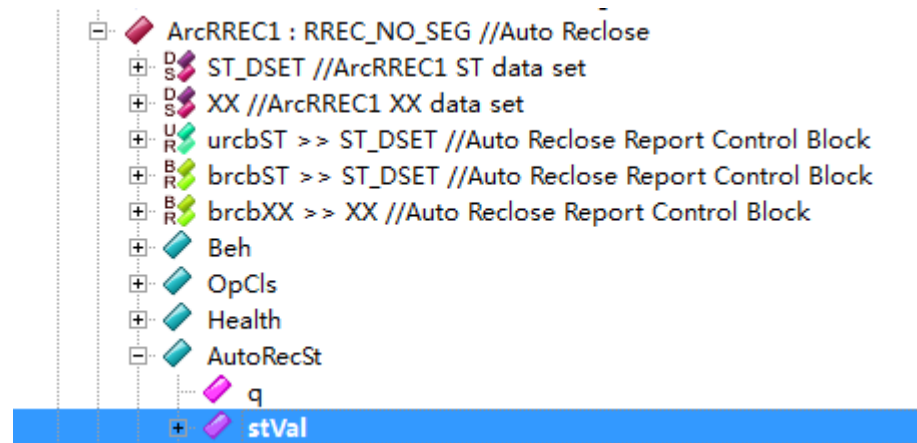
Note: Don't forget to save the project before closing the tool.

- Reopen the Project



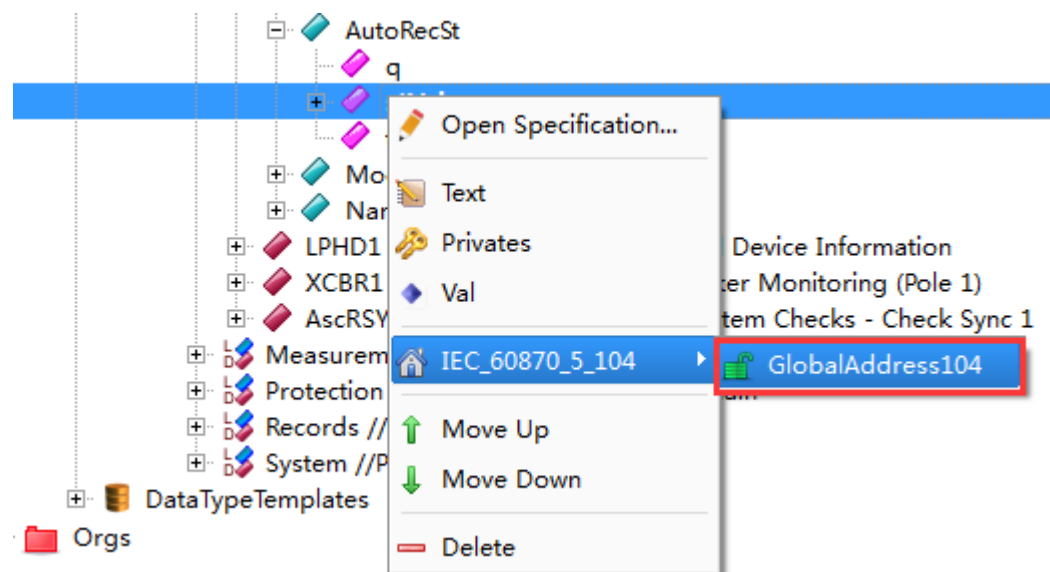
2.2.3. Create Private Element

- Select parent **Node** to add **Private Element**

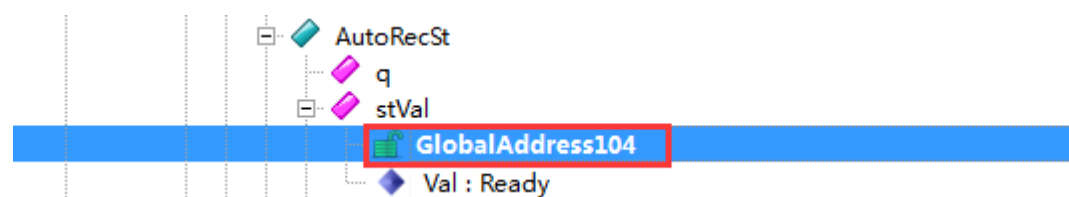


We select node **stVal** to add Private Element as model extension.

