UNIVERSITY OF MARYLAND
COLLEGE PARK

SYSTEMS ENGINEERING VALIDATION AND VERIFICATION
ENPM643

TERM PROJECT: Ontology-Enabled Validation of Systems containing Electric Terminals

FINAL REPORT

PROFESSOR: MARK AUSTIN
STUDENT: FRANCISCO J GALLO

DEC/6/2005

TABLE OF CONTENTS

1. DESCRIPTION OF PROJECT

The primary goal of this project is to develop a tool that allows the validation of connections between electrical terminals.

Secondary goals of this project are:

1) Define a framework for defining and identifying the different elements needed to interconnect components in a system. This framework is approached at a case study level and can be extended as necessary.
2) Using the framework, create an ontology that implements key aspects of it and that allows to validate the use of components included in the ontology.
3) Implement a tool that is capable of validate connectivity of electrical terminals by using the hierarchical schema defined in the ontology.
4) Use XML as a data-storage mechanism to validate formal data and to move it between applications.
5) Gain understanding of tool Protégé.

2. INTRODUCTORY REMARKS

Work done in this project was based on papers by [Mayank] and [Liang] and on reference materials provided by Professor Mark Austin. This project was strongly based on the following technologies:

1. Stanford's Protégé
2. XML
3. XSLT
4. XPATH
5. VBA
6. Excel

3. FINAL DELIVERABLES OBTAINED FROM THIS PROJECT

1. **Ontology Validation Tool**: Implemented in VBA and Excel XP. This tool is capable of importing XML data containing instances of a predefined ontology and then, allows the user to test connectivity between elements. This tool displays detailed information about compatible/incompatible attributes and works seamlessly with XML generate by Protégé. The ontology validation tool has been implemented as a macro in file "OntologyValidationTool_V.N.n.xls".
2. **Ontology in Protégé**: A Protégé project containing an ontology for electrical ports has been fully implemented. This ontology is based on the rules and knowledge gained by analyzing a Home Theater System. The Protégé project has been include in file "PortOntology.pprj".
3. **Electrical Terminal Ontology Specification**: Document (see file "Annex01_ElectricTerminalsOntologySpecification")) containing the analysis process done on a Home Theater System. This analysis provided the framework to implement the ontology in Protégé.

Other achievements done as result of this project:

1. Analyzed the XML hierarchy that Protégé creates. Documented this hierarchy in file "Annex04_ProtegeXMLSchemaAnalysis.xls" for future reference.

2. Created XSLT template using XPATH to provide style to XML generated by Protégé allowing seamless integration with XML/XSLT capable environments.

4. PROJECT IMPLEMENTATION DESCRIPTION

4.1. STEP: ANALYSIS OF BASE SYTEM AND CREATION OF BASIC RULES FOR ONTOLOGY

Due to the complexity of this step, it was decided to include it as an Annex. Please refer to file "Annex01_ElectricTerminalsOntologySpecification_v.0.2.doc" for details on how this was done.

Please note that some Annex 1 provides some important definitions that will be used from now on. It is strongly recommended to read it before continuing with present document.

4.2. STEP: IMPLEMENTATION OF ONTOLOGY IN PROTÉGÉ

Through an iterative process using Protégé, the ontology needed to describe a set of electrical terminals was implemented. This ontology is in a Protégé project file.

The Electrical Terminals Ontology (ETO) implemented in Protégé is focused in the description of the Terminals needed for a set of ports to work properly. Bear in mind that a Port is a composition of Terminals.

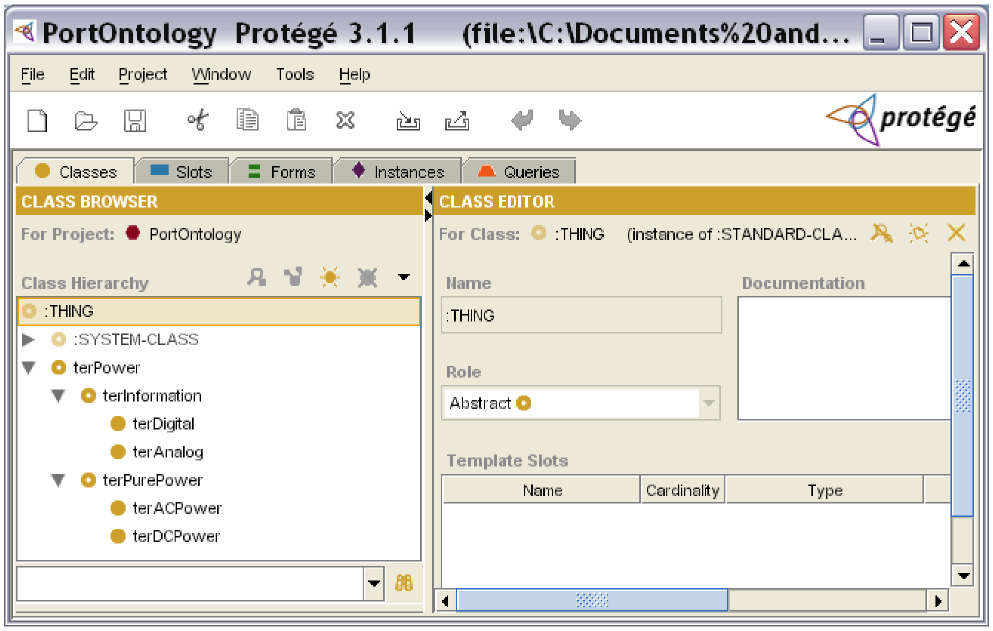
Concepts used to categorize the classes and defining their attributes are based on Liang [2]. These are: function, behavior and form.

Function has been used to define the top-level classes based on the nature of the service they provide. For example: power or information transmission.

Behavior has been used to define the bottom-level classes based on how the service is implemented. For example, digital or analog signals. Also, behavior has been used to define certain attributes of the classes, such as the operating frequency, impedance and others.

Form has been provided as an attribute in the ontology. Since describing the form of a connector implies a complexity beyond the scope of this project, the abstract approach to shape specifications suggested by Liang [3] is used. This is, instead of providing the physical details of form, a certain standard name is defined (e.g. StandardRoundTerminal). Then, the description of a certain Terminal can be progressively elaborated in additional detail by extending what the abstract shape specification means (Round connectors, with two parallel blades separated x inches, etc.) The important concept here is that for two Terminals to work as a Port, they must have the same abstract shape specification and one has to be male and the other female.

Figure 4.1 shows a screenshot of Protégé and the ETO that was implemented. In the left pane, the hierarchy of classes that compose the ontology can be appreciated. Note that the fundamental concept of how classes have been grouped is the way the energy is used (Power or Information; Analogical or Digital).



For Project: PortOntology

:THING
:SYSTEM-CLASS
terPower
terInformation
terDigital
terAnalog
terPurePower
terACPower
terDCPower

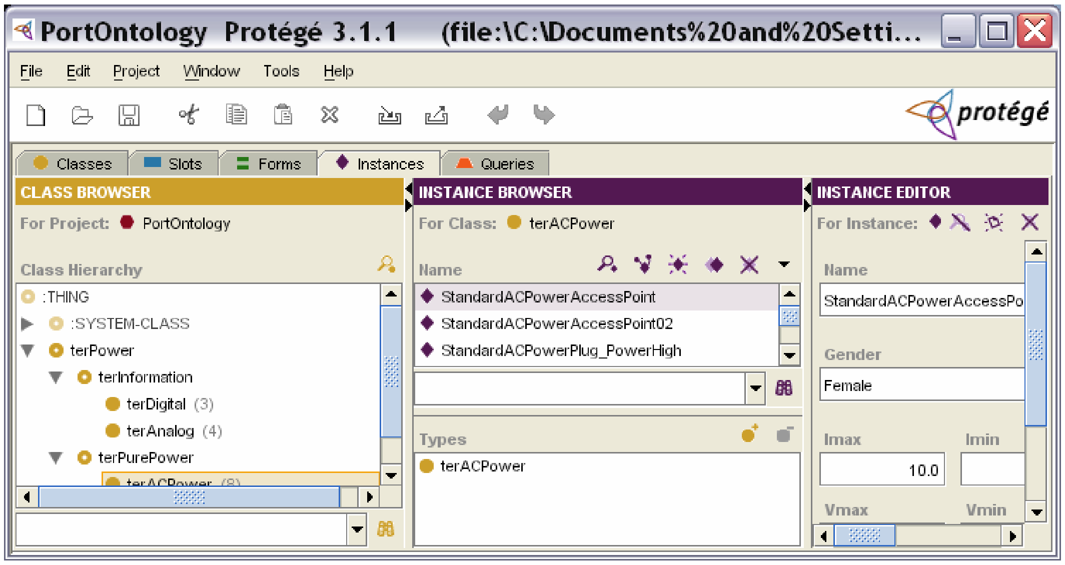
For Class: :THING (instance of :STANDARD-CLA...

Name: :THING

Role: Abstract

Figure 4.1

Figure 4.2 shows another view of Protégé in which the instances created using the ontology can be appreciated. Note that all the files created in Protégé have been submitted along with this final report.



For Project: PortOntology

:THING
:SYSTEM-CLASS
terPower
terInformation
terDigital (3)
terAnalog (4)
terPurePower
terACPower (8)

For Class: terACPower

StandardACPowerAccessPoint
StandardACPowerAccessPoint02
StandardACPowerPlug_PowerHigh

Types: terACPower

Name: StandardACPowerAccessPo

Gender: Female

Imax: 10.0

Figure 4.2

4.3. STEP: GENERATION OF XML FILES AND CREATION OF XSLT STYLE SHEETS

Once the ontology was created in Protégé, it can be exported into XML. To do this, the option shown in Figure 4.3 was selected in Protégé. This creates an XML file in the Protégé workspace folder for this project.

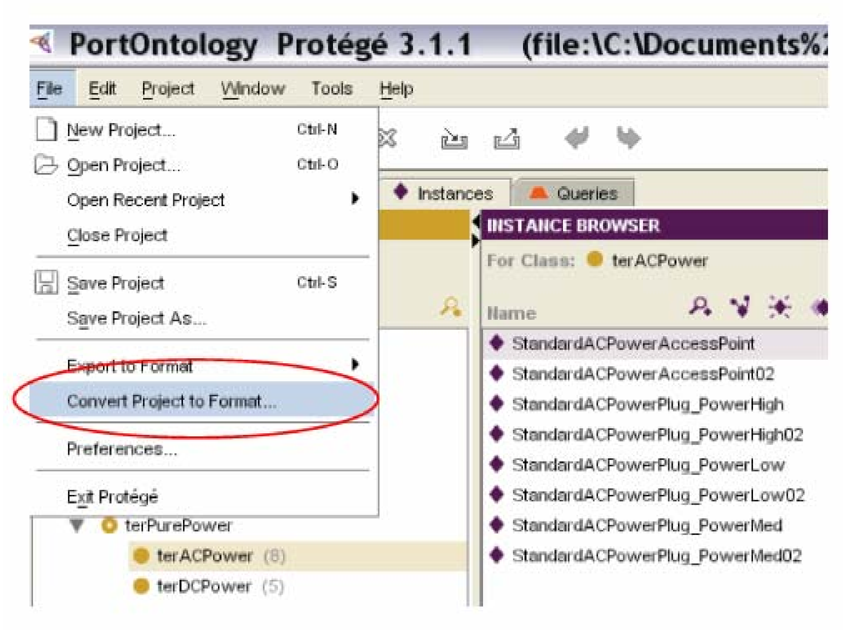

PortOntology Protégé 3.1.1 (file:\C:\Documents%
File Edit Project Window Tools Help
New Project... Ctrl-N
Open Project... Ctrl-O
Open Recent Project
Close Project
Save Project Ctrl-S
Save Project As...
Export to Format
Convert Project to Format...
Preferences...
Exit Protégé
Instances
Queries
INSTANCE BROWSER
For Class: terACPower
Name
StandardACPowerAccessPoint
StandardACPowerAccessPoint02
StandardACPowerPlug_PowerHigh
StandardACPowerPlug_PowerHigh02
StandardACPowerPlug_PowerLow
StandardACPowerPlug_PowerLow02
StandardACPowerPlug_PowerMed
StandardACPowerPlug_PowerMed02
terPurePower
terACPower (8)
terDCPower (5)

Figure 4.3

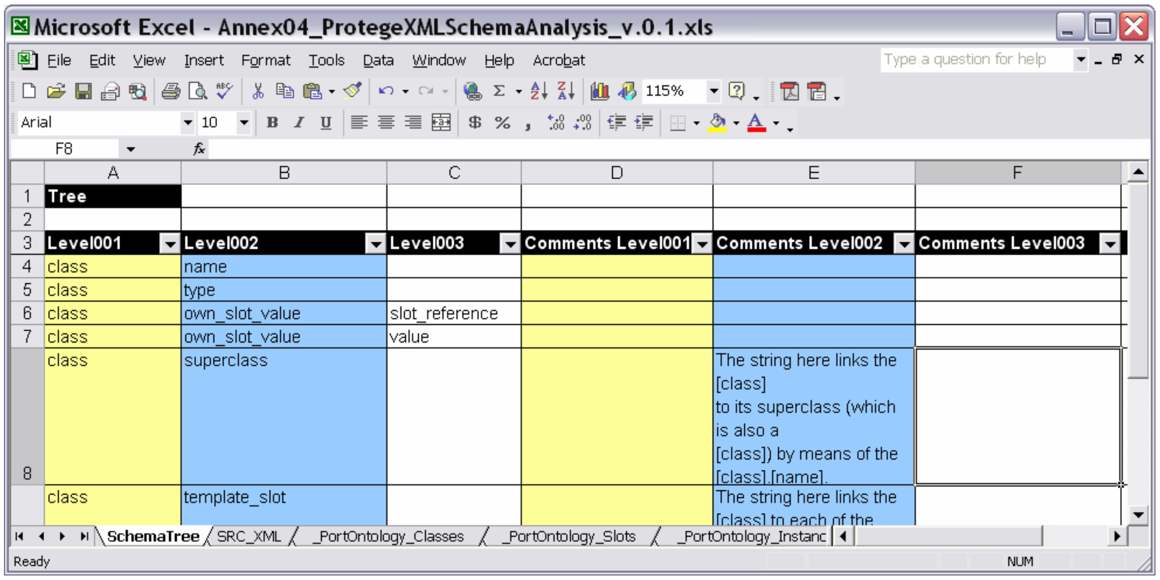
After doing this, the XML file has to be modified so it can be imported into Excel as a flat table where the hierarchy has been properly transformed. The steps to do this are explained in file "Annex05_StepsToExportImportXML_v.N.n.doc".

