Translation to Modelica

Contents

[Introduction 2](#_Toc76200313)

[Durable and Accountable File Translation Process 2](#_Toc76200314)

[Translation Workflow 3](#_Toc76200315)

[Modelica Components 3](#_Toc76200316)

[Component Naming 4](#_Toc76200317)

[XSLT Building Blocks 4](#_Toc76200318)

[Tap Changer 6](#_Toc76200319)

[Generator 7](#_Toc76200320)

[Tools 7](#_Toc76200321)

[Actions 8](#_Toc76200322)

[Associations 8](#_Toc76200323)

[Enumerations 9](#_Toc76200324)

[Selecting Particular Nodes 9](#_Toc76200325)

[WorkFlow 10](#_Toc76200326)

[Summary 11](#_Toc76200327)

[EQ Synchronous Machine 11](#_Toc76200328)

[NYPA\_500 11](#_Toc76200329)

[IEEE\_9bus 12](#_Toc76200330)

[Appendix 12](#_Toc76200331)

[XML 12](#_Toc76200332)

[Elements 12](#_Toc76200333)

[Tree Structure 13](#_Toc76200334)

[Attributes 13](#_Toc76200335)

[Namespace 13](#_Toc76200336)

[XSLT 13](#_Toc76200337)

[Template Rules 13](#_Toc76200338)

[Local Variable 14](#_Toc76200339)

[Global Variable 14](#_Toc76200340)

[Function 14](#_Toc76200341)

[Looping Through Entities 14](#_Toc76200342)

[CGMES Triples for 500-bus System 15](#_Toc76200343)

[Equipment Resource 15](#_Toc76200344)

[Dynamic Resource 15](#_Toc76200345)

[Editix Example 16](#_Toc76200346)

# Introduction

Researchers and educators perform power system dynamic analysis such as phasor simulations using the Open-Instance Power System Library (OpenIPSL) which is a library of power system component models written in the [Modelica](http://modelica.org) language. Professor [Luigi Vanfretti's](https://github.com/lvanfretti) research group [ALSETLab](https://github.com/ALSETLab) at [Rensselaer Polytechnic Institute](http://rpi.edu), Troy, NY, collaborators and friends, such as [Dietmar Winkler](https://github.com/dietmarw) and [FOSSEE](https://om.fossee.in/fellowship2018) currently develop and main OpenIPSL.

Modelica's ability to create one model by combining many types of systems allows extendability for researchers not available with other modeling software such as [PSS/E](https://new.siemens.com/global/en/products/energy/energy-automation-and-smart-grid/pss-software/pss-e.html) and [PSAT](https://www.mathworks.com/matlabcentral/answers/91079-power-system-analysis-toolbox-psat). Also, its ability to "dra and drop" pre-built components into the model is more intuitive for students. Model creation and modification is simple in OpenIPSL.

Model validation compares what the person knows to be true and the new model. Translations will make Modelica universal for modeling. Many researchers have data from PSS/E and PSAT. Therefore, moving these known models to Modelica is a good starting point. This paper will focus on moving a known PSS/E model into OpenIPSL and comparing the results.

First, a PSS/E tool will export the model parameters into a XML format. Then, XSLT translates XML to a Modelica file. Finally, we will compare results. XSLT is an extensible-style sheet language for transformations specifically for mapping XML to other formats. This paper will provide a durable and accountable file translation process and translation workflow.

# Durable and Accountable File Translation Process

The direct route of translating proprietary files from PSS/E or PSAT to Modelica is not the best way since the process is transient. Software vendor's file formats can be changed at any moment without notification or accountability.

The best way to translate is to use a certified file exchange – a contract that requires complete accounting of model parameters. This begins with a hierarchical set of standards that describes how model parameters are stored. Certified standards also allow any software vendor to participate, and the model data is ubiquitous.

The members of the European Network of Transmission System Operators ([ENTSO-E](https://www.entsoe.eu/)) are presently the predominant creators of certified standards. Their work within the International Electrotechnical Commission (IEC) Technical Committee (TC) 57 and [Work Group (WG) 13](https://www.iec.ch/dyn/www/f?p=103:14:0::::FSP_ORG_ID:2392) allows comment and adjustment from other countries and creates an open process. Their proposed standards are documented in the Common Grid Model Exchange Standard ([CGMES](https://cgmes.github.io/)). The implementation of the standard can be certified through conformance testing by UCA International Users Group ([UCAIug](https://www.ucaiug.org/default.aspx)). CGMES provides the parameter descriptions within the Common Information Model ([CIM](https://www.dmtf.org/standards/cim)) ; therefore, we will use it as our beginning.

