Pacific Northwest
NATIONAL LABORATORY

PNNL-34946

# A Power Application Developer's Guide to the Common Information Model

## An Introduction for Power Systems Engineers and Application Developers – CIM17v40

March 2023

Alexander A. Anderson
Thomas E. McDermott
Eric. G. Stephan

U.S. DEPARTMENT OF ENERGY

Prepared for the U.S. Department of Energy
under Contract DE-AC05-76RL01830

## DISCLAIMER



PACIFIC NORTHWEST NATIONAL LABORATORY
*operated by*
BATTELLE
*for the*
UNITED STATES DEPARTMENT OF ENERGY
*under Contract DE-AC05-76RL01830*

**Printed in the United States of America**

**Available to DOE and DOE contractors from the Office of Scientific and Technical Information,**
**P.O. Box 62, Oak Ridge, TN 37831-0062**
www.osti.gov
**ph: (865) 576-8401**
**fax: (865) 576-5728**
**email: reports@osti.gov**

**Available to the public from the National Technical Information Service**
**5301 Shawnee Rd., Alexandria, VA 22312**
**ph: (800) 553-NTIS (6847)**
**or (703) 605-6000**
**email: info@ntis.gov**
**Online ordering: http://www.ntis.gov**

PNNL-34946

# A Power Application Developer's Guide to the Common Information Model

An Introduction for Power Systems Engineers and Application Developers – CIM17v40

March 2023

Alexander A. Anderson
Thomas E. McDermott
Eric. G. Stephan

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99354

# Executive Summary

The electric grid is being reshaped rapidly by decarbonization efforts, smart grid technology adoption, and projects to improve the reliability, resiliency, and robustness of both the bulk electric power system and distribution networks. Each year brings significant increases in the number of distributed energy resources, electric vehicles, controllable customer assets, power electronics, and intelligent grid-edge devices. Integration of these new devices and associated operational paradigms are driving the need for more streamlined data integration workflows and data exchange between advanced applications and between utilities.

A key issue in creating the next generation of energy management system (EMS) and advanced distribution management system (ADMS) platforms will be the ability to represent and exchange power system network model data in a consistent manner. To this end, the Common Information Model (CIM) stands out as the only standardized vocabulary (or *ontology*) for defining power system network models and asset data in a comprehensive, consistent manner across the generation-transmission-distribution boundary.

The CIM is freely available to use and extend. The CIM is maintained by the UCAiug (informally known as the CIM User's Group) under an Apache 2.0 license. The CIM Users Group collaborates with the IEC and other standards communities for the development of technical and informative specifications. Although portions of the information model are referred to by the corresponding IEC standards naming, it is not necessary to purchase any of the IEC standards to use the CIM information model.

This document provides a roadmap for power system engineers and application developers not familiar with semantic modeling to start using the CIM for modeling, simulation, optimization, and development of advanced power applications. The key classes needed for defining power system topology and equipment are explained systematically. Key focus areas include modeling of lines, transformers, generators, switching equipment, loads, and distributed energy resources (DERs) through a handful of core 61970 and 61968 classes summarized in the table below.

| CIM Class | Description |
|---|---|
| `ACLineSegment` | Any overhead line or underground cable |
| `EnergyConsumer` | Load |
| `PowerTransformer` | Three-phase transformer |
| `TransformerTank` | Single-phase transformer |
| `RatioTapChanger` | Load tap changing transformer / voltage regulator |
| `ShuntCompensator` | Shunt capacitor and reactor |
| `SynchronousMachine` | Any synchronous generator / motor |
| `AsynchronousMachine` | Type I/II wind turbine / induction motor |
| `PowerElectronicsConnection` | Inverter-based generator / energy storage |
| `BatteryUnit, PhotoVoltaicUnit` | DC side of inverter-based resources |
| `Switch, Breaker, Fuse, Recloser, Sectionaliser, LoadBreakSwitch` | Switching equipment |

# Acronyms and Abbreviations

| | |
|---|---|
| ADMS | Advanced Distribution Management System |
| API | Application Programming Interface |
| CIM | Common Information Model |
| DDL | Data Definition Language |
| EA | Enterprise Architect |
| EPRI | Electric Power Research Institute |
| DER | Distributed Energy Resource |
| EMS | Energy Management System |
| ICCP | Inter-control Center Communication Protocol |
| IT | Information Technology |
| LOD | Linked Open DATA |
| mRID | master Resource Identifier |
| OMS | Outage Management System |
| OWL | Web Ontology Language |
| PMU | Phasor Measurement Units |
| PNNL | Pacific Northwest National Laboratory |
| RDF | Resource Description Framework |
| SCADA | System Control and Data Acquisition |
| UML | Unified Modeling Language |
| UUID | Universally Unique Identifier |
| XSD | eXtensible Schema Definition |
| XML | eXtensible Markup Language |

# Contents

# 1.0 Introduction

The Common Information Model (CIM) is an abstract information model that can be used to model an electrical network and the various equipment used on the network. CIM is widely used for data exchange of bulk transmission power systems and is now beginning to find increasing use for distribution modeling and analysis. By using a common model, utilities, vendors, and researchers from both academia and industry can reduce the effort and cost of data integration, and instead focus on developing increased functionality for managing and optimizing the smart grid of the future.

This report is intended as a first introduction to CIM for utility engineers, power systems researchers, and application developers, providing a broad view of the CIM and how particular profiles can be adapted for various use cases. This document is intended to be read as the second of three documents that cover the why, what, and how of using the CIM for model exchange, data exchange, and data integration.

The first document [1] outlined *why* an agreed-upon information model is needed, as well as key concepts around data integration and model exchange driven by the evolution of the electric grid and available data streams due to ongoing decarbonization efforts. Key concepts related to information modeling, including semantics, syntactics, canonical models, data profiles, data structures, and database schemes were introduced using a running example of mapping distributed energy resources to substation breakers for improved coordination. It provides five use cases for the CIM with increasing levels of data integration, ranging from manual model exchange to full utility-wide data integration of all planning, operations, and business software across a single enterprise message bus.


Figure 1: Recommended guides to read as first introduction. This document explains *why* an information model is needed. The second document [1] explains *what* is specified by CIM to represent power system networks and assets. The third document [3] explains *how* to build a custom profile and use some of the available modeling tools.

This second document outlines *what* set classes and attributes are used by the CIM to specify common power system components, such as lines, transformers, and switches. It also provides a summary of available tools and a first introduction to reading class diagrams and schemas used by CIM to specify how power system components should be modeled. The main focus is placed on the IEC 61970 and IEC 61968 portions of the information model, which describe modeling of the electrical network and physical assets. The goal is to provide an overview of modeling of common types of power systems equipment, such as lines, transformers, loads, and generators. This report will not cover the transactive market portion of the information model corresponding to the IEC 62325 standards.

This document also specifically covers the latest, stable, "gold standard" version of CIM 17v40 available from the UCAiug (also known as the CIM User Group) [2]. The upcoming draft CIM18 will introduce a number of changes and enhancements, including DER asset models, new stable namespaces, etc. This document will be updated upon the release of CIM 18.

The third recommended document is the EPRI CIM Primer [3], which explains *how* to create a CIM profile and provides a much more detailed and technical description of schemas, data structures, and implementation techniques. That report is more suitable for use by software engineers familiar with schema representations who are seeking to build converter scripts to convert CIM models and perform advanced model development tasks.

# 2.0 UML Class Diagrams

As an information model, CIM does not directly specify how those objects should be used in a database or data structure. Rather, it simply defines the vocabulary of how these objects should be described and what is the relationship between them. The CIM extensively uses Unified Modeling Language (UML) class diagrams to describe each class of objects, their attributes, and their relationships with other classes. UML provides 13 types of diagrams to define software architecture. One of them is the **UML Class Diagram**, which visually represents object hierarchies and relationships.

First a review of basic concepts and terminology related to class diagrams:

- An **object** is any particular thing that we want to describe.
- A **class** represents a specific type of object.
- A **class hierarchy** is a model of the system showing every component as a separate class. The class hierarchy should represent the real-world structure of the system.
- A **package** is a group of classes. Think of folders in a computer file explorer.
- **Inheritance** allows us to define very general "parent classes" and very specific "child classes".
- **Attributes** are the properties that describe what type of thing the class represents.
- **Associations** are the relationships between various objects and how they are connected to each other.

## 2.1 Interpretation of UML Class Diagrams

Class diagrams show all the attributes and associations of various classes in a particular package in a single picture. Throughout this report, classes and their attributes can be distinguished through a capitalization convention that classes and packages use InitialCapitals, while attributes and enumerations use camelCase. This document will use `consolas font` to highlight CIM class and attributes. To read a class diagram, such as Figure 2, remember that

- Lines with an arrowhead indicate class inheritance. For example, in the figure below, `ACLineSegment` inherits from `Conductor`, `ConductingEquipment`, `Equipment`, and then `PowerSystemResource`. `ACLineSegment` inherits all attributes and associations from its ancestors (e.g., `length`), in addition to its own attributes and ancestors.
- Lines with a diamond indicate composition. For example, `Substations` make up a `SubGeographicalRegion`, which then make up a `GeographicRegion`.
- Lines without a terminating symbol are associations. For example, `ACLineSegment` has (through inheritance) a `BaseVoltage`, `Location` and one or more `Terminal` objects.
- Italicized names at the top of each class indicate the ancestor (aka superclass), in cases where the ancestor does not appear on the diagram. For example, `PowerSystemResource` inherits from `IdentifiedObject`.

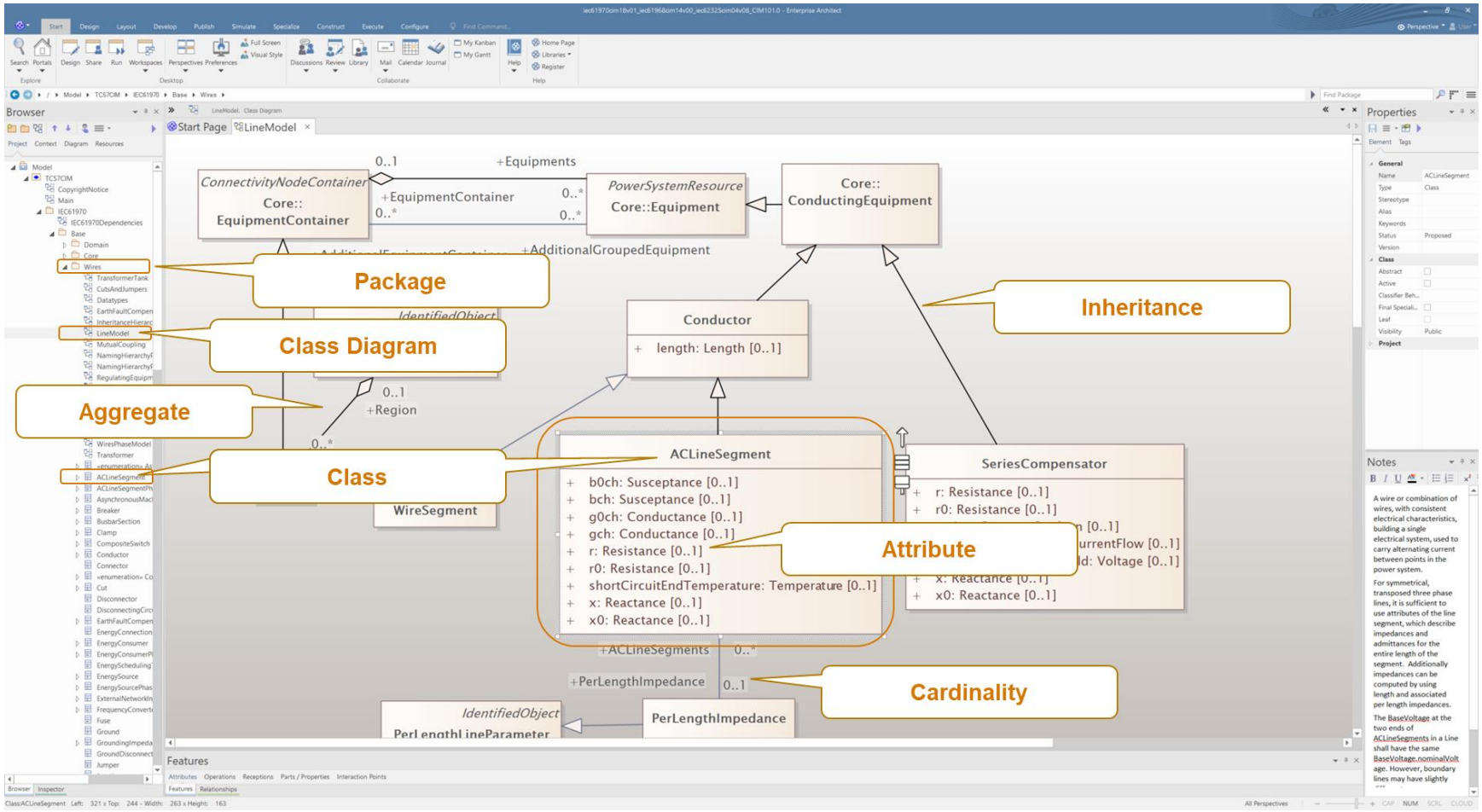


Figure 2: A UML class diagram for `ACLineSegment` and related classes viewed in Enterprise Architect. Each class is described in detail, with the ability to view technical descriptions of each attribute and the set of links to other classes throughout the entire information model.

## 2.2 Inheritance of Class Attributes and Associations

Objects specified using CIM inherit all the attributes and associations of their parent classes, as shown in Figures 3 and 4. Due to the deep inheritance hierarchies found within CIM, a particular object will contain numerous (seemingly unrelated) attributes when serialized. For example, the `ACLineSegment` class has five parent classes, as shown in Figure 3, and inherits the attributes of all its parent classes. Thus, an individual `ACLineSegment` object will contain attributes such as `mRID`, `inService`, and `length`. In most serializations, the parent classes are written out in full; however, newer approaches using knowledge graphs and JSON-LD may omit the parent class and use only the attribute name.

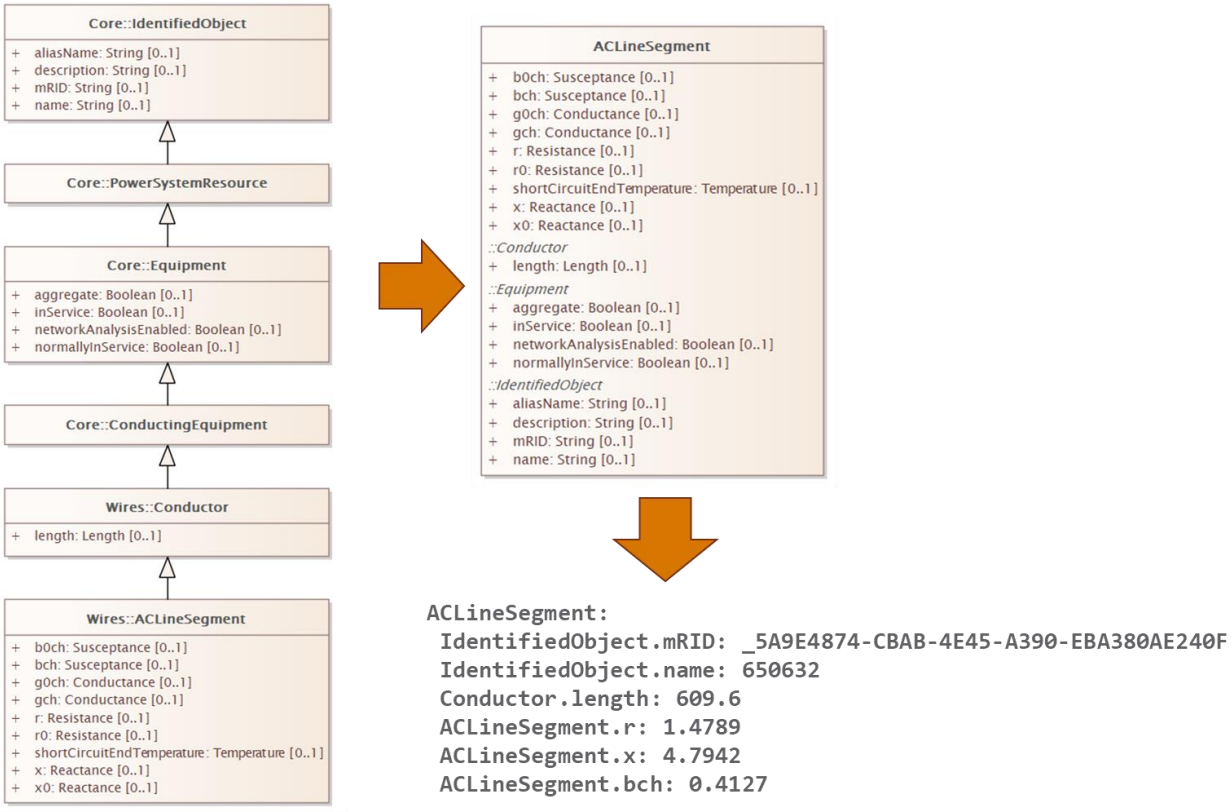


Figure 3: Inheritance of attributes from parent classes in CIM.

