

# Carnegie Mellon University Silicon Valley



## ASHRAE Standard 201 Reference Data Generator

# USER GUIDE

## Version 1.0

Point of contact:  
Dr. Steven Ray  
[Steve.ray@sv.cmu.edu](mailto:Steve.ray@sv.cmu.edu)

---

Carnegie Mellon University – Silicon Valley | NASA Ames Research Park  
Bldg 23 (MS 23-11) | Moffett Field, CA 94035

## TABLE OF CONTENTS

Introduction .....	3
System Requirements .....	3
Quick Start Instructions – TopBraid Composer Maestro .....	4
Quick Start Instructions – Topbraid Composer Free .....	8
System Design .....	11
System Execution .....	14
The Testing and Explanation Environment .....	17
Appendix .....	24

## INTRODUCTION

This tool uses of semantic modeling and reasoning techniques to build reference testing artifacts to aid in conformance testing of vendor implementations of emerging smart grid standards. By researching and demonstrating the feasibility and utility of this approach with one such standard<sup>1</sup>, the intention is that the approach will be available and applicable to all smart grid standards that are defined using the Unified Modeling Language (UML), most standards defined in XML Schema Definition (XSD), and eventually standards defined in OWL, thus providing a large return on investment to the smart grid community.

The full source code for this project has been placed in a public Github repository called fsgim-owlTesting, available for cloning at (<https://github.com/steveraysteveray/fsgim-owlTesting.git>), under GNU General Public License, Version 3.0.

For those who simply want to explore ASHRAE Standard 201 as an OWL model, a slightly cleaned-up version of the model, with shorter and more readable prefix names, has been placed into a public Github repository called fsgim-owl (<https://github.com/steveraysteveray/fsgim-owl.git>).

## SYSTEM REQUIREMENTS

The system requires a licensed copy of TopBraid Composer Maestro Edition for full functionality. This is because the system uses a SPARQLMotion script to invoke the SPIN reasoner to generate reference data. In principle, one could step through the entire process without using SPARQLMotion, in which case TopBraid Composer Free could be used to perform the SPIN reasoning. It is also possible to simply browse the reference data for the example described in this guide using TopBraid Composer Free, available at <http://www.topquadrant.com/downloads/topbraid-composer-install/>.

---

<sup>1</sup> ASHRAE SPC 201, sometimes known as the Facility Smart Grid Information Model, or FSGIM, available at [http://www.techstreet.com/ashrae/standards/ashrae-201-2016?product\\_id=1915946](http://www.techstreet.com/ashrae/standards/ashrae-201-2016?product_id=1915946)

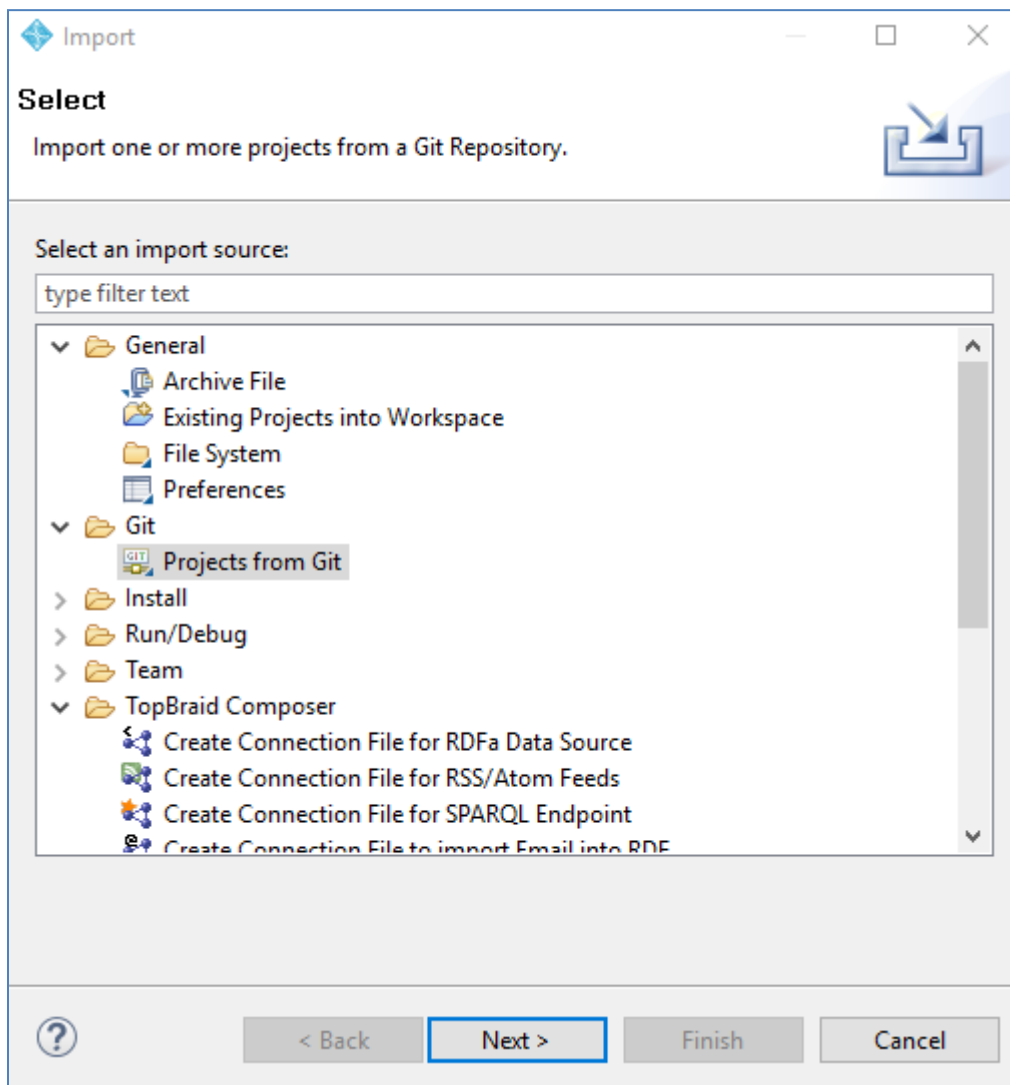
## QUICK START INSTRUCTIONS – TOPBRAID COMPOSER MAESTRO

These instructions take you through the installation of the data reference generator, running the generator with sample data, and inspecting the results. It is assumed you have previously installed a copy of TopBraid Composer Maestro, and inferencing is configured to use the TopSPIN reasoner.

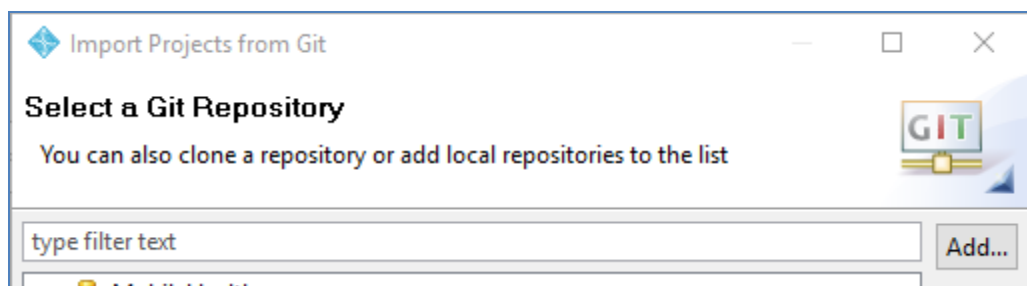
1. Clone the project from github. From a command line:

git clone <https://github.com/stevearysteveray/fsgim-owlTesting.git>

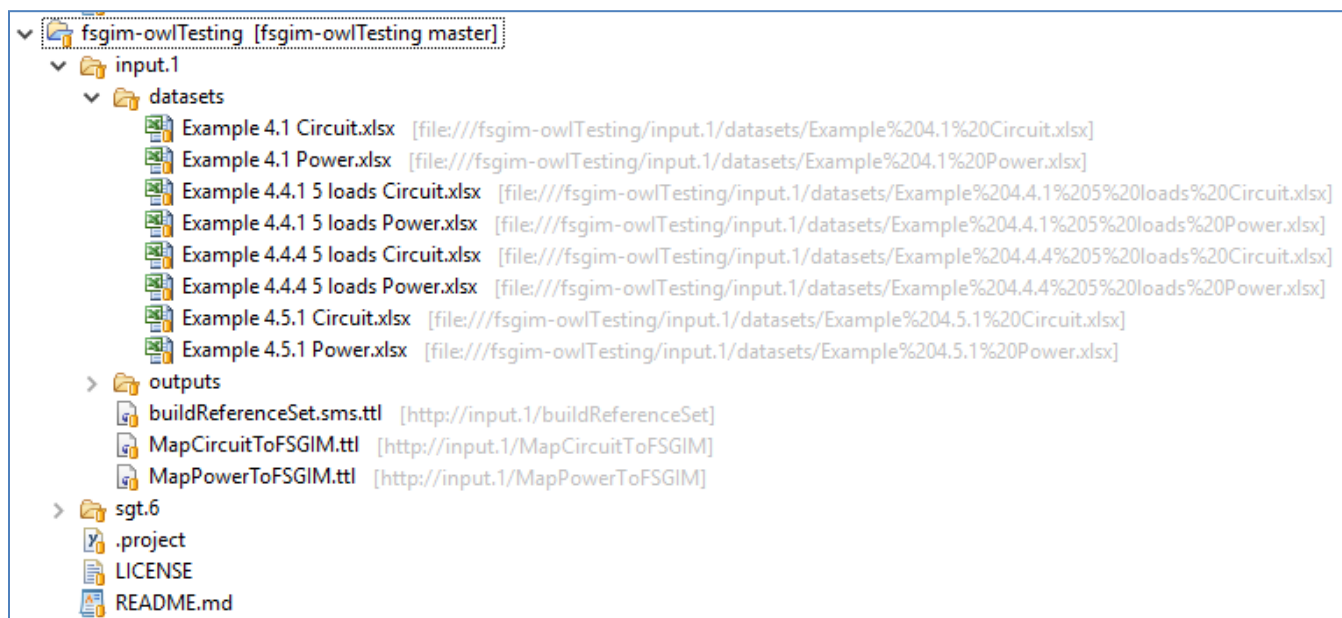
2. After starting TopBraid Composer (TBC), select File->Import->Git/Projects from Git