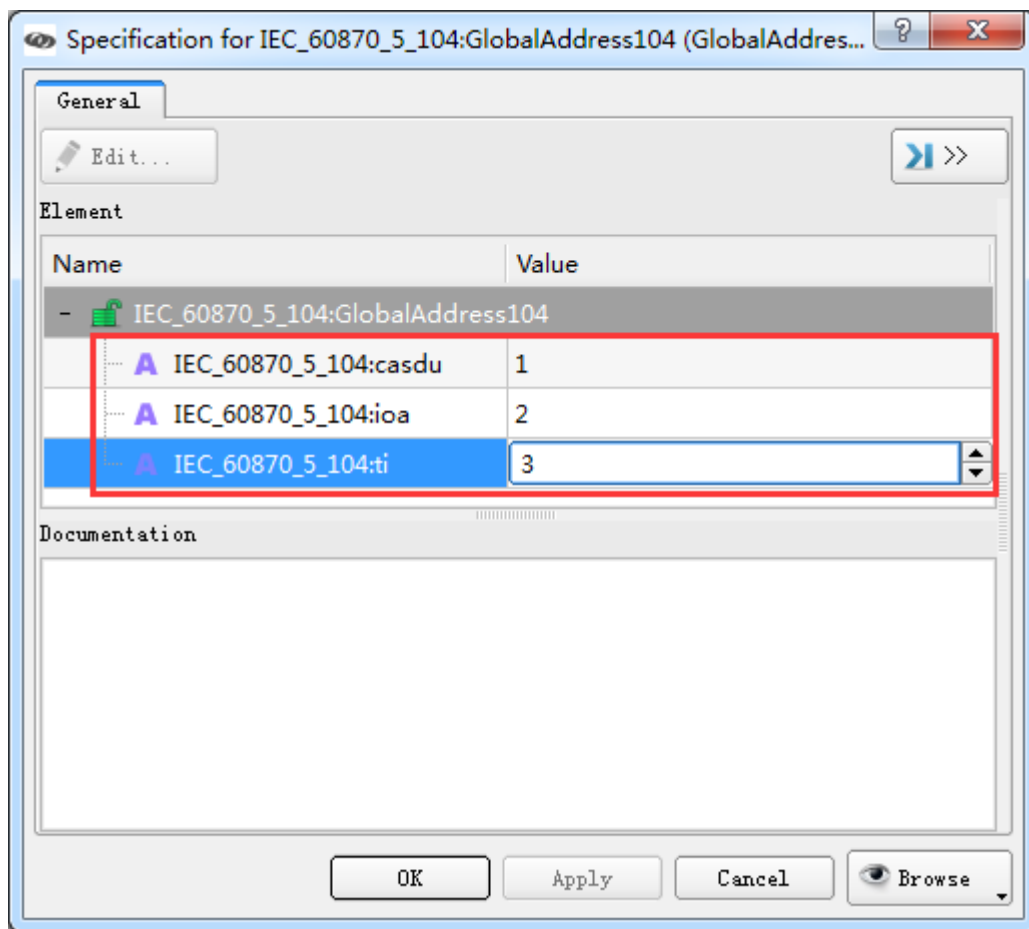
- Right click “**stVal**” to add private element



- Double click “GlobalAddress104”

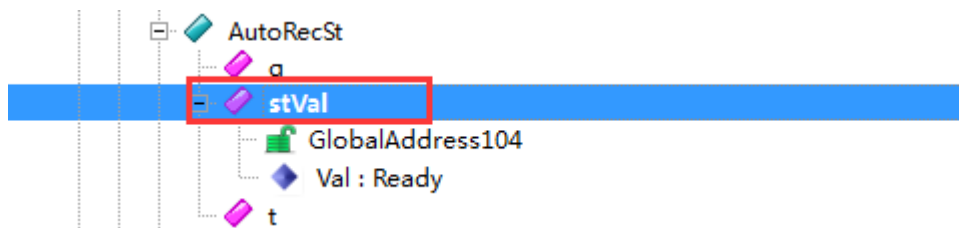


- Edit “GlobalAddress104”