Likewise, CIM classes also inherit all of the associations of their parent classes to other classes. As illustrated in Figure 4, the class `ACLineSegment` inherits the associations of its parent classes to `Location`, `EquipmentContainer`, and `BaseVoltage`. Thus, when serialized, the associations of its parent classes will appear along with its inherited attributes.

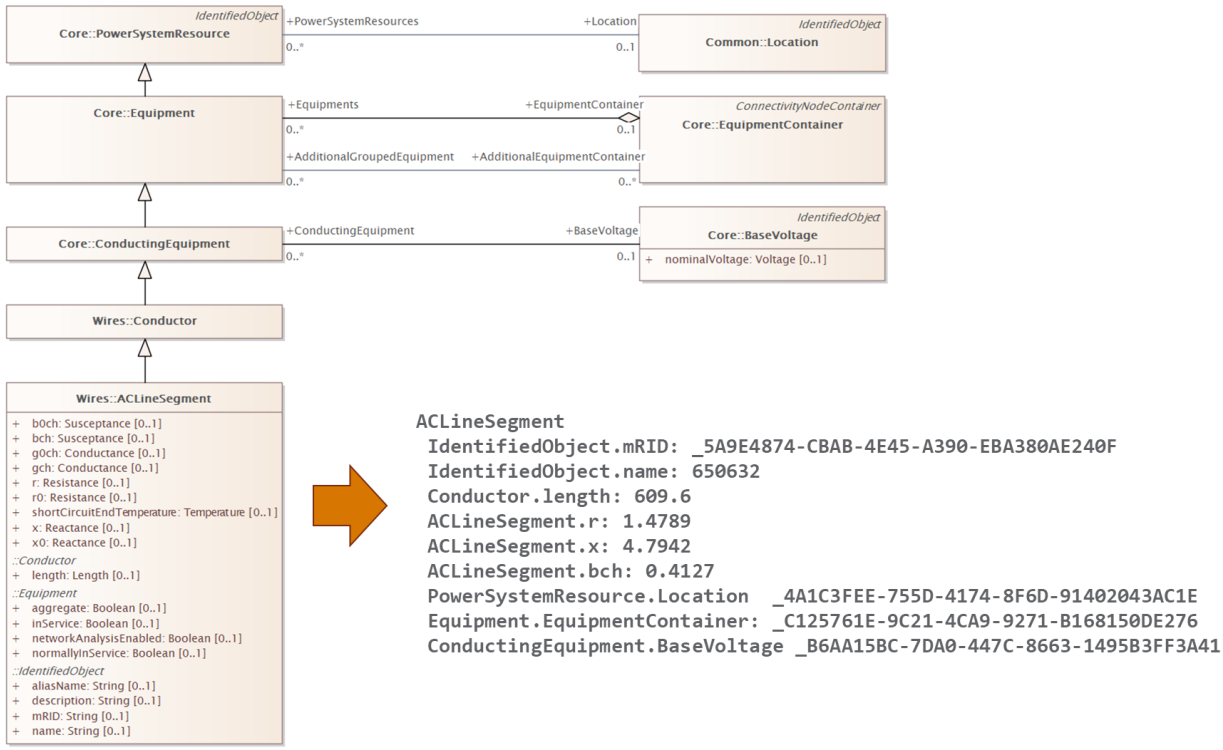


Figure 4: Inheritance of associations from parent classes within CIM.

# 3.0 Names and mRIDs

The highest-level class of objects in CIM is called `IdentifiedObject`. This class is very abstract and only contains attributes used to reference the object either by a user or in software. One of the most important attributes of `IdentifiedObject` is the master resource identifier (mRID), which is a globally unique 3- to 18-character identifier of objects; the mRID does not have to be human-readable. It is strongly recommended that this identifier be a Universally Unique Identifier (UUID), but this requirement is not enforced by the CIM UML or associated standards. This identifier is generally intended to be used by software systems and is never displayed to the user. In CIM-based applications, the mRID is often used as the unique pointer or dictionary key that allows all the properties of a particular piece of equipment to be called from memory or queried from a database.

The attributes `name`, `description`, and `aliasName`, shown in Figure 5, are intended for providing identifiers that are human-readable. It is common for names of objects within a utility to not be unique due to historical naming conventions, the results of mergers and acquisitions, and the inability of other software systems to manage uniqueness. For these reasons, there are no constraints on these names requiring them to be unique. The uniqueness of the equipment assets and properties is instead managed through the equipment mRID. Thus, it would not matter if a particular switch was named "NORTHSUB-115KV-BAY01-031" in one application but "NSB115031" in another application since both would reference the switch using its unique machine-readable mRID `"_68B7C1EF-AC11-48E2-8D85-43BB0EF41B58"`. Note again that the mRID would typically never be displayed to the user, but only used internally by the application code and databases.

When re-exporting models, a master list of mRIDs is typically kept serving as a mapping between the equipment naming within a particular application and the unique mRID that is referenced and used by all applications working from the same CIM XML power system model.

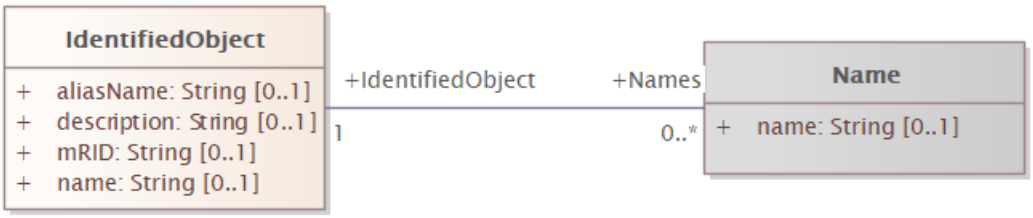


Figure 5: `IdentifiedObject` class, which is used to specify the unique name and identifier of CIM objects. The `Name` class should ***not*** be used for this purpose.

It is important to note that classes `Name`, `NameType`, and `NameTypeAuthority`, shown in Figure 5, should ***not*** be used as the primary reference for CIM objects and power system model data. These classes exist to support non-standard business-specific naming or the use of multiple names for the same device across multiple software environments.

# 4.0 Conducting Equipment and Assets

All elements of the power system model are categorized as a `PowerSystemResource`, which can be any item of equipment (e.g. a switch), an equipment container with multiple pieces of equipment (e.g. a substation or a feeder), or an organizational entity (e.g. a control area). All power systems modeling components are then divided into two categories, depending on whether the particular focus is on the electrical connectivity or the physical assets. All electrical aspects of equipment are associated with the classes inheriting from `Equipment` class in the 61970 package, as shown in Figure 6. The physical aspects of equipment are associated with the `Assets` and `AssetInfo` packages of the 61968 package.

Equipment objects have associated `GeographicalRegion`, `SubGeographicalRegion`, `Substation`, and `VoltageLevel` objects. They may further be associated with a particular `EquipmentContainer`, such as a `Feeder` or `Bay`. Any equipment that conducts electric current is covered in the set of classes inheriting from `ConductingEquipment`. Table 1 and Figure 7 summarize the most commonly used classes.

The `Assets` package contains the set of classes related to equipment maintenance and asset management. The `AssetInfo` package contains the set of classes related to physical ratings and properties of lines, transformers, switches, and other conducting equipment. Voltage ratings, current ratings, conductor geometry, insulation properties, cable diameters, impedance per line length, and similar physical properties are specified by the attributes associated with the set of classes `BusBarSectionInfo`, `SwitchInfo`, `ShuntCompensatorInfo`, `WireInfo`, `CableInfo`, `TapChangerInfo`, `TransformerEndInfo`, etc.

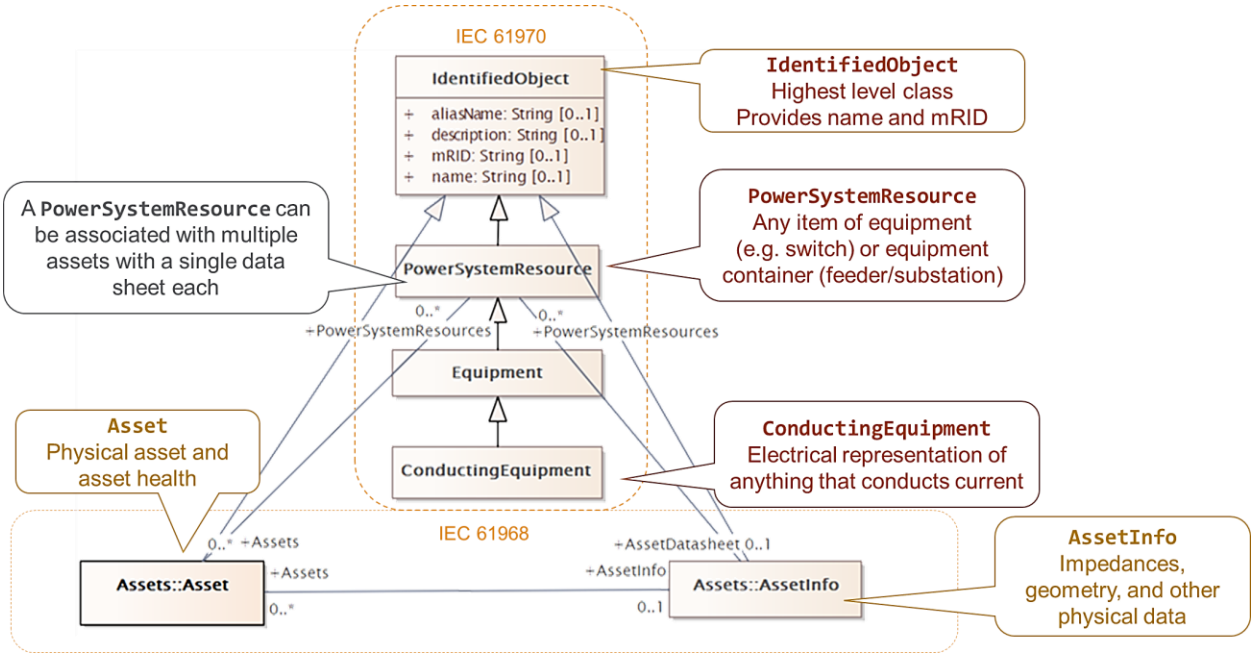


Figure 6: Organization of electrical data into 61970 CIM classes and asset data into 61968 CIM classes.

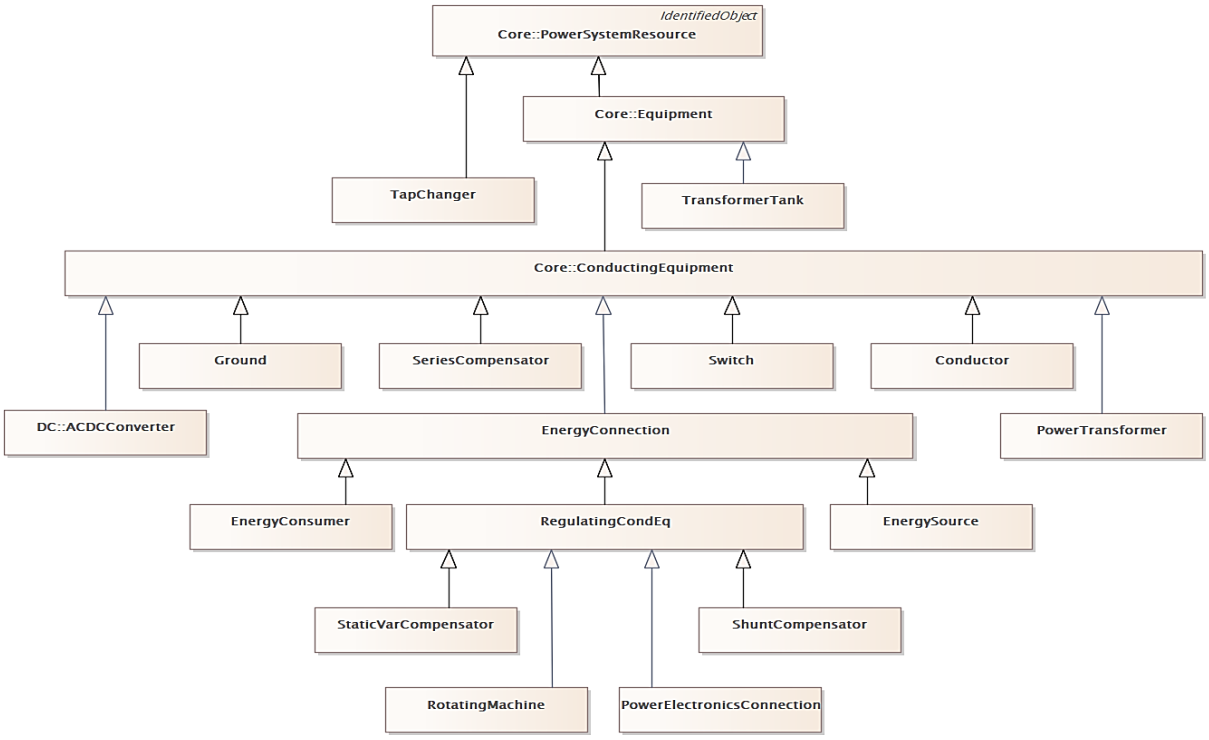


Figure 7: The most commonly used 61970 classes, which cover the majority of electrical network data

Table 1: Summaries of the most commonly used 61970 classes for electrical network data

| CIM Class | Description |
|---|---|
| `ACLineSegment` | Any overhead line or underground cable |
| `EnergyConsumer` | Load |
| `PowerTransformer` | Three-phase transformer |
| `TransformerTank` | Single-phase transformer |
| `RatioTapChanger` | Load tap changing transformer / voltage regulator |
| `ShuntCompensator` | Shunt capacitor and reactor |
| `SynchronousMachine` | Any synchronous generator / motor |
| `AsynchronousMachine` | Type I/II wind turbine / induction motor |
| `PowerElectronicsConnection` | Inverter-based generator / energy storage |
| `BatteryUnit, PhotoVoltaicUnit` | DC side of inverter-based resources |
| `Switch, Breaker, Fuse, Recloser, Sectionaliser, LoadBreakSwitch` | Switching equipment |

# 5.0 Nodes, Terminals, and Measurements

All current-carrying electrical equipment inheriting from the `ConductingEquipment` class are defined as being connected to a `Terminal` object associated with a `ConnectivityNode`. Generators, shunt capacitors, shunt reactors, and loads are connected to one `Terminal` object associated with a single `ConnectivityNode`. Lines, cables, series capacitors, and series reactors have two `Terminal` objects associated with either end of the branch. Rather than defining from/to buses, CIM identifies the end of a branch by setting the `ACDCTerminal.sequenceNumber` attribute of a terminal to values of `1` or `2` to specify the particular end. Transformers are defined using two or three terminals, with the `ACDCTerminal.sequenceNumber` attribute used to designate whether the `Terminal` is associated with the primary, secondary, or tertiary windings. Split-phase distribution transformers also use three terminals, but the two low-side terminals are associated with a single `ConnectivityNode`, as shown in Figure 8 below.