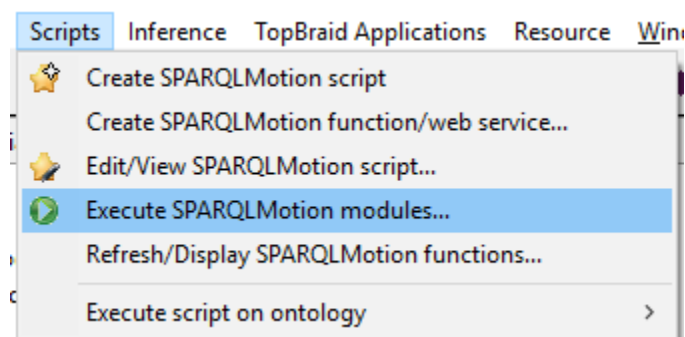
3. Select “Existing local repository”
4. Select the fsgim-owlTesting repository. If it doesn’t show up in your listing, use the “Add” button to navigate to wherever you cloned the repository.



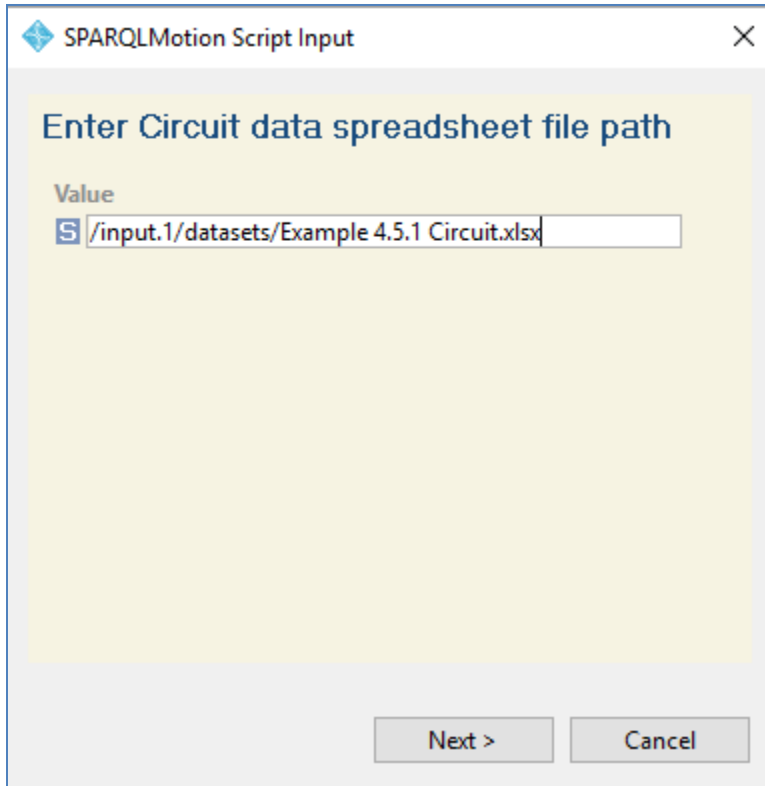
5. You should now see the fsgim-owlTesting project in the Navigator window in TBC.



6. Open the SPARQLMotion file “buildReferenceSet.sms.ttl” (double-click)
7. Select “Execute SPARQLMotion modules...” from the Scripts menu



8. After confirming the only choice of script, you are prompted for a data file that defines your circuit. In this example, we will use one of the files provided. Enter the file path as shown, press Enter (this is important, and often forgotten), and then click “Next”. Note that you are able to define your own circuits by modifying the contents of the sample Excel spreadsheets provided. Details about the contents and structure of these input files are given in the Appendix of this User Guide.



SPARQLMotion Script Input

Enter Circuit data spreadsheet file path

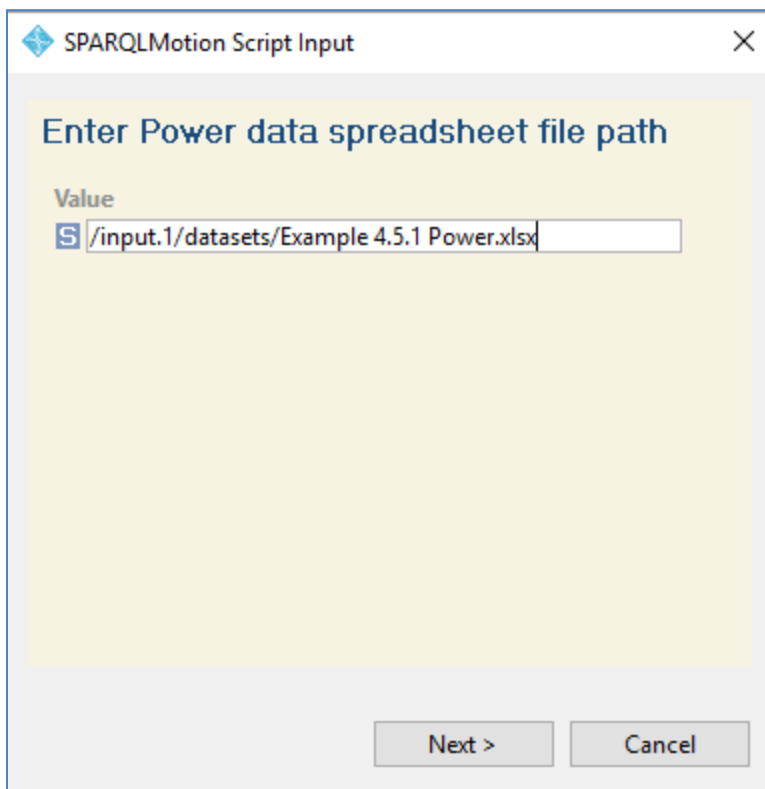
Value

S /input.1/datasets/Example 4.5.1 Circuit.xlsx

Next > Cancel

This is a screenshot of a software dialog box titled "SPARQLMotion Script Input". The dialog has a light gray border and a close button (X) in the top right corner. The main content area has a light yellow background and contains the text "Enter Circuit data spreadsheet file path". Below this, there is a label "Value" and a text input field. The input field contains the text "/input.1/datasets/Example 4.5.1 Circuit.xlsx" and has a small blue icon with a white 'S' to its left. At the bottom of the dialog, there are two buttons: "Next >" and "Cancel".

9. Do the same for the file that provides power levels.



SPARQLMotion Script Input

Enter Power data spreadsheet file path

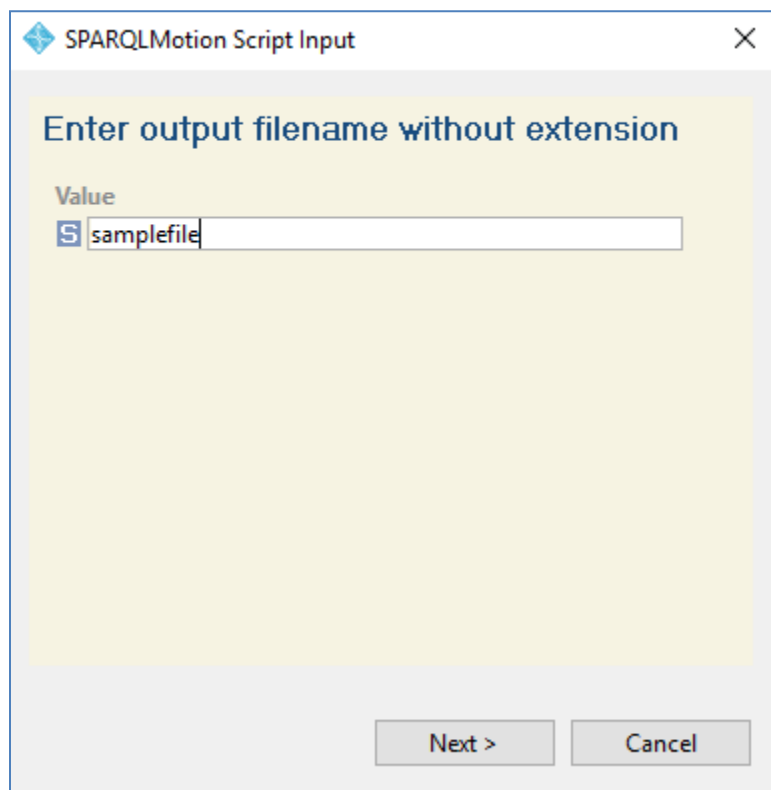
Value

S /input.1/datasets/Example 4.5.1 Power.xlsx

Next > Cancel

This is a screenshot of a software dialog box titled "SPARQLMotion Script Input". The dialog has a light gray border and a close button (X) in the top right corner. The main content area has a light yellow background and contains the text "Enter Power data spreadsheet file path". Below this, there is a label "Value" and a text input field. The input field contains the text "/input.1/datasets/Example 4.5.1 Power.xlsx" and has a small blue icon with a white 'S' to its left. At the bottom of the dialog, there are two buttons: "Next >" and "Cancel".