The modifications shown in such annex connect the XML file to the XSLT template that was implemented using XPATH. This XSLT template is in file "_PortOntology_Instances.xsl" in the folder where all the Protégé related folders have been included. Please note that file "PortOntology_Target-Instances.xml" is the final XML template that will be used to validate the connections. Files "PortOntology_Target-Classes.xml" and "PortOntology_Target-Slots.xml" are byproduct obtained during the reverse engineering process.

In order to create the XPATH code implemented in the XSLT file "_PortOntology_Instances.xsl" it was necessary to analyze the XML hierarchy created by Protégé. This analysis can be appreciated in file "Annex04_ProtegeXMLSchemaAnalysis_v.N.n.xls". A screenshot of the contents of this file has been included here for convenience, please see Figure 4.4. Note how the hierarchy is a three-tier schema in which the attributes for each instance are provided at the leaf level.

4.4. STEP: CREATION OF ONTOLOGY VALIDATION TOOL

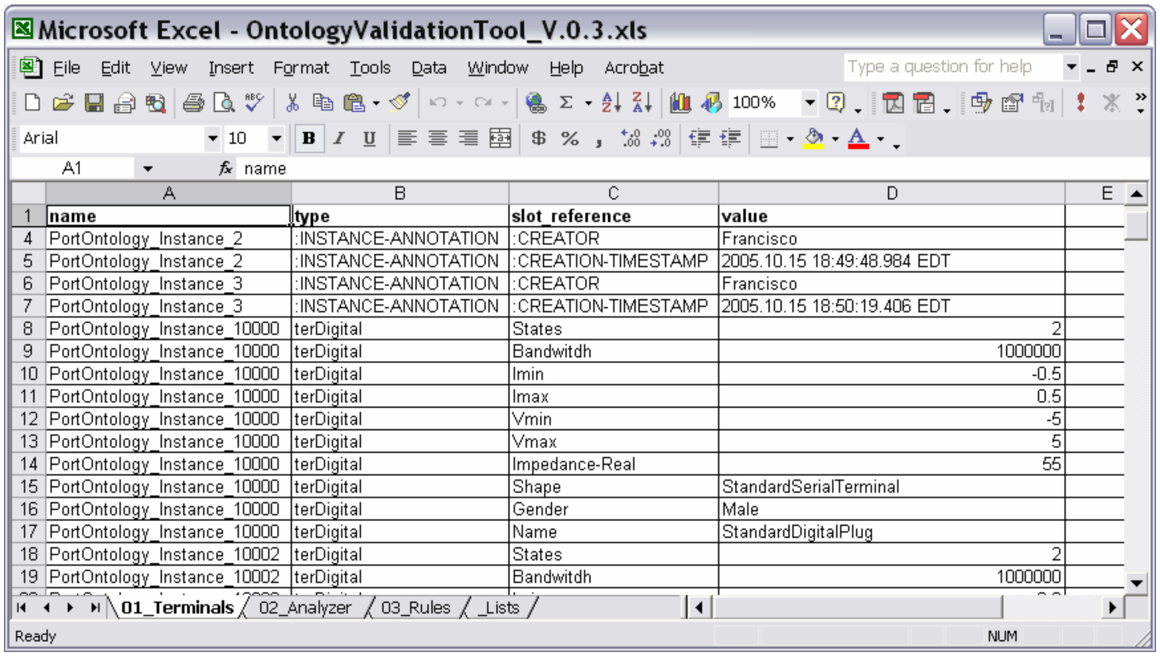
Using Excel and its built in XML capabilities, data in XML file from Protégé was imported into a tab. An analysis process of the data was done and a step-by-step tool was implemented. This tool in VBA is capable of retrieving all the instances from the XML data and then generates drop-down menus that the user can utilize to validate connectivity of terminals. Figure 4.5 shows the data imported to Excel by applying the XSLT template.



Tree

Level001	Level002	Level003	Comments Level001	Comments Level002	Comments Level003
class	name				
class	type				
class	own_slot_value	slot_reference			
class	own_slot_value	value			
class	superclass			The string here links the [class] to its superclass (which is also a [class]) by means of the [class].[name].	
class	template_slot			The string here links the [class] to each of the	

Figure 4.4

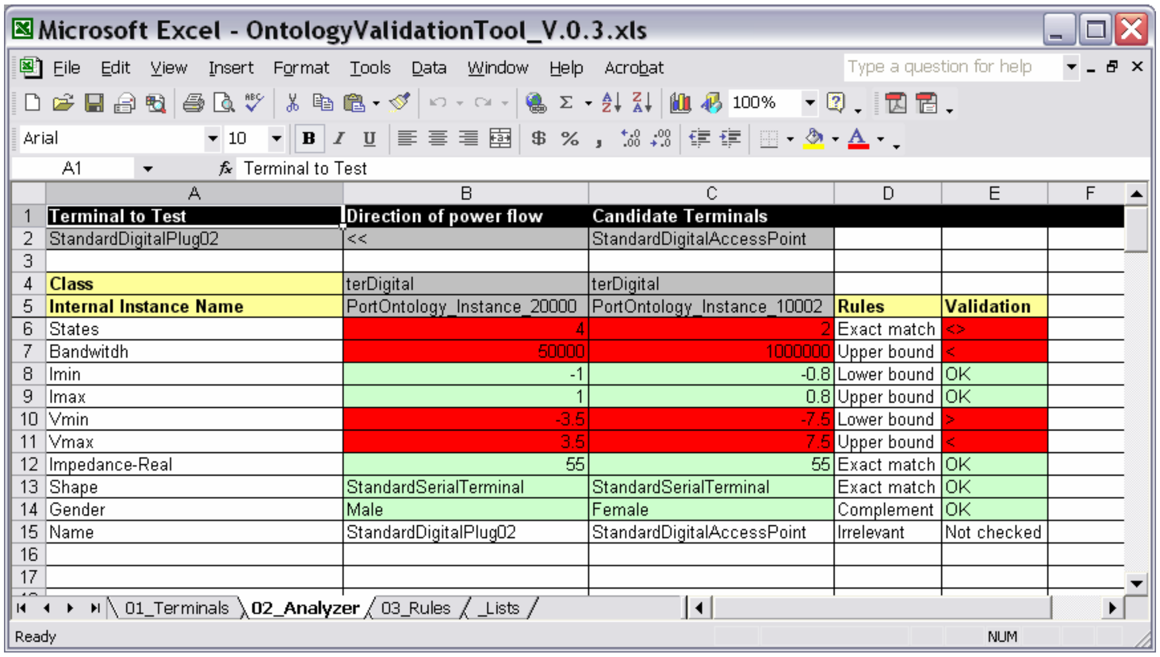

Microsoft Excel - OntologyValidationTool_V.0.3.xls
File Edit View Insert Format Tools Data Window Help Acrobat
Type a question for help
100%
Arial 10
A1
name
01_Terminals
02_Analyzer
03_Rules
_Lists
Ready
NUM

	name	type	slot_reference	value
1	name	type	slot_reference	value
4	PortOntology_Instance_2	:INSTANCE-ANNOTATION	:CREATOR	Francisco
5	PortOntology_Instance_2	:INSTANCE-ANNOTATION	:CREATION-TIMESTAMP	2005.10.15 18:49:48.984 EDT
6	PortOntology_Instance_3	:INSTANCE-ANNOTATION	:CREATOR	Francisco
7	PortOntology_Instance_3	:INSTANCE-ANNOTATION	:CREATION-TIMESTAMP	2005.10.15 18:50:19.406 EDT
8	PortOntology_Instance_10000	terDigital	States	2
9	PortOntology_Instance_10000	terDigital	Bandwitdh	1000000
10	PortOntology_Instance_10000	terDigital	Imin	-0.5
11	PortOntology_Instance_10000	terDigital	Imax	0.5
12	PortOntology_Instance_10000	terDigital	Vmin	-5
13	PortOntology_Instance_10000	terDigital	Vmax	5
14	PortOntology_Instance_10000	terDigital	Impedance-Real	55
15	PortOntology_Instance_10000	terDigital	Shape	StandardSerialTerminal
16	PortOntology_Instance_10000	terDigital	Gender	Male
17	PortOntology_Instance_10000	terDigital	Name	StandardDigitalPlug
18	PortOntology_Instance_10002	terDigital	States	2
19	PortOntology_Instance_10002	terDigital	Bandwitdh	1000000

Figure 4.5

In the Excel tool, by going to the tab “02_Analyzer” the user can access the drop-down menus generated by the software and can test different combinations of terminals for compatibility. In Figure 4.6 such tab can be appreciated. Note how parameters that are not compatible are marked in red and those that are compatible are highlighted in green. Also observe how in cell B2 the user can indicate the direction of power flow. This affects the criteria used to determine whether connectivity is possible or not.

Column “validation” displays a symbol indicating why a certain parameter is not compatible. For example in screenshot in cell E6, “<>” is displayed. This indicates that the reason why the corresponding attribute was not validated is because it differs and the rule says that an exact match is required.



Microsoft Excel - OntologyValidationTool_V.0.3.xls

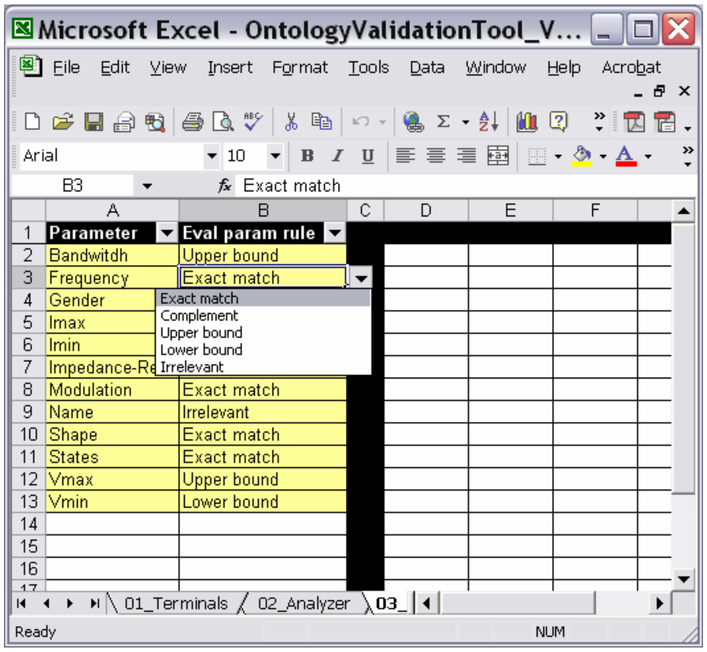
A1 Terminal to Test

	A	B	C	D	E
1	**Terminal to Test**	**Direction of power flow**	**Candidate Terminals**		
2	StandardDigitalPlug02	<<	StandardDigitalAccessPoint		
3					
4	**Class**	terDigital	terDigital		
5	**Internal Instance Name**	PortOntology_Instance_20000	PortOntology_Instance_10002	**Rules**	**Validation**
6	States	4	2	Exact match	<>
7	Bandwitdh	50000	1000000	Upper bound	<
8	Imin	-1	-0.8	Lower bound	OK
9	Imax	1	0.8	Upper bound	OK
10	Vmin	-3.5	-7.5	Lower bound	>
11	Vmax	3.5	7.5	Upper bound	<
12	Impedance-Real	55	55	Exact match	OK
13	Shape	StandardSerialTerminal	StandardSerialTerminal	Exact match	OK
14	Gender	Male	Female	Complement	OK
15	Name	StandardDigitalPlug02	StandardDigitalAccessPoint	Irrelevant	Not checked

01_Terminals / 02_Analyzer / 03_Rules / _Lists

Figure 4.6

The process uses for validating a certain attribute is based on a series of rules included in tab “03_Rules”. Figure 4.7 shows this tab. User can modify these rules with relative ease as long as they remain consistent with the Ontology in Protégé.