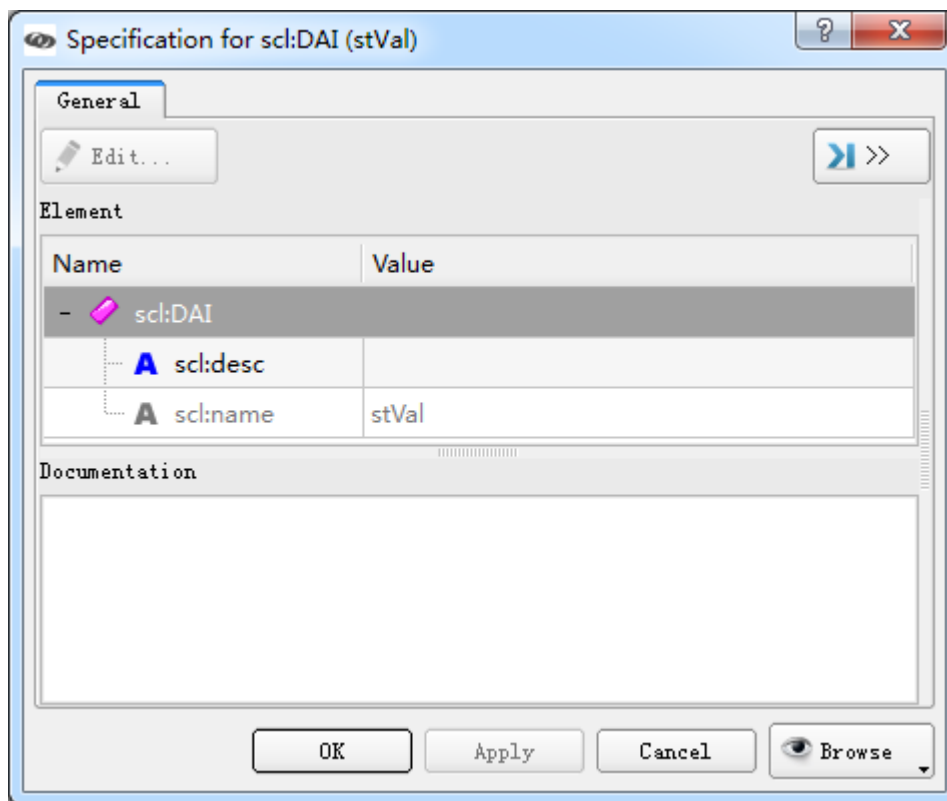


2.2.4. Create Private Attribute

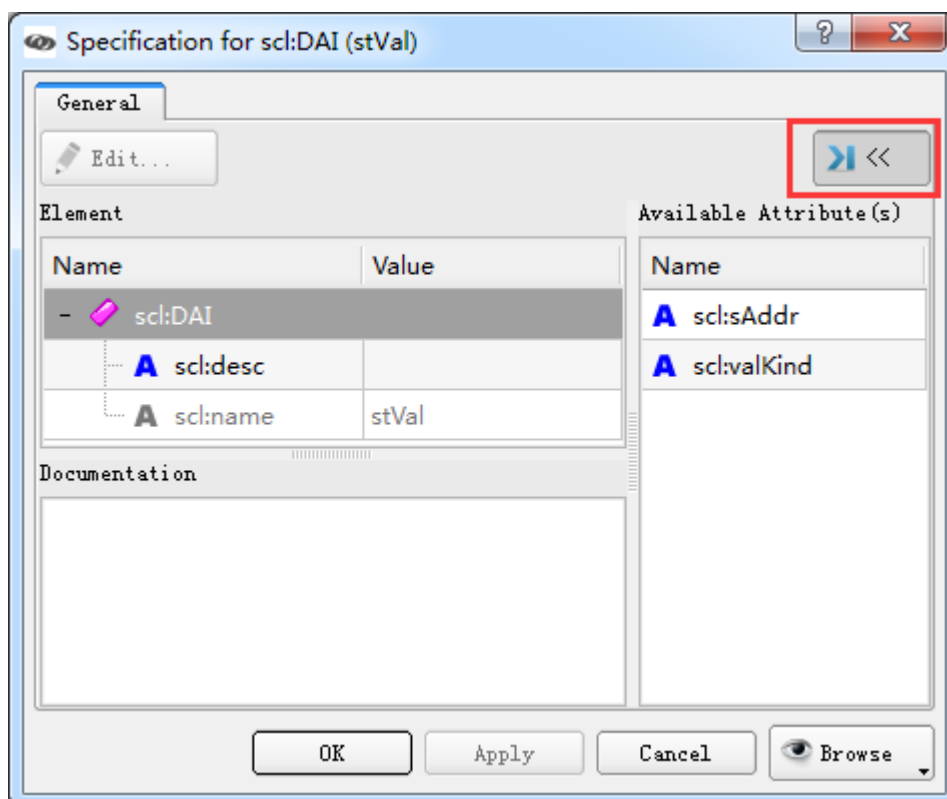
- Select parent Node to add Private Attributes



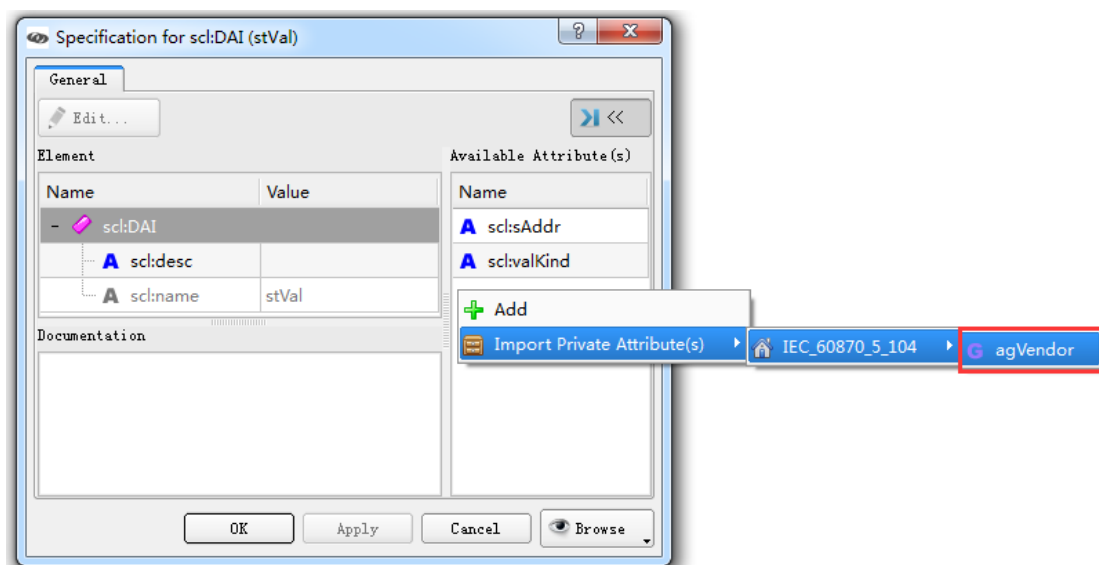
- Double click “stVal”



