PowerSystemModel (HORIZON®)
IEC 61131 Library for AcSELERATOR RTAC® Projects

SEL Automation Controllers
# Table of Contents

Section 1: PowerSystemModel (HORIZON®)

- Introduction .................................................................................. 3
- Supported Firmware Versions ......................................................... 7
- Enumerations ............................................................................... 7
- Structures .................................................................................. 7
- Classes ..................................................................................... 8
- Benchmarks ................................................................................ 43
- Examples .................................................................................... 44
- Log File Format ........................................................................... 69
- Release Notes .............................................................................. 70
Introduction

The PowerSystemModel library provides the ability to instantiate, describe, and connect different power system elements. With each scan of the logic engine task, it will collect the available measured information and determine which nodes are connected. The model provides a single voltage quantity for all devices connected without impedance (an electrical node) and a current for each branch that can be directly calculated using Kirchhoff’s current law.

The PowerSystemModel library works with the C37.118 synchrophasor stream to provide a more detailed snapshot of the system than can be provided by the raw data alone. Internally the model treats all data and parameters as three-phase data at base units (e.g., ohms and amperes) and it uses those three-phase data to perform its calculations. Each monitored input contains a quality_t structure. If validity inside this structure is set to anything other than GOOD, that measured value is not used in the calculations during this scan. Once all inputs have been validated, the model expands existing current and voltage data to as many nodes and branches as possible, and where sufficient data is available, provides a sanity check on the measured values and breaker states.

The PowerSystemModel library is primarily designed to support the following two use cases: extending system observability given a sparsely metered system and validating measurements and topology given a densely metered system. The breaker-and-a-half configuration, shown in Figure 1, is used to illustrate the different use cases.

![Figure 1: Differing Levels of Observability](image-url)
Introduction

Observability Extension: Figure 1a shows current meters installed on all three breakers. By combining these measurements and the topology information, the power system model can calculate the current injected into both of the unmetered lines.

Error Detection: Figure 1b shows current meters installed on two of the three breakers and both lines. By combining these measurements and the topology information, the power system model can calculate the current flow through the unmetered breaker and detect if a metering error has likely occurred.

Error Detection and Identification: Figure 1c shows fully redundant current. By combining these measurements and the topology information, the power system model can validate each of the redundant measurements, detect any inconsistencies, and provide an indicator that one of the meters within the collection is providing incorrect data.

Glossary

These terms are used throughout this document to describe the functionality provided by this library.

**Conducting Equipment**  Any piece of electrical equipment that is designed to carry current or that is conductively connected to the network. This does not include containers that hold this equipment. For example, a power transformer is not conducting equipment even though it can hold multiple power transformer ends which are themselves conducting equipment.

**Connectivity Node**  A representation of the connecting point for two or more terminals. The model will create these anywhere two or more terminals are connected via the bootstrap methods. At no time should the user ever instantiate a connectivity node.

**Model**  In this library, the model is a collection of conducting equipment that is connected and the data that describes that equipment.

**Power Transformer End**  A transformer winding. A given transformer may have two or more windings.

**Terminal**  The part of a piece of electrical equipment that is meant to connect it to other equipment. Every piece of conducting equipment has at least one terminal and all terminals should be attached to at least one other terminal at a connectivity node. These also serve as anchoring points for various measurement devices.

Placing a System Into the Model

As an example, the user desires to represent a substation using this library. The high-voltage side of the substation being modeled is represented by the one-line diagram shown in Figure 2:
To begin with, the user must identify the individual pieces of conducting equipment represented by this diagram. This defines what objects must be created within the library to model this system, as seen in Figure 3. Note that every piece of conducting equipment in the model has one or more terminals that are implicitly created with the object—the bus can be represented as only a single terminal.

To tie each of the pieces of equipment together, we introduce the idea of a connectivity node. These nodes provide a location to tie together as many terminals as desired. In the library the user does not need to create these nodes. They are created automatically when two or more terminals are connected. Figure 4 shows how this might be conveyed. As before, there is a connectivity node where the terminals of the three lines join the terminal of the bus.
Figure 5 shows the user indicating where data are fed into (classes named as Measurements) and read out of (classes named as Report) the model. Each of these I/O points is tied to a terminal that it monitors. The direction of current flow is standardized as being positive when flow is moving through the terminal from the conducting equipment into the connectivity node. For example; if current was flowing from left-to-right through a breaker, a measurement point on the left terminal would read as a negative value flowing into the conducting equipment, away from the connectivity node; and a measurement point on the right terminal would read as a positive value flowing away from the conducting equipment, into the connectivity node.

Though current and voltage values may be read in from measurements in whatever units desired, they must be scaled to volts or amperes with the correct sign for use in the model. Measurement classes provide a scalar that must be correctly populated to translate the units and directionality being fed into the model from the synchrophasor stream into the required magnitudes and directions.

Finally, some objects need to be part of a container to define their interaction with other objects. For example, all conducting equipment must be placed in a nominal voltage and transformer ends must be placed in a transformer with other transformer ends, as shown in Figure 6.
Supported Firmware Versions

You can use this library on any device configured using ACSELRATOR RTAC® SEL-5033 Software with firmware version R143 or higher.

Version 3.5.0.0 can be used on RTAC firmware version R132 and higher.

Enumerations

Enumerations make code more readable by allowing a specific number to have a readable textual equivalent.

enum_ValueSource

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNAVAILABLE</td>
<td>The value is not measured and cannot be calculated by the installed system.</td>
</tr>
<tr>
<td>MEASURED</td>
<td>The value is reported exactly as measured.</td>
</tr>
<tr>
<td>DIRECT_ASSOCIATION</td>
<td>The value was achieved through comparing like values connected without impedance.</td>
</tr>
<tr>
<td>CALCULATED</td>
<td>The value was achieved through linear extrapolation based on provided parameters.</td>
</tr>
<tr>
<td>UNTRUSTED</td>
<td>The value measured here conflicts with other measured values or the provided parameters. This can be set if a measured value is too different from a calculated value or switch that reports open has a calculated current.</td>
</tr>
</tbody>
</table>

enum_WindingConnection

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELTA</td>
<td>A Delta winding.</td>
</tr>
<tr>
<td>WYE</td>
<td>A Wye winding.</td>
</tr>
<tr>
<td>ZIGZAG</td>
<td>A ZigZag winding.</td>
</tr>
<tr>
<td>WYE_NEUTRAL</td>
<td>A Wye winding with neutral brought out for grounding.</td>
</tr>
<tr>
<td>ZIGZAG_NEUTRAL</td>
<td>A ZigZag winding with neutral brought out for grounding.</td>
</tr>
<tr>
<td>AUTO</td>
<td>An Autotransformer common winding.</td>
</tr>
<tr>
<td>INDEPENDENT</td>
<td>An independent winding for single-phase connections.</td>
</tr>
</tbody>
</table>

Structures

Structures provide a means to group together several memory locations (variables), making them easier to manage.
This library makes use of several acSELerator RTAC Data Types for data input, output, and storage. The layout of these structures can be found in the acSELerator RTAC SEL-5033 Software Instruction Manual.

CMV A communications structure for moving phasors.
MV A communications structure for moving analog values.
quality_t A communications structure for indicating data health.
SPS A communications structure for moving binary points.
vector_t A structure for storing phasors as an angle and a magnitude.

**Classes**

Classes are a particular implementation of a function block. They are generally initialized using bootstrap methods and provide methods and properties, which normal function blocks do not provide.

**class_PowerSystemModel (Class)**

This class contains the working algorithms for the model. It stores the connections of objects to each other, controls the order of operations each scan, and provides a centralizing point for all other features.

For the model to do its work, all elements must be tied to it before calling run. In general, this means that all terminals must be tied first using bootstrap_ConnectTerminals(). Then individual objects should be configured using their assorted bootstrap_Set and bootstrap_Configure methods. Finally, objects can be grouped in containers and transformers and meters can be added using bootstrap_Add methods.

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td>STRING</td>
<td>The name of the log file for this model’s initialization.</td>
</tr>
<tr>
<td>ABCRotation</td>
<td>BOOL</td>
<td>The rotation of the phases in this model. True indicates that after the A-phase current crosses a reference angle, B-phase current will cross it next.</td>
</tr>
</tbody>
</table>

**bootstrap_ConnectTerminals (Method)**

Call this method to connect two terminals to each other. A terminal can be used as an argument to this method more than once to connect more than two terminals to each other.

If Terminal A is connected to Terminal B and then Terminal A is connected to Terminal C, Terminal B is implicitly connected to Terminal C and the two do not need to be connected by an explicit call to this method.

This is the first bootstrap method to be called. It must be called after all objects are instantiated and before any other bootstrap method and before Run().
Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pt_terminal1</td>
<td>POINTER TO class_Terminal</td>
<td>The first terminal to connect.</td>
</tr>
<tr>
<td>pt_terminal2</td>
<td>POINTER TO class_Terminal</td>
<td>The second terminal to connect.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if the terminals are connected together and added to the model.</td>
</tr>
</tbody>
</table>

Processing

This method is intended to be called before processing to link objects together in the model. It performs the following actions:

- Stores a reference to the model in both objects.
- Connects the two terminals together at a connectivity node.
- Connects both terminals to any other terminals already attached to the connectivity node.
- Returns FALSE if the references are not stored, either because the objects are no longer in the initialization phase or a terminal is already attached to another model.

bootstrap_FinalizeConnections (Method)

To ensure proper tying of all model objects, this method must be called after all terminals have been connected by bootstrap_ConnectTerminals and before calling any other bootstrap methods or Run(). It switches the model out of the terminal connection state in preparation for all other work.

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if the model has been successfully tied together.</td>
</tr>
</tbody>
</table>

Processing

This method prompts the model to perform the following actions:

- Disable the connection of additional terminals.
- Enable the insertion of objects (e.g., class_Breakers, class_ACLineSegments, and others) into containers (i.e. class_VoltageLevels and class_PowerTransformers).
- Build internal linking structures required for additional processing.
- Return FALSE if bootstrap_ConnectTerminals() has not been successfully called or if the system state prevented the final connections.
bootstrap_ValidateModel (Method)

This verifies the state of all objects and that the model itself can operate without error. Until this method is called, no work is done by the Run() method.

Upon completion of this method, the model is ready to write out a log file summarizing the configuration of the model. This file will be completed during the first few iterations of the calls to the Run() method. Its format can be seen in Log File Format on page 69.

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if the model is ready to monitor inputs and update outputs.</td>
</tr>
</tbody>
</table>

Processing

This method is intended to be called before ever calling Run() to finalize all model configuration. It prompts the model to perform the following actions:

- Disable the attachment of any additional containers.
- Check that all model objects have every reference required for calculations.
- Return FALSE if the model is unable to calculate model state due to a configuration error. Any such error will be described in the file filename.

Run (Method)

This method drives all work done by the model. It should be called once per task scan after all configuration information has been processed.

Processing

Each time Run() is called, it performs the following tasks:

- Checks that the model has been validated and deemed healthy.
- Updates the measurements from all inputs configured during bootstrap.
- Determines the observability of the system. This decides which current values can be calculated and which cannot based on the switch states and valid measurement values available.
- Calculates the current through all observable pieces of conducting equipment and the voltage at all observable terminals. If measurements beyond the minimum required for observability are available, the method uses all available information to generate a minimum mean squared error estimate.
- If measurements are deemed to be too far from calculated values, the resulting output will be flagged as UNTRUSTED. Likewise, any switch reporting as open with more than minimal current running through it will be reported as UNTRUSTED.
- Once all measurements are calculated and outputs are flagged, the time-aligned values will be placed in the outputs of all user-provided measurement points and all switches and breakers.
**class_ConductingEquipment**

This class is never instantiated by the user. This is a category of object that is extended to allow objects of multiple types to be treated as if they are the same for the purpose of grouping and analysis. Classes that extend this class can be added to class_VoltageLevels.

**class_Measurement**

This class is never instantiated by the user. This is a category of object that is extended to allow objects of multiple types to be treated as if they are the same for the purpose of grouping and analysis. Classes that extend this class can be added to terminals for both input and output of measurement values.

**class_Terminal**

The point on a piece of conducting equipment that connects to other conducting equipment. Terminals are used to join all objects together and provide explicit points of contact for measurement units.

This class is never instantiated by the user. All instantiated conducting equipment already have terminals as member objects. The user must be aware that terminals exist to facilitate calling bootstrap methods to tie the model together and to attach measurement points.

**bootstrap_AddMeasurement (Method)**

Call this method to tie a measurement point to a specific terminal.

This is the last type of bootstrap method to be called. It must be called after all terminals are connected and all configure and set bootstrap methods have been completed.

**Inputs/Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>measurement</td>
<td>class_Measurement</td>
<td>The measurement to be applied to this terminal.</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>The measurement was successfully added to the terminal.</td>
</tr>
</tbody>
</table>

**Processing**

This method stores a reference to the provided measurement at this terminal unless the objects are no longer in the initialization phase or the terminal is attached to a bus or junction and the measurement is a current measurement. After being attached, these measurement inputs and outputs will be used with all operations acting on the model.
class_Switch

Instantiate one instance of this class for any non-load breaking switch in the model. The instance tracks the open-close state of that device. Before this class can be used, one or both of its bootstrap methods must be called.

If none of the variables being monitored are healthy on a given scan, the switch will be analyzed per its *TypicallyClosed* value and the *StO_Quality* will be set to UNAVAILABLE. If only one value is available, because only one bootstrap method is called or only one monitored value is healthy, that value will be used as the open-close state. If both values are bootstrapped and healthy and the two values conflict, the switch will be analyzed per its *TypicallyClosed* value and the *StO_Quality* will be set to UNTRUSTED.

Extended Classes

Extending a class provides full inheritance of all that classes features (methods, variables, properties). A class may only extend one other class directly, but class extension can be tiered indefinitely.

➢ class_ConductingEquipment

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING</td>
<td>The name of this object on the power system.</td>
</tr>
<tr>
<td>Description</td>
<td>STRING</td>
<td>A human-readable description of this object.</td>
</tr>
<tr>
<td>TypicallyClosed</td>
<td>BOOL</td>
<td>This switch is closed under normal operating conditions. This is the value that will be used if there is no communication with the switch.</td>
</tr>
</tbody>
</table>

Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pt_TerminalA</td>
<td>POINTER TO class_Terminal</td>
<td>The first terminal of this switch, used to attach it inside the model.</td>
</tr>
<tr>
<td>pt_TerminalB</td>
<td>POINTER TO class_Terminal</td>
<td>The second terminal of this switch, used to attach it inside the model.</td>
</tr>
<tr>
<td>StIn_IsClosed</td>
<td>BOOL</td>
<td>The switch is reporting as closed.</td>
</tr>
<tr>
<td>StIn_IsClosedQOk</td>
<td>BOOL</td>
<td>The channel communicating switch state is healthy.</td>
</tr>
<tr>
<td>StO_IsClosed</td>
<td>BOOL</td>
<td>The state the model calculates the switch to be in. The model may override an open condition if current is still detected across the link.</td>
</tr>
<tr>
<td>StO_Quality</td>
<td>enum_ValueSource</td>
<td>The confidence level in the <em>StO_IsClosed</em> flag.</td>
</tr>
</tbody>
</table>
bootstrap_ConfigureIsOpenInput (Method)

Call this method to provide a reference to the value the switch should monitor to detect an open condition.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.

Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stIn_IsOpen</td>
<td>SPS</td>
<td>The variable that will be monitored to determine the open status of this switch.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>The switch was successfully configured.</td>
</tr>
</tbody>
</table>

Processing

This method is intended to be called before processing to associate a data location with the switch. It performs the following actions:

- Stores a reference to the provided variable to be used later.
- Returns FALSE if the reference is not stored because the switch already had this bootstrap method called.

bootstrap_ConfigureIsClosedInput (Method)

Call this method to provide a reference to the value the switch should monitor to detect a closed condition.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.

Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stIn_IsClosed</td>
<td>SPS</td>
<td>The variable that will be monitored to determine the closed status of this switch.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>The switch was successfully configured.</td>
</tr>
</tbody>
</table>
Processing

This method is intended to be called before processing to associate a data location with the switch. It performs the following actions:

- Stores a reference to the provided variable to be used later.
- Returns FALSE if the reference is not stored because the switch already had this bootstrap method called.

class_Breaker

Instantiate one instance of this class for any load breaking device in the model. The instance tracks open-close state of that device. Before this class can be used, one or both of its bootstrap methods must be called.

If none of the variables being monitored are healthy on a given scan, the breaker will be analyzed per its TypicallyClosed value and the StO_Quality will be set to UNAVAILABLE. If only one value is available, because only one bootstrap method is called or only one monitored value is healthy, that value will be used as the open-close state. If both values are bootstrapped and healthy and the two values conflict, the breaker will be analyzed per its TypicallyClosed value and the StO_Quality will be set to UNTRUSTED.

Extended Classes

Extending a class provides full inheritance of all that classes features (methods, variables, properties). A class may only extend one other class directly, but class extension can be tiered indefinitely.

- class_ConductingEquipment
- class_Switch

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING</td>
<td>The name of this object on the power system.</td>
</tr>
<tr>
<td>Description</td>
<td>STRING</td>
<td>A human-readable description of this object.</td>
</tr>
<tr>
<td>TypicallyClosed</td>
<td>BOOL</td>
<td>This breaker is closed under normal operating conditions. This is the value that will be used if there is no communication with the breaker.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pt_TerminalA</td>
<td>POINTER TO class_Terminal</td>
<td>The first terminal of this breaker, used to attach it inside the model.</td>
</tr>
<tr>
<td>pt_TerminalB</td>
<td>POINTER TO class_Terminal</td>
<td>The second terminal of this breaker, used to attach it inside the model.</td>
</tr>
<tr>
<td>StIn_IsClosed</td>
<td>BOOL</td>
<td>The breaker is reporting as closed.</td>
</tr>
<tr>
<td>StIn_IsClosedQOk</td>
<td>BOOL</td>
<td>The channel communicating breaker state is healthy.</td>
</tr>
</tbody>
</table>
### Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StO_IsClosed</td>
<td>BOOL</td>
<td>The state the model calculates the breaker to be in. The model may override an open condition if current is still detected across the link.</td>
</tr>
<tr>
<td>StO_Quality</td>
<td>enum_ValueSource</td>
<td>The confidence level in the StO_IsClosed flag.</td>
</tr>
</tbody>
</table>

### bootstrap_ConfigureIsOpenInput (Method)

Call this method to provide a reference to the value the breaker should monitor to detect an open condition.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.

### Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stIn_IsOpen</td>
<td>SPS</td>
<td>The variable that will be monitored to determine the open status of this breaker.</td>
</tr>
</tbody>
</table>

### Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>The breaker was successfully configured.</td>
</tr>
</tbody>
</table>

### Processing

This method is intended to be called before processing to associate a data location with the breaker. It performs the following actions:

- Stores a reference to the provided variable to be used later.
- Returns FALSE if the reference is not stored because the breaker already had this bootstrap method called.

### bootstrap_ConfigureIsClosedInput (Method)

Call this method to provide a reference to the value the breaker should monitor to detect a closed condition.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.
Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sIn_IsClosed</td>
<td>SPS</td>
<td>The variable that will be monitored to determine the closed status of this breaker.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>The breaker was successfully configured.</td>
</tr>
</tbody>
</table>

Processing

This method is intended to be called before processing to associate a data location with the breaker. It performs the following actions:

- Stores a reference to the provided variable to be used later.
- Returns FALSE if the reference is not stored because the breaker already had this bootstrap method called.

class_EnergySource

This class is a terminating element. It represents an edge of the model being worked on. Instantiate one instance of this class anywhere it is desired to model all elements beyond this point as a metered source of electrical power.

Extended Classes

Extending a class provides full inheritance of all that classes features (methods, variables, properties). A class may only extend one other class directly, but class extension can be tiered indefinitely.

- class_ConductingEquipment

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING</td>
<td>The name of this object on the power system.</td>
</tr>
<tr>
<td>Description</td>
<td>STRING</td>
<td>A human-readable description of this object.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pt_Terminal</td>
<td>POINTER TO class_Terminal</td>
<td>The terminal of this source, used to attach it inside the model.</td>
</tr>
</tbody>
</table>
class_EnergyConsumer

This class is a terminating element. It represents an edge of the model being worked on. Instantiate one instance of this class anywhere it is desired to model all elements beyond this point as a metered electrical load.

Extended Classes

Extending a class provides full inheritance of all that classes features (methods, variables, properties). A class may only extend one other class directly, but class extension can be tiered indefinitely.

▸ class_ConductingEquipment

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING</td>
<td>The name of this object on the power system.</td>
</tr>
<tr>
<td>Description</td>
<td>STRING</td>
<td>A human-readable description of this object.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pt_Terminal</td>
<td>POINTER TO class_Terminal</td>
<td>The terminal of this load, used to attach it inside the model.</td>
</tr>
</tbody>
</table>

class_ShuntCompensator

This class is a terminating element. It represents an edge of the model being worked on. Instantiate one instance of this class anywhere it is desired to model a shunt capacitor, inductor, or resistor. The provided values are used during all operations on the model.

Inputs to this class must be in units of siemens.

Extended Classes

Extending a class provides full inheritance of all that classes features (methods, variables, properties). A class may only extend one other class directly, but class extension can be tiered indefinitely.

▸ class_ConductingEquipment

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING</td>
<td>The name of this object on the power system.</td>
</tr>
<tr>
<td>Description</td>
<td>STRING</td>
<td>A human-readable description of this object.</td>
</tr>
</tbody>
</table>
### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conductanceAPhase</td>
<td>LREAL</td>
<td>The real part of the shunt admittance of the A-phase.</td>
</tr>
<tr>
<td>susceptanceAPhase</td>
<td>LREAL</td>
<td>The reactive part of the shunt admittance of the A-phase.</td>
</tr>
<tr>
<td>conductanceBPhase</td>
<td>LREAL</td>
<td>The real part of the shunt admittance of the B-phase.</td>
</tr>
<tr>
<td>susceptanceBPhase</td>
<td>LREAL</td>
<td>The reactive part of the shunt admittance of the B-phase.</td>
</tr>
<tr>
<td>conductanceCPhase</td>
<td>LREAL</td>
<td>The real part of the shunt admittance of the C-phase.</td>
</tr>
<tr>
<td>susceptanceCPhase</td>
<td>LREAL</td>
<td>The reactive part of the shunt admittance of the C-phase.</td>
</tr>
<tr>
<td>conductanceABPhase</td>
<td>LREAL</td>
<td>The real part of the shunt admittance due to mutual coupling between the A-phase and the B-phase.</td>
</tr>
<tr>
<td>susceptanceABPhase</td>
<td>LREAL</td>
<td>The reactive part of the shunt admittance due to mutual coupling between the A-phase and the B-phase.</td>
</tr>
<tr>
<td>conductanceACPhase</td>
<td>LREAL</td>
<td>The real part of the shunt admittance due to mutual coupling between the A-phase and the C-phase.</td>
</tr>
<tr>
<td>susceptanceACPhase</td>
<td>LREAL</td>
<td>The reactive part of the shunt admittance due to mutual coupling between the A-phase and the C-phase.</td>
</tr>
<tr>
<td>conductanceBCPhase</td>
<td>LREAL</td>
<td>The real part of the shunt admittance due to mutual coupling between the B-phase and the C-phase.</td>
</tr>
<tr>
<td>susceptanceBCPhase</td>
<td>LREAL</td>
<td>The reactive part of the shunt admittance due to mutual coupling between the B-phase and the C-phase.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pt_Terminal</td>
<td>POINTER TO class_Terminal</td>
<td>The terminal of this compensator, used to attach it inside the model.</td>
</tr>
</tbody>
</table>

### class_ACLineSegment

Instantiate one instance of this class at each location where it is desired to model the connection between two points as having impedance. Each instance defaults to zero impedance and zero shunt admittance unless an appropriate bootstrapping method is called.
Extended Classes

Extending a class provides full inheritance of all that classes features (methods, variables, properties). A class may only extend one other class directly, but class extension can be tiered indefinitely.

- class_ConductingEquipment

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING</td>
<td>The name of this object on the power system.</td>
</tr>
<tr>
<td>Description</td>
<td>STRING</td>
<td>A human-readable description of this object.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pt_TerminalA</td>
<td>POINTER TO class_Terminal</td>
<td>The first terminal of this line, used to attach it inside the model.</td>
</tr>
<tr>
<td>pt_TerminalB</td>
<td>POINTER TO class_Terminal</td>
<td>The second terminal of this line, used to attach it inside the model.</td>
</tr>
</tbody>
</table>

bootstrap_SetNominalLineImpedance1Line (Method)

Call this method to set the line impedance using a 1-line model. Arguments of this method must be in units of ohms.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resistance</td>
<td>LREAL</td>
<td>The real part of the positive-sequence series impedance.</td>
</tr>
<tr>
<td>reactance</td>
<td>LREAL</td>
<td>The reactive part of the positive-sequence series impedance.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if the impedance was set based on the provided values.</td>
</tr>
</tbody>
</table>

Processing

This method sets the impedance of the line based on the provided values and returns TRUE, unless the impedance of the line was set previously or the model is no longer in the initialization phase.
bootstrap_SetNominalLineImpedance3Phase (Method)

Call this method to set the line impedance using three-phase data. Arguments of this method must be in units of ohms.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resistanceAPhase</td>
<td>LREAL</td>
<td>The real part of the series impedance of the A-phase.</td>
</tr>
<tr>
<td>reactanceAPhase</td>
<td>LREAL</td>
<td>The reactive part of the series impedance of the A-phase.</td>
</tr>
<tr>
<td>resistanceBPhase</td>
<td>LREAL</td>
<td>The real part of the series impedance of the B-phase.</td>
</tr>
<tr>
<td>reactanceBPhase</td>
<td>LREAL</td>
<td>The reactive part of the series impedance of the B-phase.</td>
</tr>
<tr>
<td>resistanceCPhase</td>
<td>LREAL</td>
<td>The real part of the series impedance of the B-phase.</td>
</tr>
<tr>
<td>reactanceCPhase</td>
<td>LREAL</td>
<td>The reactive part of the series impedance of the B-phase.</td>
</tr>
<tr>
<td>resistanceABPhase</td>
<td>LREAL</td>
<td>The real part of the series impedance due to mutual coupling between the A-phase and the B-phase.</td>
</tr>
<tr>
<td>reactanceABPhase</td>
<td>LREAL</td>
<td>The reactive part of the series impedance due to mutual coupling between the A-phase and the B-phase.</td>
</tr>
<tr>
<td>resistanceACPhase</td>
<td>LREAL</td>
<td>The real part of the series impedance due to mutual coupling between the A-phase and the C-phase.</td>
</tr>
<tr>
<td>reactanceACPhase</td>
<td>LREAL</td>
<td>The reactive part of the series impedance due to mutual coupling between the A-phase and the C-phase.</td>
</tr>
<tr>
<td>resistanceBCPhase</td>
<td>LREAL</td>
<td>The real part of the series impedance due to mutual coupling between the B-phase and the C-phase.</td>
</tr>
<tr>
<td>reactanceBCPhase</td>
<td>LREAL</td>
<td>The reactive part of the series impedance due to mutual coupling between the B-phase and the C-phase.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if the impedance was set based on the provided values.</td>
</tr>
</tbody>
</table>

Processing

This method sets the impedance of the line based on the provided values and returns TRUE, unless the impedance of the line was set previously or the model is no longer in the initialization phase.

bootstrap_SetNominalLineImpedanceWZeroSequence (Method)

Call this method to set the line impedance using a positive- and zero-sequence data model. Arguments of this method must be in units of ohms.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.
## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resistancePosSequence</td>
<td>LREAL</td>
<td>The real part of the positive-sequence series impedance.</td>
</tr>
<tr>
<td>reactancePosSequence</td>
<td>LREAL</td>
<td>The reactive part of the positive-sequence series impedance.</td>
</tr>
<tr>
<td>resistanceZeroSequence</td>
<td>LREAL</td>
<td>The real part of the zero-sequence series impedance.</td>
</tr>
<tr>
<td>reactanceZeroSequence</td>
<td>LREAL</td>
<td>The reactive part of the zero-sequence series impedance.</td>
</tr>
</tbody>
</table>

## Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if the impedance was set based on the provided values.</td>
</tr>
</tbody>
</table>

## Processing

This method sets the impedance of the line based on the provided values and returns TRUE, unless the impedance of the line was set previously or the model is no longer in the initialization phase.

### bootstrap_SetNominalShuntAdmittance1Line (Method)

Call this method to set the shunt admittance of the line using a one-line model. Arguments of this method must be in units of siemens.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.

## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conductance</td>
<td>LREAL</td>
<td>The real part of the positive-sequence shunt admittance.</td>
</tr>
<tr>
<td>susceptance</td>
<td>LREAL</td>
<td>The reactive part of the positive-sequence shunt admittance.</td>
</tr>
</tbody>
</table>

## Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if the admittance was set based on the provided values.</td>
</tr>
</tbody>
</table>

## Processing

This method sets the admittance of the line based on the provided values and returns TRUE, unless the admittance of the line was set previously or the model is no longer in the initialization phase.
bootstrap_SetNominalShuntAdmittance3Phase (Method)

Call this method to set the line’s shunt admittance using three-phase data. Arguments of this method must be in units of siemens.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conductanceAPhase</td>
<td>LREAL</td>
<td>The real part of the shunt admittance of the A-phase.</td>
</tr>
<tr>
<td>susceptanceAPhase</td>
<td>LREAL</td>
<td>The reactive part of the shunt admittance of the A-phase.</td>
</tr>
<tr>
<td>conductanceBPhase</td>
<td>LREAL</td>
<td>The real part of the shunt admittance of the B-phase.</td>
</tr>
<tr>
<td>susceptanceBPhase</td>
<td>LREAL</td>
<td>The reactive part of the shunt admittance of the B-phase.</td>
</tr>
<tr>
<td>conductanceCPhase</td>
<td>LREAL</td>
<td>The real part of the shunt admittance of the C-phase.</td>
</tr>
<tr>
<td>susceptanceCPhase</td>
<td>LREAL</td>
<td>The reactive part of the shunt admittance of the C-phase.</td>
</tr>
<tr>
<td>conductanceABPhase</td>
<td>LREAL</td>
<td>The real part of the shunt admittance due to mutual coupling between the A-phase and the B-phase.</td>
</tr>
<tr>
<td>susceptanceABPhase</td>
<td>LREAL</td>
<td>The reactive part of the shunt admittance due to mutual coupling between the A-phase and the B-phase.</td>
</tr>
<tr>
<td>conductanceACPhase</td>
<td>LREAL</td>
<td>The real part of the shunt admittance due to mutual coupling between the A-phase and the C-phase.</td>
</tr>
<tr>
<td>susceptanceACPhase</td>
<td>LREAL</td>
<td>The reactive part of the shunt admittance due to mutual coupling between the A-phase and the C-phase.</td>
</tr>
<tr>
<td>conductanceBCPhase</td>
<td>LREAL</td>
<td>The real part of the shunt admittance due to mutual coupling between the B-phase and the C-phase.</td>
</tr>
<tr>
<td>susceptanceBCPhase</td>
<td>LREAL</td>
<td>The reactive part of the shunt admittance due to mutual coupling between the B-phase and the C-phase.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns true if the admittance was set based on the provided values.</td>
</tr>
</tbody>
</table>

Processing

This method sets the admittance of the line based on the provided values and returns TRUE, unless the admittance of the line was set previously or the model is no longer in the initialization phase.
bootstrap_SetNominalShuntAdmittanceWZeroSequence (Method)

Call this method to set the line’s shunt admittance using a positive- and zero-sequence data model. Arguments of this method must be in units of siemens.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conductancePosSequence</td>
<td>LREAL</td>
<td>The real part of the positive-sequence shunt admittance.</td>
</tr>
<tr>
<td>susceptancePosSequence</td>
<td>LREAL</td>
<td>The reactive part of the positive-sequence shunt admittance.</td>
</tr>
<tr>
<td>conductanceZeroSequence</td>
<td>LREAL</td>
<td>The real part of the zero-sequence shunt admittance.</td>
</tr>
<tr>
<td>susceptanceZeroSequence</td>
<td>LREAL</td>
<td>The reactive part of the zero-sequence shunt admittance.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if the admittance was set based on the provided values.</td>
</tr>
</tbody>
</table>

Processing

This method sets the admittance of the line based on the provided values and returns TRUE, unless the admittance of the line was set previously or the model is no longer in the initialization phase.

class_PowerTransformerEnd

Instantiate one instance of this class for each transformer winding desired in the model.

Extended Classes

Extending a class provides full inheritance of all that classes features (methods, variables, properties). A class may only extend one other class directly, but class extension can be tiered indefinitely.

➤ class_ConductingEquipment

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING</td>
<td>The name of this object on the power system.</td>
</tr>
<tr>
<td>Description</td>
<td>STRING</td>
<td>A human-readable description of this object.</td>
</tr>
</tbody>
</table>
### PowerSystemModel (HORIZON®) Instruction Manual

#### Classes

##### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NominalRatio</td>
<td>REAL</td>
<td>The modifier to use as the expected change between this winding and others in the system.</td>
</tr>
<tr>
<td>ConnectionType</td>
<td>enum_WindingConnection</td>
<td>The wiring configuration of the winding.</td>
</tr>
</tbody>
</table>

##### Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pt_Terminal</td>
<td>POINTER TO class_Terminal</td>
<td>The terminal of this transformer winding, used to attach it into the model.</td>
</tr>
</tbody>
</table>

##### bootstrap_AddTapChanger (Method)

Call this method to add a tap changer that will modify the NominalRatio of this transformer end.

This is the last type of bootstrap method to be called. It must be called after all terminals are connected and all configure and set bootstrap methods have been completed.

**Inputs/Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tapChanger</td>
<td>class_TapChanger</td>
<td>The tap changer that will modify this winding.</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if the tap changer is added to this winding.</td>
</tr>
</tbody>
</table>

**Processing**

This method links the provided tap changer to this winding and returns true, unless another tap changer has already been set, the tap changer is already modifying another transformer end, or the model is no longer in the initialization phase.

##### bootstrap_SetNominalEndImpedance1Line (Method)

Call this method to set the impedance of the transformer winding from a 1-line model.