Microsoft Excel - OntologyValidationTool_V...

B3 Exact match

	A	B
1	**Parameter**	**Eval param rule**
2	Bandwitdh	Upper bound
3	Frequency	Exact match
4	Gender	
5	Imax	
6	Imin	
7	Impedance-Re	
8	Modulation	Exact match
9	Name	Irrelevant
10	Shape	Exact match
11	States	Exact match
12	Vmax	Upper bound
13	Vmin	Lower bound

Exact match
Complement
Upper bound
Lower bound
Irrelevant

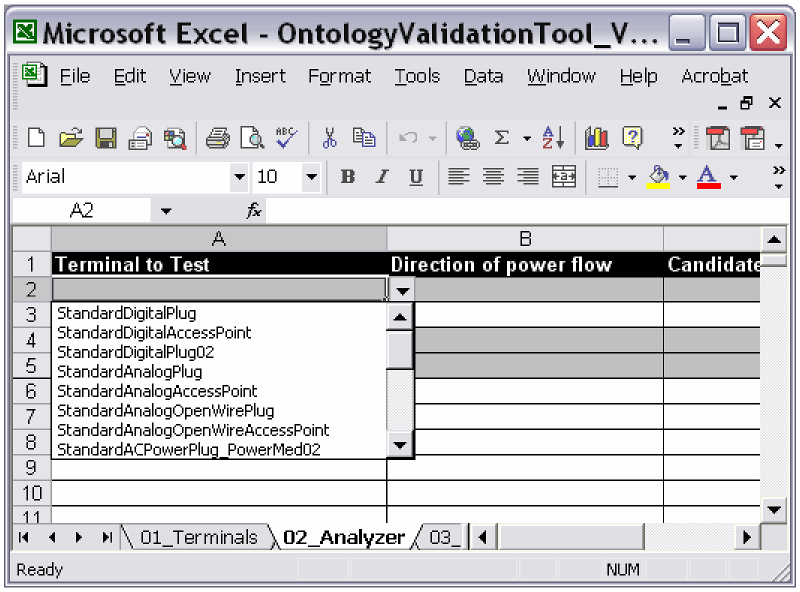
01_Terminals / 02_Analyzer / 03_

Figure 4.7

5. STEPS FOR USING THE ONTOLOGY VALIDATION TOOL

The sequence of validating terminals with the Excel-based tool is activated by pressing "Ctrl + Shift + A". The sequence of steps is:

1) Go to tab "02_Analyzer" and set cells A2 and C2 to blank.
2) Import data from XML file to tab "01_Terminals".
3) Hit "Ctrl + Shif + A". This will read the data and will identify all the instances available. Go to tab "02_Analyzer" and a drop down menu should have been generated. See Figure 5.1.



	A	B	
1	Terminal to Test	Direction of power flow	Candidate
2			

StandardDigitalPlug
StandardDigitalAccessPoint
StandardDigitalPlug02
StandardAnalogPlug
StandardAnalogAccessPoint
StandardAnalogOpenWirePlug
StandardAnalogOpenWireAccessPoint
StandardACPowerPlug_PowerMed02

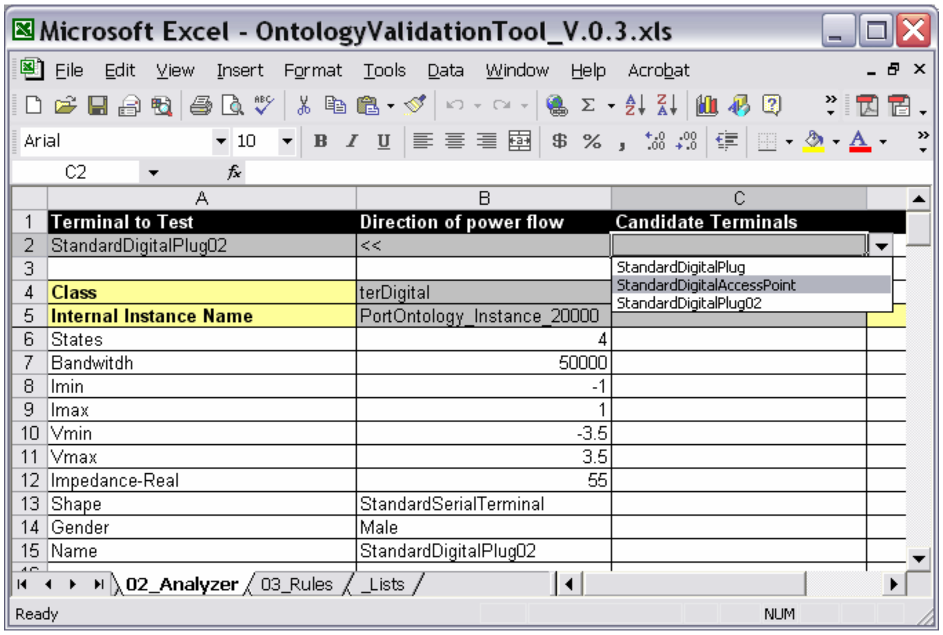
Figure 5.1

4) Select one instance in menu in cell A2.
5) Hit "Ctrl + Shift + A" again. This will:
   a. Populate the parameters of the instance you selected. See Figure 5.2.
   b. Generate a new drop down menu in cell C2. This menu contains the instances that based on class-level validation are compatible with the instance selected in cell A2.
6) Select an instance in cell C2.
7) Press "Ctrl + Shift + A". Figure 5.3 shows how the attributes of the second instance are retrieved and then validated in red and green as explained before.

In order to test a different instance in cell A2 in tab 02 you need to:

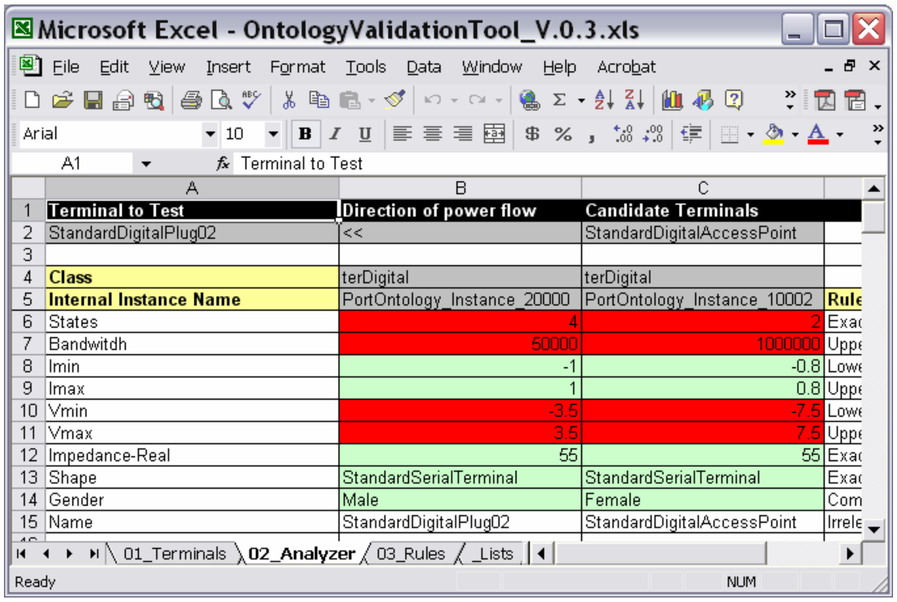
1) Delete contents of cell A2.
2) Delete contents of cell C2
3) Run tool again with "Ctrl + shift + A" and proceed to select a new instance from cell A2. This process of clearing cells is necessary to prevent inconsistencies caused by changes done to instances being tested after instance 2 has been selected.

If you want to test a different instance in cell C2, simply change it in the menu and press "Ctrl + Shift + A". It is not necessary to clear cell A2 if you are not going to change instance 1 (terminal to test).


Microsoft Excel - OntologyValidationTool_V.0.3.xls
File Edit View Insert Format Tools Data Window Help Acrobat
Arial 10
C2
02_Analyzer 03_Rules _Lists
Ready
NUM

	A	B	C
1	**Terminal to Test**	**Direction of power flow**	**Candidate Terminals**
2	StandardDigitalPlug02	<<	
3			StandardDigitalPlug
4	**Class**	terDigital	StandardDigitalAccessPoint
5	**Internal Instance Name**	PortOntology_Instance_20000	StandardDigitalPlug02
6	States	4	
7	Bandwitdh	50000	
8	Imin	-1	
9	Imax	1	
10	Vmin	-3.5	
11	Vmax	3.5	
12	Impedance-Real	55	
13	Shape	StandardSerialTerminal	
14	Gender	Male	
15	Name	StandardDigitalPlug02	

Figure 5.2


Microsoft Excel - OntologyValidationTool_V.0.3.xls
File Edit View Insert Format Tools Data Window Help Acrobat
Arial 10
A1
01_Terminals 02_Analyzer 03_Rules _Lists
Ready
NUM

Terminal to Test

	A	B	C	
1	**Terminal to Test**	**Direction of power flow**	**Candidate Terminals**	
2	StandardDigitalPlug02	<<	StandardDigitalAccessPoint	
3				
4	**Class**	terDigital	terDigital	
5	**Internal Instance Name**	PortOntology_Instance_20000	PortOntology_Instance_10002	**Rule**
6	States	4	2	Exac
7	Bandwitdh	50000	1000000	Uppe
8	Imin	-1	-0.8	Lowe
9	Imax	1	0.8	Uppe
10	Vmin	-3.5	-7.5	Lowe
11	Vmax	3.5	7.5	Uppe
12	Impedance-Real	55	55	Exac
13	Shape	StandardSerialTerminal	StandardSerialTerminal	Exac
14	Gender	Male	Female	Com
15	Name	StandardDigitalPlug02	StandardDigitalAccessPoint	Irrele

Figure 5.3

6. REFERENCES

1. Mayank, Vimal; Kositsyna, Natalya; and Austin, Mark– "Ontology-Enabled Validation of System-Level Architectures"; Institute for Systems Research, University of Maryland, College Park
2. Liang, Vei-Chung; and Paredis, Christiaan J.J. – "A Port Ontology For Automated Model Composition"; Institute for Complex Engineered Systems, Carnegie Mellon University
3. Liang, Vei-Chung; and Paredis, Christiaan J.J. – "A Port Ontology For Conceptual Design of Systems"; Journal of Computing and Information Science in Engineering - Sep/2004, Vol 4, Pgs: 206-217
4. Austin, Mark; "Introduction to Systems Engineering – Information Centric Systems Engineering"; Institute for Systems Research, University of Maryland, College Park

ANNEX 1
Electric Terminals Ontology Specification

TABLE OF CONTENTS

1. PURPOSE OF THIS DOCUMENT

The purpose of this annex is to document the process identifying the concepts needed to create an Electric Terminals Ontology (ETO) in Protégé.

Also in this annex, a naming convention will be introduced. The purpose of this naming convention is to provide an accurate an unambiguous way to name all the connections of a system with a similar complexity to the one used as a reference (Home Theater System, more details will be provided about this later). Note that since the Ontology created in Protégé used the system just as a template and did not map all the elements, the naming convention was not applied as a whole. However it was decided to leave it here as it may become useful in case that this project is enhanced in the future.

Note that the project files in Protégé are named “Port Ontology”. Don’t be confused because of this. Under the framework that will be explained in this document, Terminals are a subset of Ports, therefore by calling the project using Ports its contents can be extended without generating inconsistencies with its name.

2. SCOPE OF THE ONTOLOGY

This ETO will be created using a basic set of electrical ports as a template. More specifically the ports needed to connect the components of a Home Theater System (HTS). This ontology is based on the work done by Mayank [1] and Liang [2, 3] as well as other sources that will be appropriately cited.

The ETO aims at describing the entities whose composition defines a Port. These entities will be called Terminals. By ensuring the proper coupling of its Terminals, correct behavior of a Port is guaranteed.

3. CONSTRUCTION OF THE ETO

3.1. ANALYSIS OF BASE SYSTEM (HTS)

The HTS that will be used as a model to define the ETO corresponds to the commercial model Panasonic SC-HT380 (Figure 3.1). The goal is that after the ETO has been defined, any HTS Terminals can be modeled using it as long as they fall within the categories defined in the ontology.