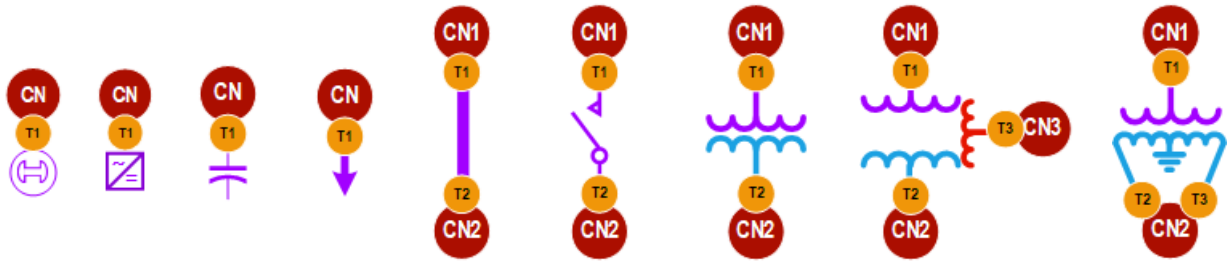


Figure 8: Configuration of `ConnectivityNode` and `Terminal` objects for (from left to right) `SynchronousMachine`, `PowerElectronicsConnection`, `ShuntCompensator`, `EnergyConsumer`, `ACLineSegment`, `LoadBreakSwitch`, two-winding `PowerTransformer`, three-winding `PowerTransformer`, and split-phase distribution `TransformerTank`.

Although this approach may seem excessively detailed and complicated compared to bus-branch modeling used by many analysis tools, the set of `ConnectivityNode` and `Terminal` objects provide a node-edge graph structure built into the power system network model. This graph structure can then be used for highly efficient topology processing and mapping of the electrical network using the `TopologicalNode` and `TopologicalIsland` objects associated with each `ConnectivityNode`.

Consider a data mapping problem in which DERs need to be associated with substation breakers: The entire analysis could be accomplished with only a topological model specifying the association of `ConnectivityNode` and `Terminal` objects to their associated DERs, lines, transformers, and switches. The node-edge graph structure built into the CIM model can then be used to determine which DERs are connected to which breakers by building a spanning tree to determine the associated `Feeder` or `TopologicalIsland` in which the DER is contained. Detailed modeling of other aspects of the power system model (such as would be needed to solve a full power flow solution) is not required, and consequently, a very simple CIM profile could be adopted with far less complexity than would be needed for full model exchange between traditional analysis software packages. The UML class diagram showing the detailed associations between nodes,

terminals, power system equipment, and how they can be organized into a particular feeder and substation is shown in Figure 9.

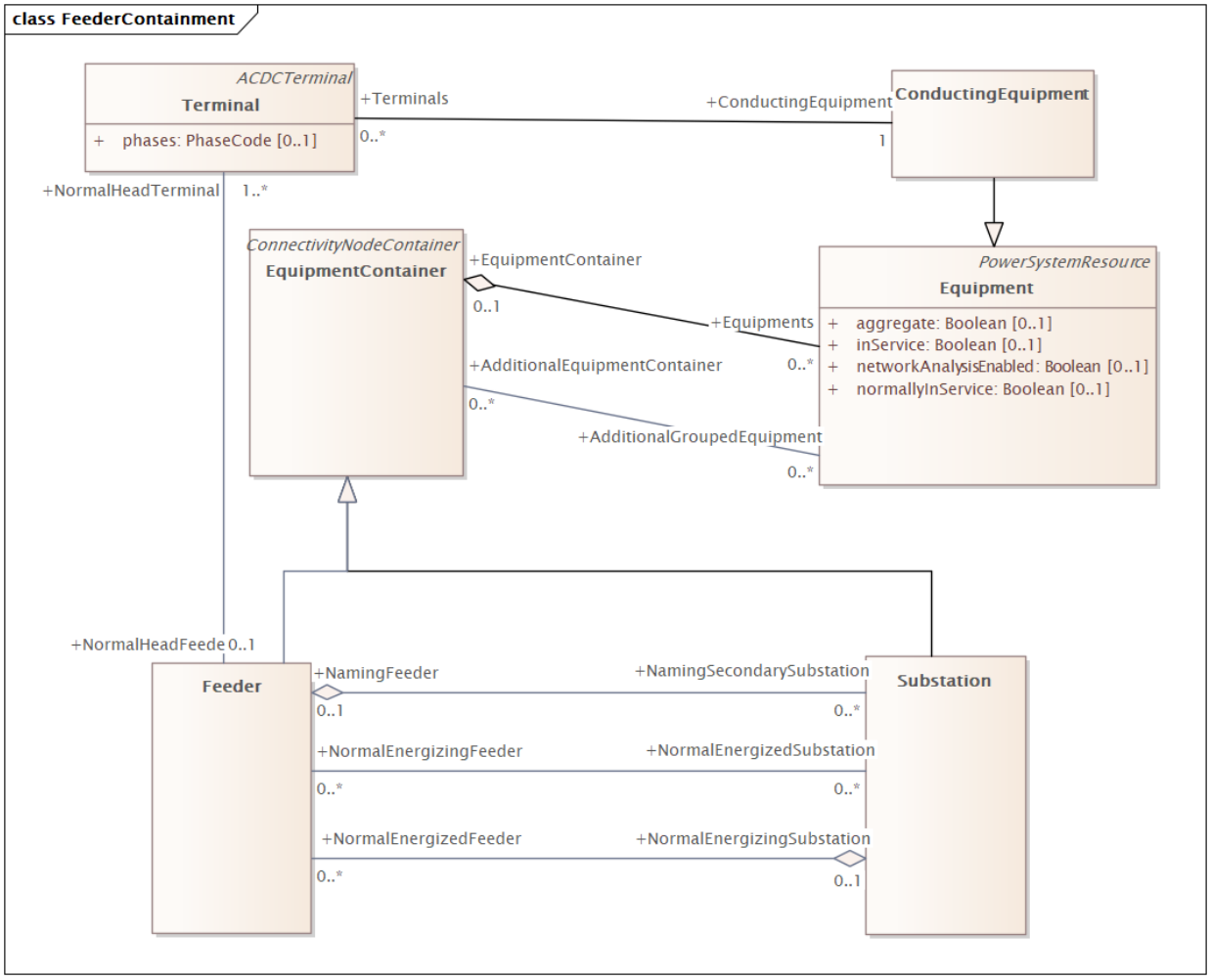


Figure 9: UML class diagram depicting how a piece of `ConductingEquipment` has a set of associated `Terminal` objects, which can be associated with a `Feeder`, which can then be associated with a `Substation`

Associated with each `Terminal` object are measurements of power system parameters. A `Measurement` object is used to represent any direct measurement, calculated value, or non-measured non-calculated value. Examples of possible `Measurement` objects include the position of a transformer tap, current measured by a current transformer (CT), calculated MW flow of a line, the oil temperature of a transformer, or whether a substation door is open. The type of a measurement is specified as a text string through the `Measurement.measurementType` attribute. CIM does not directly specify how measurements are named or defined. Rather, it provides high-level classes for the overall type of `Measurement`, such as `Analog` (e.g. voltage), `Discrete` (e.g. breaker status), and `Accumulator` (e.g. metered kWh).

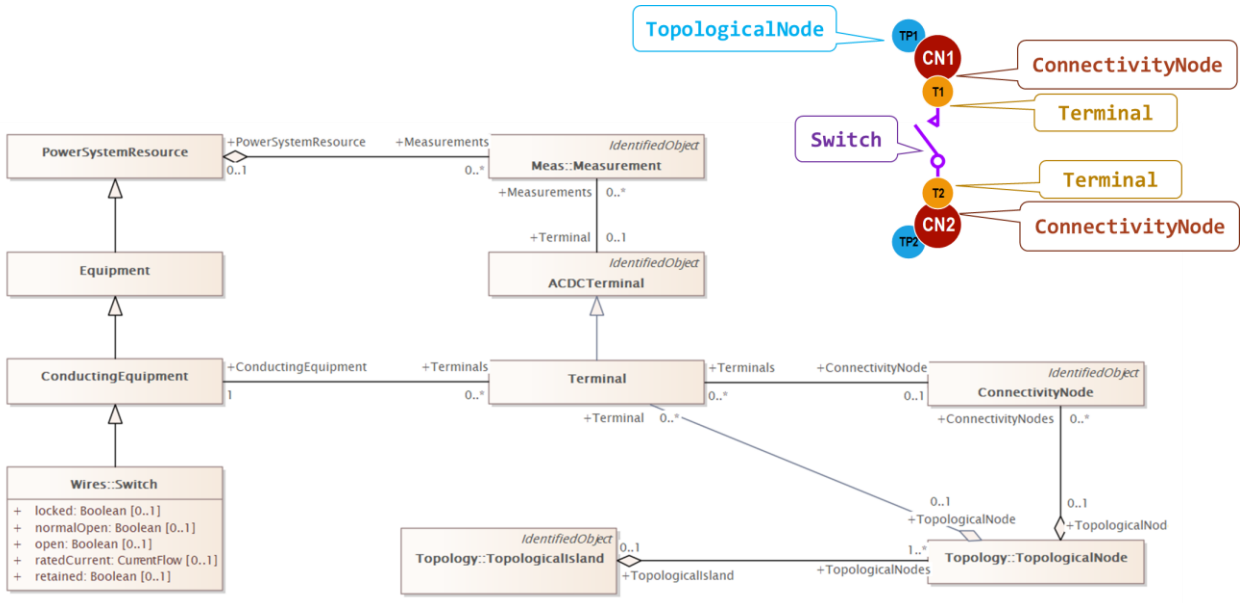


Figure 10: UML class diagram showing that every piece of `ConductingEquipment` has a set of `Terminal` objects through which it is connected to the electrical network. Each `Terminal` has an associated `ConnectivityNode`, which in turn, may have defined a `TopologicalNode` that is used to identify whether the `ConnectivityNode` is contained inside a `TopologicalIsland`. SCADA `Measurement` objects related to power flow are associated with the equipment terminals, rather than with the equipment object itself.

`Measurement` objects associated with power flow quantities (e.g. MW flow at the end of a line) are associated with the `ACDCTerminal` at which the measurement is taken. `Measurement` objects relating to properties of the equipment itself (e.g. oil temperature) are generally associated with the `PowerSystemResource` object for that piece of equipment. Figure 10 illustrates how a `Measurement` is associated to the `MeasurementValue`, `Terminal`, and `PowerSystemResource`. Because CIM objects inherit all of associations of their parent classes, the Measurement can be associated with a specific device, such as a `Switch`.

Figure 11 below highlights the different kinds of measurements available within CIM, including the classes:

- `Analog` – used for voltage, power, current and other continuous measurements
- `Discrete` – used for switch open/closed positions, transformer tap positions, etc.
- `Accumulator` – used for time-integrated measurements (such as meter kWh)

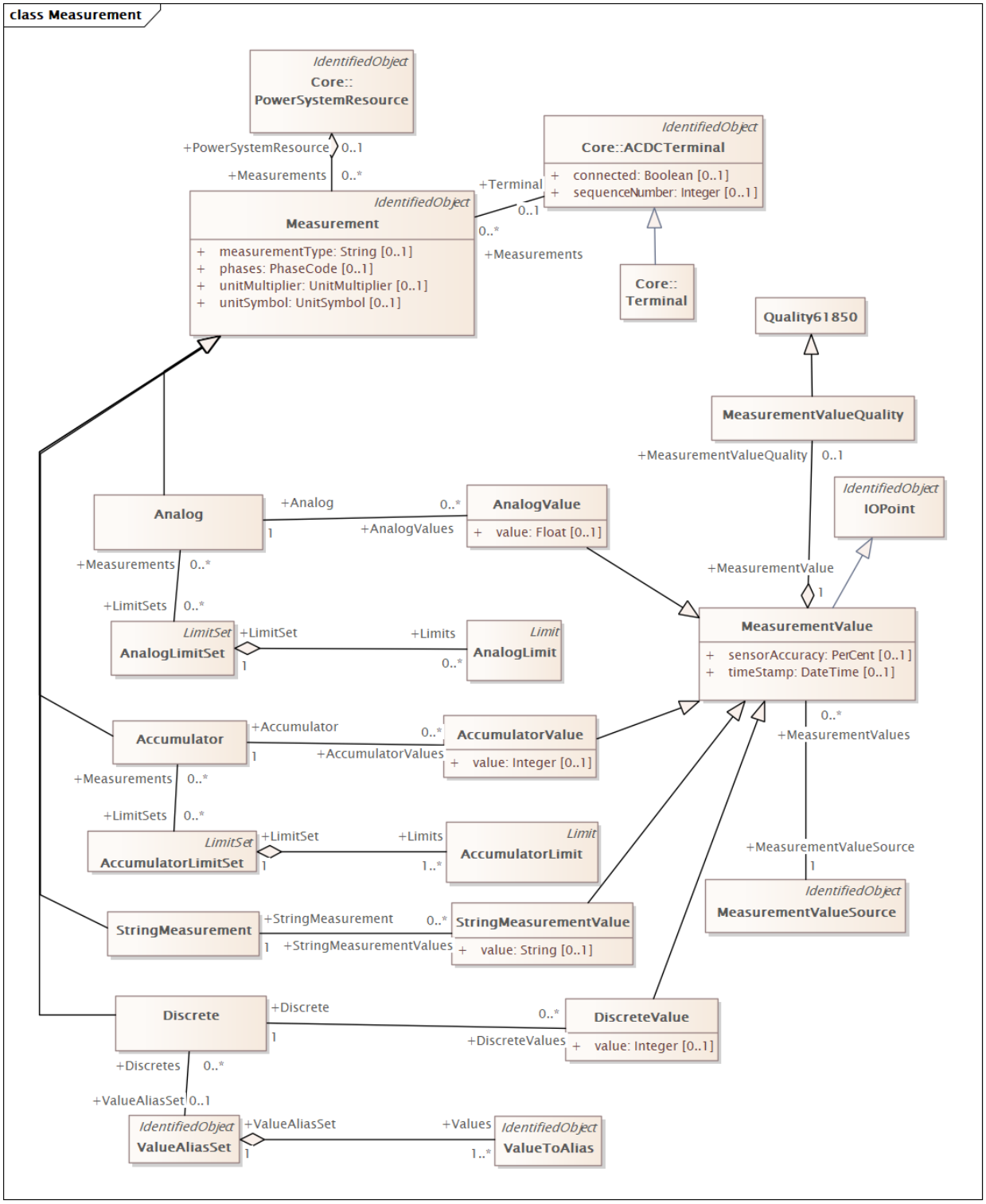


Figure 11: UML class diagram depicting the association of a `Measurement` to an `ACDCTerminal` and/or `PowerSystemResource`. A `Measurement` can be `Analog`, `Discrete`, `Accumulator`, or `StringMeasurement`. The numerical `value` of the measurement is an attribute of `AnalogValue`, `DiscreteValue`, `AccumulatorValue`, or `StringMeasurementValue`, which inherit from the `MeasurementValue` class.

# 6.0 Overhead Lines and Underground Cables

All overhead and underground AC lines are represented as `ACLineSegment` objects. Like all `ConductingEquipment`, all lines and cables are defined with two `Terminal` objects with `ACDCTerminal:sequenceNumber` set to 1 and 2 to represent the two ends of the line, rather than specifying a from-bus and to-bus. There are four different ways to specify ACLineSegment impedances and admittances. The first two use positive and zero sequence values; the third specifies the lower triangular R, X, and B values for each conductor phase; the fourth uses the conductor material, geometry, and spacing. In all cases, the `Conductor:length` attribute is required. A combination of all four methods may be used in a single model to define the network.

The first way, shown in Figure 12, is to specify the individual positive sequence and zero sequence R, X, and B values as `ACLineSegment` attributes, in a manner similar to the method used to define line impedance in many bus-branch transmission analysis tools, such as PSSE. A power system application importing the CIM model would directly use the specified attributes to run a power flow solution. The `PerLengthImpedance` attribute is left as null. Figure 12 below summarizes the classes and attributes used in this method.