Arguments of this method must be in units of ohms.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.
Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resistance</td>
<td>LREAL</td>
<td>The real part of the positive-sequence series impedance.</td>
</tr>
<tr>
<td>reactance</td>
<td>LREAL</td>
<td>The reactive part of the positive-sequence series impedance.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if the impedance was set based on the provided values.</td>
</tr>
</tbody>
</table>

Processing

This method sets the impedance of the transformer end based on the provided values and returns TRUE, unless the impedance of the transformer end was set previously or the model is no longer in the initialization phase.

bootstrap_SetNominalEndImpedance3Phase (Method)

Call this method to set the impedance of the transformer winding from three-phase data. Arguments of this method must be in units of ohms.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resistanceAPhase</td>
<td>LREAL</td>
<td>The real part of the series impedance of the A-phase.</td>
</tr>
<tr>
<td>reactanceAPhase</td>
<td>LREAL</td>
<td>The reactive part of the series impedance of the A-phase.</td>
</tr>
<tr>
<td>resistanceBPhase</td>
<td>LREAL</td>
<td>The real part of the series impedance of the B-phase.</td>
</tr>
<tr>
<td>reactanceBPhase</td>
<td>LREAL</td>
<td>The reactive part of the series impedance of the B-phase.</td>
</tr>
<tr>
<td>resistanceCPhase</td>
<td>LREAL</td>
<td>The real part of the series impedance of the C-phase.</td>
</tr>
<tr>
<td>reactanceCPhase</td>
<td>LREAL</td>
<td>The reactive part of the series impedance of the C-phase.</td>
</tr>
<tr>
<td>resistanceABPhase</td>
<td>LREAL</td>
<td>The real part of the series impedance due to mutual coupling between the A-phase and the B-phase.</td>
</tr>
<tr>
<td>reactanceABPhase</td>
<td>LREAL</td>
<td>The reactive part of the series impedance due to mutual coupling between the A-phase and the B-phase.</td>
</tr>
<tr>
<td>resistanceACPhase</td>
<td>LREAL</td>
<td>The real part of the series impedance due to mutual coupling between the A-phase and the C-phase.</td>
</tr>
<tr>
<td>reactanceACPhase</td>
<td>LREAL</td>
<td>The reactive part of the series impedance due to mutual coupling between the A-phase and the C-phase.</td>
</tr>
<tr>
<td>resistanceBCPhase</td>
<td>LREAL</td>
<td>The real part of the series impedance due to mutual coupling between the B-phase and the C-phase.</td>
</tr>
<tr>
<td>reactanceBCPhase</td>
<td>LREAL</td>
<td>The reactive part of the series impedance due to mutual coupling between the B-phase and the C-phase.</td>
</tr>
</tbody>
</table>
Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if the impedance was set based on the provided values.</td>
</tr>
</tbody>
</table>

Processing

This method sets the impedance of the transformer end based on the provided values and returns TRUE, unless the impedance of the transformer end was set previously or the model is no longer in the initialization phase.

bootstrap_SetNominalEndImpedanceWZeroSequence (Method)

Call this method to set the impedance of the transformer winding from a positive- and zero-sequence data model. Arguments of this method must be in units of ohms.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resistancePosSequence</td>
<td>LREAL</td>
<td>The real part of the positive-sequence series impedance.</td>
</tr>
<tr>
<td>reactancePosSequence</td>
<td>LREAL</td>
<td>The reactive part of the positive-sequence series impedance.</td>
</tr>
<tr>
<td>resistanceZeroSequence</td>
<td>LREAL</td>
<td>The real part of the zero-sequence series impedance.</td>
</tr>
<tr>
<td>reactanceZeroSequence</td>
<td>LREAL</td>
<td>The reactive part of the zero-sequence series impedance.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if the impedance was set based on the provided values.</td>
</tr>
</tbody>
</table>

Processing

This method sets the impedance of the transformer end based on the provided values and returns TRUE, unless the impedance of the transformer end was set previously or the model is no longer in the initialization phase.

bootstrap_SetNominalShuntAdmittance1Line (Method)

Call this method to set the shunt admittance of the transformer end from a 1-line model. Arguments of this method must be in units of siemens.
This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conductance</td>
<td>LREAL</td>
<td>The real part of the positive-sequence shunt admittance.</td>
</tr>
<tr>
<td>susceptance</td>
<td>LREAL</td>
<td>The reactive part of the positive-sequence shunt admittance.</td>
</tr>
</tbody>
</table>

### Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if the admittance was set based on the provided values.</td>
</tr>
</tbody>
</table>

### Processing

This method sets the admittance of the transformer end based on the provided values and returns TRUE, unless the admittance of the transformer end was set previously or the model is no longer in the initialization phase.

**bootstrap_SetNominalShuntAdmittance3Phase (Method)**

Call this method to set the shunt admittance of the transformer winding from three-phase data. Arguments of this method must be in units of siemens.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conductanceAPhase</td>
<td>LREAL</td>
<td>The real part of the shunt admittance of the A-phase.</td>
</tr>
<tr>
<td>susceptanceAPhase</td>
<td>LREAL</td>
<td>The reactive part of the shunt admittance of the A-phase.</td>
</tr>
<tr>
<td>conductanceBPhase</td>
<td>LREAL</td>
<td>The real part of the shunt admittance of the B-phase.</td>
</tr>
<tr>
<td>susceptanceBPhase</td>
<td>LREAL</td>
<td>The reactive part of the shunt admittance of the B-phase.</td>
</tr>
<tr>
<td>conductanceCPhase</td>
<td>LREAL</td>
<td>The real part of the shunt admittance of the C-phase.</td>
</tr>
<tr>
<td>susceptanceCPhase</td>
<td>LREAL</td>
<td>The reactive part of the shunt admittance of the C-phase.</td>
</tr>
<tr>
<td>conductanceABPhase</td>
<td>LREAL</td>
<td>The real part of the shunt admittance due to mutual coupling between the A-phase and the B-phase.</td>
</tr>
<tr>
<td>susceptanceABPhase</td>
<td>LREAL</td>
<td>The reactive part of the shunt admittance due to mutual coupling between the A-phase and the B-phase.</td>
</tr>
</tbody>
</table>
### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conductanceACPhase</td>
<td>LREAL</td>
<td>The real part of the shunt admittance due to mutual coupling between the A-phase and the C-phase.</td>
</tr>
<tr>
<td>susceptibilityACPhase</td>
<td>LREAL</td>
<td>The reactive part of the shunt admittance due to mutual coupling between the A-phase and the C-phase.</td>
</tr>
<tr>
<td>conductanceBCPhase</td>
<td>LREAL</td>
<td>The real part of the shunt admittance due to mutual coupling between the B-phase and the C-phase.</td>
</tr>
<tr>
<td>susceptibilityBCPhase</td>
<td>LREAL</td>
<td>The reactive part of the shunt admittance due to mutual coupling between the B-phase and the C-phase.</td>
</tr>
</tbody>
</table>

### Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if the admittance was set based on the provided values.</td>
</tr>
</tbody>
</table>

### Processing

This method sets the admittance of the transformer end based on the provided values and returns TRUE, unless the admittance of the transformer end was set previously or the model is no longer in the initialization phase.

### bootstrap_SetNominalShuntAdmittanceWZeroSequence (Method)

Call this method to set the shunt admittance of the transformer end from a positive- and zero-sequence data model. Arguments of this method must be in units of siemens.

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conductancePosSequence</td>
<td>LREAL</td>
<td>The real part of the positive-sequence shunt admittance.</td>
</tr>
<tr>
<td>susceptibilityPosSequence</td>
<td>LREAL</td>
<td>The reactive part of the positive-sequence shunt admittance.</td>
</tr>
<tr>
<td>conductanceZeroSequence</td>
<td>LREAL</td>
<td>The real part of the zero-sequence shunt admittance.</td>
</tr>
<tr>
<td>susceptibilityZeroSequence</td>
<td>LREAL</td>
<td>The reactive part of the zero-sequence shunt admittance.</td>
</tr>
</tbody>
</table>

### Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if the admittance was set based on the provided values.</td>
</tr>
</tbody>
</table>


Processing

This method sets the admittance of the transformer end based on the provided values and returns TRUE, unless the admittance of the transformer end was set previously or the model is no longer in the initialization phase.

class_BusbarSection

Instantiate one instance of this class for each location of a bus desired to be modeled.

Extended Classes

Extending a class provides full inheritance of all that classes features (methods, variables, properties). A class may only extend one other class directly, but class extension can be tiered indefinitely.

➤ class_ConductingEquipment

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING</td>
<td>The name of this object on the power system.</td>
</tr>
<tr>
<td>Description</td>
<td>STRING</td>
<td>A human-readable description of this object.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pt_Terminal</td>
<td>POINTER TO class_Terminal</td>
<td>The terminal of this bus, used to attach it into the model.</td>
</tr>
</tbody>
</table>

class_Junction

Instantiate one instance of this class for each location of a non-bus intersection of three or more terminals that requires a name.

Extended Classes

Extending a class provides full inheritance of all that classes features (methods, variables, properties). A class may only extend one other class directly, but class extension can be tiered indefinitely.

➤ class_ConductingEquipment
## Classes

### PowerSystemModel

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING</td>
<td>The name of this object on the power system.</td>
</tr>
<tr>
<td>Description</td>
<td>STRING</td>
<td>A human-readable description of this object.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pt_Terminal</td>
<td>POINTER TO class_Terminal</td>
<td>The terminal of this junction, used to attach it into the model.</td>
</tr>
</tbody>
</table>

### class_TapChanger

Instantiate one instance of this class for each class_PowerTransformerEnd requiring a dynamic transformer ratio.

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING</td>
<td>The name of this object on the power system.</td>
</tr>
<tr>
<td>Description</td>
<td>STRING</td>
<td>A human-readable description of this object.</td>
</tr>
<tr>
<td>DefaultStep</td>
<td>INT</td>
<td>The step value this tap changer will use if communication with the device is unhealthy. A value of zero leaves the PowerTransformer winding at its default NominalRatio.</td>
</tr>
<tr>
<td>StepSize</td>
<td>REAL</td>
<td>The size of each step in percentage.</td>
</tr>
<tr>
<td>StepHighLimit</td>
<td>INT</td>
<td>The maximum multiplier this tap changer allows.</td>
</tr>
<tr>
<td>StepLowLimit</td>
<td>INT</td>
<td>The minimum multiplier this tap changer allows.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StIn_RatioModifier</td>
<td>REAL</td>
<td>The measured value of the tap changer as a percentage.</td>
</tr>
<tr>
<td>StIn_RatioModifierQOK</td>
<td>BOOL</td>
<td>The health of the communications channel providing the ratio modifier. This will be FALSE if the value is out of bounds or the communication is flagged as invalid.</td>
</tr>
</tbody>
</table>

### bootstrap_ConfigureInputs (Method)

Call this method to provide a reference to all required variables the tap changer should monitor for its state.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.
Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ratioModifier</td>
<td>INS</td>
<td>The variable that will report the step position of the tap changer.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if the variables are successfully tied to the tap changer.</td>
</tr>
</tbody>
</table>

Processing

This method stores references to the provided variables and returns true, unless it has already been called for this tap changer.

**class_PowerTransformer**

Instantiate one instance of this class for each set of correlated windings to be modeled. This class acts as a container for class_PowerTransformerEnd objects and relates them to each other.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING</td>
<td>The name of this object on the power system.</td>
</tr>
<tr>
<td>Description</td>
<td>STRING</td>
<td>A human-readable description of this object.</td>
</tr>
<tr>
<td>PowerFactorRating</td>
<td>REAL</td>
<td>The information required to derive the maximum angle from real power this transformer allows before its behavior becomes non-linear.</td>
</tr>
<tr>
<td>RealPowerRating</td>
<td>REAL</td>
<td>The maximum real power this transformer can handle.</td>
</tr>
</tbody>
</table>

**bootstrap_AddWinding (Method)**

Call this method to attach a class_PowerTransformerEnd to this transformer as one of multiple windings.

This is the last type of bootstrap method to be called. It must be called after all terminals are connected and all configure and set bootstrap methods have been completed.

Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>winding</td>
<td>class_PowerTransformerEnd</td>
<td>The winding to add to this transformer.</td>
</tr>
</tbody>
</table>
Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if the winding was added successfully.</td>
</tr>
</tbody>
</table>

Processing

This method attaches the transformer end to this transformer and returns true, unless the transformer end is already attached to a transformer, the transformer has ends attached to a different model, or the model is no longer in the initialization stage.

class_VoltageLevel

Instantiate one voltage level for each nominal voltage desired in the model. The model uses values provided here in per-unit calculations.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING</td>
<td>The name of this object on the power system.</td>
</tr>
<tr>
<td>Description</td>
<td>STRING</td>
<td>A human-readable description of this object.</td>
</tr>
<tr>
<td>NominalVoltage</td>
<td>REAL</td>
<td>The voltage value to be used in per-unit calculations for all equipment added to this voltage level.</td>
</tr>
</tbody>
</table>

bootstrap_AddEquipment (Method)

Call this method for each piece of equipment that should be associated with this container.

Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>equipment</td>
<td>class_ConductingEquipment</td>
<td>The equipment to add to the container.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>The equipment was successfully added to the container.</td>
</tr>
</tbody>
</table>

Processing

This method is intended to be called before processing to associate equipment with this container. It performs the following actions:

- Stores a reference to the object so that it can be accessed as part of the container later.
- If the equipment can be stored in only one of this container type, prompts the equipment to store a reference to the container as well.
Returns FALSE if the object cannot be added to the container because either the objects are no longer in the initialization phase or the equipment was already added to a conflicting container.

**class_CurrentMeasurement**

Current measurement objects can be added to any class_Terminal except one correlating to a bus or junction. Measurements are complex phasors that must be scaled to present the current injection from the equipment out, in amperes. Instantiate one instance of this class for each set of current meter data that the model needs to account for.

class_CurrentMeasurement objects also reports three-phase current values after data manipulation.

**Extended Classes**

Extending a class provides full inheritance of all that classes features (methods, variables, properties). A class may only extend one other class directly, but class extension can be tiered indefinitely.

- class_Measurement

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING</td>
<td>The name of this object on the power system.</td>
</tr>
<tr>
<td>Description</td>
<td>STRING</td>
<td>A human-readable description of this object.</td>
</tr>
<tr>
<td>ScaleFactor</td>
<td>REAL</td>
<td>A multiplier applied as values enter the model.</td>
</tr>
<tr>
<td>MaxError</td>
<td>REAL</td>
<td>The maximum error percentage allowed on this meter before it is flagged as UNTRUSTED. Default is 1%.</td>
</tr>
<tr>
<td>MinimumValue</td>
<td>REAL</td>
<td>The minimum value in amperes that will be used as the denominator in calculating percentage error. Default is 1 ampere.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StIn_IsEnabled</td>
<td>BOOL</td>
<td>The values read by this measurement are allowed by the operator to be used in model calculations.</td>
</tr>
<tr>
<td>StIn_QOk</td>
<td>BOOL</td>
<td>The values provided to this measurement object are healthy.</td>
</tr>
<tr>
<td>StIn_PhaseA</td>
<td>vector_t</td>
<td>A-phase values as measured for three-phase inputs, or a decomposition to vectors for PosSequence or Sequence input.</td>
</tr>
<tr>
<td>StIn_PhaseB</td>
<td>vector_t</td>
<td>B-phase values as measured for three-phase inputs, or a decomposition to vectors for PosSequence or Sequence input.</td>
</tr>
<tr>
<td>StIn_PhaseC</td>
<td>vector_t</td>
<td>C-phase values as measured for three-phase inputs, or a decomposition to vectors for PosSequence or Sequence input.</td>
</tr>
</tbody>
</table>
## Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StIn_PosSequence</td>
<td>vector_t</td>
<td>PosSequence is calculated for three-phase inputs, or as a measured PosSequence input.</td>
</tr>
<tr>
<td>StIn_ZeroSequence</td>
<td>vector_t</td>
<td>ZeroSequence is calculated for three-phase inputs, or as a measured ZeroSequence input.</td>
</tr>
<tr>
<td>StIn_NegSequence</td>
<td>vector_t</td>
<td>NegSequence is calculated for three-phase inputs, or as a measured NegSequence input.</td>
</tr>
<tr>
<td>StO_PhaseAQuality</td>
<td>enum_ValueSource</td>
<td>The confidence level in the reported A-phase value.</td>
</tr>
<tr>
<td>StO_PhaseBQuality</td>
<td>enum_ValueSource</td>
<td>The confidence level in the reported B-phase value.</td>
</tr>
<tr>
<td>StO_PhaseCQuality</td>
<td>enum_ValueSource</td>
<td>The confidence level in the reported C-phase value.</td>
</tr>
<tr>
<td>StO_PhaseA</td>
<td>vector_t</td>
<td>The A-phase value after adjustments.</td>
</tr>
<tr>
<td>StO_PhaseB</td>
<td>vector_t</td>
<td>The B-phase value after adjustments.</td>
</tr>
<tr>
<td>StO_PhaseC</td>
<td>vector_t</td>
<td>The C-phase value after adjustments.</td>
</tr>
</tbody>
</table>

### bootstrap_ConfigureInputsPosSequence (Method)

Call this method to provide a reference to all required variables the measurement should monitor for its state. This method configures this measurement’s inputs to be read from a single value, positive-sequence. It ensures that values read will be converted from positive-sequence to three-phase for later calculations by the model.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.

## Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>BOOL</td>
<td>The variable that will report the state of the manual override of this measurement.</td>
</tr>
<tr>
<td>posSequence</td>
<td>CMV</td>
<td>The variable that will be monitored to determine the positive-sequence value of this measurement.</td>
</tr>
</tbody>
</table>

## Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if references to the variables are stored for future use.</td>
</tr>
</tbody>
</table>

## Processing

This method stores references to the data locations provided as well as the type of data expected and returns true, unless the measurement has already been given other inputs.
bootstrap_ConfigureInputsSequence (Method)

Call this method to provide a reference to all required variables the measurement should monitor for its state. This method configures this measurement’s inputs to be read from three values: positive-, negative-, and zero-sequence. It ensures that values read will be converted from sequence to three-phase for later calculations by the model.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.

Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>BOOL</td>
<td>The variable that will report the state of the manual override of this measurement.</td>
</tr>
<tr>
<td>posSequence</td>
<td>CMV</td>
<td>The variable that will be monitored to determine the positive-sequence value of this measurement.</td>
</tr>
<tr>
<td>zeroSequence</td>
<td>CMV</td>
<td>The variable that will be monitored to determine the zero-sequence value of this measurement.</td>
</tr>
<tr>
<td>negSequence</td>
<td>CMV</td>
<td>The variable that will be monitored to determine the negative-sequence value of this measurement.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if references to the variables are stored for future use.</td>
</tr>
</tbody>
</table>

Processing

This method stores references to the data locations provided as well as the type of data expected and returns true, unless the measurement has already been given other inputs.

bootstrap_ConfigureInputsThreePhase (Method)

Call this method to provide a reference to all required variables the measurement should monitor for its state. This method configures this measurement’s inputs to be read from three-phase values. It ensures that values read will be used directly in later calculations by the model.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.

Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>BOOL</td>
<td>The variable that will report the state of the manual override of this measurement.</td>
</tr>
<tr>
<td>phaseA</td>
<td>CMV</td>
<td>The variable that will be monitored to determine the A-phase value of this measurement.</td>
</tr>
</tbody>
</table>
Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>phaseB</td>
<td>CMV</td>
<td>The variable that will be monitored to determine the B-phase value of this measurement.</td>
</tr>
<tr>
<td>phaseC</td>
<td>CMV</td>
<td>The variable that will be monitored to determine the C-phase value of this measurement.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if references to the variables are stored for future use.</td>
</tr>
</tbody>
</table>

Processing

This method stores references to the data locations provided as well as the type of data expected and returns true, unless the measurement has already been given other inputs.

class_VoltageMeasurement

Voltage measurement objects can be added to any class_Terminal. Measurements are complex phasors that must be scaled to present the voltage in volts. Instantiate one instance of this class for each set of voltage meter data that the model needs to account for.

class_VoltageMeasurement objects also reports three-phase voltage values after data manipulation.

Extended Classes

Extending a class provides full inheritance of all that classes features (methods, variables, properties). A class may only extend one other class directly, but class extension can be tiered indefinitely.

  ➤ class_Measurement

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING</td>
<td>The name of this object on the power system.</td>
</tr>
<tr>
<td>Description</td>
<td>STRING</td>
<td>A human-readable description of this object.</td>
</tr>
<tr>
<td>ScaleFactor</td>
<td>REAL</td>
<td>A multiplier applied as values enter the model.</td>
</tr>
<tr>
<td>MaxError</td>
<td>REAL</td>
<td>The maximum error percentage allowed on this meter before it is flagged as UNTRUSTED. Default is 1%.</td>
</tr>
<tr>
<td>MinimumValue</td>
<td>REAL</td>
<td>The minimum value in volts that will be used as the denominator in calculating percentage error. Default is 1 volt.</td>
</tr>
</tbody>
</table>
### Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StIn_IsEnabled</td>
<td>BOOL</td>
<td>The values read by this measurement are allowed by the operator to be used in model calculations.</td>
</tr>
<tr>
<td>StIn_QOK</td>
<td>BOOL</td>
<td>The values provided to this measurement object are healthy.</td>
</tr>
<tr>
<td>StIn_PhaseA</td>
<td>vector_t</td>
<td>A-phase values as measured for three-phase inputs, or a decomposition to vectors for PosSequence or Sequence input.</td>
</tr>
<tr>
<td>StIn_PhaseB</td>
<td>vector_t</td>
<td>B-phase values as measured for three-phase inputs, or a decomposition to vectors for PosSequence or Sequence input.</td>
</tr>
<tr>
<td>StIn_PhaseC</td>
<td>vector_t</td>
<td>C-phase values as measured for three-phase inputs, or a decomposition to vectors for PosSequence or Sequence input.</td>
</tr>
<tr>
<td>StIn_PosSequence</td>
<td>vector_t</td>
<td>PosSequence is calculated for three-phase inputs, or as a measured PosSequence input.</td>
</tr>
<tr>
<td>StIn_ZeroSequence</td>
<td>vector_t</td>
<td>ZeroSequence is calculated for three-phase inputs, or as a measured ZeroSequence input.</td>
</tr>
<tr>
<td>StIn_NegSequence</td>
<td>vector_t</td>
<td>NegSequence is calculated for three-phase inputs, or as a measured NegSequence input.</td>
</tr>
<tr>
<td>StO_PhaseAQuality</td>
<td>enum_ValueSource</td>
<td>The confidence level in the reported A-phase value.</td>
</tr>
<tr>
<td>StO_PhaseBQuality</td>
<td>enum_ValueSource</td>
<td>The confidence level in the reported B-phase value.</td>
</tr>
<tr>
<td>StO_PhaseCQuality</td>
<td>enum_ValueSource</td>
<td>The confidence level in the reported C-phase value.</td>
</tr>
<tr>
<td>StO_PhaseA</td>
<td>vector_t</td>
<td>The A-phase value after adjustments.</td>
</tr>
<tr>
<td>StO_PhaseB</td>
<td>vector_t</td>
<td>The B-phase value after adjustments.</td>
</tr>
<tr>
<td>StO_PhaseC</td>
<td>vector_t</td>
<td>The C-phase value after adjustments.</td>
</tr>
</tbody>
</table>

#### bootstrap_ConfigureInputsPosSequence (Method)

Call this method to provide a reference to all required variables the measurement should monitor for its state. This method configures this measurement’s inputs to be read from a single value, positive-sequence. It ensures that values read will be converted from positive-sequence to three-phase for later calculations by the model.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.

### Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>BOOL</td>
<td>The variable that will report the state of the manual override of this measurement.</td>
</tr>
<tr>
<td>posSequence</td>
<td>CMV</td>
<td>The variable that will be monitored to determine the positive-sequence value of this measurement.</td>
</tr>
</tbody>
</table>
## Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if references to the variables are stored for future use.</td>
</tr>
</tbody>
</table>

## Processing

This method stores references to the data locations provided as well as the type of data expected and returns true, unless the measurement has already been given other inputs.

### bootstrap_ConfigureInputsSequence (Method)

Call this method to provide a reference to all required variables the measurement should monitor for its state. This method configures this measurement’s inputs to be read from three values: positive-, negative-, and zero-sequence. It ensures that values read will be converted from sequence to three-phase for later calculations by the model.

This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.

### Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>BOOL</td>
<td>The variable that will report the state of the manual override of this measurement.</td>
</tr>
<tr>
<td>posSequence</td>
<td>CMV</td>
<td>The variable that will be monitored to determine the positive-sequence value of this measurement.</td>
</tr>
<tr>
<td>zeroSequence</td>
<td>CMV</td>
<td>The variable that will be monitored to determine the zero-sequence value of this measurement.</td>
</tr>
<tr>
<td>negSequence</td>
<td>CMV</td>
<td>The variable that will be monitored to determine the negative-sequence value of this measurement.</td>
</tr>
</tbody>
</table>

## Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if references to the variables are stored for future use.</td>
</tr>
</tbody>
</table>

## Processing

This method stores references to the data locations provided as well as the type of data expected and returns true, unless the measurement has already been given other inputs.

### bootstrap_ConfigureInputsThreePhase (Method)

Call this method to provide a reference to all required variables the measurement should monitor for its state. This method configures this measurement’s inputs to be read from three-phase values. It ensures that values read will be used directly in later calculations by the model.
This type of bootstrap method must be called after tying terminals and before calling any bootstrap_Add methods.

### Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>BOOL</td>
<td>The variable that will report the state of the manual override of this measurement.</td>
</tr>
<tr>
<td>phaseA</td>
<td>CMV</td>
<td>The variable that will be monitored to determine the A-phase value of this measurement.</td>
</tr>
<tr>
<td>phaseB</td>
<td>CMV</td>
<td>The variable that will be monitored to determine the B-phase value of this measurement.</td>
</tr>
<tr>
<td>phaseC</td>
<td>CMV</td>
<td>The variable that will be monitored to determine the C-phase value of this measurement.</td>
</tr>
</tbody>
</table>

### Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Returns TRUE if references to the variables are stored for future use.</td>
</tr>
</tbody>
</table>

### Processing

This method stores references to the data locations provided as well as the type of data expected and returns true, unless the measurement has already been given other inputs.

### class_ReportedCurrent3Phase

Reported current objects can be added to any class_Terminal except one correlating to a bus or junction. Reports are complex phasors that present the current injection from the equipment out, in amperes. Instantiate one instance of this class for each unmonitored location where knowledge of the current is desired.

### Extended Classes

Extending a class provides full inheritance of all that classes features (methods, variables, properties). A class may only extend one other class directly, but class extension can be tiered indefinitely.

- class_Measurement

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING</td>
<td>The name of this object on the power system.</td>
</tr>
<tr>
<td>Description</td>
<td>STRING</td>
<td>A human-readable description of this object.</td>
</tr>
<tr>
<td>PerPhaseMaxThreshold</td>
<td>REAL</td>
<td>The value that, if exceeded by the results of any phase, will flag a possible error condition. This value is compared directly against the final unscaled output.</td>
</tr>
</tbody>
</table>
# Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StO_PhaseAQuality</td>
<td>enum_ValueSource</td>
<td>The confidence level in the reported A-phase value.</td>
</tr>
<tr>
<td>StO_PhaseBQuality</td>
<td>enum_ValueSource</td>
<td>The confidence level in the reported B-phase value.</td>
</tr>
<tr>
<td>StO_PhaseCQuality</td>
<td>enum_ValueSource</td>
<td>The confidence level in the reported C-phase value.</td>
</tr>
<tr>
<td>StO_PhaseA</td>
<td>vector_t</td>
<td>The A-phase value after adjustments.</td>
</tr>
<tr>
<td>StO_PhaseB</td>
<td>vector_t</td>
<td>The B-phase value after adjustments.</td>
</tr>
<tr>
<td>StO_PhaseC</td>
<td>vector_t</td>
<td>The C-phase value after adjustments.</td>
</tr>
<tr>
<td>PhaseAThresholdExceeded</td>
<td>BOOL</td>
<td>The magnitude of the A-phase value reported exceeds the provided threshold.</td>
</tr>
<tr>
<td>PhaseBThresholdExceeded</td>
<td>BOOL</td>
<td>The magnitude of the B-phase value reported exceeds the provided threshold.</td>
</tr>
<tr>
<td>PhaseCThresholdExceeded</td>
<td>BOOL</td>
<td>The magnitude of the C-phase value reported exceeds the provided threshold.</td>
</tr>
</tbody>
</table>

## class_ReportedCurrentSequence

Reported current objects can be added to any class_Terminal except one correlating to a bus or junction. Reports are complex phasors that present the current injection from the equipment out, in amperes. Instantiate one instance of this class for each unmonitored location where knowledge of the current is desired.

## Extended Classes

Extending a class provides full inheritance of all that classes features (methods, variables, properties). A class may only extend one other class directly, but class extension can be tiered indefinitely.

- ➤ class_Measurement

## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING</td>
<td>The name of this object on the power system.</td>
</tr>
<tr>
<td>Description</td>
<td>STRING</td>
<td>A human-readable description of this object.</td>
</tr>
<tr>
<td>positiveSeqMaxThreshold</td>
<td>REAL</td>
<td>The value that, if exceeded by the positive-sequence results, will flag a possible error condition. This value is compared directly against the final unscaled output.</td>
</tr>
<tr>
<td>zeroSeqMaxThreshold</td>
<td>REAL</td>
<td>The value that, if exceeded by the zero-sequence results, will flag a possible error condition. This value is compared directly against the final unscaled output.</td>
</tr>
</tbody>
</table>
Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>negativeSeqMaxThreshold</td>
<td>REAL</td>
<td>The value that, if exceeded by the negative-sequence results, will flag a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>possible error condition. This value is compared directly against the final</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unscaled output.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StO_Quality</td>
<td>enum_ValueSource</td>
<td>The confidence level in the reported sequence values.</td>
</tr>
<tr>
<td>StO_PosSequence</td>
<td>vector_t</td>
<td>The calculated positive-sequence value.</td>
</tr>
<tr>
<td>StO_ZeroSequence</td>
<td>vector_t</td>
<td>The calculated negative-sequence value.</td>
</tr>
<tr>
<td>StO_NegSequence</td>
<td>vector_t</td>
<td>The calculated zero-sequence value.</td>
</tr>
<tr>
<td>positiveSeqThresholdExceeded</td>
<td>BOOL</td>
<td>The magnitude of the positive-sequence value reported exceeds the provided threshold.</td>
</tr>
<tr>
<td>zeroSeqThresholdExceeded</td>
<td>BOOL</td>
<td>The magnitude of the zero-sequence value reported exceeds the provided threshold.</td>
</tr>
<tr>
<td>negativeSeqThresholdExceeded</td>
<td>BOOL</td>
<td>The magnitude of the negative-sequence value reported exceeds the provided threshold.</td>
</tr>
</tbody>
</table>

class_ReportedVoltage3Phase

Voltage report objects can be added to any class_Terminal. Reports are complex phasors that present voltage in volts. Instantiate one instance of this class for each unmonitored location where knowledge of the voltage is desired.

Extended Classes

Extending a class provides full inheritance of all that classes features (methods, variables, properties). A class may only extend one other class directly, but class extension can be tiered indefinitely.

- class_Measurement

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING</td>
<td>The name of this object on the power system.</td>
</tr>
<tr>
<td>Description</td>
<td>STRING</td>
<td>A human-readable description of this object.</td>
</tr>
</tbody>
</table>
### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PerPhaseMaxThreshold</td>
<td>REAL</td>
<td>The value that, if exceeded by the results of any phase, will flag a possible error condition. This value is compared directly against the final unscaled output.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StO_PhaseAQuality</td>
<td>enum_ValueSource</td>
<td>The confidence level in the reported A-phase value.</td>
</tr>
<tr>
<td>StO_PhaseBQuality</td>
<td>enum_ValueSource</td>
<td>The confidence level in the reported B-phase value.</td>
</tr>
<tr>
<td>StO_PhaseCQuality</td>
<td>enum_ValueSource</td>
<td>The confidence level in the reported C-phase value.</td>
</tr>
<tr>
<td>StO_PhaseA</td>
<td>vector_t</td>
<td>The A-phase value after adjustments.</td>
</tr>
<tr>
<td>StO_PhaseB</td>
<td>vector_t</td>
<td>The B-phase value after adjustments.</td>
</tr>
<tr>
<td>StO_PhaseC</td>
<td>vector_t</td>
<td>The C-phase value after adjustments.</td>
</tr>
<tr>
<td>PhaseAThresholdExceeded</td>
<td>BOOL</td>
<td>The magnitude of the A-phase value reported exceeds the provided threshold.</td>
</tr>
<tr>
<td>PhaseBThresholdExceeded</td>
<td>BOOL</td>
<td>The magnitude of the B-phase value reported exceeds the provided threshold.</td>
</tr>
<tr>
<td>PhaseCThresholdExceeded</td>
<td>BOOL</td>
<td>The magnitude of the C-phase value reported exceeds the provided threshold.</td>
</tr>
</tbody>
</table>

### `class_ReportedVoltageSequence`

Voltage report objects can be added to any `class_Terminal`. Reports are complex phasors that present voltage in volts. Instantiate one instance of this class for each unmonitored location where knowledge of the voltage is desired. All values presented are in the base units used by the library.

### Extended Classes

Extending a class provides full inheritance of all that classes features (methods, variables, properties). A class may only extend one other class directly, but class extension can be tiered indefinitely.

- `class_Measurement`

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING</td>
<td>The name of this object on the power system.</td>
</tr>
<tr>
<td>Description</td>
<td>STRING</td>
<td>A human-readable description of this object.</td>
</tr>
</tbody>
</table>
## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>positiveSeqMaxThreshold</td>
<td>REAL</td>
<td>The value that, if exceeded by the positive-sequence results, will flag a possible error condition. This value is compared directly against the final unscaled output.</td>
</tr>
<tr>
<td>zeroSeqMaxThreshold</td>
<td>REAL</td>
<td>The value that, if exceeded by the zero-sequence results, will flag a possible error condition. This value is compared directly against the final unscaled output.</td>
</tr>
<tr>
<td>negativeSeqMaxThreshold</td>
<td>REAL</td>
<td>The value that, if exceeded by the negative-sequence results, will flag a possible error condition. This value is compared directly against the final unscaled output.</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StO_Quality</td>
<td>enum_ValueSource</td>
<td>The confidence level in the reported sequence values.</td>
</tr>
<tr>
<td>StO_PosSequence</td>
<td>vector_t</td>
<td>The calculated positive-sequence value.</td>
</tr>
<tr>
<td>StO_ZeroSequence</td>
<td>vector_t</td>
<td>The calculated negative-sequence value.</td>
</tr>
<tr>
<td>StO_NegSequence</td>
<td>vector_t</td>
<td>The calculated zero-sequence value.</td>
</tr>
<tr>
<td>positiveSeqThresholdExceeded</td>
<td>BOOL</td>
<td>The magnitude of the positive-sequence value reported exceeds the provided threshold.</td>
</tr>
<tr>
<td>zeroSeqThresholdExceeded</td>
<td>BOOL</td>
<td>The magnitude of the zero-sequence value reported exceeds the provided threshold.</td>
</tr>
<tr>
<td>negativeSeqThresholdExceeded</td>
<td>BOOL</td>
<td>The magnitude of the negative-sequence value reported exceeds the provided threshold.</td>
</tr>
</tbody>
</table>

## Benchmarks

### Benchmark Platforms

The benchmarking tests recorded for this library are performed on the following platforms.

- SEL-3505
  - R135-V0 firmware
- SEL-3530
  - R135-V0 firmware
PowerSystemModel (HORIZON®)

Examples

- SEL-3555
  - Dual-core Intel i7-3555LE processor
  - 4 GB ECC RAM
  - R135-V0 firmware

Benchmark Test Descriptions

Substation Example

The posted time is the average execution time of 100 consecutive calls to the Run() method after initializing the system shown in Modeling a Substation on page 45.

Distribution Network Example

The posted time is the average execution time of 100 consecutive calls to the Run() method after initializing the system shown in Modeling a Distribution Network on page 53.

Ring Network Example

The posted time is the average execution time of 100 consecutive calls to the Run() method after initializing the system shown in Modeling a Ring Network on page 62.

Benchmark Results

<table>
<thead>
<tr>
<th>Operation Tested</th>
<th>SEL-3505</th>
<th>SEL-3530</th>
<th>SEL-3555</th>
</tr>
</thead>
<tbody>
<tr>
<td>Substation Example Run() Time</td>
<td>327,803</td>
<td>198,856</td>
<td>12,858</td>
</tr>
<tr>
<td>Distribution Example Run() Time</td>
<td>266,256</td>
<td>161,899</td>
<td>9,550</td>
</tr>
<tr>
<td>Ring Network Example Run() Time</td>
<td>164,639</td>
<td>97,313</td>
<td>6,689</td>
</tr>
</tbody>
</table>

Examples

These examples demonstrate the capabilities of this library. Do not mistake them as suggestions or recommendations from SEL.