CGMES is a set of standards that begins with XML. Extensible markup language (XML) is a standardized data structure like HTML without pre-defined tags. [CIM](https://www.dmtf.org/standards/cim) refines the XML standard by creating and defining tags. XSLT is an XML document; therefore, you must understand the basics of XML to differentiate between the documents that depend on it.

As you move from XML to CGMES CIM, you move from a generic xml data structure to CGMES power systems groupings. PSS/E adopts most if not all groupings from CGMES. OpenIPSL uses the same CGMES parameters within their components. Therefore, it is a direct map from the CGMES CIM to Modelica OpenIPSL components.

The Resource Descriptor Framework (RDF) from W3C specifications is part of CGMES. [RDF](https://en.wikipedia.org/wiki/Resource_Description_Framework) uses statements about resources in expressions of the form subject-predicate-object called triples. The subject denotes the resource, for example Synchronous Machine. The predicate denotes traits, for example Synchronous Machine q percent representing the coordinated reactive control in percent. The object denotes an aspect of the resource, for example 100 percent.

The translation moves CGMES triples into Modelica parameters. While RDF can represent more complicated data structures, CGMES uses something like relational database associations. Therefore, we'll use the rules and techniques for relational databases to transform with XSLT.

# Translation Workflow

The [CGMES equipment (EQ from IEC 61970-452) and dynamics (DY from IEC 61970-302) files](https://www.entsoe.eu/digital/cim/cim-for-grid-models-exchange/) translate to a single MO files using OneMO.xsl (called OneMO). The result is a model like the pssetomo.py translation used from OpenIPSL examples.

The xsl-files copy the triple to the Modelica parameter. The translation must copy the required Modelica parameters. The CGMES subjects align with UML or relational models. The major result is that one table does not duplicate the data in another. The PSS/E separates files into static and dynamic triples as specified in ENTSO-E models. Modelica uses parameters by component.

## Modelica Components

Power system equipment is modeled with Modelica components. A 14-bus system which has been used as example of a typical power system can be analyzed during faults or other dynamic events and is composed of the following Modelica components:

Component partial record record

OpenIPSL.Electrical

**Buses.Bus** Bus\_Template PF\_Bus\_00000

Branches

**PwLine**

**PSSE.TwoWindingTransformer** Trafos\_Template PF\_Trafos\_00000

**Loads.PSSE.Load** Loads\_Template PF\_Loads\_00000

**Banks.PSSE.Shunt**

**System.Generators** Machines\_Template PF\_Machines\_00000

The above table identifies how OpenIPSL has chosen to simplify use of measurements. The ‘record’ column, a command within Modelica use to group variables, are the raw measurements and can be quickly swapped out for new measurements. The ‘partial record’ column, also a command within Modelica groups the meta data of the record such as the name used within the model definitions.

Measured values from bus, transformers, loads, and machines are defined with the EQ (equipment) file, but these are placeholders or initial values. Measurements are meant to be used out of the SV (state variable) file and can be rapidly switched out with the record structure.

### Component Naming

Component names must be unique. This can be accomplished simply for line (PwLine) and shunts (Banks.PSSE.Shunt) because the name is the CIM ‘IdentifiedObject.name’ that are unique to each. If it is used with the record structure, the unique name should also group the parameters. For example, all buses have a voltage and angle parameter and VBUS1 and ABUS1 would be grouped by BUS1. Transformers have terminal 1 (T1) and terminal 2 (T2) and are named 470\_1\_T1 and 470\_1\_T2 for example.

## XSLT Building Blocks

The OneMO translates the EQ-file and then creates template or function calls to the DY-file. The core building blocks are as follows:

**Output** <xsl:output indent="yes" method="text"/> sets the output of the translation to text.

**Key** <xsl:key match="cim:ConnectivityNode" name="node" use="@rdf:ID"/> uses associations within the CIM. In this case, it associates terminals with their respective components. Terminals are the Modelica bus equivalent.

**Variable** <xsl:variable name="dynamic" select="document('examples\bus-14\ieee14\_DY.xml')"/> assigns a tree to a variable.

**Function** <xsl:function as="xs:string" name="gkh:compliantName">

<xsl:param as="xs:string" name="input"/>takes individual parameters and returns an individual result.

**Template** <xsl:template name="gkh:equipmentLookup"> takes an xml-tree and returns several different types of results including another tree, individual parameters, or anything in-between. This is a push-style, reaching into the xml tree-structure, of programming. You can recursively enter a tree.