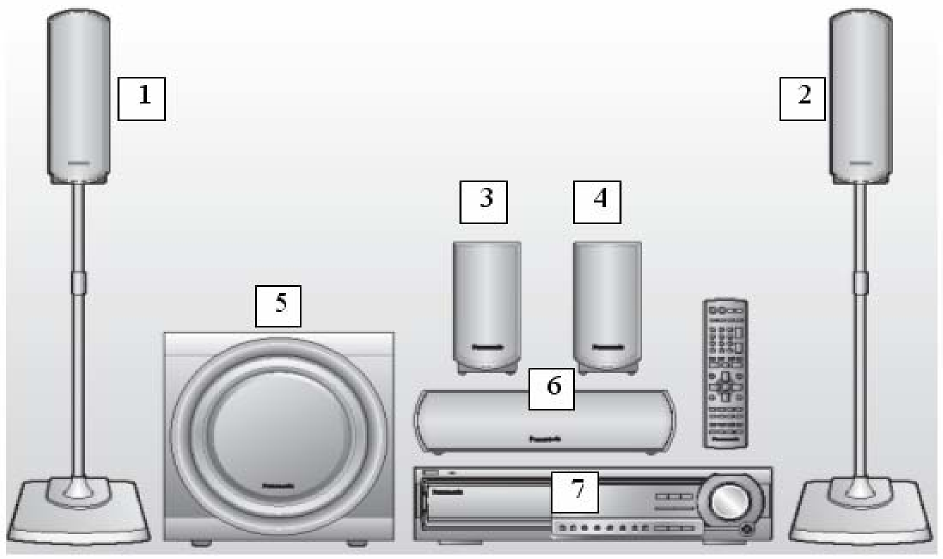


Figure 3.1

The components of interest of this HTS are:

Component Number	Component Name
1	Front-Left speaker
2	Front-Right speaker
3	Surround-Left speaker
4	Surround-Right speaker
5	Subwoofer
6	Central voice speaker
7	Central unit
8	AM Loop Antenna
9	FM Indoor Antenna

Table 3.1

In addition to these components, the HTS needs to interact with other additional external (i.e. not provided with the HTS) components:

Component Number	Component Name
E1	TV set
E2	Power outlet
E3	TV Antenna

Table 3.2

Now, some definitions are introduced:

Connector: Any cable or wire going from one component to another.

Port: Point where one component is coupled to a Connector. If a Connector is attached to a component in such a manner that the Connector cannot be unplugged from the component, by definition, no connection exists. Note that this is consistent with Liang in [2].

Terminal: Physical element whose composition defines a Port. There are two kinds of Terminals: Access Points and Plugs. For a Port to work properly, the Terminals composing it must fulfill certain conditions.

> **Access Point:** Terminal located in a component where a Connector is plugged. An Access Point can be either Male or Female.
>
> **Plug:** Terminal located at the end of a Connector that is plugged into a component. A plug can be either Male or Female.

A basic block diagram showing all the Connectors and components of the HTS is shown in Figure 3.2.

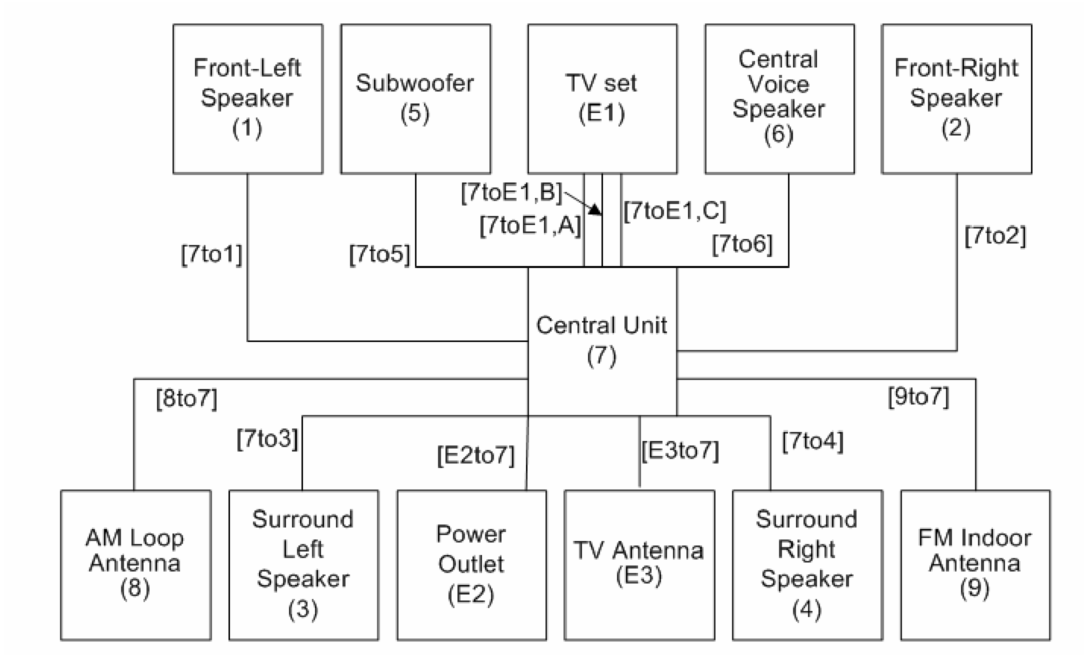


Figure 3.2

3.2. INTRODUCTION OF NAMING CONVENTION

Using diagram in Figure 3.2, a series of conventions will be defined to describe:

1. The **path** of each Connector.
2. The **location** of each Port.
3. The **location** of each Access Point.
4. The **location** of each Plug.

Note that these conventions do not include details regarding the categories suggested by Liang in [2]: form, function and behavior. Those categories will be dealt with later on.

3.2.1. CONNECTORS – PATH

The path of each Connector is labeled as [#to#]. The first number is the component that sends the signal (information/power) and the second number is the component that receives it.

When there is more than one connection between two components, the convention is [#to#,α], where α is A, B, etc. This can be seen in the connections between the Central Unit and the TV set. One connection (A) is for video and the other two for left (B) and right (C) audio channels. When there is only one connection, the “α” can be simply omitted.

3.2.2. PORTS – LOCATION

Each Connection may have one or two Ports. The Port at the end that is located next to the component that sends the signal will be called the Start Port. Similar, the Port at the other end will be called the End Port. This concept will be referred to as Directionality.

Since all the speakers and antennas have the wire attached to them and need to be plugged only to the Central Unit, these components have only either a Start or an End Port. The TV set and the Power Outlet use Ports that need to be plugged at both ends and that are not permanently attached to any of the components. Therefore, Connectors [7toE1,A], [7toE1,B], [7toE1,C], and [E2to7] have Start and End Ports.

The convention to make reference to the Directionality of a Port is [#to#.Start] or [#to#.End]. For instance, the plug that goes in the power outlet is described as [E2to7.Start] or the Video connector from the Central Unit to the TV set is described as [7toE1,A.End] where it is plugged to the TV. Note that this convention is simply an extension of the one used for Connectors.

3.2.3. ACCESS POINT – LOCATION

An Access Point will be denoted by indicating the Port it corresponds to: [#to#,α.Directionality.AP]. For instance, the RCA connector receptacle in the TV set that receives the video signal from the Central Unit is denoted as [7toE1,A.End.AP].

3.2.4. PLUG – LOCATION

A Plug will be denoted by indicating the Port it corresponds to: [#to#,α.Directionality.Pl]. For instance, the RCA connector Plug in the Connector that delivers the video signal from the Central Unit to the TV set is denoted as [7toE1,A.End.Pl].

3.3. IDENTIFICATION OF ALL CONNECTORS, PORTS AND TERMINALS

In file [Anex03_ConnectionsSummary] there are several tabs. Each one contains a different view of the set of Connectors, Ports and Terminals. This file also shows how the different elements relate to each other (i.e. how they are interconnected).