10. Finally, enter the output file name, without the file path and without an extension. This file will be created in the outputs folder in the input.1 project.



A screenshot of a dialog box titled "SPARQLMotion Script Input". The dialog has a light gray border and a close button (X) in the top right corner. The main area has a light yellow background and contains the text "Enter output filename without extension" in bold blue font. Below this, there is a label "Value" and a text input field. The input field contains the text "samplefile" and has a small blue icon with a white 'S' to its left. At the bottom of the dialog, there are two buttons: "Next >" and "Cancel".

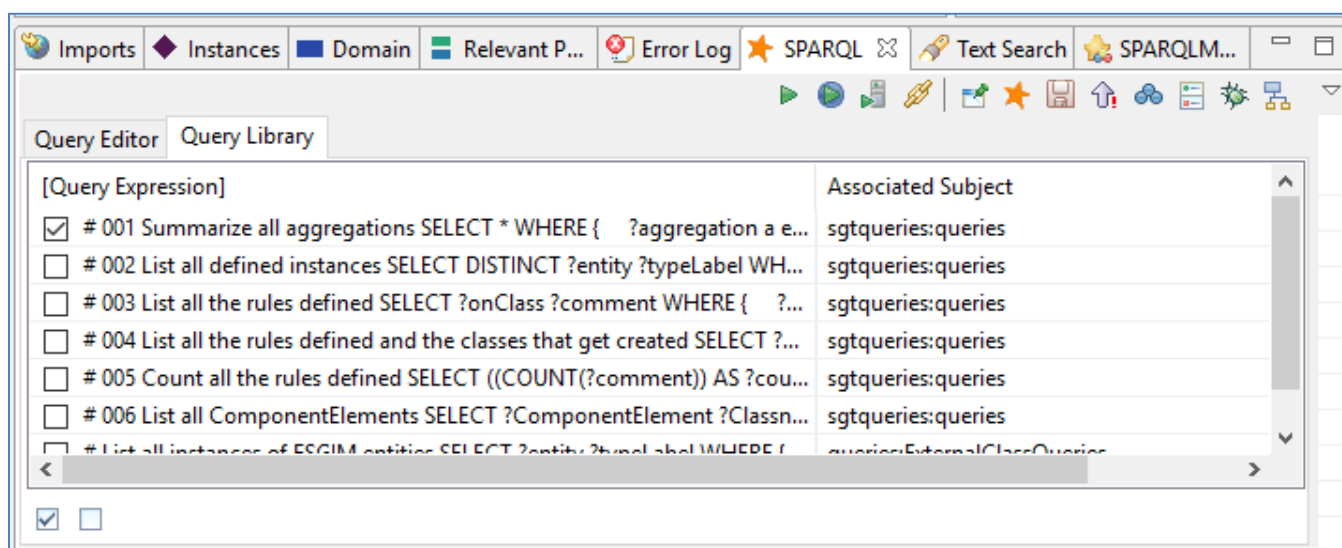
11. The system will now generate the reference data set. Please note that this example takes 4.5 minutes to complete on a computer using an Intel i7-920XM processor. Your performance will scale accordingly. When it is complete, a pop-up will state it is complete. You may have to click "OK" several times.

To view the generated reference data set, please follow the instructions below for TopBraid Composer Free.

## QUICK START INSTRUCTIONS – TOPBRAID COMPOSER FREE

The free version of the tool is not able to execute the SPARQLMotion script, but is able to view the results file. To load the sample reference data files, follow instructions 1-5 of the Quick Start Instructions for TopBraid Composer Maestro. Then,

1. Navigate to the input.1/outputs directory
2. Double-click the file Example 4.5.1.ttl (for this example), or the name of the file that you generated if using TopBraid Composer Maestro.
3. Select the “SPARQL” tab in the lower pane, and choose “Query Library”. De-select all the queries by clicking the empty box at the very bottom, then select just query #001. Your screen should look similar to this:



4. Execute query #001 by clicking the green triangle arrow pointing to the right (not the one inside the blue circle). Note that you can resize the panes by clicking and dragging your mouse. On the right side of the pane, you should see something like the following:

[aggregation]	EM	circuit	realDemand	realUncertainty	reactiveDemand	r
◆ fsgim:AggregateDemandESICircuit1	ESI	Circuit1	F 46.9	D 0.26457513...	F 16.6	I
◆ fsgim:AggregateDemandESICircuit2	ESI	Circuit2	F 46.9	D 0.26457513...	F 16.6	I
◆ fsgim:AggregateDemandFloor1Circuit2	Floor1	Circuit2	F 2.4	D 0.14142135...	F 1.5	I
◆ fsgim:AggregateDemandFloor2Circuit2	Floor2	Circuit2	F 44.5	D 0.20000000...	F 15.1	I
◆ fsgim:AggregateDemandFloor2Circuit3	Floor2	Circuit3	F 0.0	D 0.0e0	F 0.0	I
◆ fsgim:AggregateDemandFloor2Circuit4	Floor2	Circuit4	F 44.5	D 0.19999999...	F 15.1	I
◆ fsgim:AggregateDemandFloor2Circuit5	Floor2	Circuit5	F 8.4	D 0.09999999...	F 4.4	I

5. Each item with a purple diamond is an instance. Double-clicking it brings that item into the “Resource Form” pane in the center of the screen, where that instance and all its properties and relations can be examined. Literal properties will have an icon identifying the data type, such as F (float), D (double), etc. Instances properties will have a purple diamond. Classes have a brown circle.



Example 4.5.1.ttl

## Resource Form

Name: fsgim:AggregateDemandFloor2Circuit4

▼ Annotations

▼ Other Properties

explain:createdBy ▼  
☐ standard\_aggregations---collections\_rulesets\_and\_aggregations:DemandAggregation

explain:hasAggregateLoadReactiveDemand ▼

explain:hasAggregateLoadReactiveDemandUncertainty ▼

explain:hasAggregateLoadRealDemand ▼

explain:hasAggregateLoadRealDemandUncertainty ▼

explain:hasAggregateMeterReactiveDemand ▼

explain:hasAggregateMeterReactiveDemandUncertainty ▼

explain:hasAggregateMeterRealDemand ▼

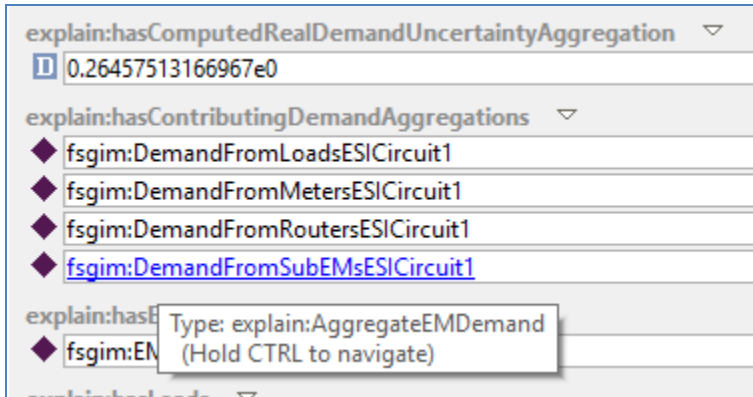
explain:hasAggregateMeterRealDemandUncertainty ▼