**For-each** <xsl:for-each select="cim:BusbarSection"> is very similar to the Template in what it takes and the results. This is a pull-style. It is like traditional languages.

Templates and For-each both can return multiple children of the main feature. If only and individual child is needed, then value-of is used, for example:  
<xsl:value-of select="gkh:defaultNumbers(cim:BasePower/cim:BasePower.basePower,100000000)"/>

Global Variables, Functions, and Templates

Output: text

**Variable:**

DY – dynamic file

TP – topology file

SV – state variable file

baseVoltages

**Key:** cim namespace

ac section – AC Line Segment

base power – Base Power

bus bar – Busbar Section

bus bar equip – BusBarSection/Equipment.EquipmentContainer

conform load – Conform Load

node – Connectivity Node

execsystem – Excitation System user Defined/ExcitationSystemDynamics.SynchronousMachineDynamics

governor – Gov Steam 0/TurbineGovernorDynamics.SynchronousMachineDynamics

shunt – Linear Shunt Compensator

non conform load – Non Conform Load

power transformer – Power Transformer

PTend - Power Transformer End/PowerTransformerEnd.PowerTransformer

ppdynamics – Proprietary Parameter Dynamics/ProprietaryParameterDynamics.ExcitationSystemUser…

pssieee2b – PssIEEE2B/PowerSystemStabilizerDynamics.ExcitationSystemDynamics

ratio tap changer – Ratio Tap Changer

tap changer – Ratio Tap Changer/RatioTapChanger.TransformerEnd

regulator – Regulating Control

sync machine – Synchronous Machine

terminal – Terminal

terminal equip – Terminal/Terminal.ConductingEquipment

voltage level – Voltage Level

**Function:** gkh namespace

compliantName(input as string?)

baseVoltage(code as string)

defaultNumbers(input as decimal?, default as decimal)

substring-after-last-match(arg as string?, regex as string)

**Template:**

gkh namespace

connectivityNode-equipmentContainer(eCcode as string)

equipmentLookup(prefix as string, keyCode as string)

cim namespace - located at the top

InitialVoltage(code as string, name as string, base as decimal)

InitialPower(code as string, name as string)

MakeMachineForCode(code as string, GenName as string) 🡪 selects dynamic parameters

EQ file

RatioTapChanger(position as integer)

PowerTransformerEnd/PowerTransformerEnd.PowerTransformer(Name as string)

RatioTapChanger/RatioTapChanger.TransformerEnd

SynchronousMachine 🡪 selects equipment name

SynchronousMachineSimplified 🡪 Modelica GENCLS

DY file

SynchronousMachineTimeConstantReactance (MainName as string) 🡪 Modelica GENROU and GENSAL

GovSteam0/TurbineGovernorDynamics.SynchronousMachineDynamics

ExcitationSystemUserDefined/ExcitationSystemDynamics.SynchronousMachineDynamics

PssIEEE2B/PowerSystemStabilizerDynamics.ExcitationSystemDynamics

ProprietaryParameterDynamics/ProprietaryParameterDynamics.ExcitationSystemUserDefined

Main Program

template /rdf:RDF

for-each – cim namespace

BusBarSection 🡪 Modelica OpenIPSL.Electrical.Buses.Bus

ACLineSegment 🡪 Modelica OpenIPSL.Electrical.Branches.PwLine

PowerTransformer 🡪 Modelica OpenIPSL.Electrical.Branches.PSSE.TwoWindingTransformer

LinearShuntCompensator 🡪 Modelica OpenIPSL.Electrical.Banks.PSSE.Shunt

ConformLoad 🡪 Modelica OpenIPSL.Electrical.Loads.PSSE.Load

SynchronousMachine 🡪 Modelica CimSystem.Generators section

Terminal 🡪 Modelica connect section

Template – gkh:equipmentLookup

Busbar

Powertrans

Acsection

Conformload

Shunt

Syncmachine

nonconformload

apply-template SynchronousMachine

for-each

TP file Topological Node 🡪 Modelica Bus\_Template

TP file Topological Node 🡪 Modelica PF\_Bus\_00000

Power Transformer End 🡪 Modelica Trafos\_Template

Power Transformer End 🡪 Modelica PF\_Trafos\_00000

TP/TopologicalNode – bus connections

TP/TopologicalNode – bus voltages

SV/SvPowerFlow – load flow

## Tap Changer

neutralU – Voltage at which the winding operates at the neutral tap setting.

neutralStep – The neutral tap step position for this winding.

normalStep – The tap step position used in “normal” network operation for this winding.

highStep – Highest possible tap step position, advance from neutral.

lowStep – Lowest possible tap step position, retard from neutral.