Implement the best practices of your organization when using these libraries. As the user of this library, you are responsible for ensuring correct implementation and verifying that the project using these libraries performs as expected.
Modeling a Substation

Objective

A user has a substation that needs to be monitored. The substation is laid out as seen in Figure 7.

Assumptions

Typically all inputs to a model would be assigned directly from a communications channel, however for ease in compilation for this example all inputs are pulled from a GVL shown below in Code Snippet 1:

```
VAR_GLOBAL
OpenStateValues : ARRAY [1..g_c_NumSwitches+g_c_NumBreakers] OF SPS := [
  (stVal := FALSE, q := (validity := good)),
  (stVal := FALSE, q := (validity := good)),
  (stVal := FALSE, q := (validity := good)),
  (stVal := FALSE, q := (validity := good)),
  (stVal := FALSE, q := (validity := good)),
  (stVal := FALSE, q := (validity := good)),
  (stVal := FALSE, q := (validity := good)),
  (stVal := FALSE, q := (validity := good)),
  (stVal := FALSE, q := (validity := good)),
  (stVal := FALSE, q := (validity := good)),
  (stVal := FALSE, q := (validity := good)),
  (stVal := FALSE, q := (validity := good)),
  (stVal := FALSE, q := (validity := good)),
  (stVal := FALSE, q := (validity := good)),
  (stVal := FALSE, q := (validity := good))
];

TapChangerValue : INS := (stVal := 1, q := (validity := good));

EnableBits : ARRAY [1..g_c_NumVoltageMeters+g_c_NumCurrentMeters] OF BOOL :=
  [g_c_NumVoltageMeters(TRUE), g_c_NumCurrentMeters(TRUE)];
```
PhaseAValues : ARRAY [1..g_c_NumVoltageMeters+g_c_NumCurrentMeters] OF CMV := [
  (q := (validity := good), instCVal := (ang := 0, mag := 1)),
  (q := (validity := good), instCVal := (ang := 0, mag := 1)),
  (q := (validity := good), instCVal := (ang := 0, mag := 2)),
  (q := (validity := good), instCVal := (ang := 0, mag := 2)),
  (q := (validity := good), instCVal := (ang := 0, mag := 4)),
  (q := (validity := good), instCVal := (ang := 0, mag := 4)),
  (q := (validity := good), instCVal := (ang := 0, mag := 1)),
  (q := (validity := good), instCVal := (ang := 0, mag := 1)),
  (q := (validity := good), instCVal := (ang := 0, mag := 1)),
  (q := (validity := good), instCVal := (ang := 0, mag := 1)),
  (q := (validity := good), instCVal := (ang := 0, mag := 2000)),
  (q := (validity := good), instCVal := (ang := 0, mag := 1000))
];

PhaseBValues : ARRAY [1..g_c_NumVoltageMeters+g_c_NumCurrentMeters] OF CMV := [
  (q := (validity := good), instCVal := (ang := 0, mag := 1)),
  (q := (validity := good), instCVal := (ang := 0, mag := 1)),
  (q := (validity := good), instCVal := (ang := 0, mag := 2)),
  (q := (validity := good), instCVal := (ang := 0, mag := 2)),
  (q := (validity := good), instCVal := (ang := 0, mag := 4)),
  (q := (validity := good), instCVal := (ang := 0, mag := 4)),
  (q := (validity := good), instCVal := (ang := 0, mag := 1)),
  (q := (validity := good), instCVal := (ang := 0, mag := 1)),
  (q := (validity := good), instCVal := (ang := 0, mag := 1)),
  (q := (validity := good), instCVal := (ang := 0, mag := 1)),
  (q := (validity := good), instCVal := (ang := 0, mag := 2000)),
  (q := (validity := good), instCVal := (ang := 0, mag := 1000))
];

PhaseCValues : ARRAY [1..g_c_NumVoltageMeters+g_c_NumCurrentMeters] OF CMV := [
  (q := (validity := good), instCVal := (ang := 0, mag := 1)),
  (q := (validity := good), instCVal := (ang := 0, mag := 1)),
  (q := (validity := good), instCVal := (ang := 0, mag := 2)),
  (q := (validity := good), instCVal := (ang := 0, mag := 2)),
  (q := (validity := good), instCVal := (ang := 0, mag := 4)),
  (q := (validity := good), instCVal := (ang := 0, mag := 4)),
  (q := (validity := good), instCVal := (ang := 0, mag := 1)),
  (q := (validity := good), instCVal := (ang := 0, mag := 1)),
  (q := (validity := good), instCVal := (ang := 0, mag := 1)),
  (q := (validity := good), instCVal := (ang := 0, mag := 1)),
  (q := (validity := good), instCVal := (ang := 0, mag := 2000)),
  (q := (validity := good), instCVal := (ang := 0, mag := 1000))
];
Solution

Once the user has identified all the elements of the model and decided the source for all required data there are only two other elements the user must create for the model to do its work. First, all configuration and bootstrap methods must be called before calling Run(). In this example this is shown as completed in a GVL, Code Snippet 3, which allows the configuration to complete before any task cycles begin. Second, the user must create a program that calls the model instance’s Run() method, as shown in Code Snippet 2.

### Code Snippet 2 _prg_RunModel_

```plaintext
PROGRAM prg_RunModel

g_Model1.Run();
```

### Code Snippet 3 _gvi_Bootstrap_

```plaintext
{attribute 'linkalways'}

VAR_GLOBAL CONSTANT
  g_c_NumSwitches : UDINT := 10;
  g_c_NumBreakers : UDINT := 8;
  g_c_ConnectionCount : UDINT := 28;
  g_c_NumVoltageMeters : UDINT := 2;
  g_c_NumCurrentMeters : UDINT := 10;
  g_c_AreaCount : UDINT := 45;
  g_c_ConfigurationCount : UDINT := 39;
END_VAR

VAR_GLOBAL

  // Instantiate any desired models here:
  g_Model1 : class_PowerSystemModel :=
    (Filename := 'GlobalModel1.log', ABCRotation := TRUE);

  // Instantiate all other components:
  g_Source1 : class_EnergySource :=
    (Name := 'Source1', Description := 'Line from elsewhere');
  g_Source2 : class_EnergySource :=
    (Name := 'Source2', Description := 'Another line');
  g_Load1 : class_EnergyConsumer := (Name := 'Load1');
  g_Load2 : class_EnergyConsumer := (Name := 'Load2');
  g_Load3 : class_EnergyConsumer := (Name := 'Load3');
  g_Load4 : class_EnergyConsumer := (Name := 'Load4');

  g_Bus1 : class_BusbarSection := (Name := 'Bus1');
  g_Bus2 : class_BusbarSection := (Name := 'Bus2');

  g_Switches : ARRAY [1..g_c_NumSwitches] OF class_Switch := [ (Name := 'Sw_Source1', TypicallyClosed := TRUE),
    (Name := 'Sw_Source2', TypicallyClosed := TRUE),
    (Name := 'Sw_Bus1Bkr', TypicallyClosed := TRUE),
    (Name := 'Sw_Bus1Transformer', TypicallyClosed := TRUE),
    (Name := 'Sw_Bus2Transformer', TypicallyClosed := TRUE),
    (Name := 'Sw_Bus2Bkr', TypicallyClosed := TRUE),
    (Name := 'Sw_Load1', TypicallyClosed := TRUE),
    (Name := 'Sw_Load2', TypicallyClosed := TRUE),
    (Name := 'Sw_Load3', TypicallyClosed := FALSE),
    (Name := 'Sw_Load4', TypicallyClosed := TRUE) ];
```
Examples

g_Breakers : ARRAY [1..g_c_NumBreakers] OF class_Breaker
:= [ (Name := 'Bkr_Source1', TypicallyClosed := TRUE),
    (Name := 'Bkr_Source2', TypicallyClosed := TRUE),
    (Name := 'Bkr_TransformerHigh', TypicallyClosed := TRUE),
    (Name := 'Bkr_TransformerLow', TypicallyClosed := TRUE),
    (Name := 'Bkr_Load1', TypicallyClosed := TRUE),
    (Name := 'Bkr_Load2', TypicallyClosed := TRUE),
    (Name := 'Bkr_Load3', TypicallyClosed := FALSE),
    (Name := 'Bkr_Load4', TypicallyClosed := TRUE) ];

g_VoltageMeters : ARRAY [1..g_c_NumVoltageMeters] OF class_VoltageMeasurement
:= [ (Name := 'Bus1Voltage', ScaleFactor := 1000),
    (Name := 'Bus2Voltage', ScaleFactor := 1000) ];

g_CurrentMeters : ARRAY [1..g_c_NumCurrentMeters] OF class_CurrentMeasurement
:= [ (Name := 'Source1', ScaleFactor := 1),
    (Name := 'Source2', ScaleFactor := 1),
    (Name := 'Bus1CurrentOut', ScaleFactor := -1),
    (Name := 'TransformerCurrentHigh', ScaleFactor := -1),
    (Name := 'TransformerCurrentLow', ScaleFactor := 1),
    (Name := 'Bus2CurrentIn', ScaleFactor := 1),
    (Name := 'Load1', ScaleFactor := -1),
    (Name := 'Load2', ScaleFactor := -1),
    (Name := 'Load3', ScaleFactor := -1),
    (Name := 'Load4', ScaleFactor := -1) ];

g_Source1Line : class_ACLineSegment := (Name := 'Line1');

g_Source2Line : class_ACLineSegment := (Name := 'Line2');

g_TransformerA : class_PowerTransformer := (Name := 'InboundTransformer');

g_TransformerAHighVoltage : class_PowerTransformerEnd :=
    (Name := 'TransEndHigh', ConnectionType := WYE, NominalRatio := 1.0);

g_TransformerALowVoltage : class_PowerTransformerEnd :=
    (Name := 'TransEndLow', ConnectionType := WYE, NominalRatio := 0.5);

g_TransformerATapChanger : class_TapChanger := (DefaultStep := 0,
    StepSize := 0.05, StepHighLimit := 16, StepLowLimit := -16);

g_HighVoltage : class_VoltageLevel;

g_LowVoltage : class_VoltageLevel;

(* Tie object terminals together. ** This should be done immediately after instantiating objects. ** This must be done before adding objects to containers. *)

g_ObjectsTied : ARRAY [1..g_c_ConnectionCount] OF BOOL
:= [ g_Model1.bootstrap_ConnectTerminals
    (g_Source1.pt_Terminal, g_Source1Line.pt_TerminalA),
    g_Model1.bootstrap_ConnectTerminals
    (g_Source1Line.pt_TerminalB, g_Switches[1].pt_TerminalA),
    g_Model1.bootstrap_ConnectTerminals
    (g_Source2.pt_Terminal, g_Source2Line.pt_TerminalA),...
g_Model1.bootstrap_ConnectTerminals
g_Source2Line.pt_TerminalB, g_Switches[2].pt_TerminalA,
g_Model1.bootstrap_ConnectTerminals
g_Switches[1].pt_TerminalB, g_Breakers[1].pt_TerminalA,
g_Model1.bootstrap_ConnectTerminals
g_Switches[2].pt_TerminalB, g_Breakers[2].pt_TerminalA,
g_Model1.bootstrap_ConnectTerminals
g_Bus1.pt_Terminal, g_Breakers[1].pt_TerminalB,
g_Model1.bootstrap_ConnectTerminals
g_Switches[2].pt_TerminalB, g_Breakers[2].pt_TerminalA,
g_Model1.bootstrap_ConnectTerminals
g_Bus1.pt_Terminal, g_Breakers[3].pt_TerminalA,
g_Model1.bootstrap_ConnectTerminals
g_Bus1.pt_Terminal, g_Breakers[4].pt_TerminalA,
g_Model1.bootstrap_ConnectTerminals
g_Bus1.pt_Terminal, g_Breakers[5].pt_TerminalA,
g_Model1.bootstrap_ConnectTerminals
g_Bus1.pt_Terminal, g_Breakers[6].pt_TerminalA,
g_Model1.bootstrap_ConnectTerminals
g_Bus1.pt_Terminal, g_Breakers[7].pt_TerminalA,
g_Model1.bootstrap_ConnectTerminals
g_Bus1.pt_Terminal, g_Breakers[8].pt_TerminalA,
g_Model1.bootstrap_ConnectTerminals
g_Bus1.pt_Terminal, g_Breakers[9].pt_TerminalA,
g_Model1.bootstrap_ConnectTerminals
g_Bus1.pt_Terminal, g_Breakers[10].pt_TerminalA,
];

// Finish connecting terminals and enable population of containers.
g_TerminalsComplete : BOOL := g_Model1.bootstrap_FinalizeConnections();

// Set object configuration:
g_ConfigsSet : ARRAY [1 .. g_c_ConfigurationCount] OF BOOL := [
Examples

```plaintext
g_Switches[1].bootstrap_ConfigureIsOpenInput (stIn_IsOpen := OpenStateValues[1]),
g_Switches[2].bootstrap_ConfigureIsOpenInput (stIn_IsOpen := OpenStateValues[2]),
g_Switches[3].bootstrap_ConfigureIsOpenInput (stIn_IsOpen := OpenStateValues[3]),
g_Switches[4].bootstrap_ConfigureIsOpenInput (stIn_IsOpen := OpenStateValues[4]),
g_Switches[5].bootstrap_ConfigureIsOpenInput (stIn_IsOpen := OpenStateValues[5]),
g_Switches[6].bootstrap_ConfigureIsOpenInput (stIn_IsOpen := OpenStateValues[6]),
g_Switches[7].bootstrap_ConfigureIsOpenInput (stIn_IsOpen := OpenStateValues[7]),
g_Switches[8].bootstrap_ConfigureIsOpenInput (stIn_IsOpen := OpenStateValues[8]),
g_Switches[9].bootstrap_ConfigureIsOpenInput (stIn_IsOpen := OpenStateValues[9]),
g_Switches[10].bootstrap_ConfigureIsOpenInput (stIn_IsOpen := OpenStateValues[10]),
g_Breakers[1].bootstrap_ConfigureIsOpenInput (stIn_IsOpen := OpenStateValues[11]),
g_Breakers[2].bootstrap_ConfigureIsOpenInput (stIn_IsOpen := OpenStateValues[12]),
g_Breakers[3].bootstrap_ConfigureIsOpenInput (stIn_IsOpen := OpenStateValues[13]),
g_Breakers[4].bootstrap_ConfigureIsOpenInput (stIn_IsOpen := OpenStateValues[14]),
g_Breakers[5].bootstrap_ConfigureIsOpenInput (stIn_IsOpen := OpenStateValues[15]),
g_Breakers[6].bootstrap_ConfigureIsOpenInput (stIn_IsOpen := OpenStateValues[16]),
g_Breakers[7].bootstrap_ConfigureIsOpenInput (stIn_IsOpen := OpenStateValues[17]),
g_Breakers[8].bootstrap_ConfigureIsOpenInput (stIn_IsOpen := OpenStateValues[18]),

g_CurrentMeters[1].bootstrap_ConfigureInputsThreePhase (enable := EnableBits[1], phaseA := PhaseAValues[1], phaseB := PhaseBValues[1], phaseC := PhaseCValues[1]),
g_CurrentMeters[2].bootstrap_ConfigureInputsThreePhase (enable := EnableBits[2], phaseA := PhaseAValues[2], phaseB := PhaseBValues[2], phaseC := PhaseCValues[2]),
g_CurrentMeters[3].bootstrap_ConfigureInputsThreePhase (enable := EnableBits[3], phaseA := PhaseAValues[3], phaseB := PhaseBValues[3], phaseC := PhaseCValues[3]),
g_CurrentMeters[4].bootstrap_ConfigureInputsThreePhase (enable := EnableBits[4], phaseA := PhaseAValues[4], phaseB := PhaseBValues[4], phaseC := PhaseCValues[4]),
g_CurrentMeters[5].bootstrap_ConfigureInputsThreePhase (enable := EnableBits[5], phaseA := PhaseAValues[5], phaseB := PhaseBValues[5], phaseC := PhaseCValues[5]),
g_CurrentMeters[6].bootstrap_ConfigureInputsThreePhase (enable := EnableBits[6], phaseA := PhaseAValues[6], phaseB := PhaseBValues[6], phaseC := PhaseCValues[6]),
g_CurrentMeters[7].bootstrap_ConfigureInputsThreePhase (enable := EnableBits[7], phaseA := PhaseAValues[7], phaseB := PhaseBValues[7], phaseC := PhaseCValues[7]),
g_CurrentMeters[8].bootstrap_ConfigureInputsThreePhase (enable := EnableBits[8], phaseA := PhaseAValues[8], phaseB := PhaseBValues[8], phaseC := PhaseCValues[8]),
```
(enable := EnableBits[8], phaseA := PhaseAValues[8],
phaseB := PhaseBValues[8], phaseC := PhaseCValues[8]),
g_CurrentMeters[9].bootstrap_ConfigureInputsThreePhase
(enable := EnableBits[9], phaseA := PhaseAValues[9],
phaseB := PhaseBValues[9], phaseC := PhaseCValues[9]),
g_CurrentMeters[10].bootstrap_ConfigureInputsThreePhase
(enable := EnableBits[10], phaseA := PhaseAValues[10],
phaseB := PhaseBValues[10], phaseC := PhaseCValues[10]),
g_VoltageMeters[1].bootstrap_ConfigureInputsThreePhase
(enable := EnableBits[11], phaseA := PhaseAValues[11],
phaseB := PhaseBValues[11], phaseC := PhaseCValues[11]),
g_VoltageMeters[2].bootstrap_ConfigureInputsThreePhase
(enable := EnableBits[12], phaseA := PhaseAValues[12],
phaseB := PhaseBValues[12], phaseC := PhaseCValues[12]),
g_Source1Line.bootstrap_SetNominalLineImpedance1Line
(resistance := 1.2, reactance := 12),
g_Source1Line.bootstrap_SetNominalShuntAdmittance1Line
(conductance := 12, susceptance := 1.2),
g_Source2Line.bootstrap_SetNominalLineImpedance1Line
(resistance := 12, reactance := 12),
g_Source2Line.bootstrap_SetNominalShuntAdmittance1Line
(conductance := 12, susceptance := 1.2),
g_TransformerAHighVoltage.bootstrap_SetNominalEndImpedance3Phase(
reactanceAPhase := 10, resistanceAPhase := 1,
reactanceBPhase := 10, resistanceBPhase := 1,
reactanceCPhase := 10, resistanceCPhase := 1,
reactanceABPhase := 1, resistanceABPhase := 0.1,
reactanceACPhase := 1, resistanceACPhase := 0.1,
reactanceBCPhase := 1, resistanceBCPhase := 0.1),
g_TransformerAHighVoltage.bootstrap_SetNominalShuntAdmittance3Phase(
conductanceAPhase := 10, susceptanceAPhase := 1,
conductanceBPhase := 10, susceptanceBPhase := 1,
conductanceCPhase := 10, susceptanceCPhase := 1,
conductanceABPhase := 1, susceptanceABPhase := 0.1,
conductanceACPhase := 1, susceptanceACPhase := 0.1,
conductanceBCPhase := 1, susceptanceBCPhase := 0.1),
g_TransformerALowVoltage.bootstrap_SetNominalEndImpedance3Phase(
reactanceAPhase := 10, resistanceAPhase := 1,
reactanceBPhase := 10, resistanceBPhase := 1,
reactanceCPhase := 10, resistanceCPhase := 1,
reactanceABPhase := 1, resistanceABPhase := 0.1,
reactanceACPhase := 1, resistanceACPhase := 0.1,
reactanceBCPhase := 1, resistanceBCPhase := 0.1),
g_TransformerALowVoltage.bootstrap_SetNominalShuntAdmittance3Phase(
conductanceAPhase := 10, susceptanceAPhase := 1,
conductanceBPhase := 10, susceptanceBPhase := 1,
conductanceCPhase := 10, susceptanceCPhase := 1,
conductanceABPhase := 1, susceptanceABPhase := 0.1,
conductanceACPhase := 1, susceptanceACPhase := 0.1,
conductanceBCPhase := 1, susceptanceBCPhase := 0.1),
g_TransformerATapChanger.bootstrap_ConfigureInputs
(ratioModifier := TapChangerValue)
);