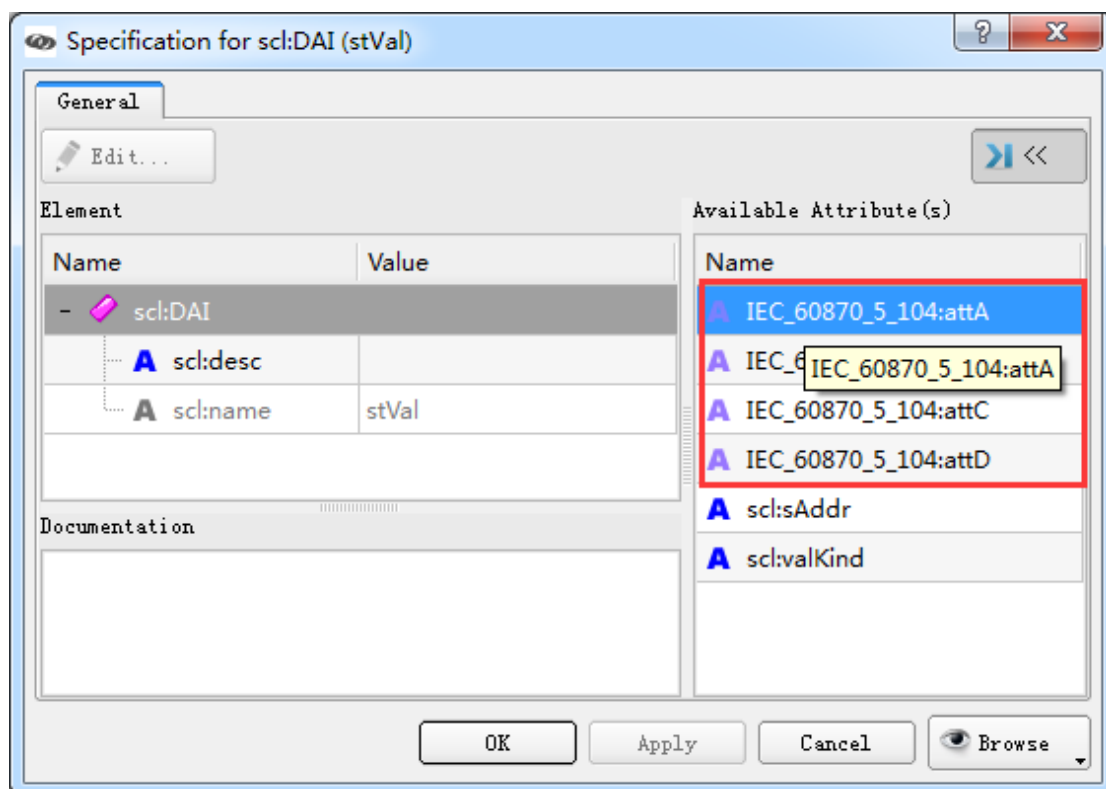
➤ Display “Attribute Box”



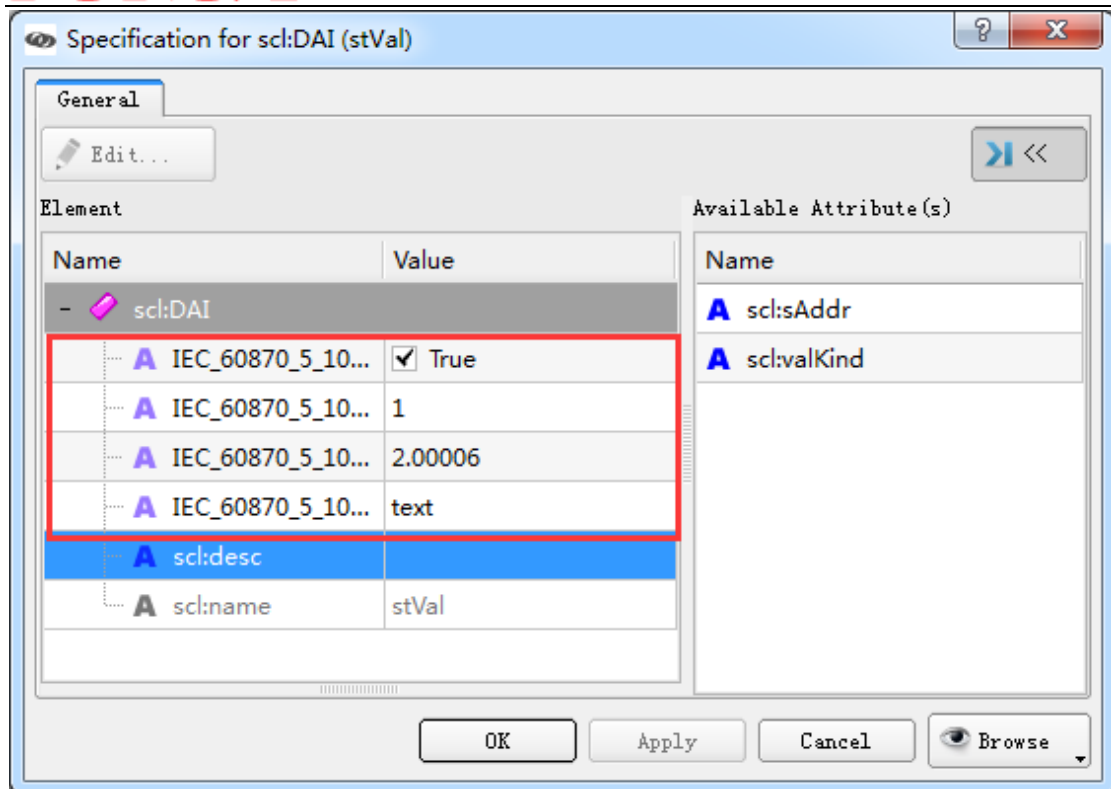
- Right click to import Private Attributes