Since OpenIPSL uses the per unit voltage at each terminal of the transformer an equation is used:

voltage = neutralU + (normalStep – neutralStep) \* RatioTapChanger.stepVoltageIncrement/100

voltage = 68.777 + (16 – 16) \* 0.64725/100

## Generator

The equipment PSS/E file (ending in EQ) contains static triples. The dynamic PSS/E file (ending in DY) contains dynamic triples. Let us work through the OpenIPSL PSSE GENROU Machine (Solid Rotor Generator represented by equal mutual inductance rotor modeling). Table 1 is from the DY file.

Table 1. Map from Modelica OpenIPSL GENROU parameters to PSS/E predicate for the dynamic triples of resource 'SynchronousMachineTimeConstantReactance'. The predicates used in this table have a prefix with the same name as the resource or 'RotatingMachineDynamics'. A period separates the prefix from the tables PSS/E predicate name. For example, the table shows 'RotatingMachineDynamics.damping' is 'damping'.

|  |  |
| --- | --- |
| **Modelica parameter description (name) unit** | **PSS/E Predicate** |
| d-axis transient open-circuit time constant (Tpd0) s | tpdo |
| d-axis sub-transient open-circuit time constant (Tppd0) s | tppdo |
| q-axis sub-transient open-circuit time constant (Tppq0) s | tppqo |
| Inertia constant (H) s | inertia |
| Speed damping (D) | damping |
| d-axis reactance (Xd) | xDirectSync |
| q-axis reactance (Xq) | xQuadSync |
| d-axis transient reactance (Xpd) | xDirectTrans |
| d-axis sub-transient reactance (Xppd) | xDirectSubtrans |
| q-axis sub-transient reactance (Xppq) |  |
| leakage reactance (Xl) | statorLeakageReactance |
| Saturation factor at 1.0 pu (S10) | saturationFactor |
| Saturation factor at 1.2 pu (S12) | saturationFactor120 |
| Armature resistance (R\_a) |  |
| q-axis transient reactance (Xpq) | xQuadTrans |
| q-axis transient open-circuit time constant (Tpq0) | tpqo |
| Sub-transient reactance (Xpp) | equal to Xppd by default. |

Finally, there is only one initialization: Initial speed deviation from nominal (w0). The overall model provides any other parameters or initializations.

Modelica parameters compared with CIM triples

Modelica OpenIPSL Directory Path: Examples->Machines->PSSE->GENROU

CIM File File and Element: …\_DY.xml

Top Element: cim:SynchronousMachineTimeConstantReactance

Sub-Element: cim:SynchronousMachineTimeConstantReactance.modelType is equal to

<http://iec.ch/TC57/2013/CIM-schema-cim16#RotorKind.roundRotor>

# Tools

An API engine translates XSLT. Common programming languages such as .NET or Python have their own API for XSLT version 1.0. The file size of the 500 bus CIM model has forced the use of XSLT version 2.0. For XSLT (and XML) development, JAPISOFT provides an IDE called [EditiX](https://www.editix.com/). It uses the [Saxonica](https://www.saxonica.com) engine. Ultimately creation of a .NET interface by using the home edition API downloaded from <http://saxon.sourceforge.net/> .

The XSL documents are a list of rules that copy information from the XML document to various output styles. We will use GenMO to operate on the CIM EQ file directly and the CIM DY file with a coded command. Here, the following method identifies a text file for the output :

*<xsl:output indent="no" method="text" />*

*MO files are text files.*

# Actions

## Associations

The rdf:ID attribute is used like a primary key. Use Foreign keys to associate data in another table. Foreign keys use the same identifier with a prefix '#' in front of the primary key. This is created with the following statement:

*concat('#',@rdf:ID)*

In CompleteMO a simpler technique of associating tables is used. A global key is assigned to a particular primary key table:

*<xsl:key name="node" match="cim:ConnectivityNode" use="@rdf:ID"/>*

The foreign key table can then link to the primary key table with the **key()** function and use a single property within it:

*<xsl:value-of select="normalize-space(key('node',cim:Terminal.ConnectivityNode/substring(@rdf:resource,2))/cim:IdentifiedObject.name)"/>*In this case the property is **cim:IdentifiedObject.name**.

When multiple properties are required, a function the passes in the primary key should be passed as shown in the Function section.