explain:hasAggregateRouterReactiveDemand ▼

explain:hasAggregateRouterReactiveDemandUncertainty ▼

explain:hasAggregateRouterRealDemand ▼

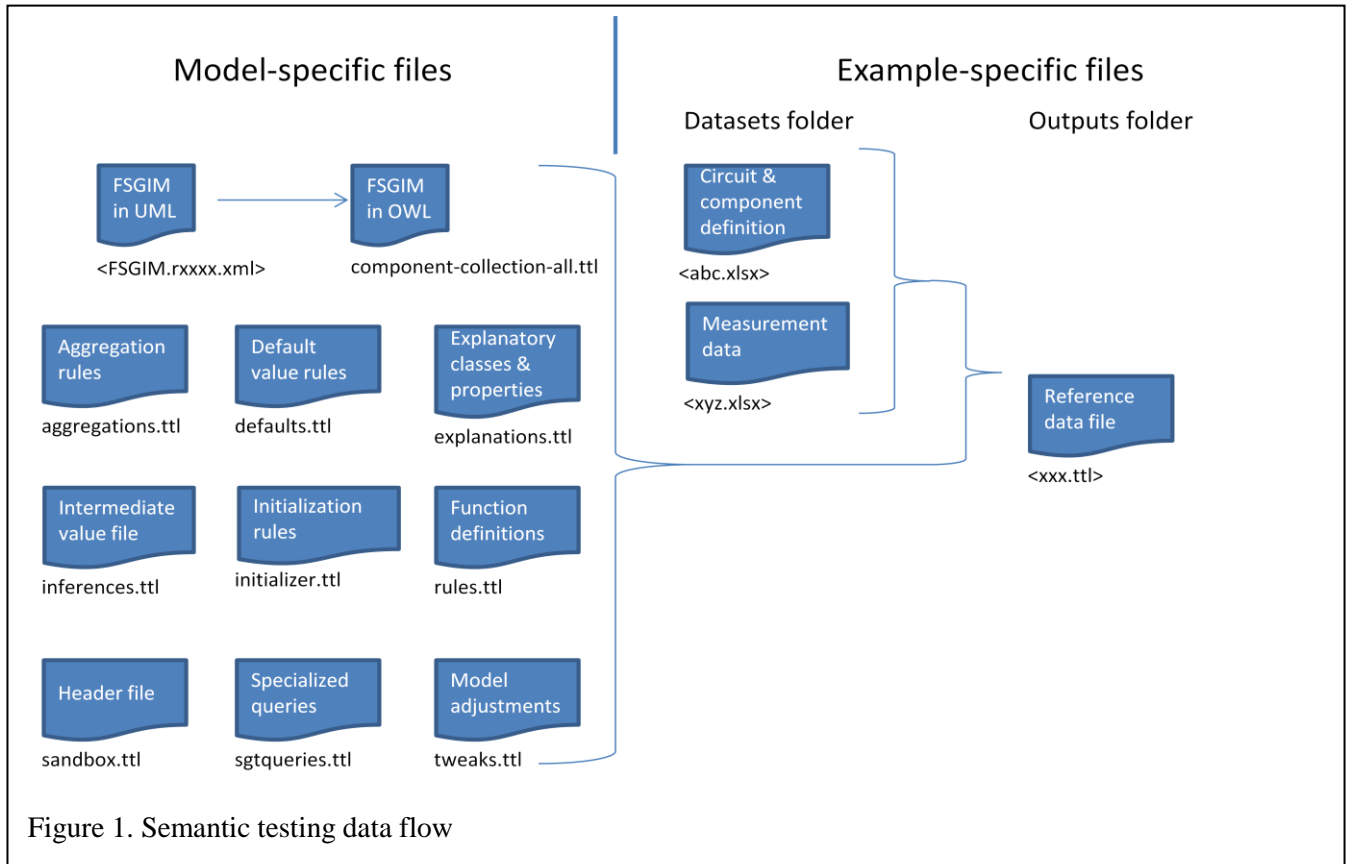
- You can navigate to a displayed object (instances or classes) by holding the CTRL key and clicking on the object name, which will become underlined. You can continue to navigate through the model by clicking on successive objects. You can also try some of the other predefined queries in the Query Library.



Navigating the reference data set is described in more detail below, in THE TESTING AND EXPLANATION ENVIRONMENT section.

## SYSTEM DESIGN

The files that support the conformance testing system are shown in Figures 1 and 2.



The files in Figures 1 and 2 are described below. The software development process was broken into two “projects” within the Eclipse software development environment. Project Input.1 handles the workflow of importing data, mapping and script files, invoking the reasoner at various stages (described later in Figure 2), and creating the final output. Project SGT.6, so named as an abbreviation of “Smart Grid Testing,” contains the rule files that encode the aggregation rules of the FSGIM standard, as well as rule files for assembling a valid instance model of a conformant circuit.

### FILES WITHIN PROJECT INPUT.1

#### DATASETS FOLDER

Every reference example is driven by two dataset files:

- A circuit definition file
- A power value file

These two files are Excel spreadsheets with predefined worksheets. They can have any name with the .xlsx extension. The location of the files need not be in the datasets folder, but is there for convenience. The structure of the two dataset files is shown in the Appendix.

#### OUTPUTS FOLDER

This folder is where the resulting reference data file will be found after running the script, with the filename given by the user, and a .ttl extension. Once it is produced, it can be browsed and queried like any other OWL file. It contains the full FSGIM schema (in OWL), the complete rule base used in generating the instance data, the reference instance data for the circuit and power readings provided in the dataset files, and a SPARQL query library for use when browsing the data. All of this data is generated so that the user can examine the schema, the rules used to generate the data, and the data itself in one session.

#### BUILDREFERENCESET.SMS.TTL

This SPARQLMotion script is the top-level routine that generates the reference data set. Its operation is described in the System Execution section of this report.

#### CIRCUITDATA.XLSX

This is a temporary file, copied from the user-defined circuit definition file and given this name so it can be found by later routines in the script.

#### MAPCIRCUITTOFSGIM.TTL

This file contains the SPINMap mappings to convert the RDF triple version of the circuit dataset spreadsheet into instances of FSGIM classes.

#### MAPPOWERTOFGIM.TTL

Similarly, this file defines the mappings for the power readings in the user-supplied power dataset to convert into instances of FSGIM measurement classes.

#### POWERDATA.XLSX

This is a temporary file, copied from the user-defined power readings file and given this name so it can be found by later routines in the script.

#### RDFCIRCUITDATA.TTL

This is an intermediate file, containing the RDF triple version of the circuit dataset spreadsheet.

#### RDFPOWERDATA.TTL

This is an intermediate file, containing the RDF triple version of the power dataset spreadsheet.

#### FILES WITHIN PROJECT SGT.6

#### AGGREGATIONS.TTL

This rules file contains all the aggregation rules. These rules are fired once all the instances have been created by the rules in initializer.ttl.

#### COMPONENT-COLLECTION-ALL.TTL

This file is produced by transforming the normative FSGIM standard from UML into OWL using a system developed under a prior grant<sup>2</sup>.

#### DEFAULTS.TTL

This file contains rules that set default values of various properties and objects if they are not specified by the user, such as:

- 002 Set default uncertainty to 0.1
- 009 Set EligibleCurtableRealDemand to ActualRealDemand if not specified
- 010 Set EligibleCurtableReactiveDemand to ActualReactiveDemand if not specified
- 011 Set MaximumCurtableRealDemand to ActualRealDemand if not specified
- 012 Set MaximumCurtableReactiveDemand to ActualReactiveDemand if not specified

#### EXPLANATIONS.TTL

This file defines additional classes and properties not found in the FSGIM standard that are used to hold intermediate calculation values and explanations that are useful for an implementer. This mechanism to automatically populate data-driven explanations is quite general and represents a new advancement in the use of reasoners for conformance testing of standards.

#### INFERENCES.TTL

This intermediate file is generated when reasoning using the rules found in initializer.ttl. It contains the FSGIM class instances that define the circuit and all mandatory supporting objects, but not the power readings or aggregations.

#### INITIALIZER.TTL

This file contains the rules defined to create all of the FSGIM class instances (but not the aggregations), driven by the data provided in the circuit dataset file.

#### RULES.TTL

This file contains the specialized SPIN function definitions needed to verify the existence of entities in a circuit, to verify the completion of intermediate calculations, and to calculate the aggregations. These functions are used by the rules defined in aggregations.ttl and initializer.ttl.

#### SANDBOX.TTL

This is the top-level OWL ontology that simply imports most of the other OWL files. It consists only of prefix and ontology declaration statements.

#### SGTQUERIES.TTL

This file contains the specialized SPARQL queries for the user to interrogate the inferred dataset. The most useful of these for browsing the reference data set is the first query: “# 001 List all defined instances”.

#### TWEAKS.TTL

---

<sup>2</sup> NIST Grant 60NANB11D170, Semantic Harmonization of Smart Grid Concepts

As described earlier, a small number of manual definitions are also supplied in this file, such as explicitly named composition associations from EnergyRouter to RouterConnectionPoint. These tweaks were needed to address implementation requirements.

## SYSTEM EXECUTION

This testing system is built as a SPARQLMotion script that takes two input arguments (the circuit and power spreadsheets) and the name of the reference data file to be created. The generation process is totally automatic at that point, with reasoning applied to specific rulesets at defined stages. The rules themselves are not written as procedural code. Instead, all of the data is brought together and a reasoning engine makes inferences about that data, creating new data as a result. The flowchart for this is shown in Figure 2.

The aggregation rulesets are a core part of the SPC 201 standard. Properly codifying these rulesets in the form of SPIN statements takes careful design attention in order to ensure their fidelity and traceability to the standard, proper sequencing for execution, and maintainability. 90 SPIN rules were written. The SPIN rules fall into two broad categories, instantiation rules and aggregation rules. Importantly, all of the rules also create explanations