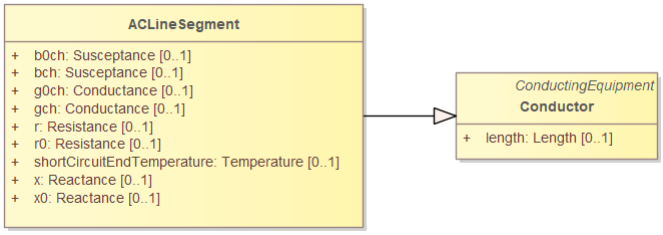


Figure 12: The first method for defining a line is by specifying the positive and zero sequence R, X, and B values directly as `ACLineSegment` attributes.

The second way, shown in Figure 13, is to specify the positive and zero sequence impedance and admittance values on a per-unit-length basis as attributes of the `PerLengthSequenceImpedance` class. A power system application importing the CIM model would multiply the specified R, X, and B values by the `Conductor:length` to determine the overall line impedance. When using this method, all the `ACLineSegment` attributes should be null.

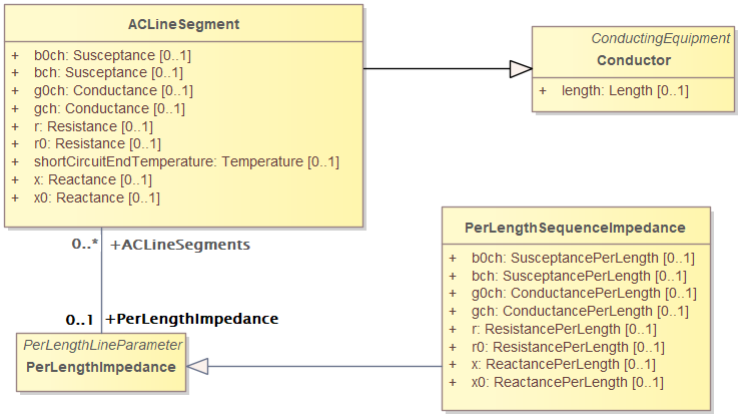


Figure 13: The second method is defining the positive and zero sequence impedances on a per unit length basis (e.g. per-km or per-mile) as attributes of the `PerLengthSequenceImpedance` class

The third way to specify line parameters, shown in Figure 14, is to define R, X, and B values for each conductor phase. This method is more useful for distribution networks for which an unbalanced power flow solution is needed. The impedance and admittance values are specified as attributes of the `PhaseImpedanceData` class, which inherits from `PerLengthPhaseImpedance`. Again, all the `ACLineSegment` attributes are left null. Only `conductorCount` from 1 to 3 is supported, and there will be 1, 3 or 6 reverse-associated `PhaseImpedanceData` instances that define the lower triangle of the Z and Y matrices per unit length. The `row` and `column` attributes must agree with `ACLineSegmentPhase:sequenceNumber`.

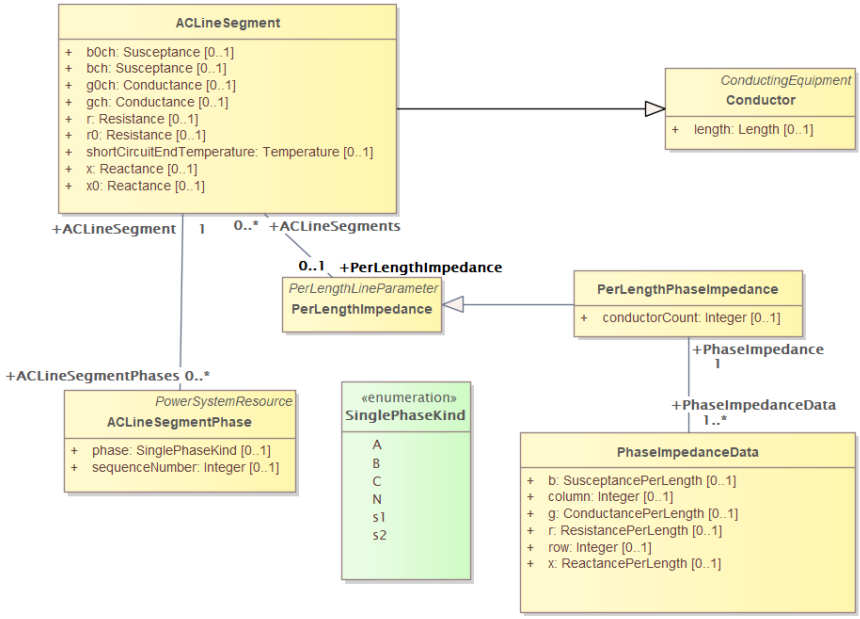


Figure 14: The third method for specifying line parameters is lower triangular R, X, and B values through `PhaseImpedanceData` attributes for each conductor phase, as specified by `ACLineSegmentPhase`

The fourth way, shown in Figure 15, is to specify wire/cable geometry and spacing data instead of impedances. Unlike the previous three methods (where the impedance values were specified through the 61970 `Wires` package), all attributes of the line are specified through physical attributes as part of the 61968 `AssetInfo` package. Conductor spacing is specified by association to `WireSpacingInfo` and `WirePosition`. Conductor geometry is specified through the attributes of `WireInfo`. Cables are specified through the `CableInfo` class and associated ConcentricNeutralCableInfo and `TapeShieldCableInfo` classes.

If there are `ACLineSegmentPhase` instances reverse-associated to the `ACLineSegment`, then per-phase modeling applies. There are several use cases for the `ACLineSegmentPhase` class:

1) single-phase, two-phase, or three-phase unbalanced primary lines
2) low-voltage secondary lines using phases s1 and s2
3) associated `WireInfo` data where the `WireSpacingInfo` association exists
4) assign specific phases to the matrix rows and columns in `PerLengthPhaseImpedance`.

It is the application's responsibility to propagate phasing through terminals to other components, and to identify any miswiring. It is also the application's responsibility to calculate the impedance and admittance values for the electrical network from the conductor geometry and spacing. The associations between all the `AssetInfo` classes described above are shown in Figure 15.

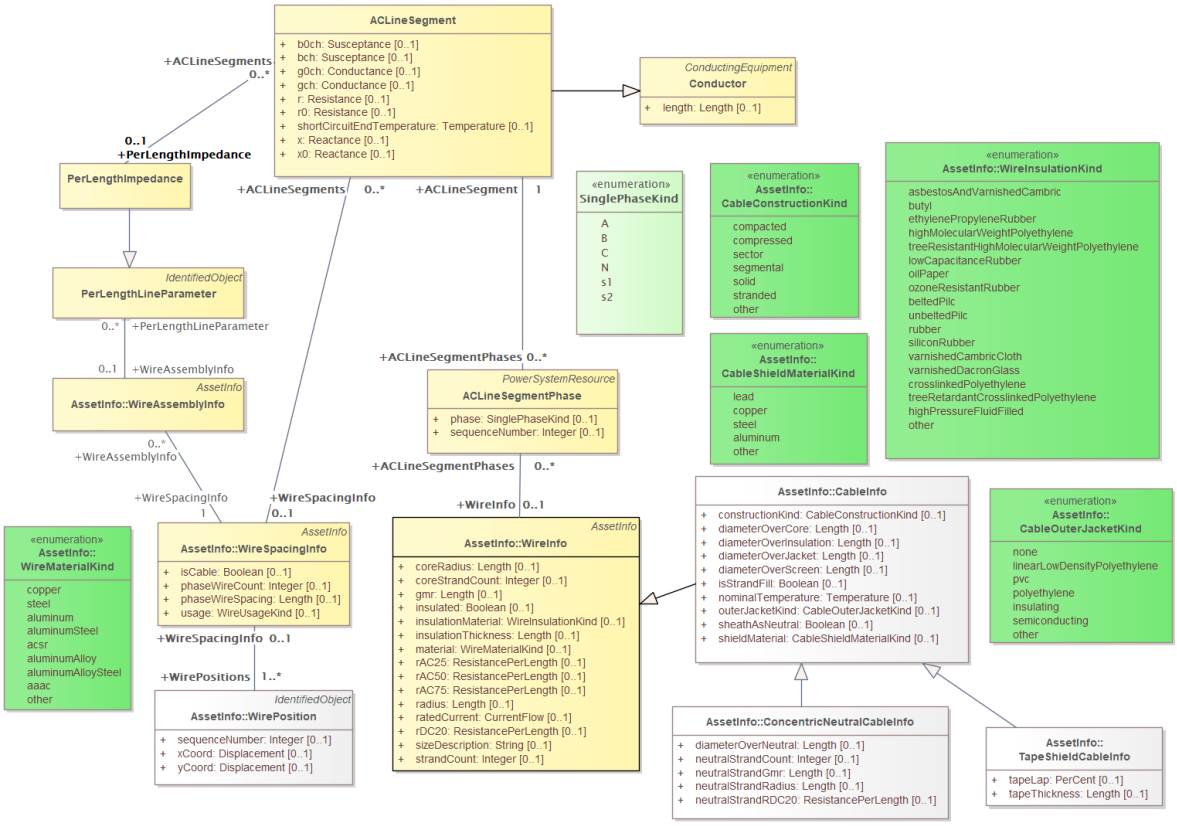


Figure 15: The fourth method for specifying line impedance is through the physical geometry and spacing of overhead conductors and underground cables, as specified by a set of classes as part of the 61968 `AssetInfo` package

# 7.0 Transformers and Tap Changers

The 61970 Wires package defines several classes for representing the different components of transformers. The first is PowerTransformer, which represents the electrical network representation of a transformer for both balanced and unbalanced circuits. The second is TransformerTank, which refers to the assembly of two or more windings placed inside a tank and can be used to model single-phase and three-phase transformers. TransformerEnd is the conducting connection point of a transformer and corresponds to the terminal of a particular winding. As with all ConductingEquipment, the PowerTransformer is connected through a set of Terminal objects, with the particular winding indicated by the TransformerEnd.endNumber attribute. The highest voltage winding should have an endNumber of 1. The endNumber does not need to match the ACDCTerminal.sequenceNumber attribute of the Terminal to which the transformer is connected.

The BaseVoltage and Terminal are associated with the TransformerEnd of each winding, rather than with the PowerTransformer itself. Figure 16 shows the associations between the different classes used to specify transformer parameters and windings.

PowerTransformer objects may be modeled with or without specifying TransformerTank objects. In both cases the PowerTransformer.vectorGroup attribute for protective relaying should be specified according to IEC transformer standards (e.g., Dy1 for many substation transformers).

The case without specifying TransformerTank objects is most suitable for balanced three-phase transformers that will not reference any reusable asset catalog data. This approach is typically used for transmission system modeling, where each transformer is unique. Each winding will have a PowerTransformerEnd that associates to both a Terminal and a BaseVoltage, and the parent PowerTransformer. The impedance and admittance parameters are defined by reverse-associated TransformerMeshImpedance between each pair of windings, and a reverse-associated TransformerCoreAdmittance for one winding. The units for these are ohms and siemens based on the winding voltage, rather than per-unit. WindingConnection is similar to PhaseShuntConnectionKind, adding Z and Zn for zig-zag connections and A for autotranformers. TransformerStarImpedance is used for conversion of three-winding transformers to separate two-winding equivalents, which is common practice in numerous power flow solvers.

If the transformer is unbalanced in any way, then TransformerTankEnd is used instead of PowerTransformerEnd, and then one or more TransformerTank objects may be used in the parent PowerTransformer. Some of the use cases are 1) center-tapped secondary, 2) open-delta and 3) EHV transformer banks. Tank-level modeling is also required if using catalog data to specify physical equipment ratings, etc. through the AssetInfo package.

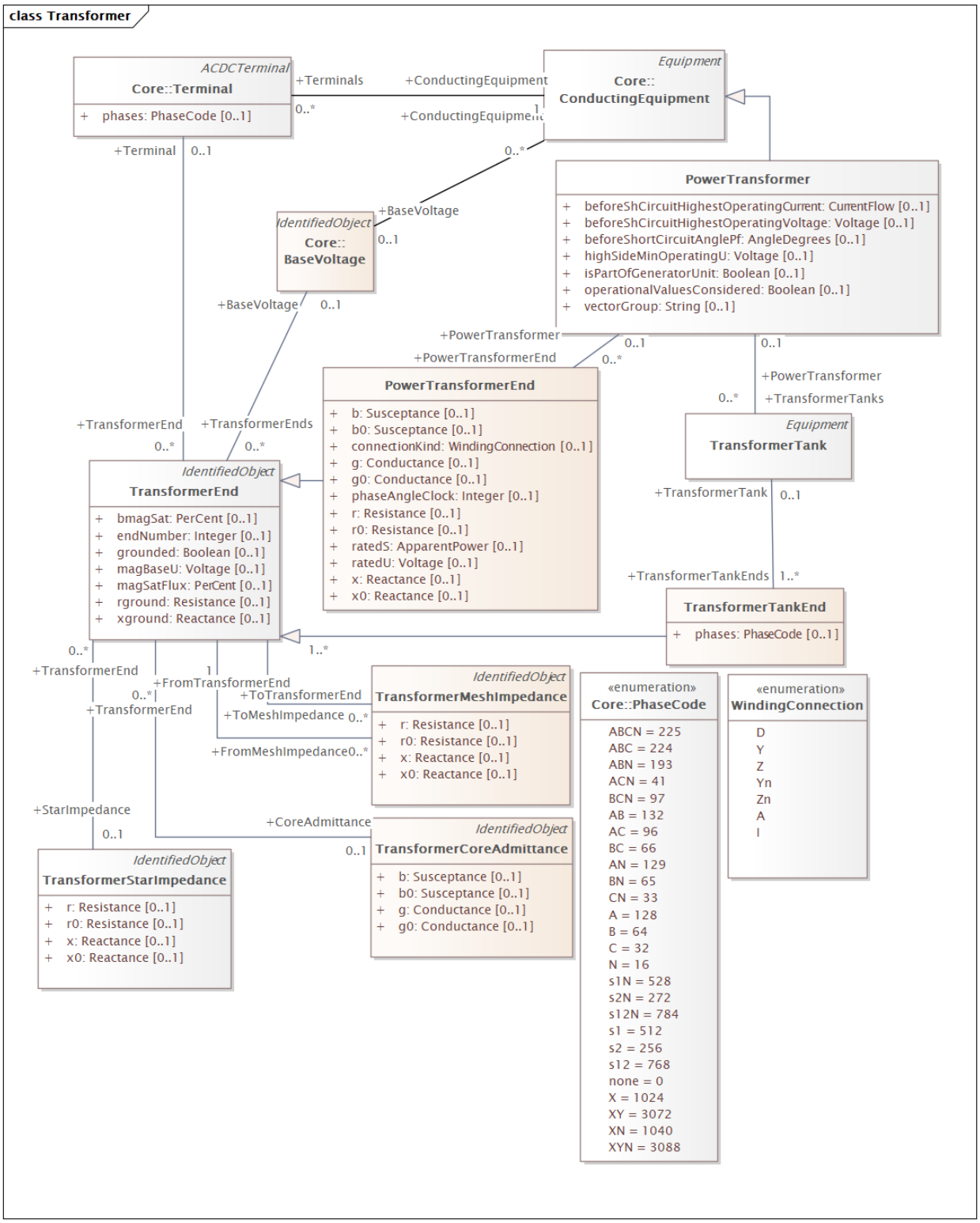


Figure 16: UML class diagram for `PowerTransformer`, `TransformerTank`, and `TransformerEnd` objects.

Many distribution software packages use the concept of catalog data, aka library data, especially for lines and transformers. This concept is implemented in CIM through the ability to define a single set of class definitions using the IEC 61968 `AssetInfo` package to save a large amount of space when defining customer secondary transformers (which typically comprise hundreds or thousands of identical poletop and pad-mounted installations). A particular transformer design and rating is then defined by creating `PowerTransformerInfo` and `TransformerTankInfo` library objects that are associated with each type of specification objects.