<xsl:key name="node" match="cim:ConnectivityNode" use="@rdf:ID"/>

|  |  |  |  |
| --- | --- | --- | --- |
|  | **name** | **match** | **use (default: @rdf:ID)** |
|  | node | cim:ConnectivityNode |  |
|  | busbar | cim:BusbarSection |  |
|  | terminal | cim:Terminal |  |
|  | powertrans | cim:PowerTransformer |  |
|  | acsection | cim:ACLineSegment |  |
|  | conformload | cim:ConformLoad |  |
|  | nonconformload | cim:NonConformLoad |  |
|  | syncmachine | cim:SynchronousMachine |  |
|  | basepower | cim:BasePower |  |
|  | shunt | cim:LinearShuntCompensator |  |
|  | PTend | cim:PowerTransformerEnd/ cim:PowerTransformerEnd. PowerTransformer | substring(@rdf:resource,2) |
|  | basevoltage | cim:BaseVoltage |  |
|  | voltagelevel | cim:VoltageLevel |  |
|  | tapchanger | cim:RatioTapChanger/ cim:RatioTapChanger. TransformerEnd | substring(@rdf:resource,2) |
|  | regulator | cim:RegulatingControl |  |

## Enumerations

Different actions are performed with enumerations because in this case they are different types of models and send different parameters to the MO file. The code below tests the equivalency of the rdf:resource attbribute with a particular enumeration and changes the text accordingly:

*<xsl:if test= "cim:SynchronousMachineTimeConstantReactance.rotorType/@rdf:resource='http://iec.ch/TC57/2013/CIM-schema-cim16#RotorKind.roundRotor'">*

*<xsl:text>GENROU</xsl:text>*

*</xsl:if>*

*<xsl:if test= "cim:SynchronousMachineTimeConstantReactance.rotorType/@rdf:resource='http://iec.ch/TC57/2013/CIM-schema-cim16#RotorKind.salientPole'">*

*<xsl:text>GENSAL</xsl:text>*

*</xsl:if>*

## Selecting Particular Nodes

The global variable **dynamic** connects to the CIM DY file. Then, like path information, the table is set with the following statement:

/rdf:RDF/cim:SynchronousMachineTimeConstantReactance/cim:SynchronousMachineDynamics.SynchronousMachine[@rdf:resource=$code]/..

The path leads down the tree, starting with the document node **rdf:RDF**, followed by the table **cim:SynchronousMachineTimeConstantReactance** and ending in the property **cim:SynchronousMachineDynamics.Synchronousmachine**. The attribute data **@rdf:resource**:

*<cim:SynchronousMachineDynamics.SynchronousMachine/@rdf:resource="#\_703939f0-9562-11e7-9e89-b46d83638f70" />*

requires the variable **$code** to be equal. In this way, it steps through each unique machine. The **$code** is the ID:

<cim:SynchronousMachineTimeConstantReactance rdf:ID="\_87e144f5-9562-11e7-9e89-b46d83638f70">

By adding '/..' to the end the top node will be the table node. The total function call code is:

*<xsl:apply-templates select= "$dynamic/rdf:RDF/cim:SynchronousMachineTimeConstantReactance/cim:SynchronousMachineDynamics.SynchronousMachine[@rdf:resource=$code]/.." >*

*<xsl:with-param name="MainName" select="$GenName" />*

*</xsl:apply-templates>*

GS0

TGD.SMD

ESUD

ESD.SMD

SMTCR

SMD.SM @rdf:resource

SM @rdf:ID

EQ file (equipment)

**SM** cim:SynchronousMachine rdf:ID

DY file (dynamic)

**SMD.SM** cim:SynchronousMachineDynamics.SynchronousMachine @rdf:resource

**SMTCR** cim:SynchronousMachineTimeConstantReactance @rdf:ID

**TGD.SMD** cim:TurbineGovernorDynamics.SynchronousMachineDynamics @rdf:resource

**GS0** cim:GovSteam0 rdf:ID

**ESD.SMD** cim:ExcitationSystemDynamics.SynchronousMachineDynamics rdf:resource

**ESUD** cim:ExcitationSystemUserDefined @rdf:resource

DY file

rdf:RDF

cim:SynchronousMachineTimeConstantReactance

cim:SynchronousMachineDynamics.Synchronousmachine

**@rdf:resource = $code**

EQ File

rdf:RDF

cim:SynchronousMachineTimeConstantReactance

WorkFlow

Code entry begins with the first template:

*<xsl:template match="/rdf:RDF" >*

*<xsl:apply-templates select="cim:SynchronousMachine" />*

*</xsl:template>*

Opening the second template that creates the machine names:

<xsl:template match="cim:SynchronousMachine">

…

<xsl:call-template name="MakeMachineForCode">…

</xsl:template>

Opening the third template to gather all the dynamic parameters:

*<xsl:template name="MakeMachineForCode" >*

*…*

*<xsl:apply-templates select= "$dynamic/rdf:RDF/cim:SynchronousMachineTimeConstantReactance/cim:SynchronousMachineDynamics.SynchronousMachine[@rdf:resource=$code]/.." >*

*<xsl:with-param name="MainName" select="$GenName" />*

*</xsl:apply-templates>*…

</xsl:template>

Opening the fourth template to gather all the static parameters:

*<xsl:template match="cim:SynchronousMachineTimeConstantReactance" >*

*…*

*</xsl:template>*

# Summary

Understanding ENTSO-E CIM in terms of database modeling techniques simplifies the paradigm for translating the CIM to Modelica. This paper relates the XSLT commands to data modeling structures which can systematically be used to map CIM to Modelica.

Many of these database modeling techniques could be used in something other than a XSLT such as Python or .NET. However, most languages support XSLT version 1.0 and can use it with existing commands with their own XSLT engine.

Rapid translation of ENTSO-E CIM will provide system planners the advantages of Modelica in their work.

# EQ Synchronous Machine

## NYPA\_500

<cim:SynchronousMachine.maxQ>9.97</cim:SynchronousMachine.maxQ>

<cim:SynchronousMachine.minQ>-7.58</cim:SynchronousMachine.minQ>

<cim:SynchronousMachine.qPercent>100</cim:SynchronousMachine.qPercent>

<cim:SynchronousMachine.type rdf:resource="http://iec.ch/TC57/2013/CIM-schema-cim16#SynchronousMachineKind.generator" />

<cim:RotatingMachine.GeneratingUnit rdf:resource="#\_49e4f9f1-6d10-11eb-a65a-74e5f963e191" />

<cim:RotatingMachine.ratedS>28</cim:RotatingMachine.ratedS>

<cim:RegulatingCondEq.RegulatingControl rdf:resource="#\_49e4f9f4-6d10-11eb-a65a-74e5f963e191" />

<cim:Equipment.EquipmentContainer rdf:resource="#\_49b54964-6d10-11eb-a65a-74e5f963e191" />

<cim:IdentifiedObject.description>108894 'Z '</cim:IdentifiedObject.description>

<cim:IdentifiedObject.name>108894\_Z </cim:IdentifiedObject.name>

## IEEE\_9bus

<cim:SynchronousMachine.maxQ>250</cim:SynchronousMachine.maxQ>

<cim:SynchronousMachine.maxU>17.16</cim:SynchronousMachine.maxU>

<cim:SynchronousMachine.minQ>-250</cim:SynchronousMachine.minQ>

<cim:SynchronousMachine.minU>17.16</cim:SynchronousMachine.minU>

<cim:SynchronousMachine.qPercent>100</cim:SynchronousMachine.qPercent>

<cim:SynchronousMachine.r>9.9E-05</cim:SynchronousMachine.r>

<cim:SynchronousMachine.type rdf:resource="http://iec.ch/TC57/2013/CIM-schema-cim16#SynchronousMachineKind.generator" />

<pti:SynchronousMachine.x>0.099</pti:SynchronousMachine.x>

<cim:RotatingMachine.GeneratingUnit rdf:resource="#\_703939ec-9562-11e7-9e89-b46d83638f70" />

<cim:RotatingMachine.ratedS>275</cim:RotatingMachine.ratedS>

<cim:RegulatingCondEq.RegulatingControl rdf:resource="#\_703939ef-9562-11e7-9e89-b46d83638f70" />

<cim:Equipment.EquipmentContainer rdf:resource="#\_7039395a-9562-11e7-9e89-b46d83638f70" />

<cim:IdentifiedObject.description>1 'G1'</cim:IdentifiedObject.description>

<cim:IdentifiedObject.name>1\_G1</cim:IdentifiedObject.name>

# Appendix

## XML

The website <https://www.w3schools.com/xml/> provides more extensive tutorial on XML. The structure is human and machine readable. We will start with CGMES defined equipment and dynamics files which have names that end in EQ and DY, respectively.

### Elements

XML Elements provide the basic structure of the XML. Opening ('<') and closing('>') tags surround an element. The closing tag has a '/'. Empty tags are elements that contain nothing, and they do not require closing tags. They have a '/' just before the '>'. For example,

*<lily age="13">This text is not here!</lily>* is the same as *<lily age="13" />* when removing 'This text is not here!' (an empty tag).