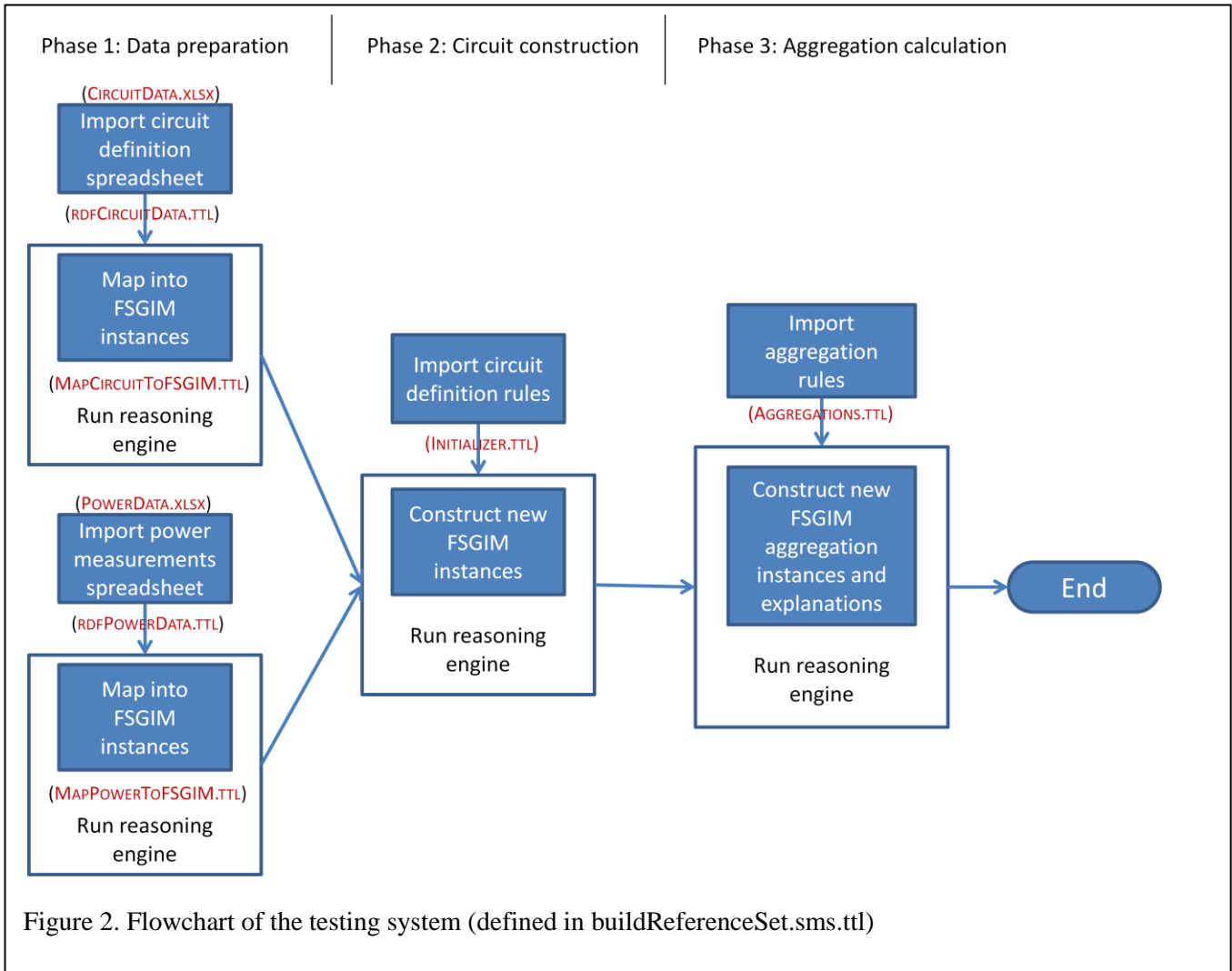


Figure 2. Flowchart of the testing system (defined in buildReferenceSet.sms.ttl)

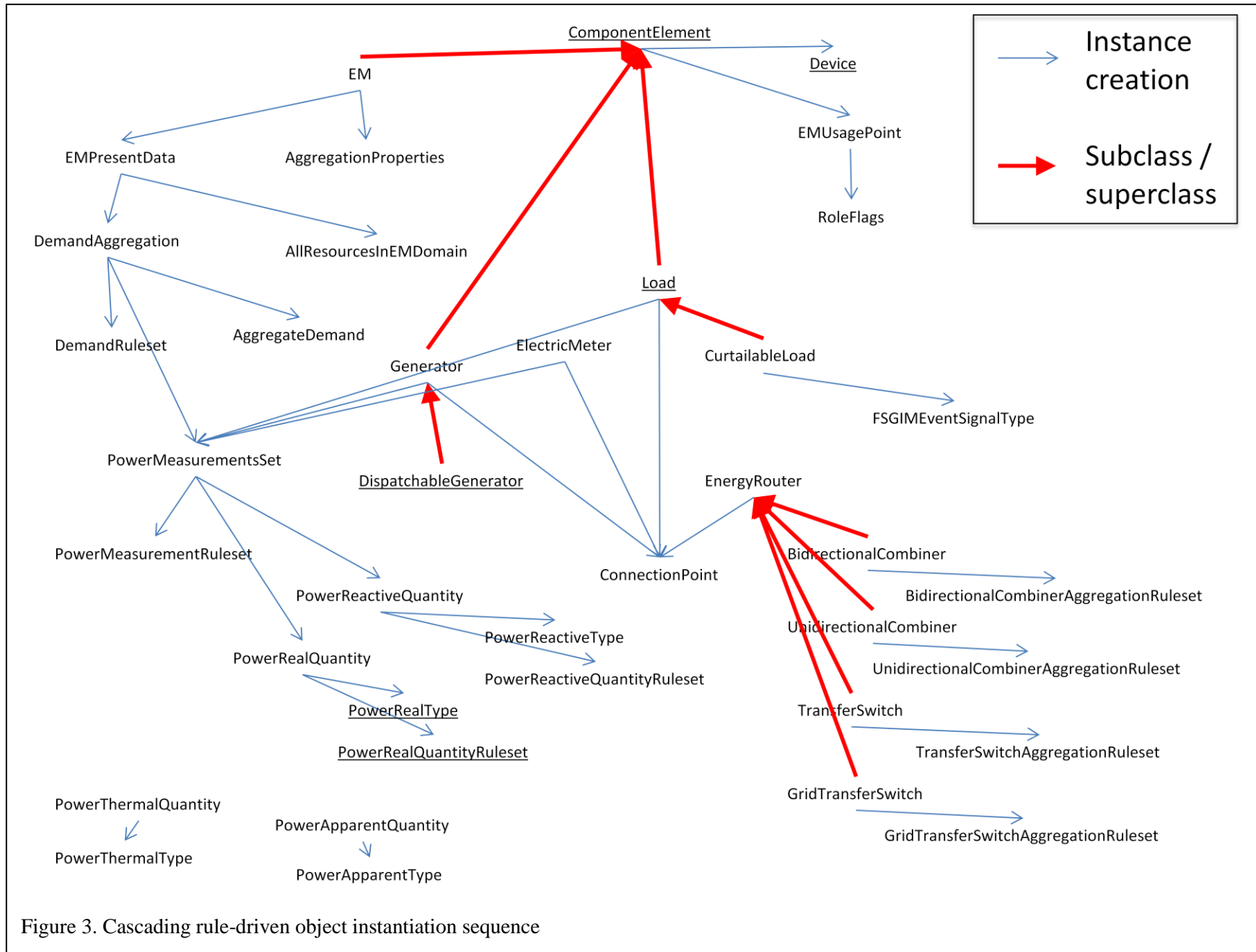
about what is generated, to benefit implementers of the standard.

The order of rule execution takes careful design. Rule firing takes place in the context of the class for which the rule is defined. All instances of that class are examined against the conditions of each of the parent class rules, and new inferences are made if the conditions are satisfied. This process is repeated for all classes containing rules. Once that is complete, if any new inferences are made, the rules for all classes are again examined. This process continues until no new inferences are made. For the circuit construction phase, this results in an iterative, cascading process, as illustrated in Figure 3. For example, when an EM is instantiated, rules will fire to create instances of `EMPresentData` and `AggregationProperties`. In turn, with the creation of an `EMPresentData` instance, new instances of `DemandAggregation` and `AllResourcesInEMDomain` are created, and so forth. Furthermore, since the EM class is a subclass of `ComponentElement`, the creation of an EM instance also triggers the creation of `Device` and `EMUsagePoint` instances.

The rules for calculating the aggregations must not execute until all the circuit object instances have been created, so all the object instantiation iterations (driven by the rules in `Initializer.ttl`) must be completed first. Then the process can begin again with the aggregation rules defined in `Aggregations.ttl` (Phase 3 in Figure 2).

Execution of the aggregation rules also requires careful design. Since aggregations are recursive in nature, with higher-level EM aggregations depending on subordinate EM aggregations, the entire aggregation process must be performed bottom-up. This is accomplished by having each EM raise a flag once all its aggregation calculations are complete. When a higher-level EM needs to calculate an aggregation, it must wait until all the subordinate EMs have raised their completion flags. So initially, the only EMs that can calculate aggregations are those without subordinate EMs, and then execution trickles up the EM hierarchy.

One significant and important difference between the model theories underlying UML and the OWL language used for this work is that OWL is based upon set theory. The implication of this is that the “redefines” association type in UML cannot be exactly represented in OWL. Rather, the original association and the “redefined” association are both present in OWL, since the subclass (or more properly, the sub-set) members are still members of the super-set. In practice, this simply means that the implementation ignores the associations that were defined by the super-set, and uses the local definitions (and constraints) only. To avoid a constraint violation for the unused super-set associations, any mandatory association constraints (i.e. cardinalities of 1 or more) defined by the super-set when a “redefines” exists, had to be manually removed from the OWL files.