- Add Private Attributes by Double click



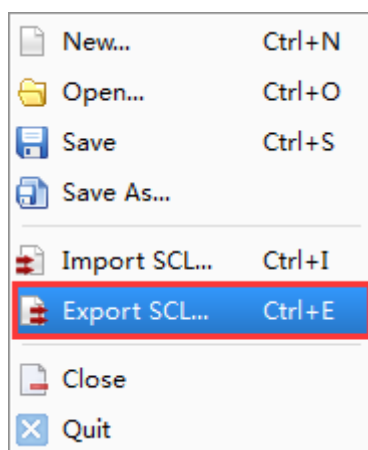
- Edit Private Attributes



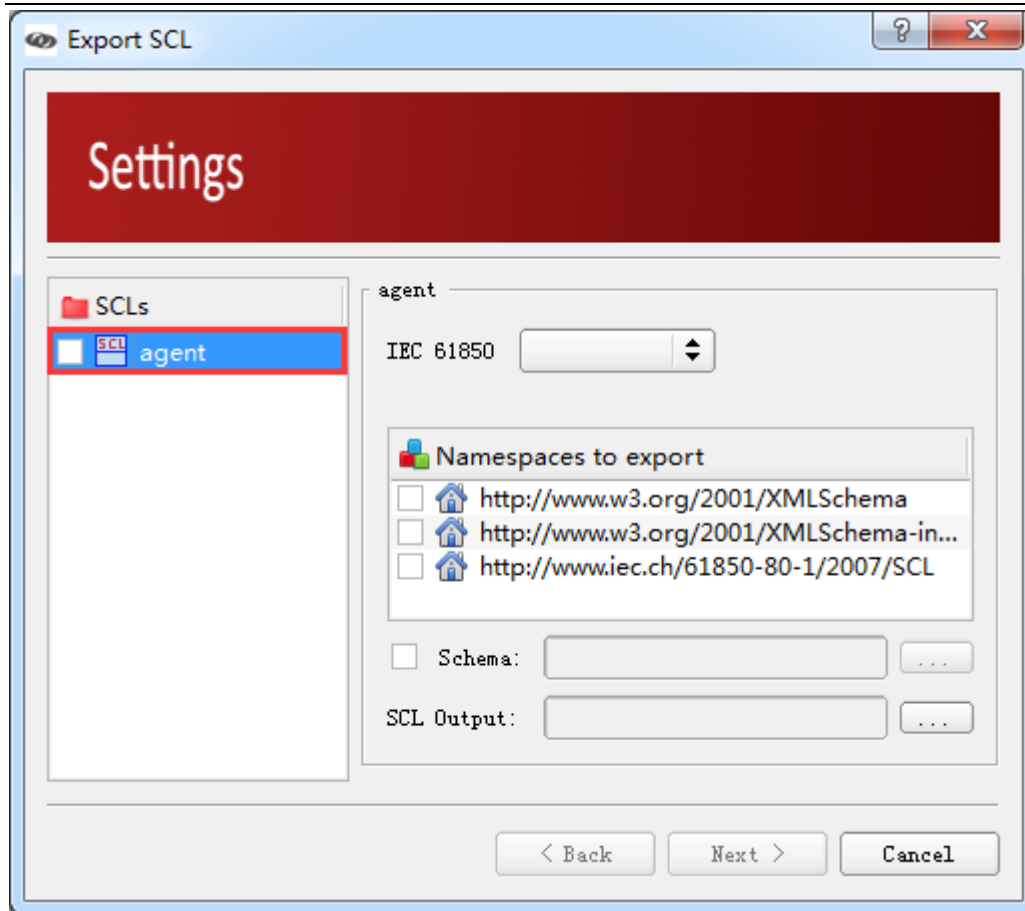
- Click OK to finish

2.2.5. Export SCL Model with Private Namespaces

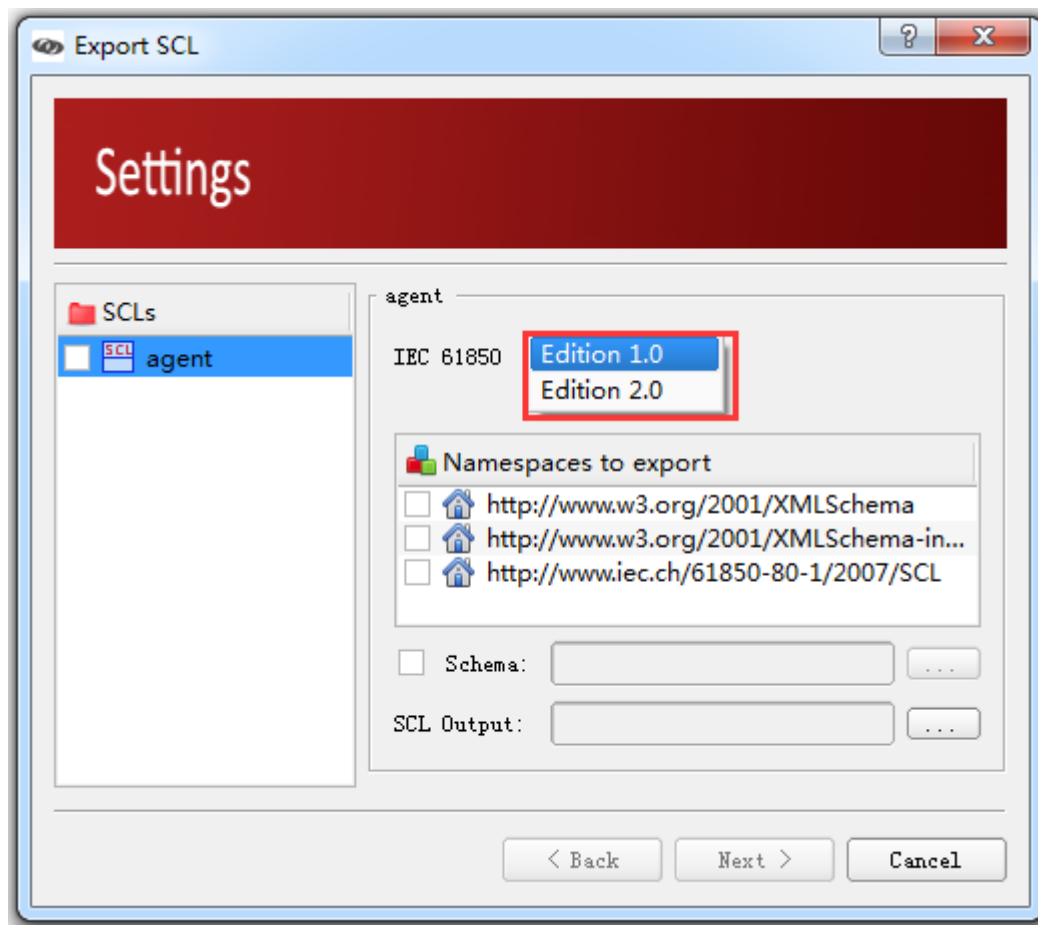
- Select **File->Export SCL...**



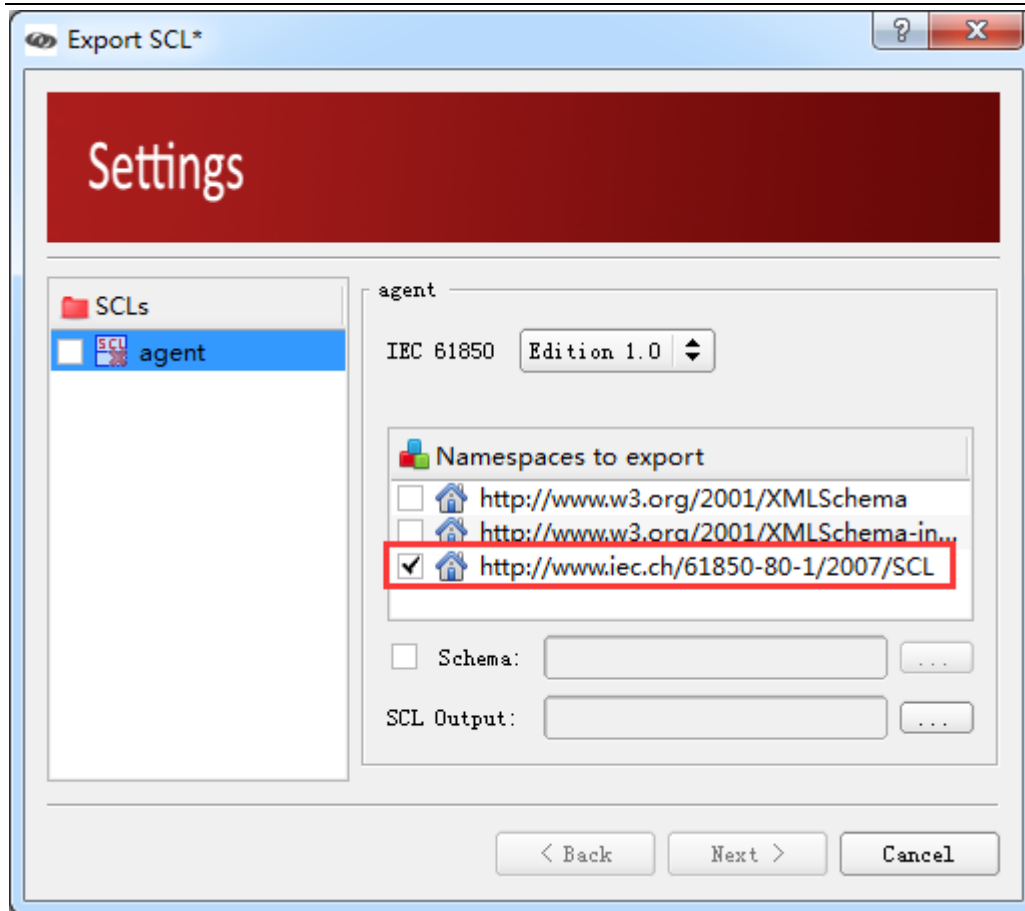
- Select the SCL to export



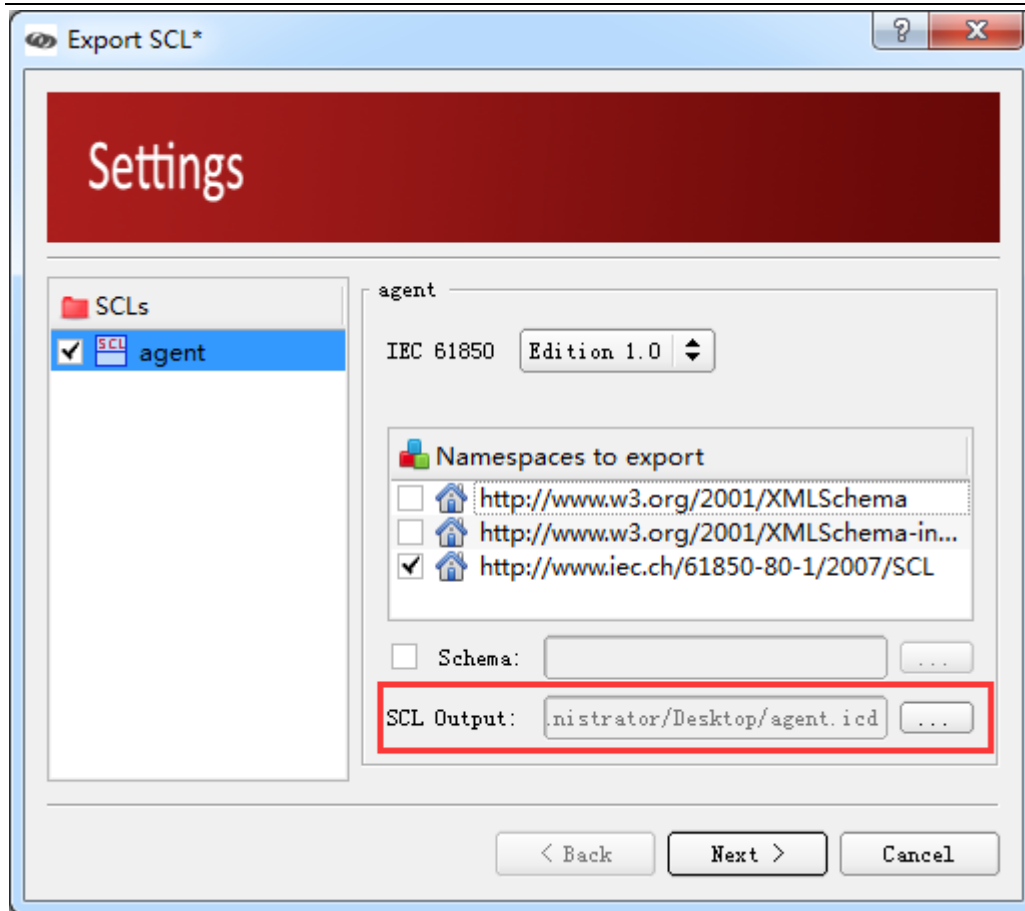