CGMES files are trees of elements. The elements form triples. The complete document is the triple's subject, which is the equipment or the dynamics files. We will look at the equipment file first. The triple's synchronous machine predicate starts with an element named "SynchronousMachine" and looks like:

<cim:SynchronousMachine …

The triple's object looks like:

<cim:SynchronousMachine.maxQ …

We will cover "cim" later within the namespace section.

### Tree Structure

According to the W3C tutorial, "XML documents form a tree structure that starts at 'the root' and branches to 'the leaves'. When looking at XML in a tree view, which is called the document object model (DOM), there is a single root node which contains all other nodes. '/rdf:RDF' is the root node of all CIM file types.

After the document element /rdf:RDF, a limited tree of elements is only two layers deep within the constraints of CGMES. The top element is comparable to a table within a database. The second layer is comparable to properties within each table. These constraints simplify the XML and make the well-known design rules from relational databases easier to use the CIM.

### Attributes

Elements can contain attributes. The only attributes used in ENTSOE CIM is an ID or resource, which are closely related, and enumerations. The ID/resource attributes are a database structure used to associate two tables of information using the primary key and foreign key links. Here, one main node from the CIM EQ file used for storing static properties:

*<cim:SynchronousMachineTimeConstantReactance rdf:ID="\_d712e95b-25db-11e7-afca-b46d83638f70">*

Here, a property within main node GovHydro1 links to the synchronous machine static node:

*<cim:TurbineGovernorDynamics.SynchronousMachineDynamics rdf:resource="#\_d712e95b-25db-11e7-afca-b46d83638f70" />*

ENTSO-E defines enumerations that identify different synchronous machine model types in this example:

*<cim:SynchronousMachineTimeConstantReactance.rotorType rdf:resource="http://iec.ch/TC57/2013/CIM-schema-cim16#RotorKind.salientPole" />*

### Namespace

Namespaces provide a way to avoid name collisions when two XML elements have the same name. In XSLT '<xsl:stylesheet xmlns:rdf…' identifies a 'rdf' namespaces near the beginning of the document. Then, 'rdf:' differentiates RDF from other uses of RDF:

*<xsl:template match="/rdf:RDF" >*

'cim', 'entsoe', 'pti', 'md' are other common namespaces within ENTSO-E CIM.

## XSLT

### Template Rules

In GenMO, after the document node, the first table is 'cim:SynchronousMachine'. The following method and command **xsl:apply-template** within the xsl element:

*<xsl:template match="/rdf:RDF" >*

*<xsl:apply-templates select="cim:SynchronousMachine" />*

*</xsl:template>*

Jumps to the corresponding scope limiting element **xsl:template** via because select = match:

<xsl:template match="cim:SynchronousMachine">

The match is not always evident.

### Local Variable

**xsl:variable** creates a variable named 'GName' that creates a text followed by a property called **cim:IdentifiedObject.name**.

*<xsl:variable name="GName">*

*<xsl:text>SM</xsl:text>*

*<xsl:value-of select="cim:IdentifiedObject.name" />*

*</xsl:variable>*For example, GName='SM3000\_1'.

The first piece of output is text generated from:

*<xsl:text>model </xsl:text>*Anything between the **xsl:text** tags goes directly to the output and is followed by a copy of the variable GName. The total output is: model SM3000\_1

### Global Variable

A global variable is a variable that is placed prior to first xsl:Template tag. At the top of the file, find the statement:

*<xsl:variable name="dynamic" select="document('IEEE\_9bus\_DY.xml')" />*

This sets the variable **dynamic** equal to the CIM DY file.

### Function

**xsl:call-template** is the equivalent of a function. The following code uses parameters **code** and **GenName** within the calling function:

*<xsl:call-template name="MakeMachineForCode">*

*<xsl:with-param name="code" select="concat('#',@rdf:ID)" />*

*<xsl:with-param name="GenName" select="$GName" />*

*</xsl:call-template>*

The function is:

*<xsl:template name="MakeMachineForCode" >*

*<xsl:param name="code" />*

*<xsl:param name="GenName"/>*

### Looping Through Entities

**For-each** loops are more traditional methods of coding the **template-apply** and are used in these circumstances:

*<xsl:for-each select="cim:BusbarSection">*

*<xsl:for-each select="cim:ACLineSegment"><xsl:for-each select="cim:ConformLoad"><xsl:for-each select="cim:PowerTransformer">*

This is a pull versus push style (template-apply) of programming. Some believe push is less efficient.