## THE TESTING AND EXPLANATION ENVIRONMENT

Once the reference data set has been created, the results are bundled into a single file, along with the entire FSGIM model (expressed in OWL) and the rule sets that generated the reference data (expressed as SPIN rules). This file can be browsed and queried using TopQuadrant's TopBraid Composer Free, available at <http://www.topquadrant.com/downloads/topbraid-composer-install/>. For the purposes of this report, the power aggregation example found in Section 4.5 of the FSGIM standard User Guide is used, and is shown in Figure 4. A screenshot of a browsing session is shown in Figure 5. At the bottom of Figure 5 is the output of a predefined

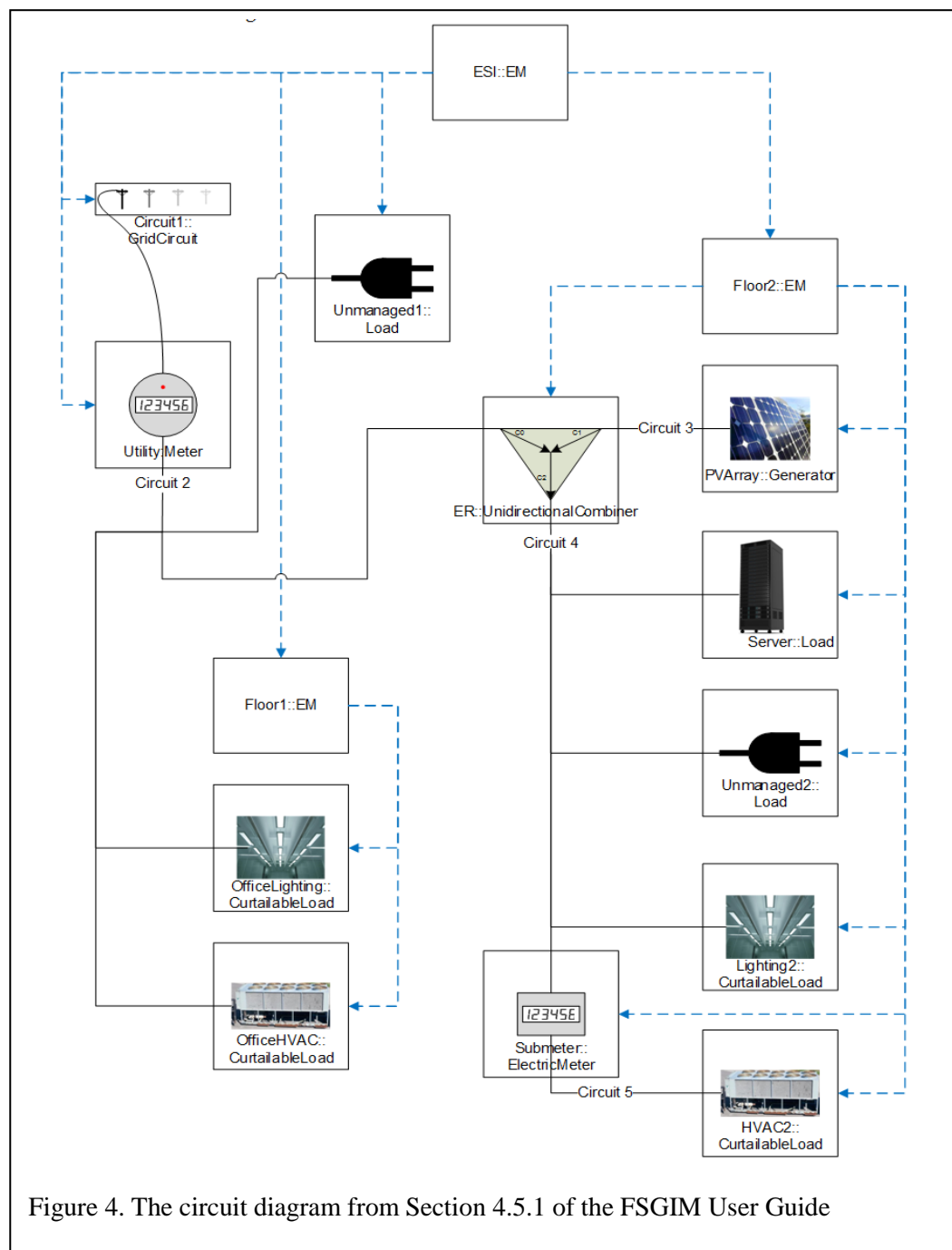


Figure 4. The circuit diagram from Section 4.5.1 of the FSGIM User Guide

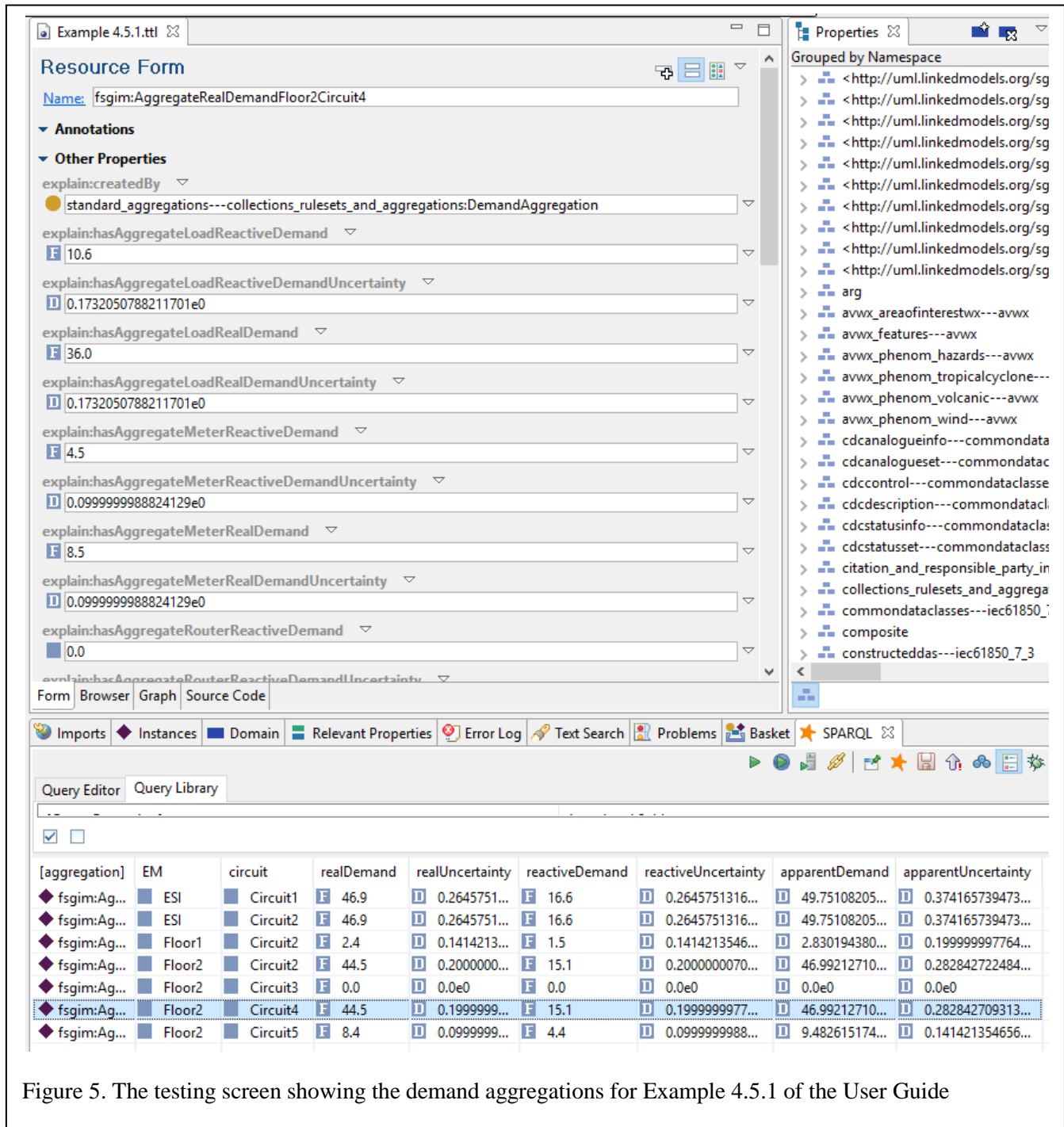


Figure 5. The testing screen showing the demand aggregations for Example 4.5.1 of the User Guide

query summarizing all the present demand aggregations, one for each Circuit managed by an energy manager (EM). The top of Figure 5 shows the window where the details of the calculations are explained. It shows that the aggregation for Circuit 4 under the energy manager named Floor2 has a contribution from one or more aggregated subordinate Loads (because the properties explain:hasAggregateLoad...Demand have non-zero values) and from one or more aggregated Meters (because the properties explain:hasAggregateMeter...Demand have non-zero values). Scrolling further down (Figure 6) shows which Loads and Meters were involved in the calculation. The

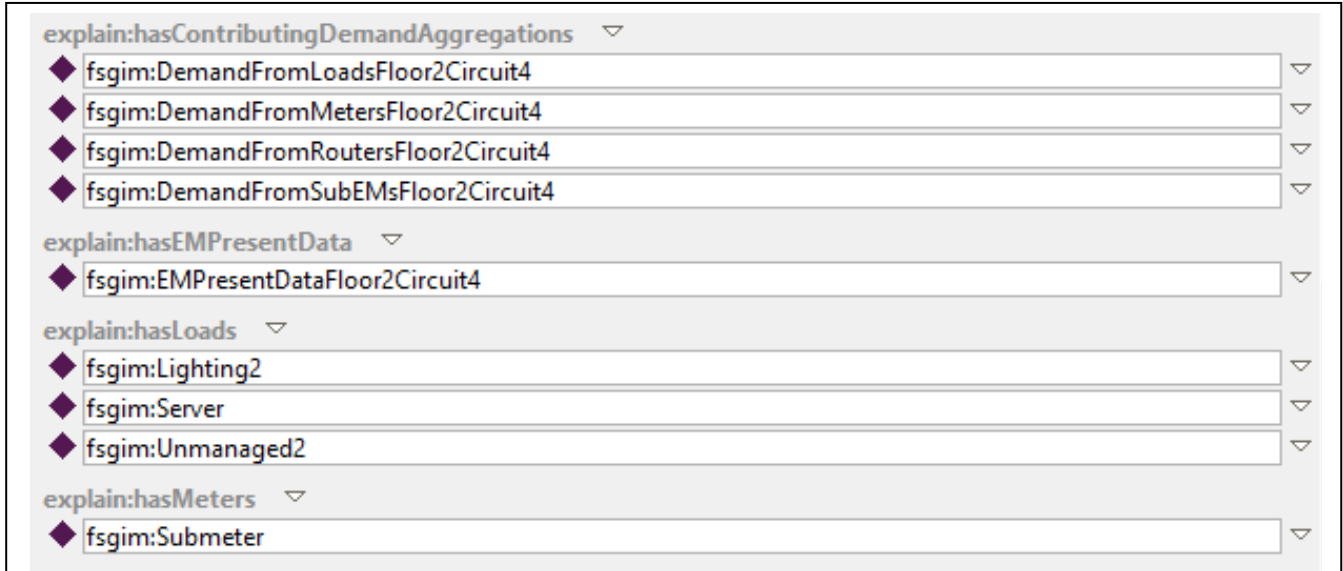


Figure 6. More details of the information supporting the aggregation calculation

purple diamonds in the figure indicate that these are instances of the model. Clicking on them allows the user to inspect them. In this way it is possible to navigate through the graph of connected instances. Clicking on fsgim:Submeter, for example, produces the display shown in Figure 7. All the properties prefixed by “explain:” are not part of the FSGIM standard and were created to help explain details for the user. Note, for example, that the explanatory property explain:measuresDemand has the value of “true”. This indicates that the system determined there are no downstream generators in the circuit, so the meter reading can be used to measure demand as well as net demand. All the other properties are part of the standard - most of the values are automatically created by the rule system.