(* Add objects to containers as desired. **
This should be the last configuration done. *)
g_AreaLoaded : ARRAY [1 .. g_c_AreaCount] OF BOOL
Examples

:= [ g_TransformerA.bootstrap_AddWinding(winding :=
g_TransformerAHighVoltage),
g_TransformerA.bootstrap_AddWinding(winding :=
g_TransformerALowVoltage),
g_TransformerALowVoltage.bootstrap_AddTapChanger
  (tapChanger := g_TransformerATapChanger),

  g_HighVoltage.bootstrap_AddEquipment(equipment :=
g_TransformerAHighVoltage),
g_HighVoltage.bootstrap_AddEquipment(equipment := g_Source1),
g_HighVoltage.bootstrap_AddEquipment(equipment := g_Source2),
g_HighVoltage.bootstrap_AddEquipment(equipment := g_Bus1),
g_HighVoltage.bootstrap_AddEquipment(equipment := g_Switches[1]),
g_HighVoltage.bootstrap_AddEquipment(equipment := g_Switches[2]),
g_HighVoltage.bootstrap_AddEquipment(equipment := g_Switches[3]),
g_HighVoltage.bootstrap_AddEquipment(equipment := g_Switches[4]),
g_HighVoltage.bootstrap_AddEquipment(equipment := g_Breakers[1]),
g_HighVoltage.bootstrap_AddEquipment(equipment := g_Breakers[2]),
g_HighVoltage.bootstrap_AddEquipment(equipment := g_Breakers[3]),
g_HighVoltage.bootstrap_AddEquipment(equipment := g_Source1Line),
g_HighVoltage.bootstrap_AddEquipment(equipment := g_Source2Line),

  g_LowVoltage.bootstrap_AddEquipment(equipment :=
g_TransformerALowVoltage),
g_LowVoltage.bootstrap_AddEquipment(equipment := g_Bus2),
g_LowVoltage.bootstrap_AddEquipment(equipment := g_Switches[5]),
g_LowVoltage.bootstrap_AddEquipment(equipment := g_Switches[6]),
g_LowVoltage.bootstrap_AddEquipment(equipment := g_Switches[7]),
g_LowVoltage.bootstrap_AddEquipment(equipment := g_Switches[8]),
g_LowVoltage.bootstrap_AddEquipment(equipment := g_Switches[9]),
g_LowVoltage.bootstrap_AddEquipment(equipment := g_Switches[10]),
g_LowVoltage.bootstrap_AddEquipment(equipment := g_Breakers[4]),
g_LowVoltage.bootstrap_AddEquipment(equipment := g_Breakers[5]),
g_LowVoltage.bootstrap_AddEquipment(equipment := g_Breakers[6]),
g_LowVoltage.bootstrap_AddEquipment(equipment := g_Breakers[7]),
g_LowVoltage.bootstrap_AddEquipment(equipment := g_Breakers[8]),
g_LowVoltage.bootstrap_AddEquipment(equipment := g_Breakers[9]),
g_LowVoltage.bootstrap_AddEquipment(equipment := g_Breakers[10]),
g_LowVoltage.bootstrap_AddEquipment(equipment := g_Load1),
g_LowVoltage.bootstrap_AddEquipment(equipment := g_Load2),
g_LowVoltage.bootstrap_AddEquipment(equipment := g_Load3),
g_LowVoltage.bootstrap_AddEquipment(equipment := g_Load4),

  g_Breakers[1].pt_TerminalB bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[1]),
g_Breakers[2].pt_TerminalB bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[2]),
g_Breakers[3].pt_TerminalA bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[3]),
g_TransformerAHighVoltage.pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[4]),
g_TransformerALowVoltage.pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[5]),
g_Breakers[4].pt_TerminalB bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[6]),
g_Breakers[5].pt_TerminalA bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[7]),
g_Breakers[6].pt_TerminalA bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[8]),
g_Breakers[7].pt_TerminalA bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[9]),

  g_Breakers[8].pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[10]),
g_Breakers[9].pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[11]),
g_Breakers[10].pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[12]),
g_Load1.pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[13]),
g_Load2.pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[14]),
g_Load3.pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[15]),
g_Load4.pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[16]),
g_Switches[1].pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[17]),
g_Switches[2].pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[18]),
g_Switches[3].pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[19]),
g_Switches[4].pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[20]),
g_Switches[5].pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[21]),
g_Switches[6].pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[22]),
g_Switches[7].pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[23]),
g_Switches[8].pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[24]),
g_Switches[9].pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[25]),
g_Switches[10].pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[26]),
g_Bus1.pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[27]),
g_Bus2.pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[28]),
g_Source1.pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[29]),
g_Source2.pt_Terminal bootstrap_AddMeasurement
    (measurement := g_CurrentMeters[30]),
]
g_Breakers[8].pt_TerminalA^.bootstrap_AddMeasurement
(measurement := g_CurrentMeters[10]),
g_Bus1.pt_Terminal^.bootstrap_AddMeasurement
(measurement := g_VoltageMeters[1]),
g_Bus2.pt_Terminal^.bootstrap_AddMeasurement
(measurement := g_VoltageMeters[2])
];

// Finalize all model configuration.
g_ModelValidated : BOOL := g_Model1.bootstrap_ValidateModel();
END_VAR

Modeling a Distribution Network

Objective

A user would like to monitor the power used on some distribution lines. The layout of the lines is as seen in Figure 8.

![Figure 8 Distribution Network of Interest](image)

Assumptions

Typically all inputs to a model would be assigned directly from a communication channel, however for ease in compilation for this example all inputs are pulled from a GVL shown in Code Snippet 4. Because the user needs to add single-phase monitoring, they must create dummy inputs for the non-existent phases as the input sources for both voltage and current.

```
VAR_GLOBAL
  g_DummyVoltage : CMV
    := (q := (validity := invalid), instCVal := (ang := 0, mag := 0));
  g_DummyCurrent : CMV
    := (q := (validity := good), instCVal := (ang := 0, mag := 0));

OpenStateValues2 : ARRAY [1..g_c_NumSwitches2+g_c_NumBreakers2] OF SPS
:= [ (stVal := FALSE, q := (validity := good)),
    (stVal := FALSE, q := (validity := good)) ];
```

Code Snippet 4 gvl_TaqHarness
Examples

(stVal := FALSE, q := (validity := good)),
(stVal := FALSE, q := (validity := good)),
(stVal := FALSE, q := (validity := good)),
(stVal := FALSE, q := (validity := good))

];

TapChangerValueN : INS := (stVal := 0, q := (validity := good));
TapChangerValueS : INS := (stVal := 0, q := (validity := good));

EnableBits2 : ARRAY [1..g_c_NumVoltageMeters2+g_c_NumCurrentMeters2] OF
           BOOL :=
           [g_c_NumVoltageMeters2(TRUE), g_c_NumCurrentMeters2(TRUE)];

PhaseAValues2 : ARRAY [1..g_c_NumVoltageMeters2+g_c_NumCurrentMeters2]
           OF CMV
           := [ (q := (validity := good), instCVal := (ang := 0, mag := 1)),
                 (q := (validity := good), instCVal := (ang := 0, mag := 1)),
                 (q := (validity := good), instCVal := (ang := 0, mag := 0.5)),
                 (q := (validity := good), instCVal := (ang := 0, mag := 0.25)),
                 (q := (validity := good), instCVal := (ang := 0, mag := 0.25)),
                 (q := (validity := good), instCVal := (ang := 0, mag := 25)),
                 (q := (validity := invalid), instCVal := (ang := 0, mag := 0)),
                 (q := (validity := invalid), instCVal := (ang := 0, mag := 0))
           ];

PhaseBValues2 : ARRAY [1..g_c_NumVoltageMeters2+g_c_NumCurrentMeters2]
           OF CMV
           := [ (q := (validity := good), instCVal := (ang := 0, mag := 1)),
                 (q := (validity := good), instCVal := (ang := 0, mag := 1)),
                 (q := (validity := good), instCVal := (ang := 0, mag := 0.5)),
                 (q := (validity := good), instCVal := (ang := 0, mag := 0.25)),
                 (q := (validity := good), instCVal := (ang := 0, mag := 25)),
                 (q := (validity := invalid), instCVal := (ang := 0, mag := 0)),
                 (q := (validity := invalid), instCVal := (ang := 0, mag := 5)),
                 (q := (validity := invalid), instCVal := (ang := 0, mag := 0))
           ];

PhaseCValues2 : ARRAY [1..g_c_NumVoltageMeters2+g_c_NumCurrentMeters2]
           OF CMV
           := [ (q := (validity := good), instCVal := (ang := 0, mag := 1)),
                 (q := (validity := good), instCVal := (ang := 0, mag := 1)),
                 (q := (validity := good), instCVal := (ang := 0, mag := 0.5)),
                 (q := (validity := good), instCVal := (ang := 0, mag := 0.25)),
                 (q := (validity := good), instCVal := (ang := 0, mag := 25)),
                 (q := (validity := invalid), instCVal := (ang := 0, mag := 0)),
                 (q := (validity := invalid), instCVal := (ang := 0, mag := 0)),
                 (q := (validity := good), instCVal := (ang := 0, mag := 5))
           ];

END_VAR
Solution

Once the user has identified all the elements of the model and decided the source for all required data there are only two other elements the user must create for the model to do its work. First, all configuration and bootstrap methods must be called before calling Run(). In this example this is shown as completed in a GVL, Code Snippet 6, which allows the configuration to complete before any task cycles begin. Second, the user must create a program that calls the model instance’s Run() method, Code Snippet 5:

Code Snippet 5  prg_RunModel

```
PROGRAM prg_RunModel

 g_Model2.Run();
```

Code Snippet 6  gvl_Bootstrap

```{attribute 'linkalways'}
VAR_GLOBAL CONSTANT
  g_c_NumSwitches2 : UDINT := 4;
  g_c_NumBreakers2 : UDINT := 2;
  g_c_ConnectionCount2 : UDINT := 28;
  g_c_NumVoltageMeters2 : UDINT := 4;
  g_c_NumCurrentMeters2 : UDINT := 5;
  g_c_AreaCount2 : UDINT := 49;
  g_c_ConfigurationCount2 : UDINT := 37;

  g_c_SeriesRealImpedancePerMile : LREAL := 0.186;
  g_c_SeriesReactiveImpedancePerMile : LREAL := 0.415;
  g_c_ShuntRealAdmittancePerMile : LREAL := 0;
  g_c_ShuntReactiveAdmittancePerMile : LREAL := 1.044E-5;
END_VAR

VAR_GLOBAL
  // Instantiate any desired models here:
  g_Model2 : class_PowerSystemModel :=
    (Filename := 'GlobalModel2.log', ABCRotation := TRUE);

  // Instantiate all other components:
  g_SourceN : class_EnergySource :=
    (Name := 'Source1', Description := 'North Line');
  g_SourceS : class_EnergySource :=
    (Name := 'Source2', Description := 'South Line');
  g_LoadAB : class_EnergyConsumer := (Name := 'LoadAB');
  g_LoadBC : class_EnergyConsumer := (Name := 'LoadBC');
  g_LoadCA : class_EnergyConsumer := (Name := 'LoadCA');
  g_LoadA : class_EnergyConsumer := (Name := 'LoadA');
  g_LoadB : class_EnergyConsumer := (Name := 'LoadB');
  g_LoadC : class_EnergyConsumer := (Name := 'LoadC');

  g_Bus : class_BusbarSection := (Name := 'Bus');
  g_Junction1 : class_Junction := (Name := 'Source Line Junction');
  g_Junction2 : class_Junction := (Name := 'Junction N Shunt');
  g_Junction3 : class_Junction := (Name := 'Junction S Shunt');
  g_Junction4 : class_Junction := (Name := 'Junction two phase split');
  g_Junction5 : class_Junction := (Name := 'Junction one phase split');

  g_Switches2 : ARRAY [1..g_c_NumSwitches2] OF class_Switch
```


```plaintext
Examples

:= [ (Name := 'Sw_SourceS', TypicallyClosed := TRUE),
     (Name := 'Sw_SourceTrans', TypicallyClosed := TRUE),
     (Name := 'Sw_LineN', TypicallyClosed := TRUE),
     (Name := 'Sw_LineS', TypicallyClosed := TRUE) ];

g_Breakers2 : ARRAY [1..g_c_NumBreakers2] OF class_Breaker
:= [ (Name := 'Bkr_LineN', TypicallyClosed := TRUE),
     (Name := 'Bkr_LineS', TypicallyClosed := TRUE) ];

g_VoltageMeters2 : ARRAY [1..g_c_NumVoltageMeters2] OF
    class_VoltageMeasurement
:= [ (Name := 'TransformerOutVolts', ScaleFactor := 1000),
     (Name := 'ALineVoltage', ScaleFactor := 1000),
     (Name := 'BLineVoltage', ScaleFactor := 1000),
     (Name := 'CLineVoltage', ScaleFactor := 1000) ];

g_CurrentMeters2 : ARRAY [1..g_c_NumCurrentMeters2] OF
    class_CurrentMeasurement
:= [ (Name := 'CurrentN', ScaleFactor := 1),
     (Name := 'CurrentS', ScaleFactor := 1),
     (Name := 'CurrentA', ScaleFactor := -1),
     (Name := 'CurrentB', ScaleFactor := -1),
     (Name := 'CurrentC', ScaleFactor := -1) ];

g_LineN1 : class_ACLineSegment := (Name := 'LineN1');
g_LineN2 : class_ACLineSegment := (Name := 'LineN2');
g_LineS1 : class_ACLineSegment := (Name := 'LineS1');
g_LineS2 : class_ACLineSegment := (Name := 'LineS2');

g_TransformerIn : class_PowerTransformer := (Name :=
    'InboundTransformer');
g_TransformerN : class_PowerTransformer := (Name := 'LineNTransformer');
g_TransformerS : class_PowerTransformer := (Name := 'LineSTransformer');

g_TransformerInHigh : class_PowerTransformerEnd :=
    (Name := 'InHighVolts', ConnectionType := WYE, NominalRatio := 1.0);
g_TransformerInLow : class_PowerTransformerEnd :=
    (Name := 'InLowVolts', ConnectionType := WYE, NominalRatio := 0.2);
g_TransformerNTap : class_TapChanger := (DefaultStep := 0,
    StepSize := 0.05, StepHighLimit := 16, StepLowLimit := -16);
g_TransformerSHigh : class_PowerTransformerEnd :=
    (Name := 'LineSHighVolts', ConnectionType := WYE, NominalRatio := 1.0);
g_TransformerSLow : class_PowerTransformerEnd :=
    (Name := 'LineSLowVolts', ConnectionType := WYE, NominalRatio := 0.21739);
g_TransformerSTap : class_TapChanger := (DefaultStep := 0,
    StepSize := 0.05, StepHighLimit := 16, StepLowLimit := -16);
```
g_HighVoltage2 : class_VoltageLevel;
g_MidVoltage2 : class_VoltageLevel;
g_LowVoltage2 : class_VoltageLevel;

g_NorthShunt : class_ShuntCompensator := ( Name := 'North Line Shunt Cap',
conductanceAPhase := 0,
susceptanceAPhase := 25,
conductanceBPhase := 0,
susceptanceBPhase := 25,
conductanceCPhase := 0,
susceptanceCPhase := 25 );
g_SouthShunt : class_ShuntCompensator := ( Name := 'South Line Shunt Cap',
conductanceAPhase := 0,
susceptanceAPhase := 25,
conductanceBPhase := 0,
susceptanceBPhase := 25,
conductanceCPhase := 0,
susceptanceCPhase := 25 );