ANNEX 2
VBA Code Of Ontology Validation Tool

```
'*************************
'GLOBAL OPTIONS DECLARATION
'*************************

'Vars declarations is made explicit
Option Explicit

'***************************
'GLOBAL CONSTANTS DECLARATION
'***************************

Const MOD_MAXINSTANCES = 1000 'Determines de max number of distinct
instances to be processed
Const MOD_ANALIZER_TESTINSTCELL = "A2" 'Cell where the terminal to be
analyzed will be inserted
Const MOD_ANALIZER_CANDIDATES = "C2"

'Tab names
Const MOD_TAB_SRC_DATA = "01_Terminals"
Const MOD_TAB_ANALYZER = "02_Analyzer"
Const MOD_TAB_RULES = "03_Rules"
Const MOD_TAB_LISTS = "_Lists"

'Names names
Const MOD_NAME_INSTANCES = "slot_reference_name"
Const MOD_NAME_INSTANCESCLASS = "slot_reference_name_class"

'Ini rows
Const MOD_SRC_FIRSTROW = 2 'First row with actual data (after headings) in src tab
Const MOD_LISTS_FIRSTROW = 2
Const MOD_RULES_FIRSTROW = 2
Const MOD_ANALYZER_FIRSTROW = 6



'***************************
'GLOBAL VARIABLES DECLARATION
'***************************


'**********
'MAIN: START
'**********

'Macro key: Ctrl + Shift + A
```

```
Sub Main()

    '***************************
    'LOCAL CONSTANTS DECLARATION
    '***************************
    Const FLAG_DEBUG_SHOW = False
    Const NMBR_OF_CHECKS = 5

    '**********************
    'LOCAL VARS DECLARATION
    '**********************
    Dim longSrcRecords As Long
    Dim longTotInstances As Long
    Dim intStep As Integer ' This variable tells the macro in which part of the execution
next part should start
    Dim chrDataToDetermineStep(NMBR_OF_CHECKS) As String
    Dim bln_ExitFlag As Boolean

    Dim strTermSelected As String ' Terminal selected by usr in drop-down menu
    Dim longBegin As Long 'Row where the selected instance (by the usr) begins
    Dim longEnd As Long 'Row where the selected instance (by the usr) ends.
    Dim strTerClass As String 'Class the terminal selected by user belongs to

    Dim longParamsTotRows As Long 'Total number of params in tab
"MOD_TAB_ANALYZER" after the params have been read for a certain instance

    Dim longCount01 As Long
    Dim strRuleParamInstance As String 'Var to store the rule associated with a certain
parameter of an instance
    Dim strCurrentParamtoEval As String 'Stores temporarily the current param being eval

    Dim strPowerFlowDirection As String 'Var to store the direction power flow.

    '**************
    'LOCAL VARS INI
    '**************

    'the exit flag is initialized to False
    bln_ExitFlag = False

    longParamsTotRows = 0

    ' Var intStep is used to indicate the macro where to start each time it is run.
    ' To determine the value of intStep, a series of checks are performed.
    ' In general, if a certain cell is blank, is because the associated step has not yet
happened.
```

```
    'This array stores the values of the cells to be checked to determine the step.
    ' Check 1 -> Has data been imported to tab "MOD_TAB_SRC_DATA"? associated
cell is read here.
    Sheets(MOD_TAB_SRC_DATA).Select
    chrDataToDetermineStep(0) = Range("A1")
    'Check 2 -> Has the user selected a certain terminal to validate? associated cell is read
here.
    Sheets(MOD_TAB_ANALYZER).Select
    chrDataToDetermineStep(1) = Range("A2")
    'Check 3 -> Has the user selected a candidate compatible terminal after drop down 2
has been generated? associated cell is read here
    Sheets(MOD_TAB_ANALYZER).Select
    chrDataToDetermineStep(2) = Range("C2")

    'Now, each check will be evaluated to determine the step in which the macro has to
start
    'If no data exists in the tab where the src data should be imported, the step is = 0.
    If chrDataToDetermineStep(0) = "" Then

        Sheets(MOD_TAB_LISTS).Select
        Range("C2") = 0
        intStep = 0

        'Drop down menues are forced to be blank
        Sheets(MOD_TAB_ANALYZER).Select
        Range("A2") = ""
        Range("C2") = ""

    'If data exists in src tab, but the user has not selected a terminal to validate for
connection, then the step is = 1
    ElseIf chrDataToDetermineStep(1) = "" Then

        Sheets(MOD_TAB_LISTS).Select
        Range("C2") = 1
        intStep = 1

    'If data exists in src tab and the user has selected a teminal to validate for connection,
then the step is = 2
    ElseIf chrDataToDetermineStep(1) <> "" Then

        Sheets(MOD_TAB_LISTS).Select
        Range("C2") = 2
        intStep = 2

    End If
    'If data exists in src tab and the user has selected a terminal to validate for connection
and
    'the user has selected a candidate terminal to test, then the step is = 3
    If chrDataToDetermineStep(1) <> "" And chrDataToDetermineStep(2) <> "" Then

        Sheets(MOD_TAB_LISTS).Select
        Range("C2") = 3
        intStep = 3

        'DEBUG LINES
        If FLAG_DEBUG_SHOW Then

            MsgBox ("[Main]" & Chr(13) & Chr(13) & _
                "Step 3 has been activated")

        End If
        'END DEBUG LINES

    End If


    '**********
    'SHEETS INI
    '**********

    'A clearing process is activated if the process is in step 0, 1 or 2
    If intStep = 0 Or intStep = 1 Or intStep = 2 Then

        'Certain target cells are cleared (before copying params for selected instance)
        Call Sub_CellsRangeClear(MOD_TAB_ANALYZER, "A4", "F100")
        Call Sub_CellsRangeClear(MOD_TAB_LISTS, "E2", "E100")

        'Lists (validations) are set to "any value" by default
        Sheets(MOD_TAB_ANALYZER).Select
        Range("A2").Select
        With Selection.Validation
            .Delete
            .Add Type:=xlValidateInputOnly, AlertStyle:=xlValidAlertStop, Operator _
            :=xlBetween
        End With
        Range("C2").Select
        With Selection.Validation
            .Delete
            .Add Type:=xlValidateInputOnly, AlertStyle:=xlValidAlertStop, Operator _
            :=xlBetween
        End With

    End If

    'Clearing process activated if process is in step 0, 1, 2 or 3
    If intStep = 0 Or intStep = 1 Or intStep = 2 Or intStep = 3 Then

        'Appropriate tab is selected
        Sheets(MOD_TAB_ANALYZER).Select

        'Default color (white) is applied to cells in the Analyzer tab
        Range("A6:F100").Select
        Selection.Interior.ColorIndex = 2 'White is applied

    End If


    '*****************
    'CORE FUNCTIONALITY
    '*****************

    '---------------
    'LOGIC FOR STEP 0
    '---------------

    'If no records exist in src tab, a warning is displayed to user
    If intStep = 0 Then

        MsgBox ("[Main]" & Chr(13) & Chr(13) & "No data was detected in tab '" &
MOD_TAB_SRC_DATA & "'." & Chr(13) _
            & "Please check." & Chr(13) & "The program will now end.")

        bln_ExitFlag = True

        'Cell A1 is selected for user convenience
        Range("A1").Select

        'Residual lists are cleared
        Call Sub_CellsRangeClear(MOD_TAB_LISTS, "A2", "A100")

    End If

    '---------------
    'LOGIC FOR STEP 1
    '---------------
    'In this step:
```

```
    '1) A list with the names of all terminals is created.
    '2) Using such list, a drop-down menu is created.

    If intStep = 1 And bln_ExitFlag = False Then

        'Cells are cleared - where the list will be populated
        Call Sub_CellsRangeClear(MOD_TAB_LISTS, "A" &
CStr(MOD_LISTS_FIRSTROW), "A" & CStr(MOD_MAXINSTANCES))

        'Total of records in src tab is calculated using the appropriate function
        longSrcRecords = Func_RecordsRelevantCount_Long(MOD_TAB_SRC_DATA,
MOD_SRC_FIRSTROW, "A")

        'Call sub that populates list of distinct instances
        Call Sub_ValueLookupCols(MOD_TAB_SRC_DATA, MOD_TAB_LISTS, "C",
"D", "Name", MOD_LISTS_FIRSTROW, "A", MOD_SRC_FIRSTROW,
longSrcRecords + MOD_SRC_FIRSTROW - 1)

        'Total of items in list of instances is counted
        longTotInstances = Func_RecordsRelevantCount_Long(MOD_TAB_LISTS,
MOD_LISTS_FIRSTROW, "A")

        'Name to define list to generate drop-menu created
        Sheets(MOD_TAB_LISTS).Select
        Range("A" & CStr(MOD_LISTS_FIRSTROW) & ":A" & CStr(longTotInstances +
MOD_LISTS_FIRSTROW - 1)).Select
        ActiveWorkbook.Names.Add Name:=MOD_NAME_INSTANCES,
RefersToR1C1:= _
            "=_Lists!R" & CStr(MOD_LISTS_FIRSTROW) & "C1:R" &
CStr(longTotInstances + MOD_LISTS_FIRSTROW - 1) & "C1"

        'Cell A1 is selected for usr convenience in the tab where the name was defined
        Range("A1").Select

        'Drop down list is inserted
        Sheets(MOD_TAB_ANALYZER).Select
        Range(MOD_ANALIZER_TESTINSTCELL).Select

        With Selection.Validation
            .Delete
            .Add Type:=xlValidateList, AlertStyle:=xlValidAlertStop, Operator:= _
            xlBetween, Formula1:="=" & MOD_NAME_INSTANCES
            .IgnoreBlank = True
            .InCellDropdown = True
            .InputTitle = ""
            .ErrorTitle = ""
            .InputMessage = ""
            .ErrorMessage = ""
            .ShowInput = True
            .ShowError = True
        End With

    End If

    '----------------
    'LOGIC FOR STEP 2
    '----------------
    'In this step:
    '1) User selects a terminal from the drop down menu.
    '2) Information about that terminal is displayed.
    '3) New drop down is generated.
    '4) User selects a candidate terminal to test.

    If intStep = 2 And bln_ExitFlag = False Then

        'Value selected by user in drop-down menu is stored
        Sheets(MOD_TAB_ANALYZER).Select
        strTermSelected = Range("A2")

        'The Begin row with info for the instance is retrieved
        longBegin = Func_InstanceBoundsRetrieve_Long(MOD_TAB_SRC_DATA,
strTermSelected, "BEGIN")

        'The End row with info for the instance is retrieved
        longEnd = Func_InstanceBoundsRetrieve_Long(MOD_TAB_SRC_DATA,
strTermSelected, "END")

        'DEBUG LINES
        If FLAG_DEBUG_SHOW Then

            MsgBox ("[Main]" & Chr(13) & Chr(13) & "Slot Ref Name evaluated = " &
strTermSelected & _
            Chr(13) & "Begin Bound = " & longBegin & Chr(13) & "End bound = " &
longEnd)

        End If
        'END DEBUG LINES

        'Cells with params for selected instance are copied to the tab where the drop-down
menu is
        Call Sub_InstanceParamsCopy(MOD_TAB_SRC_DATA,
MOD_TAB_ANALYZER, "C" & CStr(longBegin), "D" & CStr(longEnd), "A6")
```

```
    'Class and Internal Instance Name are now copied
    Call Sub_InstanceParamsCopy(MOD_TAB_SRC_DATA,
MOD_TAB_ANALYZER, "A" & CStr(longBegin), "A" & CStr(longBegin), "B5")
    Call Sub_InstanceParamsCopy(MOD_TAB_SRC_DATA,
MOD_TAB_ANALYZER, "B" & CStr(longBegin), "B" & CStr(longBegin), "B4")

    'Labels are inserted for the class and internal instance name
    Sheets(MOD_TAB_ANALYZER).Select
    Range("A4") = "Class"
    Range("A5") = "Internal Instance Name"

    'Format is applied to these cells
    Range("A4:A5").Select
    With Selection.Interior
      .ColorIndex = 36
      .Pattern = xlSolid
    End With
    Range("B4:B5").Select
    With Selection.Interior
      .ColorIndex = 15
      .Pattern = xlSolid
    End With
    Columns("B:B").EntireColumn.AutoFit
    Range("A4:A5").Select
    Selection.Font.Bold = True
    Range("A6").Select

    'Now, a list will be created will all instances belonging to the same class as the
instance selected by the user.
    'This list will be created in the "MOD_TAB_LISTS" tab.
    'This sub-step is CLASS LEVEL validation.
    strTerClass = Range("B4")

    'DEBUG LINES
    If FLAG_DEBUG_SHOW Then

      MsgBox ("[Main]" & Chr(13) & Chr(13) & _
        "Class name to look for = " & strTerClass)

    End If
    'END DEBUG LINES

    'Total of records in src tab is re-calculated using the appropriate function
    longSrcRecords = Func_RecordsRelevantCount_Long(MOD_TAB_SRC_DATA,
MOD_SRC_FIRSTROW, "A")
```

```
    'Sub that finds each instance belonging to the same class as the instance selected by
user is called
    Call Sub_ValueLookupColsRefined(MOD_TAB_SRC_DATA,
MOD_TAB_LISTS, "C", "B", "D", "Name", strTerClass, MOD_LISTS_FIRSTROW,
"E", MOD_SRC_FIRSTROW, longSrcRecords + MOD_SRC_FIRSTROW - 1)

    '-----------------------------------------------
    'DROP DOWN MENU 2 FOR INSTANCES OF THE SAME CLASS

    'Total of instances of selected class is counted
    longTotInstances = Func_RecordsRelevantCount_Long(MOD_TAB_LISTS,
MOD_LISTS_FIRSTROW, "E")

    'Name to define list to generate drop-menu 2 created
    Sheets(MOD_TAB_LISTS).Select
    Range("E" & CStr(MOD_LISTS_FIRSTROW) & ":E" & CStr(longTotInstances +
MOD_LISTS_FIRSTROW - 1)).Select
    ActiveWorkbook.Names.Add Name:=MOD_NAME_INSTANCESCLASS,
RefersToR1C1:= _
        "=_Lists!R" & CStr(MOD_LISTS_FIRSTROW) & "C5:R" &
CStr(longTotInstances + MOD_LISTS_FIRSTROW - 1) & "C5"

    'Cell A1 is selected for usr convenience in the tab where the name was defined
    Range("A1").Select

    'Drop down list is inserted
    Sheets(MOD_TAB_ANALYZER).Select
    Range(MOD_ANALIZER_CANDIDATES).Select

    With Selection.Validation
        .Delete
        .Add Type:=xlValidateList, AlertStyle:=xlValidAlertStop, Operator:= _
        xlBetween, Formula1:="=" & MOD_NAME_INSTANCESCLASS
        .IgnoreBlank = True
        .InCellDropdown = True
        .InputTitle = ""
        .ErrorTitle = ""
        .InputMessage = ""
        .ErrorMessage = ""
        .ShowInput = True
        .ShowError = True
    End With

  End If
```

```
    '----------------
    'LOGIC FOR STEP 3
    '----------------
    'In this step:
    '1) Parameters of candidate terminal are retrieved.
    '2) Parameters of candidate terminal are compared to the paramerters of base terminal
(Test Terminal)
    'to determine compatibility.

    If intStep = 3 And bln_ExitFlag = False Then

        'Value selected by user in drop-down menu is stored (cadidate terminal in this case)
        Sheets(MOD_TAB_ANALYZER).Select
        strTermSelected = Range("C2")

        'The Begin row with info for the instance is retrieved
        longBegin = Func_InstanceBoundsRetrieve_Long(MOD_TAB_SRC_DATA,
strTermSelected, "BEGIN")

        'The End row with info for the instance is retrieved
        longEnd = Func_InstanceBoundsRetrieve_Long(MOD_TAB_SRC_DATA,
strTermSelected, "END")

        'DEBUG LINES
        If FLAG_DEBUG_SHOW Then

            MsgBox ("[Main]" & Chr(13) & Chr(13) & "Slot Ref Name evaluated
(Candidate) = " & strTermSelected & _
                Chr(13) & "Begin Bound = " & longBegin & Chr(13) & "End bound = " &
longEnd)

        End If
        'END DEBUG LINES

        'Cells with params for selected instance are copied to the tab where the drop-down
menu is
        Call Sub_InstanceParamsCopy(MOD_TAB_SRC_DATA,
MOD_TAB_ANALYZER, "D" & CStr(longBegin), "D" & CStr(longEnd), "C6")

        'Class and Internal Instance Name are now copied
        Call Sub_InstanceParamsCopy(MOD_TAB_SRC_DATA,
MOD_TAB_ANALYZER, "A" & CStr(longBegin), "A" & CStr(longBegin), "C5")
        Call Sub_InstanceParamsCopy(MOD_TAB_SRC_DATA,
MOD_TAB_ANALYZER, "B" & CStr(longBegin), "B" & CStr(longBegin), "C4")

        'Format is applied to these cells
        Sheets(MOD_TAB_ANALYZER).Select
        Range("C4:C5").Select
        With Selection.Interior
            .ColorIndex = 15
            .Pattern = xlSolid
        End With
        Columns("C:C").EntireColumn.AutoFit
        Range("A1").Select

        '------------------------------------------------
        'RANGE OF CELLS (ROWS) OF TEST PARAMS IS DETERMINED

        longParamsTotRows =
Func_RecordsRelevantCount_Long(MOD_TAB_ANALYZER,
MOD_ANALYZER_FIRSTROW, "B")

                'DEBUG LINES
        If FLAG_DEBUG_SHOW Then

            MsgBox ("[Main]" & Chr(13) & Chr(13) & _
                "Range of rows of params for analysis [Lower] = " &
MOD_ANALYZER_FIRSTROW & Chr(13) & _
                "Range of rows of params for analysis [Upper] = " &
CStr(longParamsTotRows + MOD_ANALYZER_FIRSTROW - 1))

        End If
        'END DEBUG LINES