**Resource Form**

Name: fsgim:Submeter

▼ Annotations

▼ Other Properties

explain:measuresDemand ▼  
B true

explain:metersCircuit ▼  
◆ fsgim:Circuit5

explain:metersPowerOf ▼  
◆ fsgim:HVAC2

device---device\_and\_model\_components:hasMRID ▼  
fsgim:Submeter


device---device\_and\_model\_components:hasName ▼  
Submeter

device---device\_and\_model\_components:hasNameType ▼  
Meter

device---device\_and\_model\_components:hasNameTypeAuthority ▼  
ASHRAE201 Standard

Figure 7. Inspecting a particular instance

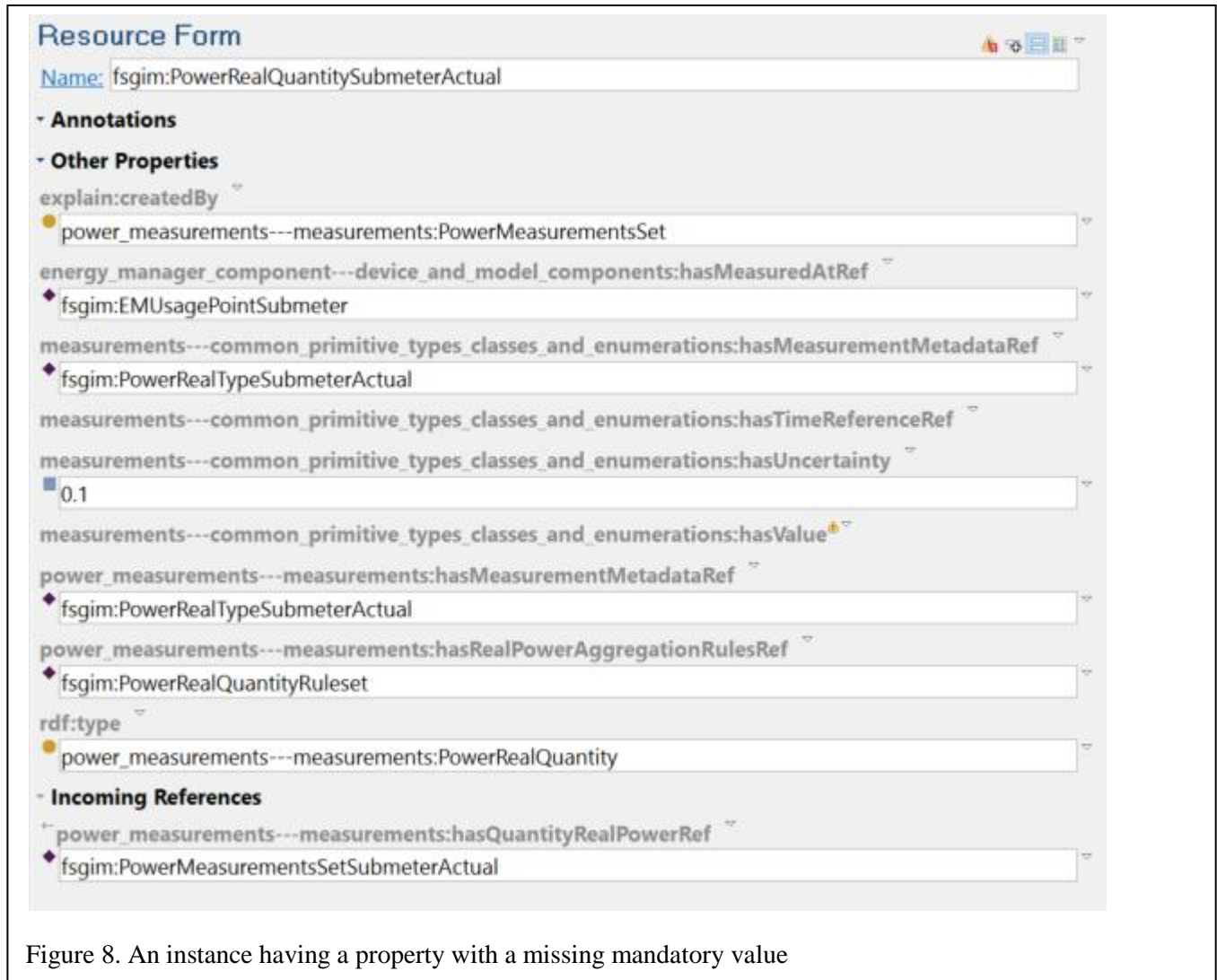
#### INTEGRITY TESTING

One kind of testing that is relatively straightforward to perform in this environment is integrity testing – that is, checking that no model constraints have been violated. The most straightforward constraints to test are cardinality constraints and range constraints. These are highlighted in the Resource Form with a small warning icon, . An example is shown in Figure 8, where the same fsgim:Submeter instance as in Figure 7 was not supplied with a power reading (a mandatory value) in the input file. Hovering the mouse over either yellow triangle pops up an explanation about what constraint is being violated.

Cardinality constraints are automatically generated from the normative UML model for FSGIM, so it is very easy to check a given dataset for this kind of integrity once it has been imported into the environment. Other, more complex constraints can also be written, of course. There is an unending list of possible constraints that could be defined to check for any of a large number of conditions. Our initial approach, however, is to first go the route of generating conforming reference data sets that can be compared with externally created data sets claiming to represent the same situation. This approach, if carried further, would compare the reference data with the test data and raise a flag whenever two values differ. The “explain” properties described here are intended to help with diagnosing *why* two such values differ.

## AGGREGATION CALCULATION

Reference data sets are automatically created, along with detailed explanations, all derived from a relatively small defining dataset (described in the Appendix). A side-by-side comparison can be made, either automatically or manually, to highlight where the two resulting datasets differ, and the explanations can be examined to figure out why the reference data has the values it does. In some ways, this is like an augmented debugging tool, except rather than just showing the correct answer, the system shows the intermediate work along the way. Returning to Figure 5, you can see that 36kW of real power and 10.6kW of reactive power are drawn by the Loads, and 8.5kW



of real power and 4.5kW of reactive power is measured by all Meters on the circuit. Each of these objects can be examined in more detail by simply clicking on the appropriate name, thus allowing the user to navigate through the object model. In general, each clause in each aggregation ruleset (see Figure 9) can be treated in the same way.

<i>Ruleset</i>
<b>DemandRuleset</b>
name = DemandRuleset nameType = Standard Rulesets nameTypeAuthority = ASHRAE 201 Standard
<p><i>notes</i></p> <p>Calculate "metered demand":</p> <ul style="list-style-type: none"> <li>For each direct subordinate that is an instance of the ElectricMeter Class where <i>only</i> <ol style="list-style-type: none"> <li>direct subordinate instances of the Load Class (including instances of the CurtailableLoad Class) and/or</li> <li>direct subordinate instances of the EM Class where EM.hasElectricalGenerators is False</li> </ol> are connected to its output ConnectionPoint and where ElectricMeter.powerReading is available, sum ElectricMeter.powerReading.</li> </ul> <p>Calculate "demand from unmetered loads":</p> <ul style="list-style-type: none"> <li>For each direct subordinate that is an instance of the Load Class (including instances of the CurtailableLoad Class) that is not included in "metered demand" and where Load.actualDemand is available, sum Load.actualDemand.</li> </ul> <p>Calculate "demand from energy managers":</p> <ul style="list-style-type: none"> <li>For each direct subordinate that is an instance of the EM Class that is not included in "metered demand" and where EMPresentData.presentAggregateDemand is available, sum the subordinate EMPresentData.presentAggregateDemand.aggregateQuantity.</li> </ul> <p>Calculate final result:</p> <ul style="list-style-type: none"> <li>Sum "metered demand", "demand from unmetered loads", and "demand from energy managers."</li> </ul>

Figure 9. The demand ruleset standard aggregation.