The rated voltage, rated current, and resistance of each winding are defined as attributes of a `TransformerEndInfo` object that is created for each transformer winding. It is important that the `TransformerEndInfo.endNumber` of the physical asset match the `TransformerEnd.endNumber` of its representation in the electrical circuit. The shunt admittances are defined by `NoLoadTest` on a winding / end, with usually just one such test. The impedances are defined by a set of attributes

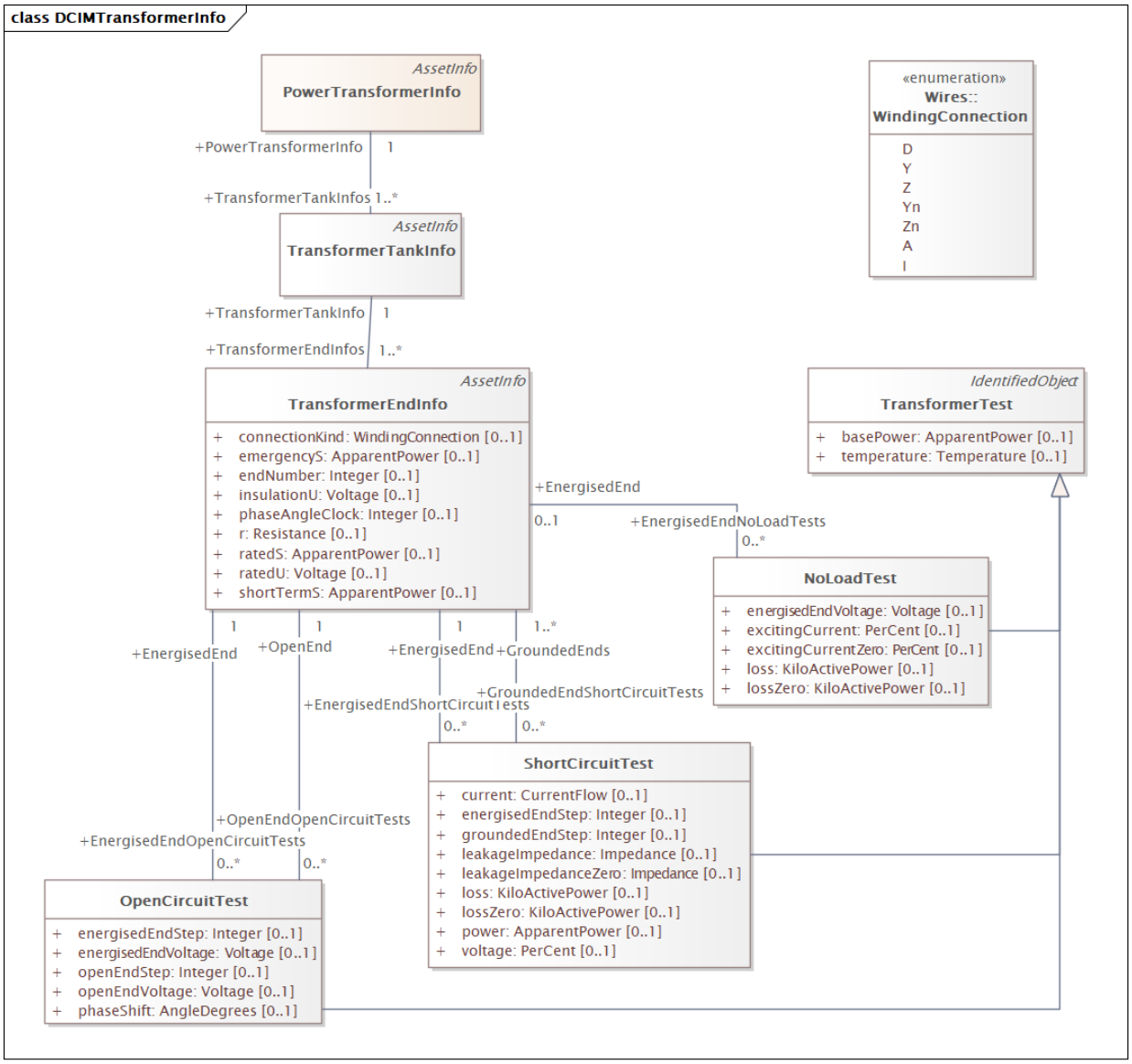


Figure 17: UML class diagram of AssetInfo classes used to specify transformer ratings

of ShortCircuitTest; one winding / end will be energized, and one or more of the others will be grounded in these tests. The complete list of asset properties is summarized in Figure 17. Note that these classes are associated with TransformerTankInfo (not PowerTransformerInfo) because transformer testing is done on tanks.

The TapChanger class is used to model both phase-shifting transformers and voltage regulators through the PhaseTapChanger and RatioTapChanger classes, which are associated with the particular TransformerEnd. The highest, lowest, and neutral tap positions available are specified as positive integers such that a TapChanger with 16 tap positions would have attributes of lowStep set to 0, neutralStep set to 8, and highStep set to 16. If a particular application uses a range of -8 to 8 for the same transformer tap range, it is the responsibility of application to convert the tap ranges to the format used internally.

If a voltage regulator uses line drop compensation, then those parameters will be defined as attributes of the TapChangerControl class, which inherits from RegulatingControl. RegulatingControl is a higher-level class that is used to specify the control mode and setpoints for numerous types of RegulatingCondEq, such capacitors, reactors, SVC, and generator automatic voltage regulation controls. Whether a particular device is regulating voltage, activePower, reactivePower, etc. is specified by the RegulatingControl:mode attribute. Other control settings, such as targetValue, targetDeadband, etc. are also attributes of RegulatingControl, as shown in Figure 18 below.

In summary, a single-phase line voltage regulator modeled in CIM includes a PowerTransformer, a TransformerTank, a TransformerTankEnd, a RatioTapChanger, and a TapChangerControl. The CT and PT parameters of a voltage regulator can only be described via the AssetInfo mechanism, described below. The RegulationControl.mode must be voltage. Older CIM versions used the tculControlMode attribute, which is now deprecated.

The AssetInfo package in the 61968 package defines the TapChangerInfo class with a set of attributes ctRating, ctRatio, and ptRatio needed for line drop compensator settings in voltage regulators. Catalog data is a one-to-many relationship. In this case, many TapChangers can share the same TapChangerInfo data, which saves space and provides consistency. Older versions of CIM had many-to-many catalog relationships, but now only one AssetDataSheet may be associated per Equipment.

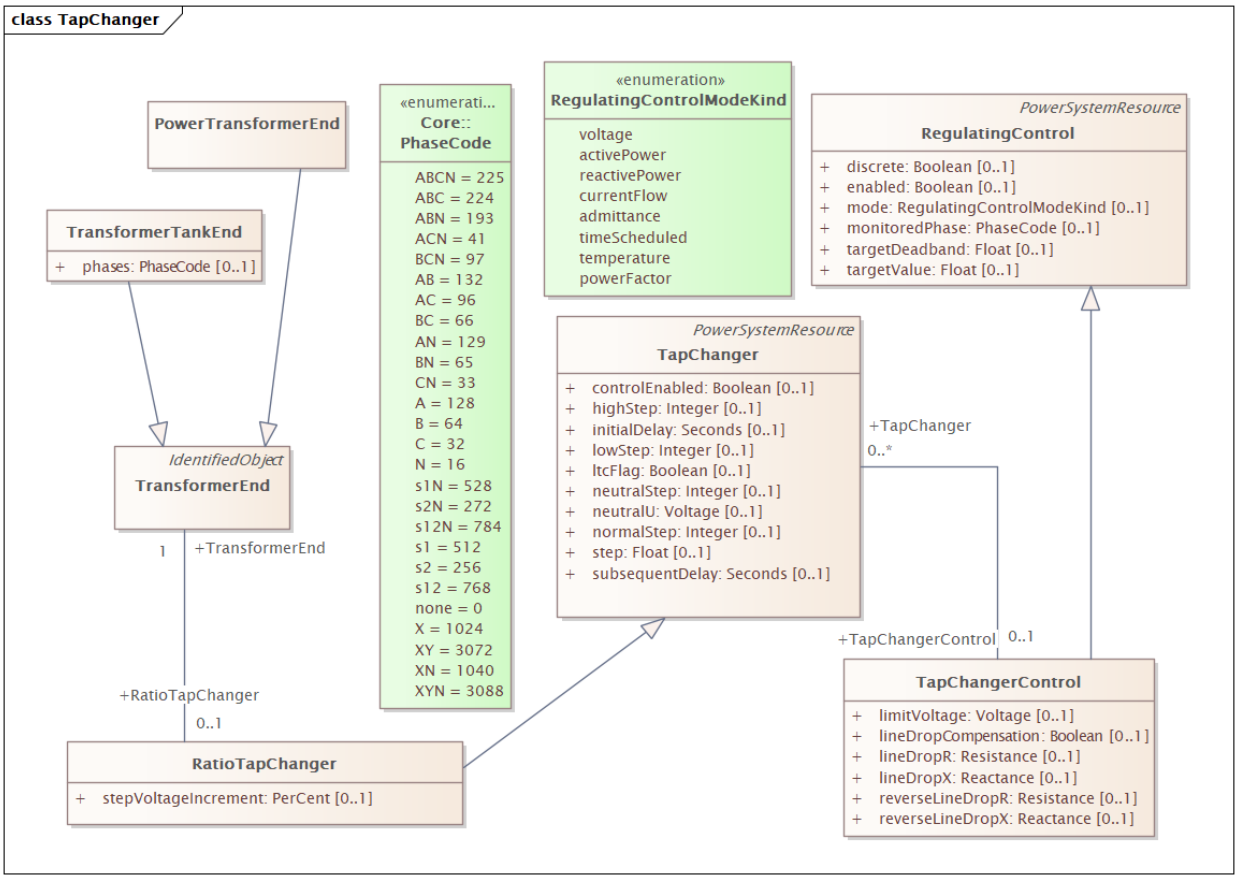


Figure 18: UML class diagram for modeling of voltage regulators. Classes and attributes for phase-shifting transformers are not shown.

# 8.0 Breakers, Fuses, Reclosers, and other Switching Devices

There are eight different kinds of `Switch` supported in the CIM, all with zero impedance. They will all behave the same in power flow analysis, and all would require many more attributes than are defined in CIM to support protection analysis. The classes of `Switch` defined in CIM are divided into two categories depending on whether the device is capable of interrupting fault currents, as shown in Figure 19 below. Multiple switches in a single vault or pad-mounted switchgear can be grouped using the `CompositeSwitch` class.

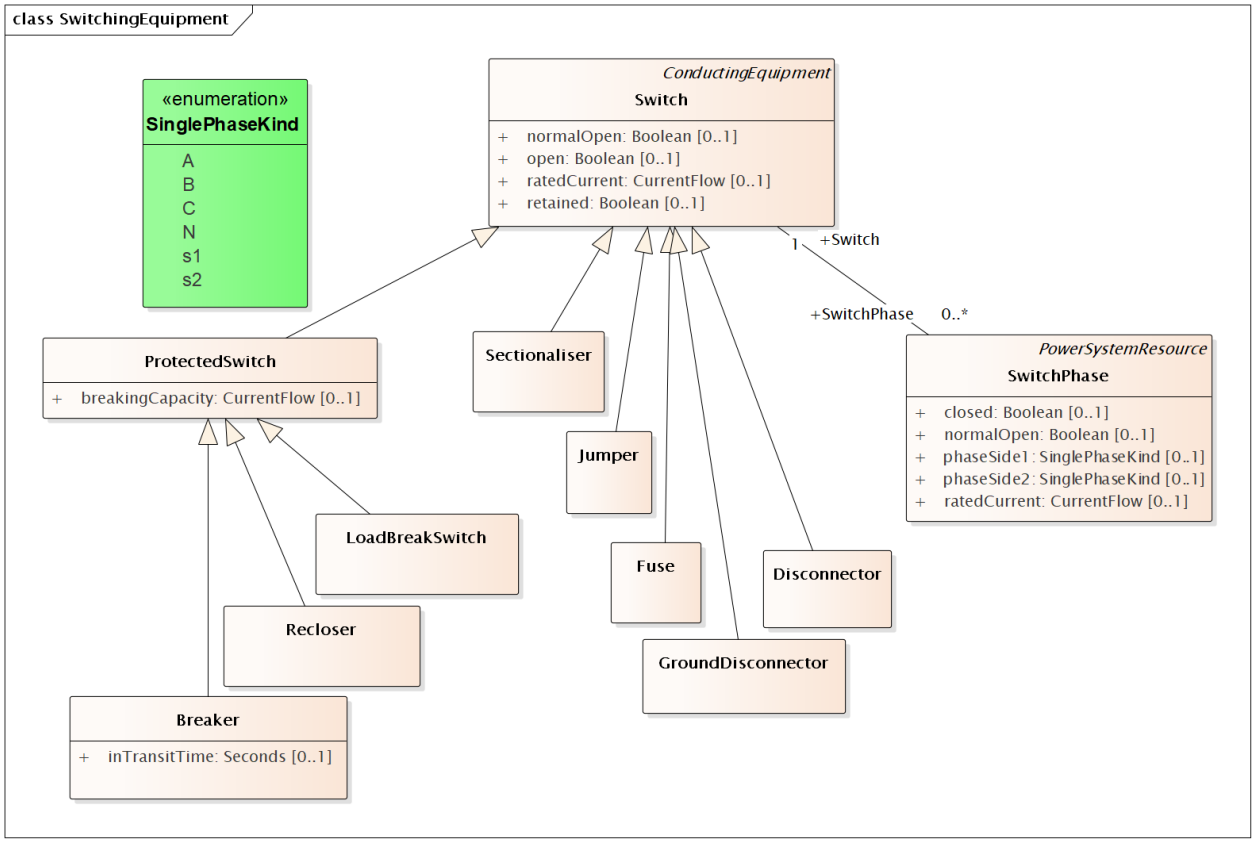


Figure 19: UML class diagram of switching equipment classes defined in CIM

Devices inheriting from `ProtectedSwitch` include:

1) `LoadBreakSwitch`: a generic mechanical switching device capable of breaking current under normal operating conditions

2) `Recloser`: a pole-mounted fault interrupter with built-in phase/ground relays and CT

3) `Breaker`: a substation circuit breaker

Other switching devices include:

1) `Sectionaliser`: a switching device locked open to isolate a faulted section, with or without any load-breaking capability (note that the class is spelled with a letter "s" rather than "z").

2) `Fuse`: a protective fuse used on lines, transformers, and other equipment. Note that `Fuse` does not inherit from `ProtectedSwitch` because fuses do not use a protection relay.

3) `Jumper`: A short section of conductor with negligible impedance that is manually installed by field crews

4) `Disconnector`: a manually-operated or motor-operated switch used to isolate circuits or equipment in a substation, not capable of breaking any significant current

5) `GroundDisconnector`: a manually-operated or motor-operated switch used to ground equipment for maintenance in a substation.

In unbalanced circuits, individual phases of a switch are modeled as attributes of the `SwitchPhase` class. The use cases for `SwitchPhase` include 1) single-phase, two-phase, and secondary switches, 2) one or two conductors open in a three-phase switch or 3) transpositions, in which case `phaseSide1` and `phaseSide2` would be different. `RatedCurrent` may be different among the phases, e.g., individual fuses on the same pole may have different ratings. Note that the `Switch` class uses the Boolean attribute `Switch:open` (with `0` meaning closed and `1` open), while the `SwitchPhase` class uses the attribute `SwitchPhase.closed`.

# 9.0 Shunt Capacitors and Reactors

Shunt capacitors and reactors are both defined using the class `ShuntCompensator`, which covers both individual shunts and switchable banks. If the shunt has bank or section admittance values that differ, then the `NonLinearShuntCompensator` class is used. Otherwise, the `LinearShuntCompensator` class is used.

CIM does not use the kVAr or MVAr rating of the capacitor bank or reactor, but rather the positive and zero sequence conductance and susceptance of the shunt module, specified by the `bPerSection`, `gPerSection`, `b0PerSection`, and `g0PerSection` attributes of `LinearShuntCompensator`. Capacitor banks that are controlled on a single-phase basis define a `ShuntCompensatorPhase` object for each phase, with the conductance and susceptance of each phase defined as attributes of `LinearShuntCompensatorPhase`. The Boolean attribute `ShuntCompensator.sections` specifies number of capacitor bank sections currently switched in, with a value of `0` indicating that all sections are switched out, `1` meaning one capacitor bank section is switched in, and so on.