        'END: RANGE OF CELLS (ROWS) OF TEST PARAMS IS DETERMINED
        '-----------------------------------------------------

        'Direction of power flow is read
        Sheets(MOD_TAB_ANALYZER).Select
        strPowerFlowDirection = Range("B2")

        '--------------------------------------------------------------------
        'LOOP TO EVALUATE EACH PARAMETER ACCORDING TO THE RULE
ASSOCIATED TO IT

        For longCount01 = MOD_ANALYZER_FIRSTROW To (longParamsTotRows +
MOD_ANALYZER_FIRSTROW - 1)

            'Appropriate tab is selected
            Sheets(MOD_TAB_ANALYZER).Select

            'Param to eval is read and stored
```

```
            strCurrentParamtoEval = Range("A" & CStr(longCount01))

            'Rule for parameter in loop is determined using the appropriate function
            strRuleParamInstance = Func_InstanceRuleRetrieve_String(MOD_TAB_RULES,
strCurrentParamtoEval, "A", "B")

            'Appropriate tab is selected
            Sheets(MOD_TAB_ANALYZER).Select

            'Rule is written in Analysis tab for ease of use
            Range("D" & CStr(longCount01)) = strRuleParamInstance

            'Param is evaluated for compatibility
            Call Sub_ParamValidate(MOD_TAB_ANALYZER, strPowerFlowDirection,
strRuleParamInstance, "B" & CStr(longCount01), "C" & CStr(longCount01), "E" &
CStr(longCount01))

        Next longCount01

        'END: LOOP TO EVALUATE EACH PARAMETER ACCORDING TO THE
RULE ASSOCIATED TO IT
        '--------------------------------------------------------------------------

        'Appropriate tab is selected
        Sheets(MOD_TAB_ANALYZER).Select

        'Headings and format are applied
        Range("D5:E5").Select
        With Selection.Interior
            .ColorIndex = 36
            .Pattern = xlSolid
        End With
        Columns("D:E").EntireColumn.AutoFit
        Range("D5:E5").Select
        Selection.Font.Bold = True

        Range("D5") = "Rules"
        Range("E5") = "Validation"

        Range("A1").Select

    End If

End Sub
'*********
'MAIN: END
```

```
'*********

'Purpose of sub: clears contents of a range of cells
'
'
'PARAMETERS:
'1) par_strTab -> Tab where cells to clean are located
'2) par_strCellIni -> Cell where the range starts
'3) par_strCellEnd -> cell where the range ends
'

Sub Sub_CellsRangeClear(par_strTab As String, par_strCellIni As String,
par_strCellEnd As String)

    '**************************
    'LOCAL CONSTANTS DECLARATION
    '**************************
    Const FLAG_DEBUG_SHOW = False

    '*********************
    'LOCAL VARS DECLARATION
    '*********************

    '*************
    'LOCAL VARS INI
    '*************

    '*****************
    'CORE FUNCTIONALITY
    '*****************
    Sheets(par_strTab).Select

    Range(par_strCellIni & ":" & par_strCellEnd).Select
    Selection.ClearContents

    'First cell is select for usr convenience
    Range("A1").Select

End Sub


'Purpose of sub: Copies parameters of an instance in a target tab and column
'
'
'PARAMETERS:
'1) par_strTabSrc -> Tab where cells to copy are located
```

```
'2) par_strTabTarget -> Destination tab where values will be copied
'3) par_strCellIni -> Cell where the range of values to be copied starts (upper left)
'4) par_strCellEnd -> cell where the range of values to be copied  ends (bottom right)
'5) par_strCellTarget -> Upper left cell where the vlaues will be copied
'

Sub Sub_InstanceParamsCopy(par_strTabSrc As String, par_strTabTarget As String,
par_strCellIni As String, par_strCellEnd As String, par_strCellTarget As String)

    '**************************
    'LOCAL CONSTANTS DECLARATION
    '**************************
    Const FLAG_DEBUG_SHOW = False

    '*********************
    'LOCAL VARS DECLARATION
    '*********************

    '*************
    'LOCAL VARS INI
    '*************

    '****************
    'CORE FUNCTIONALITY
    '****************
    Sheets(par_strTabSrc).Select
    Range(par_strCellIni & ":" & par_strCellEnd).Select
    Selection.Copy
    Sheets(par_strTabTarget).Select
    Range(par_strCellTarget).Select
    ActiveSheet.Paste
    Range("A1").Select

    'Original tab is selected again
    Sheets(par_strTabSrc).Select
    Range("A1").Select

End Sub


'Purpose of sub: Determines if a certain attribute of a candidate instance is compatible
with the same attribute in the
'test instance (terminal). This sub considers the direction in which power flows.
'
'
'PARAMETERS:
```

```
'1) par_strTabSrc -> Tab where attributes to validate are located.
'2) par_strPowerFlow -> Direction of power flow. Values are ">>" or "<<" to be
interpreted as arrows (to right and to left)
'3) par_strRule -> Indicates the rule that applies to the attribute under eval. Valid values
are only those included in tab "_Lists".
'4) par_strCellAttribute01 -> Cell containint the attribute corresponding to the Test
terminal.
'5) par_strCellAttribute02 -> Cell containint the attribute corresponding to the Candidate
terminal.
'6) par_strCellEvalTarget -> Cell where the analysis result will be written.
'

Sub Sub_ParamValidate(par_strTabSrc As String, par_strPowerFlow As String,
par_strRule As String, par_strCellAttribute01 As String, par_strCellAttribute02 As
String, par_strCellEvalTarget As String)

  '***************************
  'LOCAL CONSTANTS DECLARATION
  '***************************
  Const FLAG_DEBUG_SHOW = False

  '**********************
  'LOCAL VARS DECLARATION
  '**********************
  Dim intColorCodeMark 'var to store the color code
  Dim strValidationCode 'var to store message to usr showing result of validation

  '**************
  'LOCAL VARS INI
  '**************
  intColorCodeMark = 2 'By default the color code is set to white
  strValidationCode = "Not checked" 'By default is set to OK

  '******************
  'CORE FUNCTIONALITY
  '******************

  'Appropriate sheet is selected
  Sheets(par_strTabSrc).Select

  'DEBUG LINES
  If FLAG_DEBUG_SHOW Then

    MsgBox ("[Sub_ParamValidate]" & Chr(13) & Chr(13) & _
      "Parameters received:" & Chr(13) & _
      "Power flow direction = " & par_strPowerFlow & _
```

```
        "Rule received = " & par_strRule & _
        "Cell with param 1 = " & par_strCellAttribute01 & _
        "Cell with param 2 = " & par_strCellAttribute02 & _
        "Target cell to include user message = " & par_strCellEvalTarget)

End If
'END DEBUG LINES

'-------------------
'RULE 1 = "Exact Match"
'--------
'Power Flow direction = Irrelevant in this case
'--------
If par_strRule = "Exact match" Then

    'DEBUG LINES
    If FLAG_DEBUG_SHOW Then

        MsgBox ("[Sub_ParamValidate]" & Chr(13) & Chr(13) & _
            "Entered: Section for 'Exact match'")

    End If
    'END DEBUG LINES

    If Range(par_strCellAttribute01) = Range(par_strCellAttribute02) Then

        'Color code is set to green
        intColorCodeMark = 35

        'Validation code is set to "OK"
        strValidationCode = "OK"

    Else

        'Color code is set to red
        intColorCodeMark = 3

        'Validation code is set to "OK"
        strValidationCode = "<>"

    End If

End If

'-------------------
'RULE 2 = "Complement"
```

```
    '--------
    'Power Flow direction = Irrelevant in this case
    '--------
    If par_strRule = "Complement" Then

        'DEBUG LINES
        If FLAG_DEBUG_SHOW Then

            MsgBox ("[Sub_ParamValidate]" & Chr(13) & Chr(13) & _
                "Entered: Section for 'Complement'")

        End If
        'END DEBUG LINES

        If Range(par_strCellAttribute01) = "Male" And Range(par_strCellAttribute02) =
"Female" Then

            'Color code is set to green
            intColorCodeMark = 35

            'Validation code is set to "OK"
            strValidationCode = "OK"

        ElseIf Range(par_strCellAttribute01) = "Female" And
Range(par_strCellAttribute02) = "Male" Then

            'Color code is set to green
            intColorCodeMark = 35

            'Validation code is set to "OK"
            strValidationCode = "OK"

        Else

            'Color code is set to red
            intColorCodeMark = 3

            'Validation code is set to "OK"
            strValidationCode = "="

        End If

    End If

    '-------------------
    'RULE 3A = "Upper Bound"
```

```
'--------
'Power Flow direction = ">>"
'--------
If par_strRule = "Upper bound" And par_strPowerFlow = ">>" Then

   'DEBUG LINES
   If FLAG_DEBUG_SHOW Then

      MsgBox ("[Sub_ParamValidate]" & Chr(13) & Chr(13) & _
         "Entered: Section for 'Upper bound' with flow '>>'")

   End If
   'END DEBUG LINES

   If Range(par_strCellAttribute01) <= Range(par_strCellAttribute02) Then

      'Color code is set to green
      intColorCodeMark = 35

      'Validation code is set to "OK"
      strValidationCode = "OK"

   Else

      'Color code is set to red
      intColorCodeMark = 3

      'Validation code is set to "OK"
      strValidationCode = ">"

   End If

End If

'--------------------
'RULE 3B = "Upper Bound"
'--------
'Power Flow direction = "<<"
'--------
If par_strRule = "Upper bound" And par_strPowerFlow = "<<" Then

   'DEBUG LINES
   If FLAG_DEBUG_SHOW Then

      MsgBox ("[Sub_ParamValidate]" & Chr(13) & Chr(13) & _
         "Entered: Section for 'Upper bound' with flow '<<'")
```

```
    End If
    'END DEBUG LINES

    If Range(par_strCellAttribute01) >= Range(par_strCellAttribute02) Then

        'Color code is set to green
        intColorCodeMark = 35

        'Validation code is set to "OK"
        strValidationCode = "OK"

    Else

        'Color code is set to red
        intColorCodeMark = 3

        'Validation code is set to "OK"
        strValidationCode = "<"

    End If

End If

'--------------------
'RULE 4A = "Lower Bound"
'--------
'Power Flow direction = ">>"
'--------
If par_strRule = "Lower bound" And par_strPowerFlow = ">>" Then

    'DEBUG LINES
    If FLAG_DEBUG_SHOW Then

        MsgBox ("[Sub_ParamValidate]" & Chr(13) & Chr(13) & _
            "Entered: Section for 'Lower bound' with flow '>>'")

    End If
    'END DEBUG LINES

    If Range(par_strCellAttribute01) >= Range(par_strCellAttribute02) Then

        'Color code is set to green
        intColorCodeMark = 35

        'Validation code is set to "OK"
        strValidationCode = "OK"

    Else

        'Color code is set to red
        intColorCodeMark = 3

        'Validation code is set to "OK"
        strValidationCode = "<"

    End If

End If

'--------------------
'RULE 4B = "Lower Bound"
'--------
'Power Flow direction = "<<"
'--------
If par_strRule = "Lower bound" And par_strPowerFlow = "<<" Then

    'DEBUG LINES
    If FLAG_DEBUG_SHOW Then

        MsgBox ("[Sub_ParamValidate]" & Chr(13) & Chr(13) & _
            "Entered: Section for 'Lower bound' with flow '<<'")

    End If
    'END DEBUG LINES

    If Range(par_strCellAttribute01) <= Range(par_strCellAttribute02) Then

        'Color code is set to green
        intColorCodeMark = 35

        'Validation code is set to "OK"
        strValidationCode = "OK"

    Else

        'Color code is set to red
        intColorCodeMark = 3

        'Validation code is set to "OK"
        strValidationCode = ">"
        End If

    End If

    '--------------------------------
    'RESULTS OF ANALYSIS ARE DISPLAYED

    'Color is applied to evaluated cells (params-attributes)
    Range(par_strCellAttribute01 & ":" & par_strCellAttribute02).Select
    With Selection.Interior
        .ColorIndex = intColorCodeMark
        .Pattern = xlSolid
    End With

    'Color is applied to cell with validation results
    Range(par_strCellEvalTarget).Select
    With Selection.Interior
        .ColorIndex = intColorCodeMark
        .Pattern = xlSolid
    End With

    'Message to user is inserted in target cell
    Range(par_strCellEvalTarget) = strValidationCode

End Sub


'Purpose of sub: Identifies all distinct instances based on a lookup value in a list. Then,
each distinct instance is
'inserted into a target tab in Excel.
'
'
'PARAMETERS:
'1) par_strTabSrc -> Tab where the records to analyze are located
'2) par_strTabTarget -> Tab where analyzed values will be inserted
'3) par_strColLookup -> Column where the reference value to detect desired rows is
located
'4) par_strColLookupInstance -> Column where the value that corresponds to the lookup
value is located
'5) par_strLookupValue -> Value to find matches (lookup value)
'6) par_intTargetRow -> First row in target tab where matching values will be insterted
'7) par_strTargetCol -> First col in target tab where matching values will be insterted
'8) par_intSrcFirstRow -> First row with data in src tab
'9) par_longSrcLastRow -> Last row with data in src tab
```

```
Sub Sub_ValueLookupCols(par_strTabSrc As String, par_strTabTarget As String,
par_strColLookup As String, par_strColLookupInstance As String, par_strLookupValue
As String, par_intTargetRow As Integer, par_strTargetCol As String, par_intSrcFirstRow
As Integer, par_longSrcLastRow As Long)

    '***************************
    'LOCAL CONSTANTS DECLARATION
    '***************************
    Const FLAG_DEBUG_SHOW = False


    '**********************
    'LOCAL VARS DECLARATION
    '**********************
    Dim strInstancesDistinct(MOD_MAXINSTANCES) As String ' Array that will
contain all the distinct instances found based on the look par.

    Dim longCount01 As Long
    Dim longCount02 As Long
    Dim longTotDistinct As Long

    '**************
    'LOCAL VARS INI
    '**************
    longCount02 = 0
    longTotDistinct = 0

    '******************
    'CORE FUNCTIONALITY
    '******************

    'Src tab is selected
    Sheets(par_strTabSrc).Select

    'The array with the distinct instances is populated
    For longCount01 = par_intSrcFirstRow To par_longSrcLastRow

        If Range(par_strColLookup & CStr(longCount01)) = par_strLookupValue Then

            strInstancesDistinct(longCount02) = Range(par_strColLookupInstance &
CStr(longCount01))

            longCount02 = longCount02 + 1
            longTotDistinct = longCount02 ' This stores the final number of distinct instances

        End If
    Next longCount01

    'The array with each of the distinct instances is written to the destination tab
    Sheets(par_strTabTarget).Select

    For longCount01 = 0 To longTotDistinct

        Range(par_strTargetCol & CStr(longCount01 + par_intTargetRow)) =
strInstancesDistinct(longCount01)

    Next longCount01

    'Autofit is applied to target col
    Columns(par_strTargetCol & ":" & par_strTargetCol).EntireColumn.AutoFit

    'First cell is select for usr convenience
    Range("A1").Select

End Sub

'Purpose of sub: Identifies all distinct instances based on a lookup value in a list. Then,
each distinct instance is
'inserted into a target tab in Excel. Additional to the lookup value, a reference parameter
has to match.
'
'PARAMETERS:
'1) par_strTabSrc -> Tab where the records to analyze are located
'2) par_strTabTarget -> Tab where analyzed values will be inserted
'3) par_strColLookup -> Column where the reference value to detect desired rows is
located
'4) par_strColRefParam -> Column where the reference parameter to filter rows is located
'5) par_strColLookupInstance -> Column where the value that corresponds to the lookup
value is located
'6) par_strLookupValue -> Value to find matches (lookup value)
'7) par_strRefParam -> Value to filter rows
'8) par_intTargetRow -> First row in target tab where matching values will be insterted
'9) par_strTargetCol -> First col in target tab where matching values will be insterted
'10) par_intSrcFirstRow -> First row with data in src tab
'11) par_longSrcLastRow -> Last row with data in src tab

Sub Sub_ValueLookupColsRefined(par_strTabSrc As String, par_strTabTarget As
String, par_strColLookup As String, par_strColRefParam As String,
par_strColLookupInstance As String, par_strLookupValue As String, par_strRefParam