- Select the Edition option



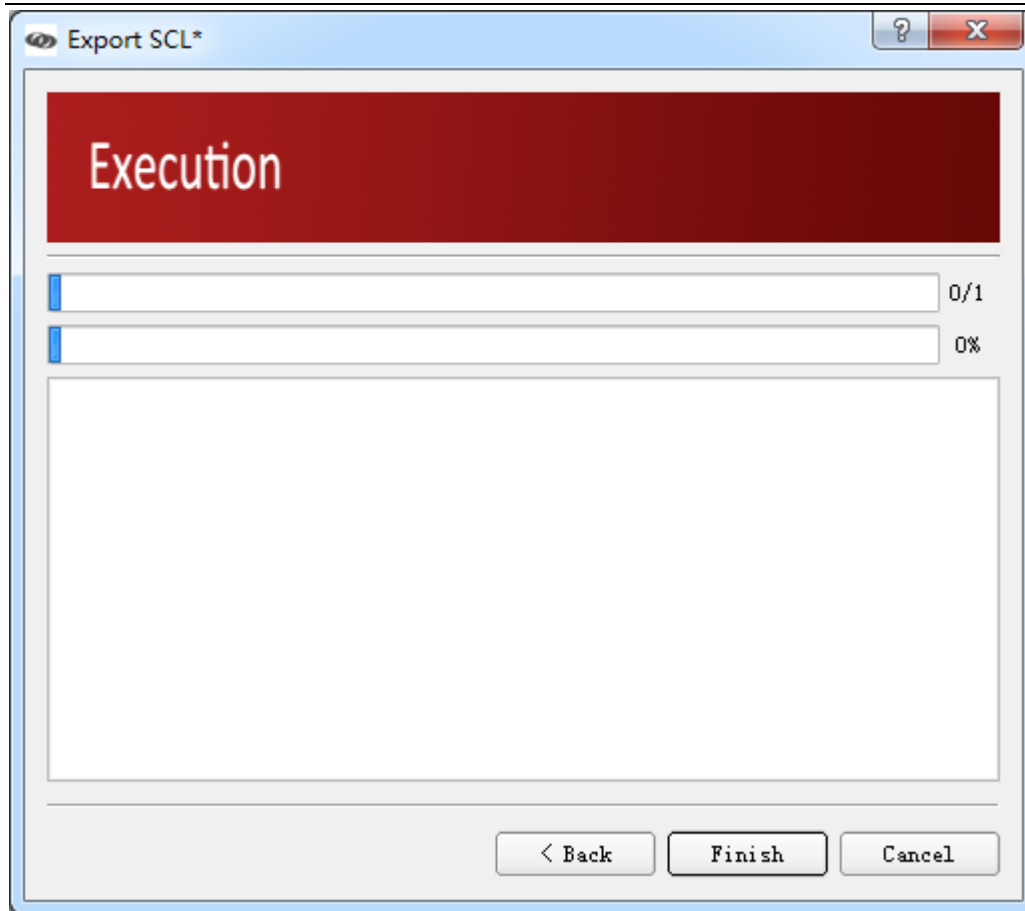
- Check the Private Namespaces to export



- Type in the output SCL file name



➤ Click Next



- Click Finish
- Overview of the Private Namespaces XML data

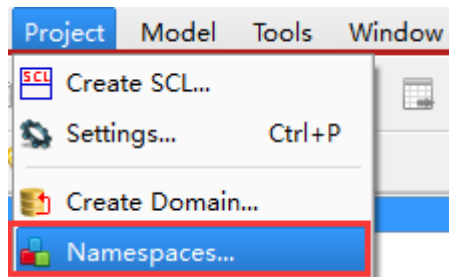
```

</DAI>
<DAI desc="" name="t"/>
</DOI>
<DOI desc="" name="AutoRecSt">
<DAI desc="" name="q"/>
<DAI