## CGMES Triples for 500-bus System

### Equipment Resource

The following predicates for readability:

AC Line Segment

Base Voltage

Busbar Section

Conform Load

Conform Load Group

Connectivity Node

Control Area

Current Limit

Geographical Region

Hydro Generating Unit

Linear Shunt Compensator

Load Area

Load Response Characteristic

Non-Conform Load Group

Operational Limit Set

Operational Limit Type

Phase Tap Changer Table

Phase Tap Changer Table Point

Phase Tap Changer Tabular

Power Transformer

Power Transformer End

Ratio Tap Changer

Ratio Tap Changer Table Point

Regulating Control

Sub Geographical Region

Sub Load Area

Substation

Synchronous Machine

Tap Changer Control

Terminal

Voltage Level

Voltage Limit

Wind Generating Unit

### Dynamic Resource

Exc AC1A

Exc AC2A

Exc IEEE AC1A

Exc IEEE AC2A

Exc IEEE DC1A

Exc IEEE ST1A

Exc IEEE ST4B

Excitation System User Defined (IEEE T1)

Exc SEXS

Gov GAST

Gov Hydro1

Gov Steam0

Gov Steam SGO

Proprietary Parameter Dynamics

Pss IEEE 2B

Synchronous Machine Simplified

Synchronous Machine Time Constant Reactance

## Editix Example

I recommend the book *XSLT: Mastering XML Transformations*, 2nd Edition which covers XSLT 2.0 for a more complete coverage. Purchase a used version from Amazon for about $6 including shipping.

Editix is an Integrated Development Environment (IDE) for XML and XSLT. It develops the XSLT code for translation.

IEEE\_9bus\_EQ.xml

IEEE\_9bus\_DY.xml

OneMO.xsl

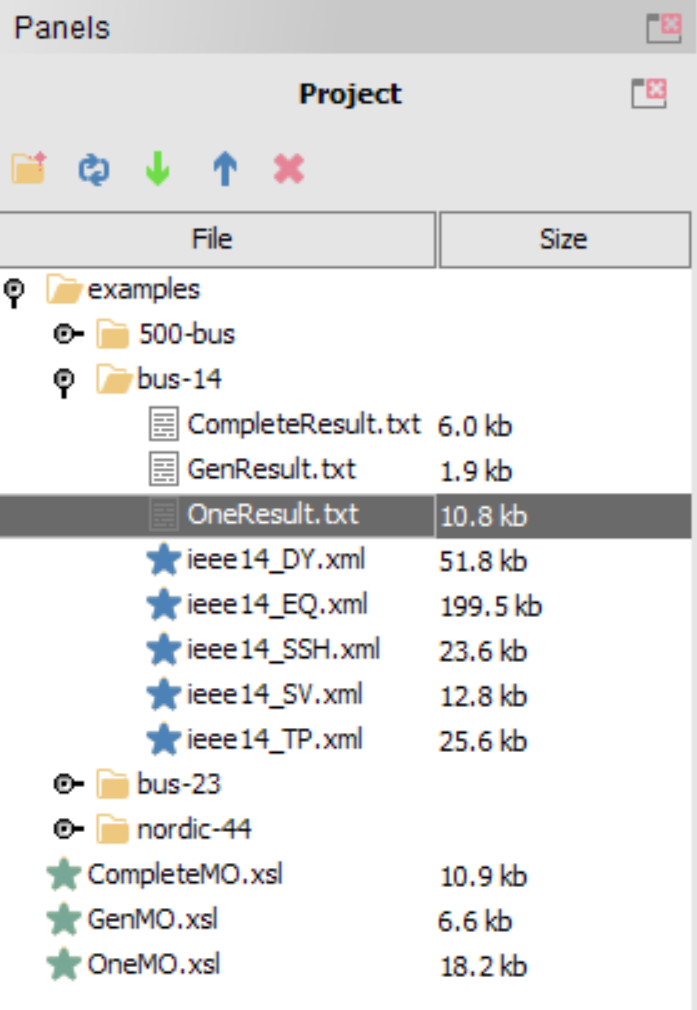
Result.txt

SM1\_G1.mo

SM2\_G2.mo

SM3\_G3.mo

IEEE\_9bus.mo

1. File -> Open Project -> NYPAModelTransformation\ModelTransf-Tool\Prototype\src\cim  
    (<https://www.editix.com/doc/manual19/index.html#mozTocId804917> )  
   
2. Double-click on 'OneMO.xsl'
3. XSLT/XQuery -> Transform A Document  
   