As String, par_intTargetRow As Integer, par_strTargetCol As String, par_intSrcFirstRow
As Integer, par_longSrcLastRow As Long)
```

```
    '***************************
    'LOCAL CONSTANTS DECLARATION
    '***************************
    Const FLAG_DEBUG_SHOW = False


    '**********************
    'LOCAL VARS DECLARATION
    '**********************
    Dim strInstancesDistinct(MOD_MAXINSTANCES) As String ' Array that will
contain all the distinct instances found based on the look par.

    Dim longCount01 As Long
    Dim longCount02 As Long
    Dim longTotDistinct As Long

    '**************
    'LOCAL VARS INI
    '**************
    longCount02 = 0
    longTotDistinct = 0

    '******************
    'CORE FUNCTIONALITY
    '******************

    'DEBUG LINES
    If FLAG_DEBUG_SHOW Then

        MsgBox ("[Sub_ValueLookupColsRefined]" & Chr(13) & Chr(13) & _
            "Function has been called")

    End If
    'END DEBUG LINES

    'Src tab is selected
    Sheets(par_strTabSrc).Select

    'DEBUG LINES
    If FLAG_DEBUG_SHOW Then

        MsgBox ("[Sub_ValueLookupColsRefined]" & Chr(13) & Chr(13) & _
            "Loop limits are:" & Chr(13) & _
            "Lower = " & par_intSrcFirstRow & Chr(13) & _
            "Upper = " & par_longSrcLastRow)
    End If
    'END DEBUG LINES

    'The array with the distinct instances is populated
    For longCount01 = par_intSrcFirstRow To par_longSrcLastRow

        'DEBUG LINES
        If FLAG_DEBUG_SHOW Then

            MsgBox ("[Sub_ValueLookupColsRefined]" & Chr(13) & Chr(13) & _
                "Row = " & longCount01 & Chr(13) & _
                "Lookup value = " & par_strLookupValue & Chr(13) & _
                "Lookup under eval = " & Range(par_strColLookup & CStr(longCount01)) &
Chr(13) & _
                "Ref param value = " & par_strRefParam & Chr(13) & _
                "Ref param value eval = " & Range(par_strColRefParam &
CStr(longCount01)))

        End If
        'END DEBUG LINES

        If Range(par_strColLookup & CStr(longCount01)) = par_strLookupValue And
Range(par_strColRefParam & CStr(longCount01)) = par_strRefParam Then

            strInstancesDistinct(longCount02) = Range(par_strColLookupInstance &
CStr(longCount01))

            longCount02 = longCount02 + 1
            longTotDistinct = longCount02 ' This stores the final number of distinct instances

        End If

    Next longCount01

    'The array with each of the distinct instances is written to the destination tab
    Sheets(par_strTabTarget).Select

    For longCount01 = 0 To longTotDistinct

        Range(par_strTargetCol & CStr(longCount01 + par_intTargetRow)) =
strInstancesDistinct(longCount01)

    Next longCount01

    'Autofit is applied to target col
    Columns(par_strTargetCol & ":" & par_strTargetCol).EntireColumn.AutoFit

    'First cell is select for usr convenience
    Range("A1").Select

End Sub

'Purpose of Function: Identifies the row where the information of a certain instance
begins or the row where it ends, depending
'on the params received.
'
'
'PARAMETERS:
'1) par_strTab -> Tab where the bound will be searched.
'2) par_strInstanceSlotRefName -> Slot Reference Name of the instance to look for.
'3) par_strBoundDefine -> [BEGIN or END] to indicate if the function shall return the
upper or lower bound.
'
'RETURNS:
'1) Row where the upper or lower bound is located

Function Func_InstanceBoundsRetrieve_Long(par_strTab As String,
par_strInstanceSlotRefName As String, par_strBoundDefine As String) As Long

    '***************************
    'LOCAL CONSTANTS DECLARATION
    '***************************
    Const FLAG_DEBUG_SHOW = False

    '**********************
    'LOCAL VARS DECLARATION
    '**********************
    Dim longCount01 As Long
    Dim longRefRow As Long 'Row where the Slot Reference Name is located
    Dim longBegin As Long 'Row where the instance rows begin
    Dim longEnd As Long 'Row where the instance rows end
    Dim longSrcRecords As Long

    Dim blnBeginFlag As Boolean 'Flag to indicate that the Begin bound has already been
found
    Dim blnEndFlag As Boolean 'Flag to indicate that the End bound has already been
found

    Dim strInstanceInternalName As String

    '**************
    'LOCAL VARS INI
    '**************
    longBegin = 0
    longEnd = 0

    blnBeginFlag = False
    blnEndFlag = False

    '*****************
    'CORE FUNCTIONALITY
    '*****************

    'Desired tab is selected
    Sheets(par_strTab).Select

    'Total number of records with relevant data is detected
    longSrcRecords = Func_RecordsRelevantCount_Long(par_strTab,
MOD_SRC_FIRSTROW, "A")
    longSrcRecords = longSrcRecords + MOD_SRC_FIRSTROW - 1

    'Row where the Slot Reference Name is, is identified
    longRefRow = Func_RecordRowPos_Long(par_strTab, par_strInstanceSlotRefName,
"D")

    'Instance (internal) name is identified using the row where the Slot Reference Name is
    strInstanceInternalName = Range("A" & CStr(longRefRow))

    'DEBUG LINES
    If FLAG_DEBUG_SHOW Then

        MsgBox ("[Func_InstanceBoundsRetrieve_Long]" & Chr(13) & Chr(13) & _
            "Instance internal name = " & strInstanceInternalName)

    End If
    'END DEBUG LINES

    'Desired tab is selected again for safety
    Sheets(par_strTab).Select

    'loop to find the first occurrence of the "strInstanceInternalName"
    For longCount01 = 1 To longSrcRecords

        'DEBUG LINES
        If FLAG_DEBUG_SHOW Then

            MsgBox ("[Func_InstanceBoundsRetrieve_Long]" & Chr(13) & Chr(13) & _
            "Value in cell [A" & longCount01 & "] = " & Range("A" &
CStr(longCount01)) & Chr(13) & _
            "Value in cell [A" & longCount01 + 1 & "] = " & Range("A" &
CStr(longCount01 + 1)) & Chr(13) & _
            "Instance Internal Name = " & strInstanceInternalName & Chr(13) & _
            Chr(13) & "blnBeginFlag = " & blnBeginFlag)

      End If
      'END DEBUG LINES

      'Begin bound is detected inside this If
      If Range("A" & CStr(longCount01)) = strInstanceInternalName And blnBeginFlag
= False Then

         ' When the first occurrence of the Internal Instance name is found, the row
number is stored
         longBegin = longCount01

         'The Begin bound flag is set to true
         blnBeginFlag = True

         'DEBUG LINES
         If FLAG_DEBUG_SHOW Then

            MsgBox ("[Func_InstanceBoundsRetrieve_Long]" & Chr(13) & Chr(13) & _
               "Begin bound detected! Row = " & longBegin & Chr(13) & _
               "Value in cell [A" & longCount01 & "] = " & Range("A" &
CStr(longCount01)) & Chr(13) & _
               "Value in cell [A" & longCount01 + 1 & "] = " & Range("A" &
CStr(longCount01 + 1)) & Chr(13) & _
               "Instance Internal Name = " & strInstanceInternalName & Chr(13) & _
               Chr(13) & "blnBeginFlag after detection = " & blnBeginFlag)

         End If
         'END DEBUG LINES

      End If

      'End bound is detected inside this If
      If Range("A" & CStr(longCount01)) = strInstanceInternalName And Range("A" &
CStr(longCount01 + 1)) <> strInstanceInternalName And blnEndFlag = False Then

         'Row of last occurrence of the Internal Instance name is stored
         longEnd = longCount01

         'The End bound flag is set to true
```

```
            blnEndFlag = True

            'DEBUG LINES
            If FLAG_DEBUG_SHOW Then

               MsgBox ("[Func_InstanceBoundsRetrieve_Long]" & Chr(13) & Chr(13) & _
                  "End bound detected! Row = " & longEnd & Chr(13) & _
                  "Value in cell [A" & longCount01 & "] = " & Range("A" &
CStr(longCount01)) & Chr(13) & _
                  "Value in cell [A" & longCount01 + 1 & "] = " & Range("A" &
CStr(longCount01 + 1)) & Chr(13) & _
                  "Instance Internal Name = " & strInstanceInternalName & Chr(13) & _
                  Chr(13) & "blnEndFlag after detection = " & blnEndFlag)

            End If
            'END DEBUG LINES

            'Exit condition is forced
            longCount01 = longSrcRecords - 1


         End If

      Next longCount01

       'DEBUG LINES
      If FLAG_DEBUG_SHOW Then

         MsgBox ("[Func_InstanceBoundsRetrieve_Long]" & Chr(13) & Chr(13) & _
            "Values to return are: " & Chr(13) & _
            "Begin bound = " & longBegin & Chr(13) & _
            "End bound = " & longEnd)

      End If
      'END DEBUG LINES

      'The upper or lower bound is returned depending on the paramter received
      If par_strBoundDefine = "BEGIN" Then

         Func_InstanceBoundsRetrieve_Long = longBegin

      ElseIf par_strBoundDefine = "END" Then

         Func_InstanceBoundsRetrieve_Long = longEnd

      End If
End Function


'Purpose of Function: Goes to the tab where the rules for the attributes of a terminal are
defined and retrieves the rule for a certain
'attribute received as a lookup paramater.'
'
'PARAMETERS:
'1) par_strTabRules -> Tab where the list of rules is.
'2) par_strParamName -> Name of the attribute of the terminal to look for.
'3) par_strLookupCol -> Column where the lookup value is supossed to be in the Rules
tab.
'4) par_strRuleCol -> Column where the rules are.
'
'RETURNS:
'1) The rule words (e.g. "Exact match" or "Upper Bound")

Function Func_InstanceRuleRetrieve_String(par_strTabRules As String,
par_strParamName As String, par_strLookupCol As String, par_strRuleCol As String)
As String

    '***************************
    'LOCAL CONSTANTS DECLARATION
    '***************************
    Const FLAG_DEBUG_SHOW = False

    '**********************
    'LOCAL VARS DECLARATION
    '**********************
    Dim longCount01 As Long
    Dim LongTotRows As Long
    Dim strRule As String

    '**************
    'LOCAL VARS INI
    '**************
    LongTotRows = 0

    '******************
    'CORE FUNCTIONALITY
    '******************

    'Total number of rows in Rules tab is retrieved
    LongTotRows = Func_RecordsRelevantCount_Long(par_strTabRules, 
MOD_RULES_FIRSTROW, "A")

    'loop to search for desired value
    For longCount01 = MOD_RULES_FIRSTROW To (LongTotRows + 
MOD_RULES_FIRSTROW - 1)

        'DEBUG LINES
        If FLAG_DEBUG_SHOW Then

            MsgBox ("[Func_InstanceRuleRetrieve_String]" & Chr(13) & Chr(13) & _
                "Rules tab received = " & par_strTabRules & Chr(13) & _
                "Last row with relevant data = " & CStr(LongTotRows + 
MOD_RULES_FIRSTROW - 1) & Chr(13) & _
                "Attribute to look for = " & par_strParamName & Chr(13) & _
                "Current rule attribute in Rules tab = " & Range(par_strLookupCol & 
CStr(longCount01)) & Chr(13) & _
                "Type of rule (current) = " & Range(par_strRuleCol & CStr(longCount01)))

        End If
        'END DEBUG LINES

        If Range(par_strLookupCol & CStr(longCount01)) = par_strParamName Then

            strRule = Range(par_strRuleCol & CStr(longCount01))

            'DEBUG LINES
            If FLAG_DEBUG_SHOW Then

                MsgBox ("[Func_InstanceRuleRetrieve_String]" & Chr(13) & Chr(13) & _
                    "Match found!" & Chr(13) & _
                    "Attribute to look for = " & par_strParamName & Chr(13) & _
                    "Current rule attribute in Rules tab = " & Range(par_strLookupCol & 
CStr(longCount01)) & Chr(13) & _
                    "Type of rule (Value to be returned) = " & Range(par_strRuleCol & 
CStr(longCount01)))

            End If
            'END DEBUG LINES

            'Exit condition is set
            longCount01 = (LongTotRows + MOD_RULES_FIRSTROW - 1)

        End If

    Next longCount01
    'DEBUG LINES
    If FLAG_DEBUG_SHOW Then

        MsgBox ("[Func_InstanceRuleRetrieve_String]" & Chr(13) & Chr(13) & _
            "Return value = " & strRule)

    End If
    'END DEBUG LINES


    'Resulting value is returned
    Func_InstanceRuleRetrieve_String = strRule

End Function


'Purpose of Function: Finds a value in a certain column and returns the row where it is located.
'
'
'PARAMETERS:
'1) par_strTab -> Tab where the value will be searched.
'2) par_strValueLookFor -> Value to look for.
'3) par_strColumn -> Column where the value will be searched.
'
'RETURNS:
'1) Row where the upper or lower bound is located. If the value is not found, the function will return 0.

Function Func_RecordRowPos_Long(par_strTab As String, par_strValueLookFor As String, par_strColumn As String) As Long

    '***************************
    'LOCAL CONSTANTS DECLARATION
    '***************************
    Const FLAG_DEBUG_SHOW = False

    '**********************
    'LOCAL VARS DECLARATION
    '**********************
    Dim longCount01
    Dim longSrcRecords
    Dim longPosRow

    '**************
    'LOCAL VARS INI
    '**************
    longPosRow = 0

    '*****************
    'CORE FUNCTIONALITY
    '*****************

    'Desired tab is selected
    Sheets(par_strTab).Select

    'Total number of records with relevant data is detected
    longSrcRecords = Func_RecordsRelevantCount_Long(par_strTab,
MOD_SRC_FIRSTROW, "A")
    longSrcRecords = longSrcRecords + MOD_SRC_FIRSTROW - 1

    'Loop to find desired record
    For longCount01 = 1 To longSrcRecords

        If Range(par_strColumn & CStr(longCount01)) = par_strValueLookFor Then

            longPosRow = longCount01

            'Exit condition from the For loop
            longCount01 = longSrcRecords

        End If

    Next longCount01

    If longPosRow = 0 Then

        MsgBox ("[Func_RecordRowPos_Long]" & Chr(13) & Chr(13) & "Value in
parameter not found in specified column.")

    End If

    'DEBUG LINES

    If FLAG_DEBUG_SHOW Then

        MsgBox ("[Func_RecordRowPos_Long]" & Chr(13) & Chr(13) & "Record = " &
par_strValueLookFor & _
            "was found in" & Chr(13) & "Row = " & longPosRow)

    End If
    'END DEBUG LINES

    'Resulting value is returned
    Func_RecordRowPos_Long = longPosRow

End Function


'Sub author: fgallo
'
'Purpose of function: This sub-function returns the number of records AFTER the
original intFirstRow value provided as a
'parameter. This is, if intFirstRow = 10 and there are 15 valid records after that row (i.e.
row 25 in Excel), then this
'function will return 15, NOT 25. If no records are detected, this function will return 0.
'
'This function assumes that a blank cell marks the end of valid rows.
'
'PARAMETERS:
'1) par_strTab -> Tab where records out of date range will be deleted
'2) par_intFirstRow -> First row where date range evaluation will start. This MUST be
the first row where actual data is supposed to exist.
'3) par_strColEval -> Column where the data will be evaluated
'
'RETURNS
'1) Number of relevant records AS LONG


Function Func_RecordsRelevantCount_Long(par_strTab As String, par_intFirstRow As
Integer, par_strColEval As String) As Long

    '***************************
    'LOCAL CONSTANTS DECLARATION
    '***************************
    Const FLAG_DEBUG_SHOW = False

    '***************************
    'LOCAL VARIABLES DECLARATION
    '***************************
    Dim longCount01 As Long
    Dim longTotRelevantRows As Long

    '*******************
    'LOCAL VARIABLES INI
    '*******************
    longCount01 = par_intFirstRow

    '*****************
    'CORE FUNCTIONALITY
    '*****************

    'Appropriate tab is selected
    Sheets(par_strTab).Select

    'Loop that will determine the number of rows with data
    Do While IsEmpty(Range(par_strColEval & CStr(longCount01))) = False

        longCount01 = longCount01 + 1

    Loop

    'The number of records is adjusted by substracting the starting row
    longTotRelevantRows = longCount01 - par_intFirstRow

    'Test lines

    If FLAG_DEBUG_SHOW Then

        MsgBox ("[Func_RecordsRelevantCount_Long] - return value" & Chr(13) &
Chr(13) & "Relevant records detected = " & longTotRelevantRows)

    End If

    'End test lines

    'Resulting value is returned
    Func_RecordsRelevantCount_Long = longTotRelevantRows

End Function
```