IEC_60870_5_104:attA="true" IEC_60870_5_104:attB="1" IEC_60870_5_104:attC="2.00006" IEC_60870_5_104:attD="text" desc="" name="stVal">
<IEC_60870_5_104:GlobalAddress104 IEC_60870_5_104:casdu="1" IEC_60870_5_104:ioa="2" IEC_60870_5_104:ti="3"/>
</Val>Ready</Val>
</DAI>
<DAI desc="" name="t"/>

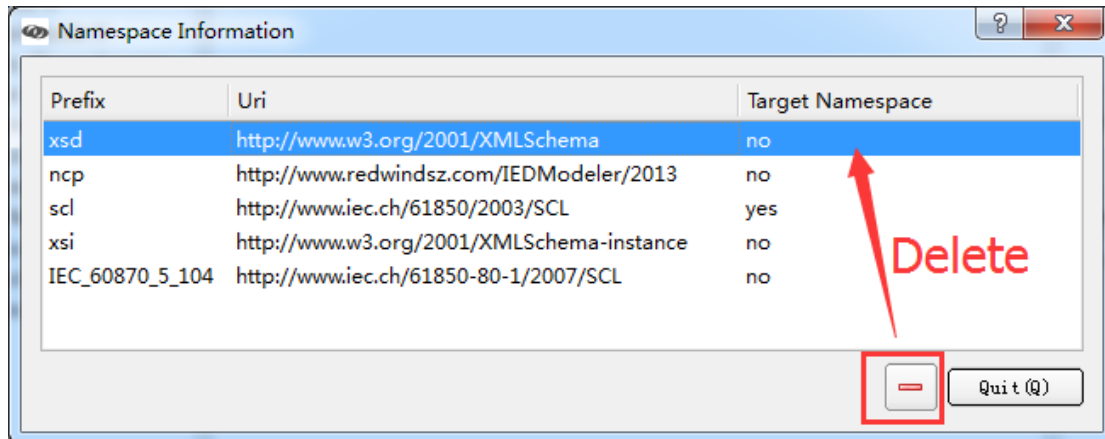
```

2.3 Project Namespaces Management

Choose **Project->Namespaces...**



Namespace Information dialog:



There are three columns in **Namespace Information**.

Prefix: Unique identifier representing an Uri

Uri: Uniform resource identifier (URI) is a namespace name

Target Namespace: Only one target namespace per project is allowed

Note:

- 1) Users cannot import **Private Namespaces** that are already used by project
- 2) Users cannot delete **Target Namespace** and preserved **Namespace** (denoted as ncp)
- 3) Elements from a deleted **Namespace** cannot be exported