(* Tie object terminals together.  
** This should be done immediately after instantiating objects.  
** This must be done before adding objects to containers. *)
g_ObjectsTied2 : ARRAY [1 .. g_c_ConnectionCount2] OF BOOL := [
ge_Model2.bootstrap_ConnectTerminals (g_Junction1.pt_Terminal, g_SourceN.pt_Terminal),
ge_Model2.bootstrap_ConnectTerminals (g_Junction1.pt_Terminal, g_Switches2[1].pt_TerminalB),
ge_Model2.bootstrap_ConnectTerminals (g_SourceS.pt_Terminal, g_Switches2[1].pt_TerminalA),
ge_Model2.bootstrap_ConnectTerminals (g_Junction1.pt_Terminal, g_Switches2[2].pt_TerminalA),
ge_Model2.bootstrap_ConnectTerminals (g_Switches2[2].pt_TerminalB, g_TransformerInHigh.pt_Terminal),
ge_Model2.bootstrap_ConnectTerminals (g_Bus.pt_Terminal, g_TransformerInLow.pt_Terminal),
ge_Model2.bootstrap_ConnectTerminals (g_Bus.pt_Terminal, g_Switches2[3].pt_TerminalA),
ge_Model2.bootstrap_ConnectTerminals (g_Switches2[3].pt_TerminalB, g_Breakers2[1].pt_TerminalA),
ge_Model2.bootstrap_ConnectTerminals (g_Breakers2[1].pt_TerminalB, g_TransformerNHigh.pt_Terminal),
ge_Model2.bootstrap_ConnectTerminals (g_Bus.pt_Terminal, g_Switches2[4].pt_TerminalA),
ge_Model2.bootstrap_ConnectTerminals (g_Switches2[4].pt_TerminalB, g_Breakers2[2].pt_TerminalA),
ge_Model2.bootstrap_ConnectTerminals (g_Breakers2[2].pt_TerminalB, g_TransformerSHigh.pt_Terminal),
//The equipment on the north line after the transformer
g_Model2.bootstrap_ConnectTerminals
  (g_TransformerNLow.pt_Terminal, g_LineN1.pt_TerminalA),
g_Model2.bootstrap_ConnectTerminals
  (g_Junction2.pt_Terminal, g_LineN1.pt_TerminalB),
g_Model2.bootstrap_ConnectTerminals
  (g_Junction2.pt_Terminal, g_NorthShunt.pt_Terminal),
g_Model2.bootstrap_ConnectTerminals
  (g_Junction2.pt_Terminal, g_LineN2.pt_TerminalA),
g_Model2.bootstrap_ConnectTerminals
  (g_Junction4.pt_Terminal, g_LineN2.pt_TerminalB),
g_Model2.bootstrap_ConnectTerminals
  (g_Junction4.pt_Terminal, g_LoadAB.pt_Terminal),
g_Model2.bootstrap_ConnectTerminals
  (g_Junction4.pt_Terminal, g_LoadBC.pt_Terminal),
g_Model2.bootstrap_ConnectTerminals
  (g_Junction4.pt_Terminal, g_LoadCA.pt_Terminal),
// The equipment on the south line after the transformer
  g_Model2.bootstrap_ConnectTerminals
  (g_TransformerSLow.pt_Terminal, g_LineS1.pt_TerminalA),
g_Model2.bootstrap_ConnectTerminals
  (g_Junction3.pt_Terminal, g_LineS1.pt_TerminalB),
g_Model2.bootstrap_ConnectTerminals
  (g_Junction3.pt_Terminal, g_SouthShunt.pt_Terminal),
g_Model2.bootstrap_ConnectTerminals
  (g_Junction3.pt_Terminal, g_LineS2.pt_TerminalA),
g_Model2.bootstrap_ConnectTerminals
  (g_Junction5.pt_Terminal, g_LineS2.pt_TerminalB),
g_Model2.bootstrap_ConnectTerminals
  (g_Junction5.pt_Terminal, g_LoadA.pt_Terminal),
g_Model2.bootstrap_ConnectTerminals
  (g_Junction5.pt_Terminal, g_LoadB.pt_Terminal),
g_Model2.bootstrap_ConnectTerminals
  (g_Junction5.pt_Terminal, g_LoadC.pt_Terminal)
];
// Finish connecting terminals and enable population of conatiners.
  g_TerminalsComplete2 : BOOL := g_Model2.bootstrap_FinalizeConnections();
// Set object configuration:
  g_ConfigsSet2 : ARRAY [1 .. g_c_ConfigurationCount2] OF BOOL
  := [ g_Switches2[1].bootstrap_ConfigureIsOpenInput(stIn_IsOpen :=
    OpenStateValues2[1]),
        g_Switches2[2].bootstrap_ConfigureIsOpenInput(stIn_IsOpen :=
    OpenStateValues2[2]),
        g_Switches2[3].bootstrap_ConfigureIsOpenInput(stIn_IsOpen :=
    OpenStateValues2[3]),
        g_Switches2[4].bootstrap_ConfigureIsOpenInput(stIn_IsOpen :=
    OpenStateValues2[4]),
        g_Breakers2[1].bootstrap_ConfigureIsOpenInput(stIn_IsOpen :=
    OpenStateValues2[5]),
        g_Breakers2[2].bootstrap_ConfigureIsOpenInput(stIn_IsOpen :=
    OpenStateValues2[6]),
        g_CurrentMeters2[1].bootstrap_ConfigureInputsThreePhase(
          enable := EnableBits2[1], phaseA := PhaseAValues2[1],
          phaseB := PhaseBValues2[1], phaseC := PhaseCValues2[1]),
        g_CurrentMeters2[2].bootstrap_ConfigureInputsThreePhase(
          enable := EnableBits2[2], phaseA := PhaseAValues2[2],
          phaseB := PhaseBValues2[2], phaseC := PhaseCValues2[2]),
Examples

```python
# Bootstrap configuring inputs for current meters
G_CurrentMeters2[3].bootstrap_ConfigureInputsThreePhase(
    enable := EnableBits2[3], phaseA := PhaseAValues2[3],
    phaseB := g_DummyCurrent, phaseC := g_DummyCurrent),
G_CurrentMeters2[4].bootstrap_ConfigureInputsThreePhase(
    enable := EnableBits2[4], phaseA := g_DummyCurrent,
    phaseB := PhaseBValues2[4], phaseC := g_DummyCurrent),
G_CurrentMeters2[5].bootstrap_ConfigureInputsThreePhase(
    enable := EnableBits2[5], phaseA := g_DummyCurrent,
    phaseB := g_DummyCurrent, phaseC := PhaseCValues2[5]),
G_VoltageMeters2[1].bootstrap_ConfigureInputsThreePhase(
    enable := EnableBits2[6], phaseA := PhaseAValues2[6],
    phaseB := PhaseBValues2[6], phaseC := PhaseCValues2[6]),
G_VoltageMeters2[2].bootstrap_ConfigureInputsThreePhase(
    enable := EnableBits2[7], phaseA := PhaseAValues2[7],
    phaseB := g_DummyVoltage, phaseC := g_DummyVoltage),
G_VoltageMeters2[3].bootstrap_ConfigureInputsThreePhase(
    enable := EnableBits2[8], phaseA := g_DummyVoltage,
    phaseB := PhaseBValues2[8], phaseC := g_DummyVoltage),
G_VoltageMeters2[4].bootstrap_ConfigureInputsThreePhase(
    enable := EnableBits2[9], phaseA := g_DummyVoltage,
    phaseB := g_DummyVoltage, phaseC := PhaseCValues2[9]),

G_LineN1.bootstrap_SetNominalLineImpedance1Line(
    resistance := g_c_SeriesRealImpedancePerMile * 45,
    reactance := g_c_SeriesReactiveImpedancePerMile * 45),
G_LineN1.bootstrap_SetNominalShuntAdmittance1Line(
    conductance := g_c_ShuntRealAdmittancePerMile * 45,
    susceptance := g_c_ShuntReactiveAdmittancePerMile * 45),
G_LineN2.bootstrap_SetNominalLineImpedance1Line(
    resistance := g_c_SeriesRealImpedancePerMile * 2,
    reactance := g_c_SeriesReactiveImpedancePerMile * 2),
G_LineN2.bootstrap_SetNominalShuntAdmittance1Line(
    conductance := g_c_ShuntRealAdmittancePerMile * 2,
    susceptance := g_c_ShuntReactiveAdmittancePerMile * 2),
G_LineS1.bootstrap_SetNominalLineImpedance1Line(
    resistance := g_c_SeriesRealImpedancePerMile * 37,
    reactance := g_c_SeriesReactiveImpedancePerMile * 37),
G_LineS1.bootstrap_SetNominalShuntAdmittance1Line(
    conductance := g_c_ShuntRealAdmittancePerMile * 37,
    susceptance := g_c_ShuntReactiveAdmittancePerMile * 37),
G_LineS2.bootstrap_SetNominalLineImpedance1Line(
    resistance := g_c_SeriesRealImpedancePerMile * 2,
    reactance := g_c_SeriesReactiveImpedancePerMile * 2),
G_LineS2.bootstrap_SetNominalShuntAdmittance1Line(
    conductance := g_c_ShuntRealAdmittancePerMile * 2,
    susceptance := g_c_ShuntReactiveAdmittancePerMile * 2),
G_TransformerInHigh.bootstrap_SetNominalEndImpedance1Line(
    reactance := 10, resistance := 1),
G_TransformerInHigh.bootstrap_SetNominalShuntAdmittance1Line(
    conductance := 10, susceptance := 1),
G_TransformerInLow.bootstrap_SetNominalEndImpedance1Line(
    reactance := 10, resistance := 1),
G_TransformerInLow.bootstrap_SetNominalShuntAdmittance1Line(
    conductance := 10, susceptance := 1),
```
g_TransformerNLow.bootstrap_SetNominalEndImpedance1Line(reactance := 10, resistance := 1),
g_TransformerNLow.bootstrap_SetNominalShuntAdmittance1Line(conductance := 10, susceptance := 1),
g_TransformerSHigh.bootstrap_SetNominalEndImpedance1Line(reactance := 10, resistance := 1),
g_TransformerSHigh.bootstrap_SetNominalShuntAdmittance1Line(conductance := 10, susceptance := 1),
g_TransformerSLow.bootstrap_SetNominalEndImpedance1Line(reactance := 10, resistance := 1),
g_TransformerSLow.bootstrap_SetNominalShuntAdmittance1Line(conductance := 10, susceptance := 1),
g_TransformerNTap.bootstrap_ConfigureInputs(ratioModifier := TapChangerValueN),
g_TransformerSTap.bootstrap_ConfigureInputs(ratioModifier := TapChangerValueS)
]

(* Add objects to containers as desired. ** This should be the last configuration done. *)
g_AreaLoaded2 : ARRAY [1 .. g_c_AreaCount2] OF BOOL := [
  g_TransformerIn.bootstrap_AddWinding(winding :=
    g_TransformerInHigh),
  g_TransformerIn.bootstrap_AddWinding(winding :=
    g_TransformerInLow),
  g_TransformerN.bootstrap_AddWinding(winding :=
    g_TransformerNHigh),
  g_TransformerN.bootstrap_AddWinding(winding :=
    g_TransformerNLow),
  g_TransformerNLow.bootstrap_AddTapChanger(tapChanger := g_TransformerNTap),
  g_TransformerS.bootstrap_AddWinding(winding :=
    g_TransformerSHigh),
  g_TransformerS.bootstrap_AddWinding(winding :=
    g_TransformerSLow),
  g_TransformerSLow.bootstrap_AddTapChanger(tapChanger := g_TransformerSTap),
  g_HighVoltage2.bootstrap_AddEquipment(equipment :=
    g_TransformerInHigh),
  g_HighVoltage2.bootstrap_AddEquipment(equipment := g_SourceN),
  g_HighVoltage2.bootstrap_AddEquipment(equipment := g_SourceS),
  g_HighVoltage2.bootstrap_AddEquipment(equipment := g_Junction1),
  g_HighVoltage2.bootstrap_AddEquipment(equipment :=
    g_Switches2[1]),
  g_HighVoltage2.bootstrap_AddEquipment(equipment :=
    g_Switches2[2]),
  g_MidVoltage2.bootstrap_AddEquipment(equipment :=
    g_TransformerInLow),
  g_MidVoltage2.bootstrap_AddEquipment(equipment := g_Bus),
  g_MidVoltage2.bootstrap_AddEquipment(equipment :=
    g_Switches2[3]),
  g_MidVoltage2.bootstrap_AddEquipment(equipment :=
    g_Switches2[4]),
  g_MidVoltage2.bootstrap_AddEquipment(equipment :=
    g_Breakers2[1]),
g_MidVoltage2.bootstrap_AddEquipment(equipment :=
g_Breakers2[2]),
g_MidVoltage2.bootstrap_AddEquipment(equipment :=
g_TransformerNHigh),
g_MidVoltage2.bootstrap_AddEquipment(equipment :=
g_TransformerSHigh),
g_LowVoltage2.bootstrap_AddEquipment(equipment :=
g_TransformerNLow),
g_LowVoltage2.bootstrap_AddEquipment(equipment :=
g_TransformerSLow),
g_LowVoltage2.bootstrap_AddEquipment(equipment := g_LineN1),
g_LowVoltage2.bootstrap_AddEquipment(equipment := g_LineN2),
g_LowVoltage2.bootstrap_AddEquipment(equipment := g_LineS1),
g_LowVoltage2.bootstrap_AddEquipment(equipment := g_LineS2),
g_LowVoltage2.bootstrap_AddEquipment(equipment := g_Junction2),
g_LowVoltage2.bootstrap_AddEquipment(equipment := g_Junction3),
g_LowVoltage2.bootstrap_AddEquipment(equipment := g_Junction4),
g_LowVoltage2.bootstrap_AddEquipment(equipment := g_Junction5),
g_LowVoltage2.bootstrap_AddEquipment(equipment := g_NorthShunt),
g_LowVoltage2.bootstrap_AddEquipment(equipment := g_SouthShunt),
g_LowVoltage2.bootstrap_AddEquipment(equipment := g_LoadAB),
g_LowVoltage2.bootstrap_AddEquipment(equipment := g_LoadBC),
g_LowVoltage2.bootstrap_AddEquipment(equipment := g_LoadCA),
g_LowVoltage2.bootstrap_AddEquipment(equipment := g_LoadA),
g_LowVoltage2.bootstrap_AddEquipment(equipment := g_LoadB),
g_LowVoltage2.bootstrap_AddEquipment(equipment := g_LoadC),
g_Breakers2[1].pt_TerminalB^.bootstrap_AddMeasurement
  (measurement := g_CurrentMeters2[1]),
g_Breakers2[2].pt_TerminalB^.bootstrap_AddMeasurement
  (measurement := g_CurrentMeters2[2]),
g_LoadA.pt_Terminal^.bootstrap_AddMeasurement
  (measurement := g_CurrentMeters2[3]),
g_LoadB.pt_Terminal^.bootstrap_AddMeasurement
  (measurement := g_CurrentMeters2[4]),
g_LoadC.pt_Terminal^.bootstrap_AddMeasurement
  (measurement := g_CurrentMeters2[5]),
g_Bus.pt_Terminal^.bootstrap_AddMeasurement
  (measurement := g_VoltageMeters2[1]),
g_LoadA.pt_Terminal^.bootstrap_AddMeasurement
  (measurement := g_VoltageMeters2[2]),
g_LoadB.pt_Terminal^.bootstrap_AddMeasurement
  (measurement := g_VoltageMeters2[3]),
g_LoadC.pt_Terminal^.bootstrap_AddMeasurement
  (measurement := g_VoltageMeters2[4])
);

// Finalize all model configuration.
  g_ModelValidated2 : BOOL := g_Model2.bootstrapValidateModel();
END_VAR
Modeling a Ring Network

Objective

A user would like to monitor the power supplied on a ring network as seen in Figure 9.

![Ring Network Diagram](image)

Figure 9: Ring of Network of Interest

Assumptions

Typically all inputs to a model would be assigned directly from a communication channel, however for ease in compilation for this example all inputs are pulled from a GVL shown in Code Snippet 7.