Note that `aVRDelay` is really a capacitor control parameter, to be used in conjunction with `RegulatingControl` on the right-hand side of Figure XX. The `RegulatingControl` class associates to the controlled capacitor bank via `RegulatingCondEq`, and to the monitored location via the `Terminal` object to which the shunt is connected. There is no support for a PT or CT ratio, so `targetDeadband` and `targetValue` have to be in primary volts, amps, vars, etc. Capacitor banks may respond to any of the `RegulatingControlModeKind` choices, but it is not expected that capacitor switching will successfully regulate to the `targetValue`

Static VAr Compensators (SVCs) are defined in a separate class, `StaticVarCompensator`, which also inherits from `RegulatingCondEq`, but does not share any other attributes or associations with `ShuntCompensator`. SVC settings (such as `voltageSetpoint`, `slope`, and `sVCControlMode`) are specified as attributes of the `StaticVarCompensator` class. Any further attributes must be defined as part of the `Dynamics` package.

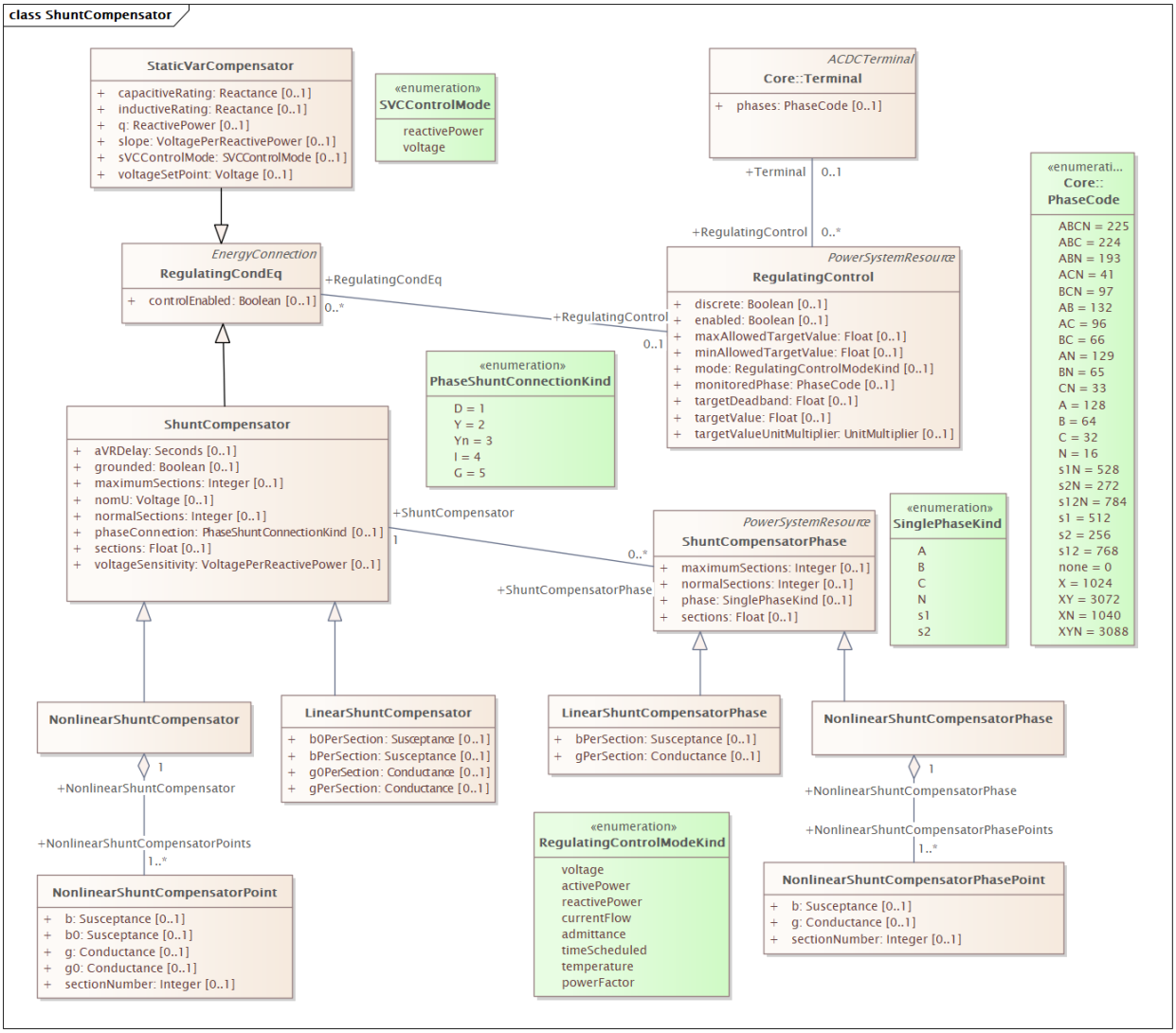


Figure 20: UML class diagram for `ShuntCompensator` objects used to model shunt capacitors and reactors. SVCs are defined as an entirely separate class inheriting from `RegulatingCondEq` but are still included in this diagram for completeness of modeling shunt reactive control equipment.

# 10.0 Load Modeling

CIM offers multiple means of modeling loads in various levels of detail using the classes `EnergyConsumer`, `ConformLoad`, and `NonConformLoad`. Figure 21 below illustrates four levels of modeling accuracy, which will be described in detail below.

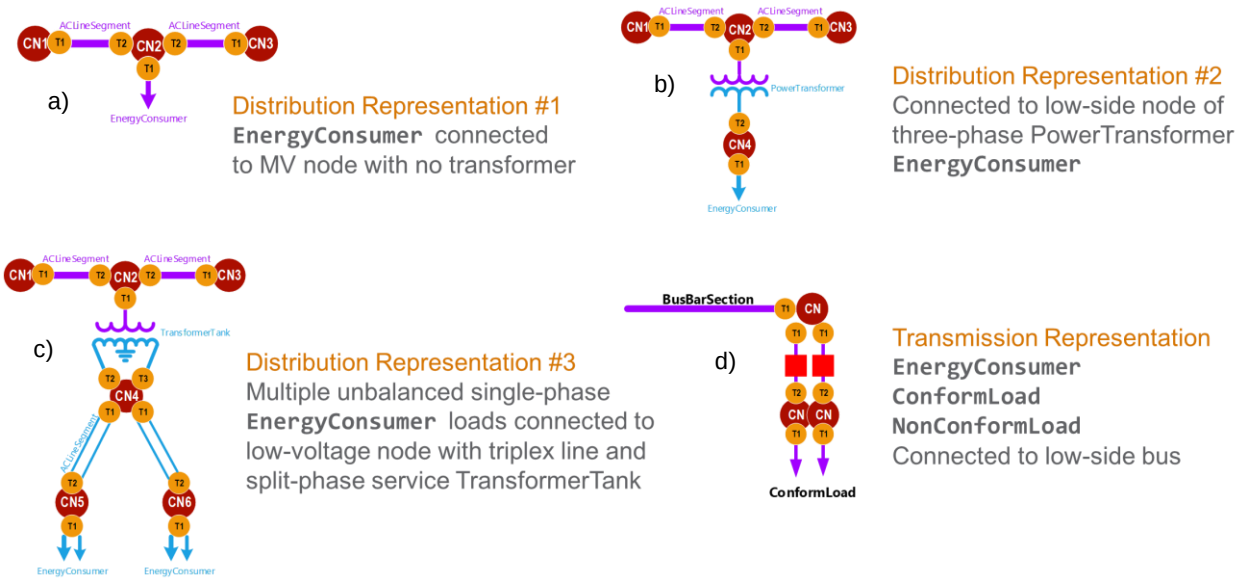


Figure 21: Distribution load modeling approaches with varying levels of detail of secondary equipment supported by CIM. `ConnectivityNode` objects are represented with large red circles and `Terminal` objects with smaller orange circles.

## 10.1 Distribution Load Modeling

Customer loads can be represented at three levels of detail depending on the requirements of the modeling effort. Each load in the distribution feeder is defined as an `EnergyConsumer` object, which may be a single metered customer or composed of multiple customers, as specified by the `customerCount` attribute. Like all `ConductingEquipment`, every `EnergyConsumer` has an associated `Terminal` to which it is connected.

The simplest method places an aggregated `EnergyConsumer` object at the terminal where the high-side of the service transformer would be connected, as depicted in Figure 21 a). A single `Terminal` object from the `EnergyConsumer` is associated with the `ConnectivityNode` to which it is connected. The first two representations can be used for both single-phase and three-phase loads.

The second level of detail includes the customer transformer as a simple two-winding `PowerTransformer` object with a single `Terminal` associated with the `EnergyConsumer` object and the `ConnectivityNode` to which the transformer low-side winding is connected, as depicted in Fig 21 b). Multiple customers served by the service transformer can be aggregated into a single `EnergyConsumer` object representing the total load.

The last and most detailed representation explicitly models the split-phase secondary transformer, two-phase triplex lines, and individual unbalanced single-phase customer loads, as shown in Figure 21 c). Multiple customers are often connected to the low-side of the transformer, each with a length of overhead or underground secondary triplex line. Due to the split-phase configuration, the individual s1 and s2 phases of the `EnergyConsumer` and triplex `ACLineSegment` are considered a single object with each modeled with just one Terminal for both phases. However, the split-phase `TransformerTank` is topologically represented by three `Terminal` and two `ConnectivityNode` objects because the s1 and s2 phases are connected to different windings of the service transformer.

Phasing of loads is specified by the `EnergyConsumerPhase:phase` attribute. For three-phase delta loads, the `EnergyConsumer:phaseConnection` is `D` and the three reverse-associated `EnergyConsumerPhase` instances will have `phase=A` for the AB load, `phase=B` for the BC load and `phase=C` for the AC load. A three-phase wye load may have either `Y` or `Yn` for the `phaseConnection`.

Load values can be defined in two ways. The first is by directly specifying the real and reactive power values of the load through the `EnergyConsumer:p` and `EnergyConsumer:q` attributes. The second is as a ZIP load with specified constant impedance (Z), constant current (I), and constant power (P) components specified by the attributes of the `LoadResponseCharacteristic` class.

Split-phase secondary loads use `phase=s1` and `phase=s2` to represent the unbalanced load components of household loads in North America. Plug loads and lighting are connected at 120V from `s1` or `s2` phase to neutral and appear as unbalanced load components specified by the `EnergyConsumerPhase:p` and `EnergyConsumerPhase:q` attributes for each `s1` or `s2` phase load object. Larger loads are connected at 240V from `s1` to `s2` are divided equally into the phase load `p` and `q` values. If the secondary triplex lines and split-phase transformer are explicitly modeled, `phaseConnection=Y`.

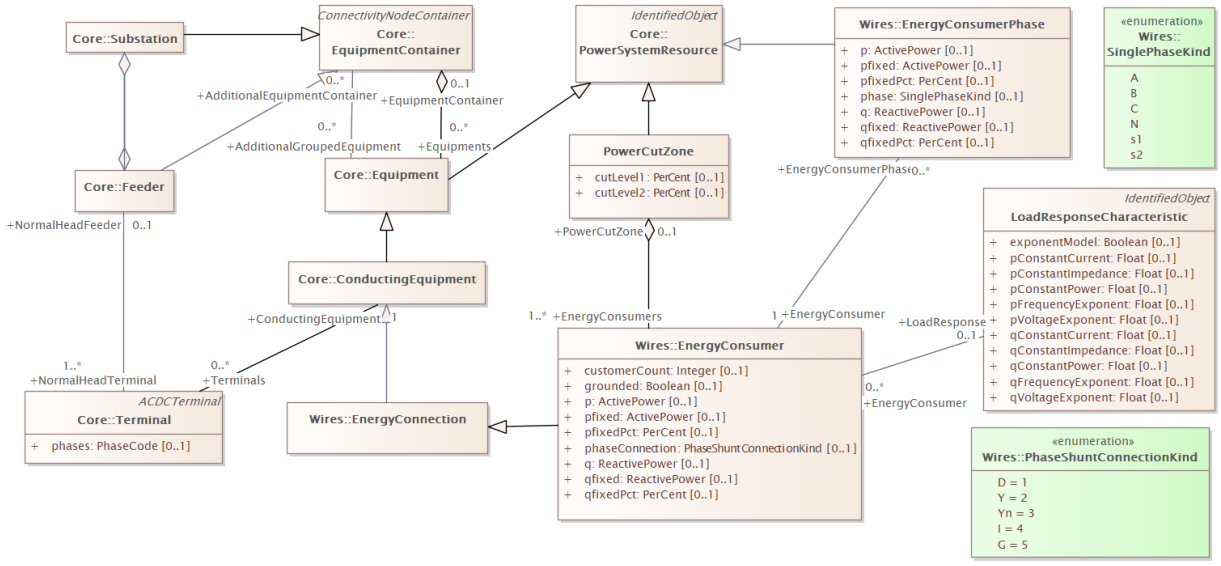


Figure 22: UML class diagram showing the associations between EnergyConsumer, Feeder, Substation, and PowerCutZone.

## 10.2 Transmission Load Modeling

The `EnergyConsumer` class can also be used to represent transmission loads at the substation level. All transmission loads are assumed to be three-phase balanced, and so `EnergyConsumerPhase` is not specified. Loads are categorized as `ConformLoad` and `NonConformLoad` depending on whether the load follows a daily pattern. `ConformLoad` objects are the most common representation of aggregated feeder load at the substation level. `NonConformLoad` objects are sometimes used in studies and simulations to represent load shedding (with a negative value) or cold load pickup values after a blackout. All `EnergyConsumer` objects that follow the same load curve can be aggregated into a `ConformLoadGroup` or `NonConformLoadGroup`, as shown in Figure 23.

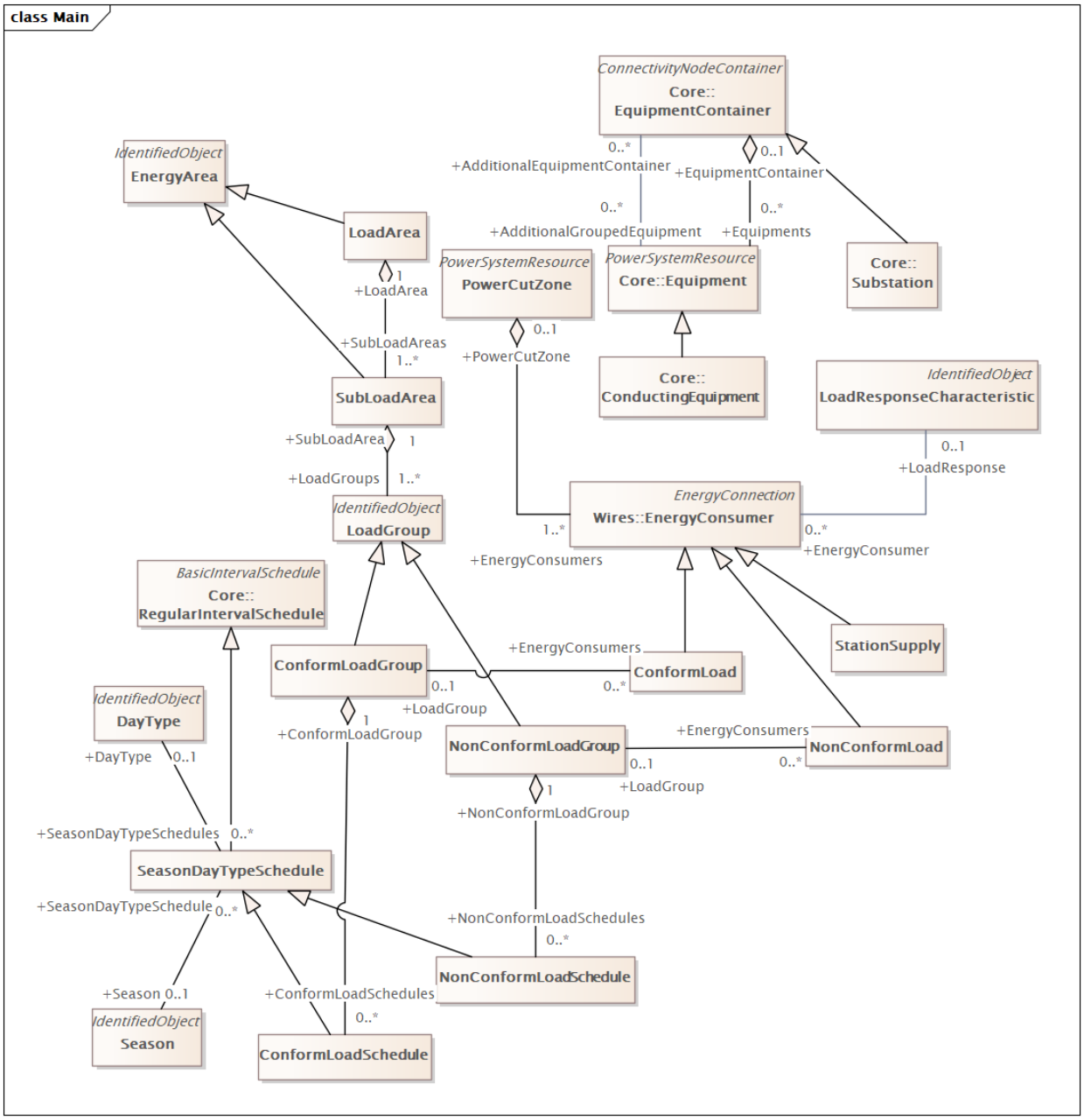


Figure 23: Representation and grouping of conforming and nonconforming loads at the transmission level

Real and reactive load curves for all loads within a particular group are specified using either `ConformLoadSchedule` or `NonConformLoadSchedule`, which both inherit from the `SeasonDayTypeSchedule` class. This is also the parent class for schedules of `Switch`, `TapChanger`, and `RegulatingControl` objects, as shown in Figure 24 below. Each schedule is associated with a `DayType` and a `Season`, which are used to group similar days, such as weekdays, weekends, and holidays.