**ACRONYMS**

<b>Acronym</b>	<b>Definition</b>	<b>URL</b>
ASHRAE	American Society of Heating, Refrigerating and Air-Conditioning Engineers	<a href="http://www.ashrae.org/">http://www.ashrae.org/</a>
OMG	Object Management Group	<a href="http://www.omg.org">http://www.omg.org</a>
OWL	Ontology Web Language (W3C)	<a href="http://www.w3.org/TR/owl2-overview">http://www.w3.org/TR/owl2-overview</a>
RDF	Resource Description Framework (W3C)	<a href="http://www.w3.org/RDF/">http://www.w3.org/RDF/</a>
SPC201P	ASHRAE Standard Project Committee 201 (SPC 201) Facility Smart Grid Information Model	<a href="http://spc201.ashraepcs.org/">http://spc201.ashraepcs.org/</a>
SPARQL	SPARQL Query Language for RDF	<a href="http://www.w3.org/TR/rdf-sparql-query">http://www.w3.org/TR/rdf-sparql-query</a>
SPIN	SPARQL Inferencing Notation	<a href="http://www.w3.org/Submission/2011/02/">http://www.w3.org/Submission/2011/02/</a>
UML	Unified Modeling Language (OMG)	<a href="http://www.uml.org/">http://www.uml.org/</a>
W3C	World-Wide Web Consortium	<a href="http://www.w3.org/">http://www.w3.org/</a>
XSD	XML Schema Definition (W3C)	<a href="http://www.w3.org/TR/xmlschema11-1/">http://www.w3.org/TR/xmlschema11-1/</a>

## APPENDIX

Listed below are the structures of the two spreadsheets used to drive the reference data set generator. The data shown corresponds to the circuit shown in Figure 4.

### CIRCUIT DATASET

This spreadsheet contains 12 worksheets, shown below. With the exception of the first worksheet which is used for version control, each worksheet corresponds to a class depicted graphically in a circuit diagram such as Figure 4, namely: GridCircuit, Circuit, EM, CurtailableLoad, Load, Generator, DispatchableGenerator, ElectricMeter, UnidirectionalCombiner, BidirectionalCombiner and TransferSwitch. The first row of each worksheet is locked. The user fills in the appropriate values in rows 2 onward.

#### WORKSHEET TESTCASE

A user-defined test case name and version

	A	B
1	TestCaseName	TestCaseVersion
2	Example 4.5.1	1

#### WORKSHEET GRIDCIRCUIT

The name of each instance of class GridCircuit

	A
1	GridCircuitName
2	Circuit1

#### WORKSHEET CIRCUIT

The name of each instance of class Circuit

	A
1	CircuitName
2	Circuit2
3	Circuit3
4	Circuit4
5	Circuit5



WORKSHEET EM

Order of rows is not important in any of the worksheets, and a given entity can appear in multiple rows. This sheet captures which entities are supervised by each energy manager (EM class). So, for example, energy manager ESI supervises an EM named Floor1, a Load named Unmanaged1, a Meter named Utility, and an energy manager named Floor2.

	A	B	C	D	E	F
1	EMName	hasEM	hasLoad	HasMeter	hasGenerator	hasEnergyRouter
2	ESI	Floor1	Unmanaged1	Utility		
3	Floor2		HVAC2	Submeter	PVArray	ER
4	Floor2		Lighting2			
5	Floor2		Server			
6	ESI	Floor2				
7	Floor1		OfficeLighting			
8	Floor1		OfficeHVAC			
9	Floor2		Unmanaged2			

WORKSHEET CURTAILABLELOAD

Every Load (and CurtailableLoad) has an input connection to a Circuit, which is identified in this and the next worksheet.

	A	B
1	CurtailableLoadName	CurtailableLoadInput
2	Lighting2	Circuit4
3	HVAC2	Circuit5
4	OfficeLighting	Circuit2
5	OfficeHVAC	Circuit2

WORKSHEET LOAD

	A	B
1	LoadName	LoadInput
2	Server	Circuit4
3	Unmanaged1	Circuit2
4	Unmanaged2	Circuit4

WORKSHEET GENERATOR

Every Generator (and DispatchableGenerator) has an output connection to a Circuit, identified in this worksheet. A Generator has a mandatory attribute indicating what kind of storage capability it has, which can be one of “none”, “electricalStorage” or “thermalStorage”. For other reasons not relevant to this discussion, these values are uniquely identified in this testing system as “ST\_None”, “ST\_ElectricalStorage”, and “ST\_ThermalStorage”.

	A	B	C
1	GeneratorName	GeneratorOutput	hasGeneratorStorage
2	PVArray	Circuit3	ST_None

WORKSHEET DISPATCHABLEGENERATOR

	A	B	C
4	DispatchableGeneratorName	DispatchableGeneratorOutput	hasDispatchableGeneratorStorage
5			

WORKSHEET ELECTRICMETER

An ElectricMeter has both an input connection (nearer to the Grid) and an output connection (further from the Grid). The Circuit for each is identified here.

	A	B	C
1	ElectricMeterName	ElectricMeterInput	ElectricMeterOutput
2	Submeter	Circuit4	Circuit5
3	Utility	Circuit1	Circuit2

ROUTERS

There are three worksheets corresponding to the three subclasses of Routers. Each Router has three connection points as described in the FSGIM standard. The Circuits connected to each is identified here. In addition, there are four mandatory Boolean properties per connection, and one enumerated property. For each connection, the properties are:

- mayBeInput
- mayBeOutput
- mayBeBidirectional
- mayBeDisconnected
- presentState (either “input” or “output”, encoded here as “CPS\_Input” and “CPS\_Output”)

The abbreviations are not part of the standard, and were invented solely for these spreadsheets in the interests of having manageable column widths. They can be decoded as follows:

- UR, BR or TR – Unidirectional Combiner, Bidirectional Combiner, or Transfer Switch
- CP0, CP1 or CP2 – Identifying the ConnectionPoint being described
- MI – MayBeInput
- MO – MayBeOutput
- MB – MayBeBidirectional

- MD – MayBeDisconnected
- PS – presentState

The example of Figure 4 has only a UnidirectionalCombiner, so only one of the three worksheets has any values.

#### WORKSHEET UNIDIRECTIONALCOMBINER

	A	B	C
1	UnidirectionalCombinerName	UnidirectionalCombinerConnectionPoint0	UnidirectionalCombinerConnectionPoint1
2	ER	Circuit2	Circuit3

D	E	F	G	H	I
UnidirectionalCombinerConnectionPoint2	URCP0MB	URCP0MD	URCP0MI	URCP0MO	URCP0PS
Circuit4	FALSE	FALSE	TRUE	FALSE	CPS_Input

J	K	L	M	N	O	P	Q	R	S
URCP1MB	URCP1MD	URCP1MI	URCP1MO	URCP1PS	URCP2MB	URCP2MD	URCP2MI	URCP2MO	URCP2PS
FALSE	FALSE	TRUE	FALSE	CPS_Input	FALSE	FALSE	TRUE	FALSE	CPS_Output

#### WORKSHEET BIDIRECTIONALCOMBINER

	A	B	C
1	BidirectionalCombinerName	BidirectionalCombinerConnectionPoint0	BidirectionalCombinerConnectionPoint1
2			

D	E	F	G	H	I
BidirectionalCombinerConnectionPoint2	BRCP0MB	BRCP0MD	BRCP0MI	BRCP0MO	BRCP0PS

J	K	L	M	N	O	P	Q	R	S
BRCP1MB	BRCP1MD	BRCP1MI	BRCP1MO	BRCP1PS	BRCP2MB	BRCP2MD	BRCP2MI	BRCP2MO	BRCP2PS

#### WORKSHEET TRANSFERSWITCH

	A	B	C
1	TransferSwitchName	TransferSwitchConnectionPoint0	TransferSwitchConnectionPoint1
2			

D	E	F	G	H	I
TransferSwitchConnectionPoint2	TRCP0MB	TRCP0MD	TRCP0MI	TRCP0MO	TRCP0PS

J	K	L	M	N	O	P	Q	R	S
TRCP1MB	TRCP1MD	TRCP1MI	TRCP1MO	TRCP1PS	TRCP2MB	TRCP2MD	TRCP2MI	TRCP2MO	TRCP2PS

## POWER DATA

The power spreadsheet is much simpler in structure, and has only two worksheets.

### WORKSHEET TESTCASE

A user-defined test case name and version

	A	B
1	TestCaseName	TestCaseVersion
2	Example 4.5.1	1

### WORKSHEET COMPONENTELEMENT

This worksheet simply provides the real and reactive values for demand, supply, or net demand for any ComponentElement where that information is available (as appropriate for the given ComponentElement). Values for Loads are interpreted as Demand, values for Generators are interpreted as Supply, and values for Meters are interpreted as NetDemand. Note that the demand values for “Unmanaged1” and “Unmanaged2” can either be shown as zero, or not appear at all. These two Loads are intended in the example to represent plug loads of unknown demand. Thus, the aggregation algorithms cannot compute their values. The best that can be done is to use meter readings further upstream. This is discussed in the User Guide. Note also that Example 4.5.1 in the User Guide does not supply uncertainty values for the demand. The testing system defaults all unspecified power uncertainty values to 0.1 kW or kvar. Support for other units or magnitudes has not been implemented (yet) in this testing system.

	A	B	C	D	E
1	ComponentElementName	Real	RealUncertainty	Reactive	ReactiveUncertainty
2	Lighting2	1		0.8	
3	HVAC2	8.4		4.4	
4	PVArray	4		0	
5	Server	35		9.8	
6	Submeter	8.5		4.5	
7	Utility	46.5		18.1	
8	OfficeLighting	1		0.8	
9	OfficeHVAC	1.4		0.7	
10	Unmanaged1	0		0	
11	Unmanaged2	0		0	