**Code Snippet 7  gvl_TagHarness**

```plaintext
VAR_GLOBAL

OpenStateValues3 : ARRAY [1..1+g_c_NumBreakers3] OF SPS
:= [ (stVal := FALSE, q := (validity := good)),
     (stVal := FALSE, q := (validity := good)),
     (stVal := FALSE, q := (validity := good)),
     (stVal := FALSE, q := (validity := good)),
     (stVal := FALSE, q := (validity := good)),
     (stVal := FALSE, q := (validity := good)),
     (stVal := FALSE, q := (validity := good)),
     (stVal := FALSE, q := (validity := good)),
     (stVal := FALSE, q := (validity := good)),
     (stVal := FALSE, q := (validity := good)),
     (stVal := FALSE, q := (validity := good)),
     (stVal := FALSE, q := (validity := good)) ];

EnableBits3 : ARRAY [1..g_c_NumVoltageMeters3+g_c_NumCurrentMeters3] OF BOOL :=
     [g_c_NumVoltageMeters3(TRUE), g_c_NumCurrentMeters3(TRUE)];

PhaseAValues3 : ARRAY [1..g_c_NumVoltageMeters3+g_c_NumCurrentMeters3] OF CMV
```

62 | PowerSystemModel (HORIZON®) | Instruction Manual | Date Code 20180924
Solution

Once the user has identified all the elements of the model and decided the source for all required data there are only two other elements the user must create for the model to do its work. First, the user must call all configuration and bootstrap methods before calling Run(). In this example, this is shown as completed in a GVL, Code Snippet 9, which allows the configuration to complete before any task cycles begin. Second, the user must create a program that calls the model instance’s Run() method, Code Snippet 8:

```plaintext
Code Snippet 8  prg_RunModel

PROGRAM prg_RunModel

  g_Model2.Run();

END_VAR
```
Code Snippet 9  gvl_Bootstrap

{attribute 'linkalways'}

VAR_GLOBAL CONSTANT
  g_c_NumBreakers3 : UDINT := 8;
  g_c_ConnectionCount3 : UDINT := 21;
  g_c_NumVoltageMeters3 : UDINT := 3;
  g_c_NumCurrentMeters3 : UDINT := 7;
  g_c_AreaCount3 : UDINT := 40;
  g_c_ConfigurationCount3 : UDINT := 31;
END_VAR

VAR_GLOBAL

// Instantiate any desired models here:
  g_Model3 : class_PowerSystemModel :=
    (Filename := 'GlobalModel3.log', ABCRotation := TRUE);

// Instantiate all other components:
  g_Source12K : class_EnergySource :=
    (Name := 'Utility');
  g_Source600 : class_EnergySource :=
    (Name := 'Generation');
  g_Load120 : class_EnergyConsumer := (Name := 'Load120');
  g_Load600_1 : class_EnergyConsumer := (Name := 'LargeLoad1');
  g_Load600_2 : class_EnergyConsumer := (Name := 'LargeLoad2');
  g_Load240 : class_EnergyConsumer := (Name := 'Load240');

  g_JunctionW : class_Junction := (Name := 'W Junction');
  g_JunctionE : class_Junction := (Name := 'E Junction');
  g_JunctionS : class_Junction := (Name := 'S Junction');

  g_Switch : class_Switch := (Name := 'Sw_Generation');

  g_Breakers3 : ARRAY [1..g_c_NumBreakers3] OF class_Breaker := [
    (Name := 'Bkr_W', TypicallyClosed := TRUE),
    (Name := 'Bkr_WNW', TypicallyClosed := TRUE),
    (Name := 'Bkr_NW', TypicallyClosed := TRUE),
    (Name := 'Bkr_NE', TypicallyClosed := TRUE),
    (Name := 'Bkr_ESE', TypicallyClosed := TRUE),
    (Name := 'Bkr_SE', TypicallyClosed := TRUE),
    (Name := 'Bkr_SW', TypicallyClosed := TRUE),
    (Name := 'Bkr_WSW', TypicallyClosed := TRUE)
  ];

  g_VoltageMeters3 : ARRAY [1..g_c_NumVoltageMeters3] OF class_VoltageMeasurement := [
    (Name := 'Volts_W', ScaleFactor := 1),
    (Name := 'Volts_E', ScaleFactor := 1),
    (Name := 'Volts_S', ScaleFactor := 1)
  ];

  g_CurrentMeters3 : ARRAY [1..g_c_NumCurrentMeters3] OF class_CurrentMeasurement := [
    (Name := 'CurrentW', ScaleFactor := 1),
    (Name := 'CurrentNW', ScaleFactor := -1),
    (Name := 'CurrentE', ScaleFactor := -1),
    (Name := 'CurrentSE', ScaleFactor := -1),
    (Name := 'CurrentS', ScaleFactor := 1),
    (Name := 'CurrentSW', ScaleFactor := -1),
    (Name := 'CurrentRing', ScaleFactor := 1)
  ]
g_Transformer12Kto600 : class_PowerTransformer := (Name := 'Transformer12K');
g_Transformer600to120 : class_PowerTransformer := (Name := 'Transformer120');
g_Transformer600to240 : class_PowerTransformer := (Name := 'Transformer240');

g_12Kin600 : class_PowerTransformerEnd :=
    (Name := '12K Winding', ConnectionType := WYE, NominalRatio := 1.0);
g_12Kout600 : class_PowerTransformerEnd :=
    (Name := '600 Winding', ConnectionType := WYE, NominalRatio := 0.05);
g_600in120 : class_PowerTransformerEnd :=
    (Name := '600 Winding 120', ConnectionType := WYE, NominalRatio := 1.0);
g_600out120 : class_PowerTransformerEnd :=
    (Name := '120 Winding', ConnectionType := WYE, NominalRatio := 0.2);
g_600in240 : class_PowerTransformerEnd :=
    (Name := '600 Winding 240', ConnectionType := WYE, NominalRatio := 1.0);
g_600out240 : class_PowerTransformerEnd :=
    (Name := '240 Winding', ConnectionType := WYE, NominalRatio := 0.4);

g_12KVolts : class_VoltageLevel;
g_600Volts : class_VoltageLevel;
g_240Volts : class_VoltageLevel;
g_120Volts : class_VoltageLevel;

(* Tie object terminals together. ** This should be done immediately after instantiating objects. ** This must be done before adding objects to containers. *)
g_ObjectsTied3 : ARRAY [1 .. g_c_ConnectionCount3] OF BOOL :=
    [   g_Model3.bootstrap_ConnectTerminals (g_Source12K.pt_Terminal, g_Breakers3[1].pt_TerminalA),
        g_Model3.bootstrap_ConnectTerminals (g_Breakers3[1].pt_TerminalB, g_12Kin600.pt_Terminal),
        g_Model3.bootstrap_ConnectTerminals (g_JunctionW.pt_Terminal, g_12Kout600.pt_Terminal),
        g_Model3.bootstrap_ConnectTerminals (g_JunctionW.pt_Terminal, g_Breakers3[2].pt_TerminalA),
        g_Model3.bootstrap_ConnectTerminals (g_Breakers3[2].pt_TerminalB, g_600in120.pt_Terminal),
        g_Model3.bootstrap_ConnectTerminals (g_Breakers3[2].pt_TerminalB, g_Breakers3[3].pt_TerminalA),
        g_Model3.bootstrap_ConnectTerminals (g_Breakers3[3].pt_TerminalB, g_Breakers3[4].pt_TerminalA),
        g_Model3.bootstrap_ConnectTerminals (g_JunctionE.pt_Terminal, g_Breakers3[4].pt_TerminalB),
        g_Model3.bootstrap_ConnectTerminals (g_JunctionE.pt_Terminal, g_Load600_1.pt_Terminal),
        g_Model3.bootstrap_ConnectTerminals (g_JunctionE.pt_Terminal, g_Breakers3[5].pt_TerminalA),
    ];
g_Model3.bootstrap_ConnectTerminals
g_Breakers3[5].pt_TerminalB, g_600in240.pt_Terminal),
g_Model3.bootstrap_ConnectTerminals
g_Breakers3[5].pt_TerminalB, g_Breakers3[6].pt_TerminalA),
g_Model3.bootstrap_ConnectTerminals
g_JunctionS.pt_Terminal, g_Breakers3[6].pt_TerminalB),
g_Model3.bootstrap_ConnectTerminals
g_JunctionS.pt_Terminal, g_Switch.pt_TerminalB),
g_Model3.bootstrap_ConnectTerminals
g_Switch.pt_TerminalA, g_Source600.pt_Terminal),
g_Model3.bootstrap_ConnectTerminals
g_JunctionS.pt_Terminal, g_Breakers3[7].pt_TerminalA),
g_Model3.bootstrap_ConnectTerminals
g_Breakers3[7].pt_TerminalB, g_Load600_2.pt_Terminal),
g_Model3.bootstrap_ConnectTerminals
g_Breakers3[7].pt_TerminalB, g_Breakers3[8].pt_TerminalA),
g_Model3.bootstrap_ConnectTerminals
g_JunctionW.pt_Terminal, g_Breakers3[8].pt_TerminalB),
g_Model3.bootstrap_ConnectTerminals
g_600out120.pt_Terminal, g_Load120.pt_Terminal),
g_Model3.bootstrap_ConnectTerminals
g_600out240.pt_Terminal, g_Load240.pt_Terminal)
];

// Finish connecting terminals and enable population of containers.
g_TerminalsComplete3 := g_Model3.bootstrap_FinalizeConnections();

// Set object configuration:

g_ConfigsSet3 := ARRAY [1 .. g_c_ConfigurationCount3] OF BOOL := [
  g_Switch.bootstrap_ConfigureIsOpenInput(stIn_IsOpen :=
    OpenStateValues3[1]),
  g_Breakers3[1].bootstrap_ConfigureIsOpenInput(stIn_IsOpen :=
    OpenStateValues3[2]),
  g_Breakers3[2].bootstrap_ConfigureIsOpenInput(stIn_IsOpen :=
    OpenStateValues3[3]),
  g_Breakers3[3].bootstrap_ConfigureIsOpenInput(stIn_IsOpen :=
    OpenStateValues3[4]),
  g_Breakers3[4].bootstrap_ConfigureIsOpenInput(stIn_IsOpen :=
    OpenStateValues3[5]),
  g_Breakers3[5].bootstrap_ConfigureIsOpenInput(stIn_IsOpen :=
    OpenStateValues3[6]),
  g_Breakers3[6].bootstrap_ConfigureIsOpenInput(stIn_IsOpen :=
    OpenStateValues3[7]),
  g_Breakers3[7].bootstrap_ConfigureIsOpenInput(stIn_IsOpen :=
    OpenStateValues3[8]),
  g_Breakers3[8].bootstrap_ConfigureIsOpenInput(stIn_IsOpen :=
    OpenStateValues3[9]),
  g_CurrentMeters3[1].bootstrap_ConfigureInputsThreePhase(
    enable := EnableBits3[1], phaseA := PhaseAValues3[1],
    phaseB := PhaseBValues3[1], phaseC := PhaseCValues3[1]),
  g_CurrentMeters3[2].bootstrap_ConfigureInputsThreePhase(
    enable := EnableBits3[2], phaseA := PhaseAValues3[2],
    phaseB := PhaseBValues3[2], phaseC := PhaseCValues3[2]),
  g_CurrentMeters3[3].bootstrap_ConfigureInputsThreePhase(
    enable := EnableBits3[3], phaseA := PhaseAValues3[3],
    phaseB := PhaseBValues3[3], phaseC := PhaseCValues3[3]),
]
g_CurrentMeters3[4].bootstrap_ConfigureInputsThreePhase
  (enable := EnableBits3[4], phaseA := PhaseAValues3[4],
   phaseB := PhaseBValues3[4], phaseC := PhaseCValues3[4]),
g_CurrentMeters3[5].bootstrap_ConfigureInputsThreePhase
  (enable := EnableBits3[5], phaseA := PhaseAValues3[5],
   phaseB := PhaseBValues3[5], phaseC := PhaseCValues3[5]),
g_CurrentMeters3[6].bootstrap_ConfigureInputsThreePhase
  (enable := EnableBits3[6], phaseA := PhaseAValues3[6],
   phaseB := PhaseBValues3[6], phaseC := PhaseCValues3[6]),
g_CurrentMeters3[7].bootstrap_ConfigureInputsThreePhase
  (enable := EnableBits3[7], phaseA := PhaseAValues3[7],
   phaseB := PhaseBValues3[7], phaseC := PhaseCValues3[7]),
g_VoltageMeters3[1].bootstrap_ConfigureInputsThreePhase
  (enable := EnableBits3[8], phaseA := PhaseAValues3[8],
   phaseB := PhaseBValues3[8], phaseC := PhaseCValues3[8]),
g_VoltageMeters3[2].bootstrap_ConfigureInputsThreePhase
  (enable := EnableBits3[9], phaseA := PhaseAValues3[9],
   phaseB := PhaseBValues3[9], phaseC := PhaseCValues3[9]),
g_VoltageMeters3[3].bootstrap_ConfigureInputsThreePhase
  (enable := EnableBits3[10], phaseA := PhaseAValues3[10],
   phaseB := PhaseBValues3[10], phaseC := PhaseCValues3[10]),
g_12Kin600.bootstrap_SetNominalEndImpedance1Line(
   reactance := 10, resistance := 1),
g_12Kin600.bootstrap_SetNominalShuntAdmittance1Line(
   conductance := 10, susceptance := 1),
g_12Kout600.bootstrap_SetNominalEndImpedance1Line(
   reactance := 10, resistance := 1),
g_12Kout600.bootstrap_SetNominalShuntAdmittance1Line(
   conductance := 10, susceptance := 1),
g_600in240.bootstrap_SetNominalEndImpedance1Line(
   reactance := 10, resistance := 1),
g_600in240.bootstrap_SetNominalShuntAdmittance1Line(
   conductance := 10, susceptance := 1),
g_600out240.bootstrap_SetNominalEndImpedance1Line(
   reactance := 10, resistance := 1),
g_600out240.bootstrap_SetNominalShuntAdmittance1Line(
   conductance := 10, susceptance := 1),
g_600in120.bootstrap_SetNominalEndImpedance1Line(
   reactance := 10, resistance := 1),
g_600in120.bootstrap_SetNominalShuntAdmittance1Line(
   conductance := 10, susceptance := 1),
g_600out120.bootstrap_SetNominalEndImpedance1Line(
   reactance := 10, resistance := 1),
g_600out120.bootstrap_SetNominalShuntAdmittance1Line(
   conductance := 10, susceptance := 1)
];

(* Add objects to containers as desired. **
   This should be the last configuration done. *)
g_AreaLoaded3 : ARRAY [1 .. g_c_AreaCount3] OF BOOL := [
  g_Transformer12Kto600.bootstrap_AddWinding(winding :=
    g_12Kin600),
  g_Transformer12Kto600.bootstrap_AddWinding(winding :=
    g_12Kout600),
g_Transformer600to240.bootstrap_AddWinding(winding := g_600in240),
g_Transformer600to240.bootstrap_AddWinding(winding := g_600out240),
g_Transformer600to120.bootstrap_AddWinding(winding := g_600in120),
g_Transformer600to120.bootstrap_AddWinding(winding := g_600out120),
g_12KVolts.bootstrap_AddEquipment(equipment := g_12Kin600),
g_12KVolts.bootstrap_AddEquipment(equipment := g_Breakers3[1]),
g_12KVolts.bootstrap_AddEquipment(equipment := g_Source12K),
g_600Volts.bootstrap_AddEquipment(equipment := g_12Kout600),
g_600Volts.bootstrap_AddEquipment(equipment := g_600in240),
g_600Volts.bootstrap_AddEquipment(equipment := g_600in120),
g_600Volts.bootstrap_AddEquipment(equipment := g_JunctionW),
g_600Volts.bootstrap_AddEquipment(equipment := g_JunctionE),
g_600Volts.bootstrap_AddEquipment(equipment := g_JunctionS),
g_600Volts.bootstrap_AddEquipment(equipment := g_Source600),
g_600Volts.bootstrap_AddEquipment(equipment := g_Load600_1),
g_600Volts.bootstrap_AddEquipment(equipment := g_Load600_2),
g_600Volts.bootstrap_AddEquipment(equipment := g_Breakers3[2]),
g_600Volts.bootstrap_AddEquipment(equipment := g_Breakers3[3]),
g_600Volts.bootstrap_AddEquipment(equipment := g_Breakers3[4]),
g_600Volts.bootstrap_AddEquipment(equipment := g_Breakers3[5]),
g_600Volts.bootstrap_AddEquipment(equipment := g_Breakers3[6]),
g_600Volts.bootstrap_AddEquipment(equipment := g_Breakers3[7]),
g_600Volts.bootstrap_AddEquipment(equipment := g_Breakers3[8]),
g_600Volts.bootstrap_AddEquipment(equipment := g_Switch),
g_240Volts.bootstrap_AddEquipment(equipment := g_600out240),
g_240Volts.bootstrap_AddEquipment(equipment := g_Load240),
g_120Volts.bootstrap_AddEquipment(equipment := g_600out120),
g_120Volts.bootstrap_AddEquipment(equipment := g_Load120),
g_JunctionW.pt_Terminal^.bootstrap_AddMeasurement(measurement := g_VoltageMeters3[1]),
g_JunctionE.pt_Terminal^.bootstrap_AddMeasurement(measurement := g_VoltageMeters3[1]),
g_JunctionS.pt_Terminal^.bootstrap>AddMeasurement(measurement := g_VoltageMeters3[1]),
g_12Kout600.pt_Terminal^.bootstrap>AddMeasurement(measurement := g_CurrentMeters3[1]),
g_600in120.pt_Terminal^.bootstrap>AddMeasurement(measurement := g_CurrentMeters3[2]),
g_Load600_1.pt_Terminal^.bootstrap>AddMeasurement(measurement := g_CurrentMeters3[2]),
g_Load600_2.pt_Terminal^.bootstrap>AddMeasurement(measurement := g_CurrentMeters3[2]),
g_Switch.pt_TerminalB^.bootstrap>AddMeasurement(measurement := g_CurrentMeters3[5]),
g_Breakers3[4].pt_TerminalB^.bootstrap>AddMeasurement(measurement := g_CurrentMeters3[7])
Log File Format

A log file is written to a file defined by the class_PowerSystemModel object’s Filename variable. This file is written in four parts.

First is a summary of connections. This should show a count of terminals, equipment, and nodes (locations where terminals are tied). Next is a list of quantities for each type of equipment. These two sets of numbers can be compared to the system desired to be modeled to validate that all pieces of the model are tied together. Third, any errors that might have been encountered during validation are printed. This prints any errors detected in a particular device, followed by the name of the device in which the errors were encountered. Finally, a summary of windings per transformer is provided. A sample log file for Modeling a Substation on page 45 with an added voltage tying error is presented below:

This model is constructed from 50 terminals connected to 30 pieces of equipment by 22 nodes.

This model is comprised of the following equipment:
- Energy Sources: 2
- Energy Consumers: 4
- Switches: 10
- Breakers: 8
- Lines: 2
- Buses and Junctions: 2
- Shunt Compensators: 0
- Transformers: 1

Conducting equipment TransEndLow not in Voltage Level.
Errors in transformer winding TransEndLow.
Transformer InboundTransformer tied to model with 2 windings.
## Release Notes

<table>
<thead>
<tr>
<th>Version</th>
<th>Summary of Revisions</th>
<th>Date Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5.1.0</td>
<td>▶ Allows new versions of ACSELERATOR RTAC to compile projects for previous firmware versions without SEL IEC types “Cannot convert” messages. ▶ Must be used with R143 firmware or later.</td>
<td>20180921</td>
</tr>
<tr>
<td>3.5.0.0</td>
<td>▶ Initial release.</td>
<td>20150924</td>
</tr>
</tbody>
</table>