The load curve itself is defined as a series of `RegularTimePoint` objects, where the time between consecutive points is equal. Following the practice of many datastores to omit repeated values, the set of time points do not need to be sequential. The timestamp associated with a particular point is not specified as a date/time, but rather using the `sequenceNumber` attribute. The timestamp is calculated by multiplying the value of the `sequenceNumber` attribute by the time interval between points in the load curve. The starting time has a `sequenceNumber` of zero and is specified by the `startTime` attribute of the `BasicIntervalSchedule` class.

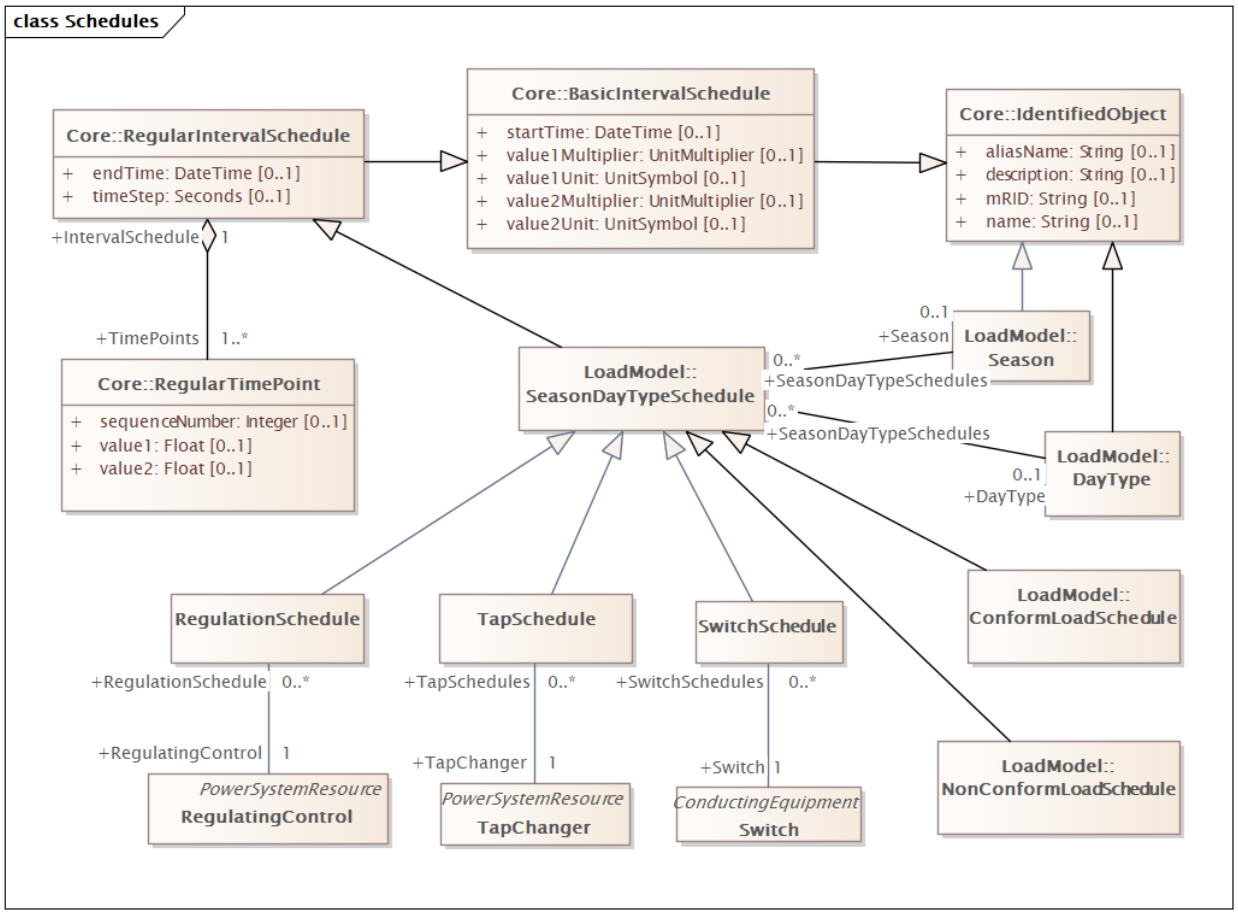


Figure 24: UML class diagram of schedule objects used to describe load curves, as well as pre-programmed schedules used by some equipment, such as tap changers.

# 11.0 Generator Modeling

Representation of conventional generators in CIM is divided into three packages across the 61970 information model. The set of classes in the `Wires` package contains the electrical representation of units for use in basic power flow studies and modeling. The set of classes in the `Generation` package contains prime mover information for use in a dispatcher training simulator and power applications, such as economic dispatch. Finally, the set of classes in the `Dynamics` package is used for transient stability and electromechanical simulations.

## 11.1 Synchronous and Asynchronous Machines

The two primary classes used for representation of generators for use in power flow studies and modeling of the electrical network are `SynchronousMachine` and `AsynchronousMachine`. All conventional thermal, hydro, and other synchronously-connected (i.e. rotating at an even multiple of 50Hz or 60Hz) is represented using the `SynchronousMachine` class. This class is also used for large motor loads connected through synchronous motors. Induction motors and Type I / Type II wind turbines are represented by the AsynchronousMachine class. The operating modes and usage are specified as enumerations, rather than as separate classes, as shown in Figure 25.

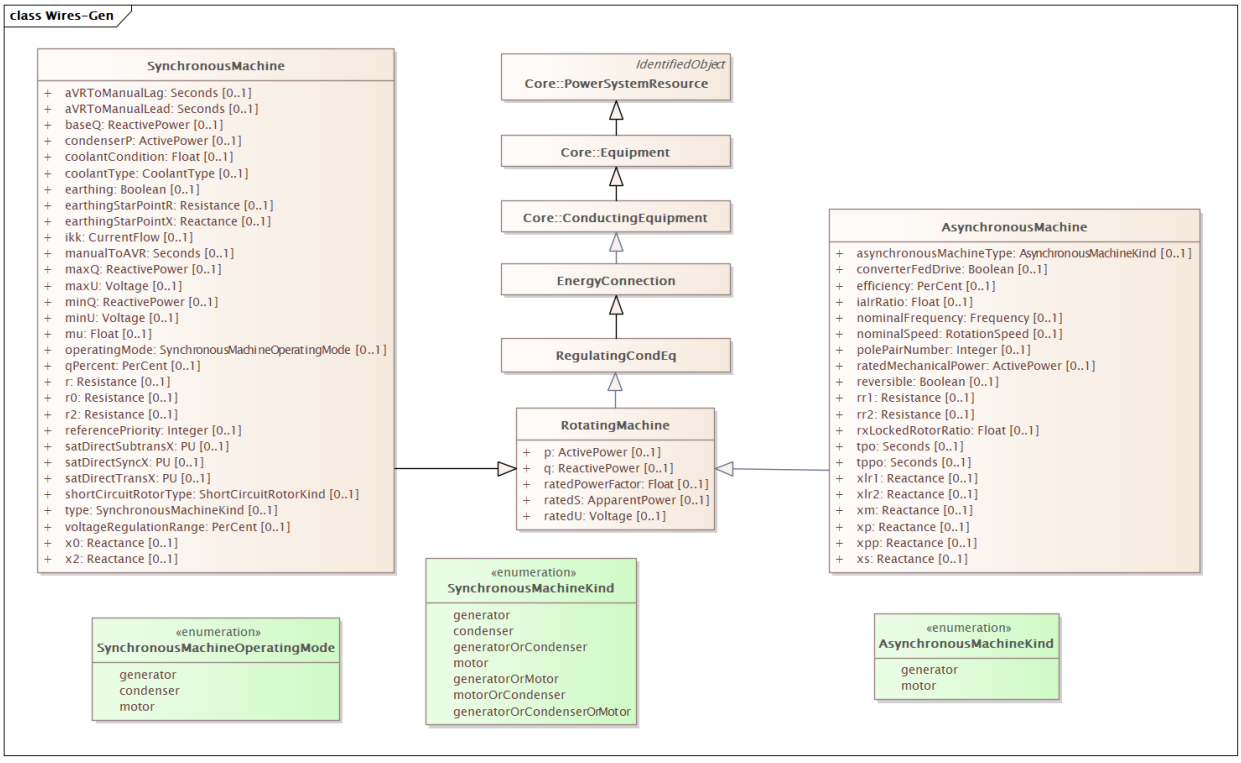


Figure 25: Usage of SynchronousMachine and AsynchronousMachine for generators and motor loads.

## 11.2 Production Cost Modeling

Detailed production cost models for economic dispatch, unit commitment, and other optimization studies are included in the 61970 `Production` package. These classes are separate from the electrical parameters defined in the `Wires` package and `Dynamics` package. The `Production` classes contain detailed parameters needed for specifying heat curves, ramp rates, startup/shutdown times, etc. A high-level summary is provided in Figure 26. The package also contains detailed models of plant component time curves (not shown) for timeseries optimization.

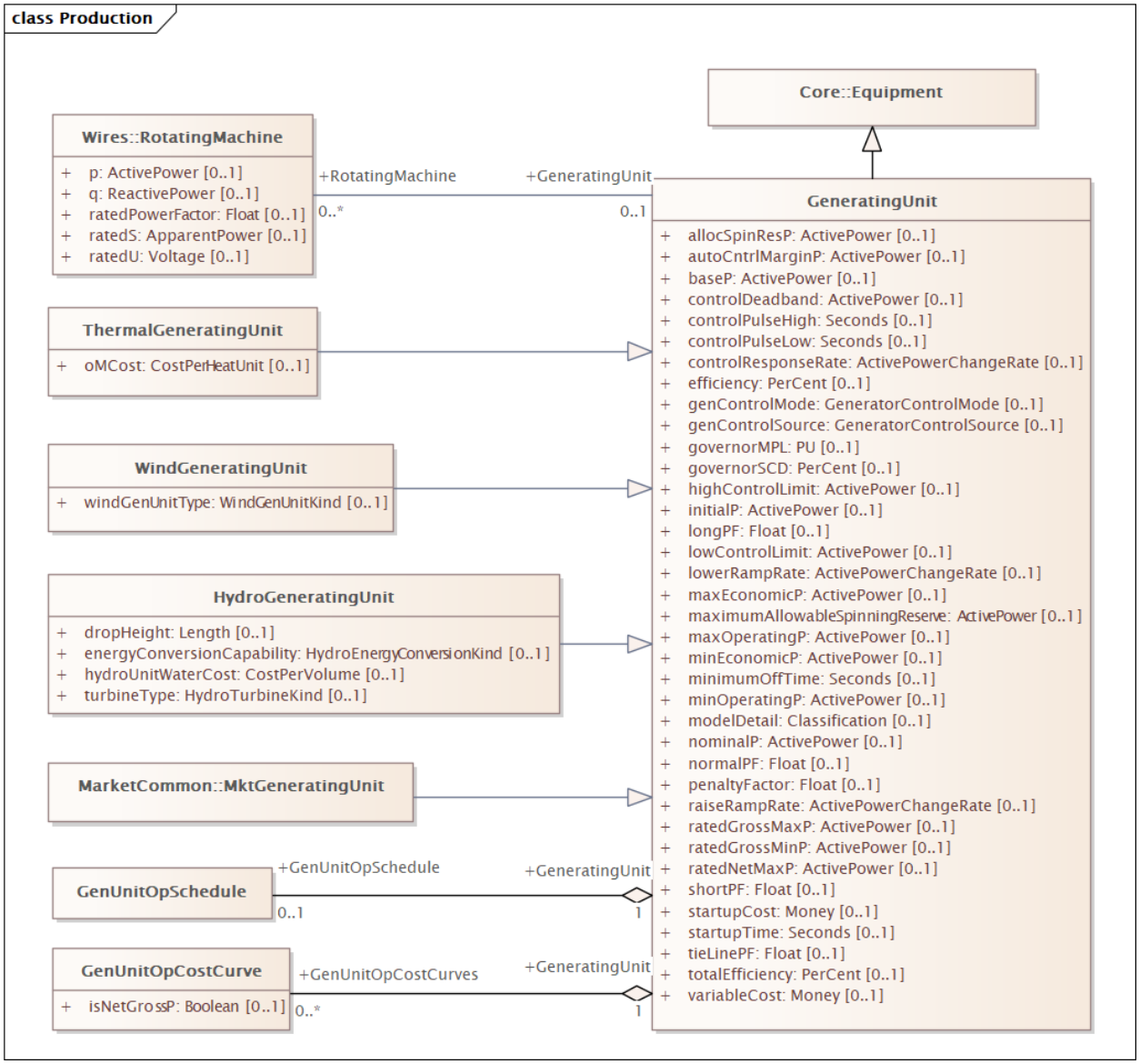


Figure 26: High-level summary of the `Production` package for generator production cost modeling

## 11.3 Long-Term Dynamics Modeling

Timeseries power system simulation can be divided into three categories. The first is quasistatic power flows solutions. These methods solve a single static snapshot at each timestep. No modeling of generator governor dynamics is needed. The only required CIM classes are `SynchronousMachine` and `AsynchronousMachine`. The second is long-term dynamics solutions, which are used by operator training simulators / dispatcher training simulators and implement basic models of the generator prime mover (i.e. turbine) and governor used for primary and secondary frequency control. These solvers typically solve a power flow solution every one second and use a timeseries integration of governor response to determine frequency, interchange, and area control error (ACE). These simulations require the governor models found in the `GenerationTrainingSimulation` package described here. The third is transient stability and other full dynamic solutions, which require detailed models of the generator inertia, governors, excitation, stabilizers, etc. (which are described in Section 11.4).

The `GenerationTrainingSimulation` package contains detailed models inheriting from `PrimeMover` to describe the basic long-term response of different kinds of generation. Each of the classes shown in Figure 27 contain detailed prime mover models and key modeling parameters (such as `HydroTurbine.waterStartingTime`) needed to tune large power system models for use in an operator training simulator.