ANNEX 3
List of References

Ref Number	Ref description
1	Mayank, Vimal; Kositsyna, Natalya; and Austin, Mark– “Ontology-Enabled Validation of System-Level Architectures”; Institute for Systems Research, University of Maryland, College Park
2	Liang, Vei-Chung; and Paredis, Christiaan J.J. – “A Port Ontology For Automated Model Composition”; Institute for Complex Engineered Systems, Carnegie Mellon University
3	Liang, Vei-Chung; and Paredis, Christiaan J.J. – “A Port Ontology For Conceptual Design of Systems”; Journal of Computing and Information Science in Engineering - Sep/2004, Vol 4, Pgs: 206-217
4	Austin, Mark; “Introduction to Systems Engineering – Information Centric Systems Engineering”; Institute for Systems Research, University of Maryland, College Park

ANNEX 4
Protégé XML Schema Analysis

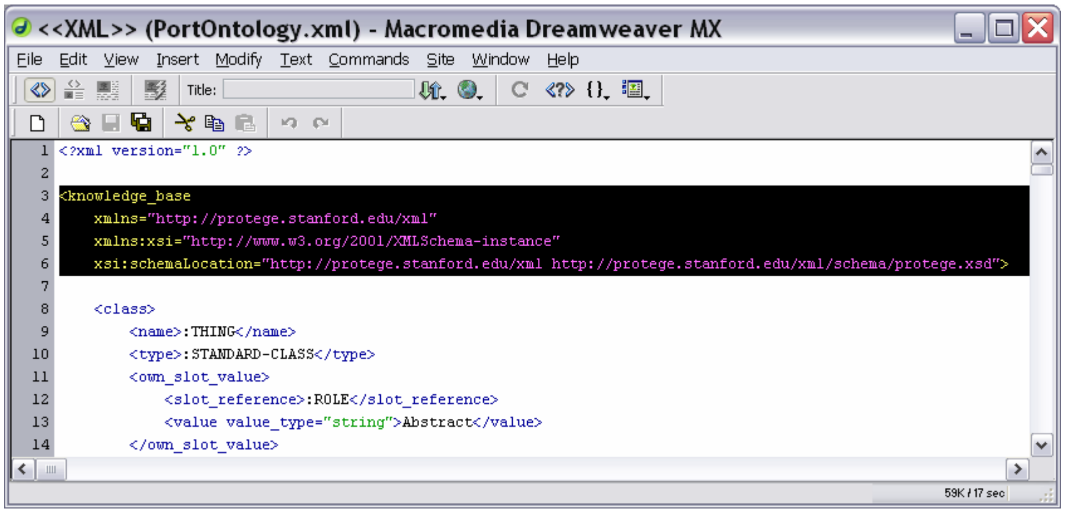
Tree					
Level001	**Level002**	**Level003**	**Comments Level001**	**Comments Level002**	**Comments Level003**
class	name				
class	type				
class	own_slot_value	slot_reference			
class	own_slot_value	value			
class	superclass			The string here links the [class] to its superclass (which is also a [class]) by means of the [class].[name].	
class	template_slot			The string here links the [class] to each of the [slots] associated to it by means of the [slot].[name].	
class	template_facet_value	slot_reference			
class	template_facet_value	facet_reference			
class	template_facet_value	value			
slot	name				
slot	type				
slot	own_slot_value	slot_reference			
slot	own_slot_value	value			
simple_instance	name				
simple_instance	type				
simple_instance	own_slot_value	slot_reference			
simple_instance	own_slot_value	value			

ANNEX 5
Steps To Export/Import XML

1. Export the data from Protégé using the option "convert project to format" from the "File" menu.
2. Rename the resulting XML file to "PortOntology_Target-Classes.xml". Open this file with a text editor.
3. By default, Protégé will use a certain schema. This schema is defined below:

```
<knowledge_base
        xmlns="http://protege.stanford.edu/xml"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://protege.stanford.edu/xml
http://protege.stanford.edu/xml/schema/protege.xsd">
```

This schema reference should be the 2nd element in the XML file:


<<XML>> (PortOntology.xml) - Macromedia Dreamweaver MX
File Edit View Insert Modify Text Commands Site Window Help
Title:
59K / 17 sec

```
<?xml version="1.0" ?>

<knowledge_base
    xmlns="http://protege.stanford.edu/xml"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://protege.stanford.edu/xml http://protege.stanford.edu/xml/schema/protege.xsd">

    <class>
        <name>:THING</name>
        <type>:STANDARD-CLASS</type>
        <own_slot_value>
            <slot_reference>:ROLE</slot_reference>
            <value value_type="string">Abstract</value>
        </own_slot_value>
```

Replace it with the line:

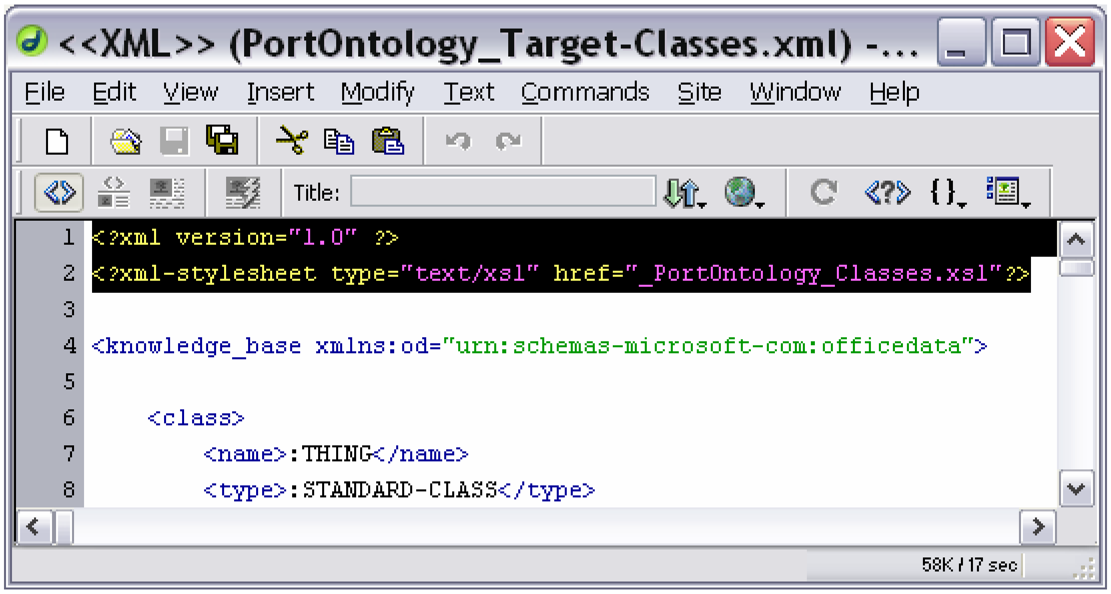
```
<knowledge_base xmlns:od="urn:schemas-microsoft-com:officedata">
```

This changes the reference to a schema that Excel and IE can work with.

4. **After** the first line (?xml version="1.0"), insert a line that reads:

```
<?xml-stylesheet type="text/xsl" href="_PortOntology_Classes.xsl"?>
```

Then the file at this time should look like:


<<XML>> (PortOntology_Target-Classes.xml) -...
File Edit View Insert Modify Text Commands Site Window Help
Title:
58K / 17 sec

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="_PortOntology_Classes.xsl"?>

<knowledge_base xmlns:od="urn:schemas-microsoft-com:officedata">

    <class>
        <name>:THING</name>
        <type>:STANDARD-CLASS</type>
```

5. Now this file is referencing the appropriate stylesheet "_PortOntology_Classes.xsl".
6. Now you can import the XML file to Excel or you can open it with IE.
7. Repeat this process for the Instances and the Slots.