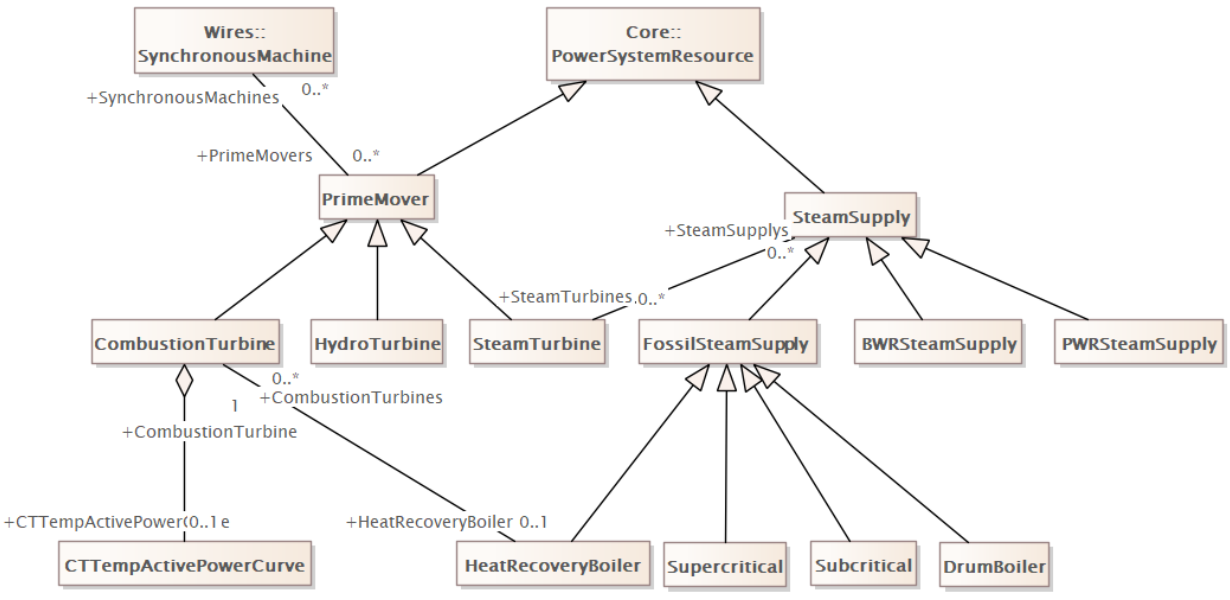


Figure 27: High-level summary of generator long-term dynamics modeling classes

## 11.4 Transient Stability and Dynamics Modeling

CIM provides an extremely detailed set of packages for power system dynamics modeling covering dynamic models of synchronous/asynchronous machines, generator governors, power system stabilizers, high-voltage direct current (HVDC) resources, static VAr compensators, voltage control systems, and composite load models. The package also contains several dozen classes for all standard generation and wind turbine model representations used in industry and found within the literature. Description of the classes and attributes used for dynamic modeling is beyond the scope of this report. A high-level summary of the dynamics packages available within CIM is presented in Figure 28.

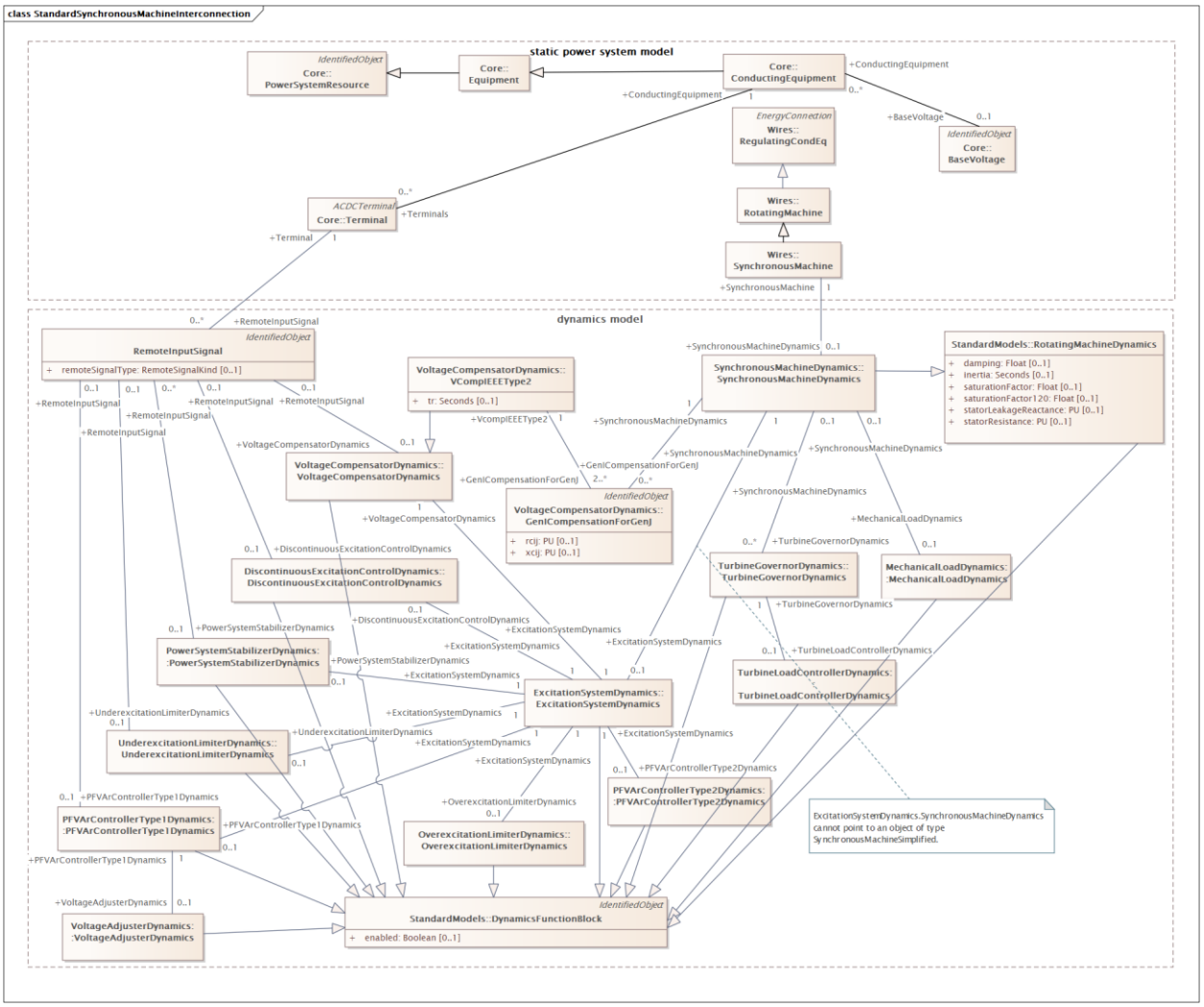


Figure 28: High-level summary of dynamics modeling packages included in the 61970 information model

# 12.0 Distributed Energy Resources

## 12.1 Inverter Power Flow Modeling

CIM 17 has introduced detailed modeling of distributed energy resources (DERs) with even more detailed models to be added in the upcoming draft CIM 18. The first set of modeling classes are contained within the `Wires` and `Production` packages. The inverter is specified as a `PowerElectronicsConnection` with attributes for the rated voltage and maximum real / reactive / apparent power that can be produced by the inverter. Each `PowerElectronicsConnection` inverter is associated with a single `Terminal` object on the AC side of the device. No explicit modeling of the DC connectivity is included. Single-phase inverters can be specified by defining each phase component as a `PowerElectronicsConnectionPhase` associated with the overall `PowerElectronicsConnection`. The DC source behind the inverter is specified through `Production` package as a `PhotoVoltaicUnit`, `BatteryUnit`, or `PowerElectronicsWindUnit`. The minimum and maximum power of each DC source is specified through the source class, as shown in Figure 28. Basic inverter control modes are specified as an enumeration.

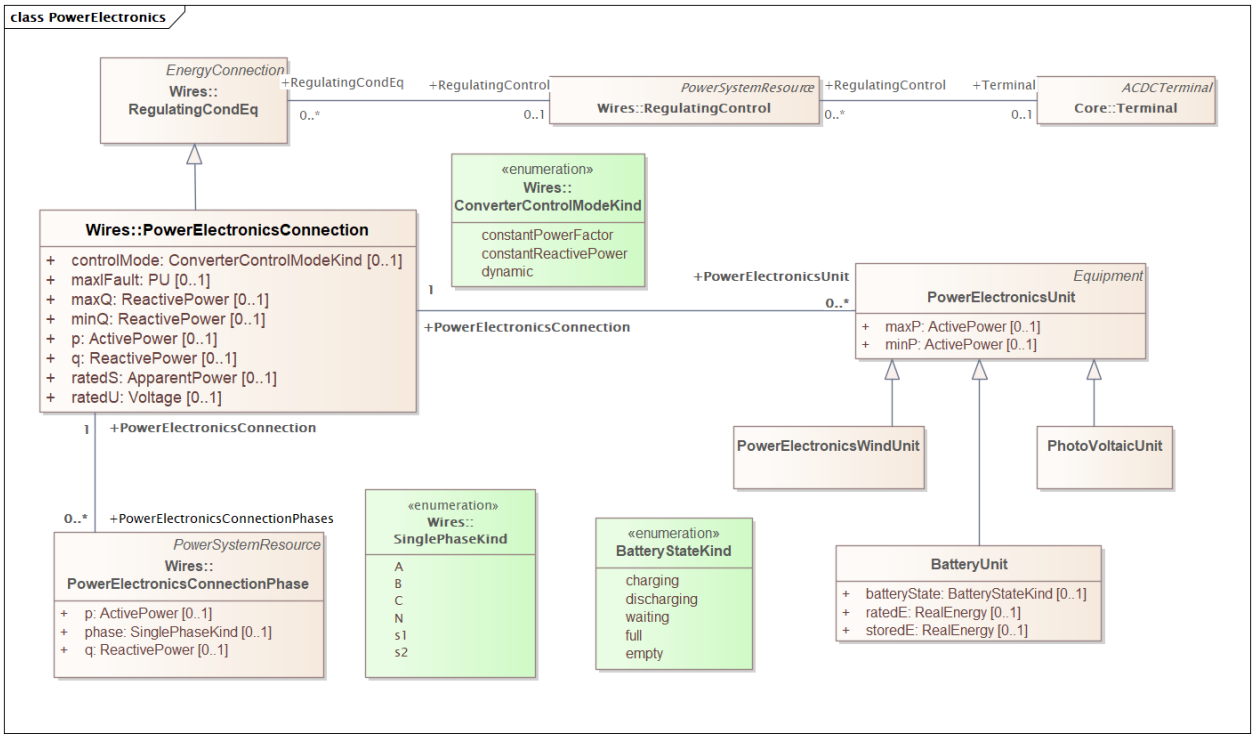


Figure 29: Modeling of DERs with the inverter specified as a `PowerElectronicsConnection` and individual DC resources as `PhotoVoltaicUnit` and `BatteryUnit`.

## 12.2 IEEE 1547-2018 Inverter Dynamics Modeling

IEC Standard 61970-302, 2nd Edition has added an initial set of inverter 61970 dynamics classes, which have been extended further by the GridAPPS-D project in the CIMHub package [4], [5]. The approach taken in this initial version is similar to that of transmission generator dynamics modeling with a set of dynamics objects defined for each inverter. The upcoming draft CIM 18 will likely revise these definitions to introduce an asset-based approach with 61968 standard nameplate definitions for common DERs which can be defined once and then referenced across the entire service territory of a utility.

The `DERIEEEType1` class is used to describe smart inverter functions and other DER behavior during time-series power flow, as shown in Figure 30. The CIM classes and attributes generally map to the interoperability tables in the IEEE standards, except for different capitalization convention, with the use of camelCase by CIM and the use of capital letters and underscores in the IEEE tables. Both `PowerElectronicsConnection` inverters and `RotatingMachine` objects can be associated to `DERIEEEType1` for supplemental nameplate and rating information in the network model. Preliminary values for these attributes would be available from an application to interconnect DER, and then updated as the project moves through commissioning to operational status. Detailed descriptions of the classes and attributes are available from IEEE 1547-2018 [6], IEEE 1547.1-2020 [7], and IEEE P1547.2/D6.2 (Annex F).

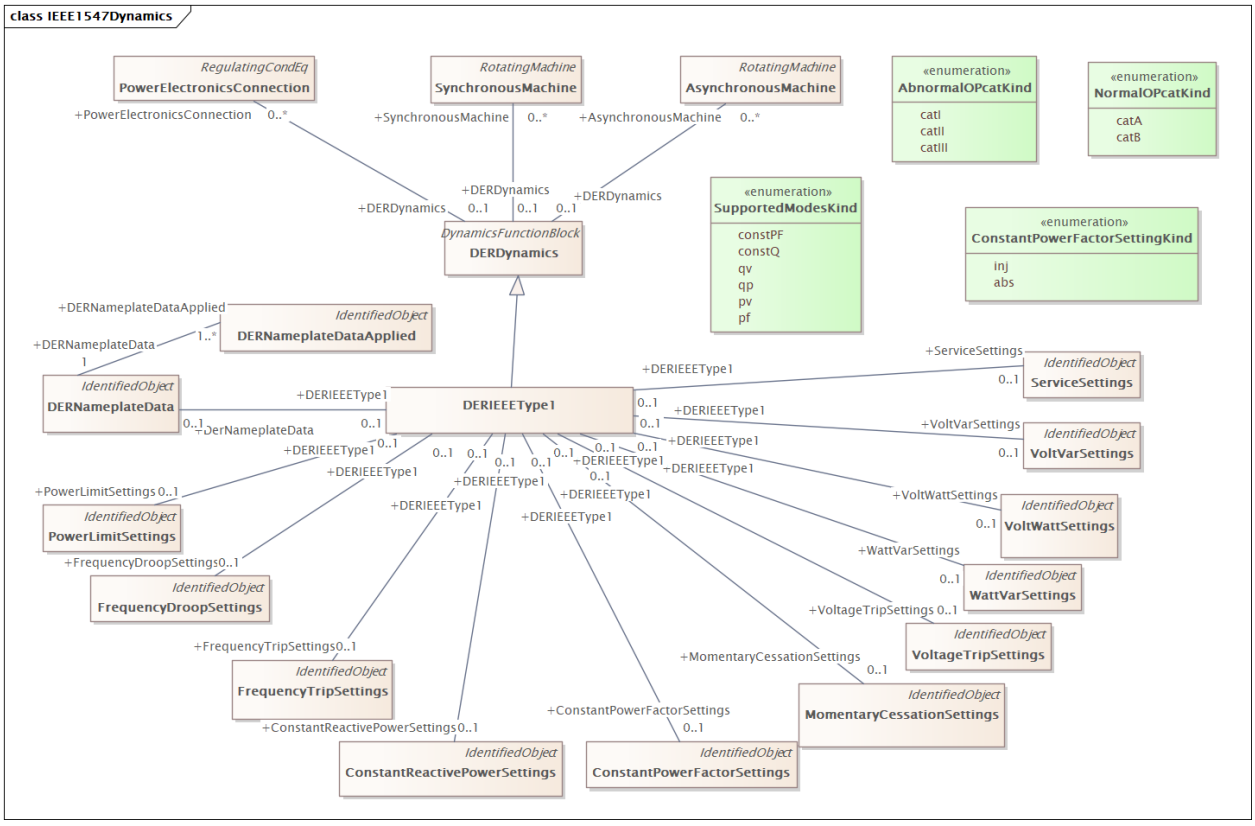


Figure 30: DERIEEEType1 and associations to classes needed to define IEEE 1547 dynamics settings.

# 13.0 Conclusion

This report introduced the set of key concepts related to adoption of the CIM to enable data integration and data exchange, looking towards creation of the next generation of advanced power applications that will be needed to ensure reliable, resilient, robust, safe, and economical operations. The document provided an overview of the key modeling classes and attributes needed for most power system modeling, simulation, and optimization tasks. An overview was provided of the modeling strategy for key power systems equipment, including lines, transformers, generators, switching equipment, loads, shunt devices, and distributed energy resources. The core classes for electrical modeling from the 61970 packages and asset modeling from the 61968 packages were introduced for each type of equipment discussed.

Power systems engineers and application developers seeking to use the CIM are encouraged to download the CIM information model (available freely under an Apache 2.0 license) from the CIM Users Group website [2]. The current, past, and draft versions of the CIM information model are available freely through the UCAiug website as set of Enterprise Architect `.eap` files and CIMTool `.xmi` files. These files can be used to generate data profiles and database schemas for use with custom software implementations and open-source modeling tools. A free viewer for the .eap files, Enterprise Architect Lite, is available from the Sparx EA downloads page[1].

The companion to this report, *Enabling Data Integration and Data Exchange with the Common Information Model* [1], provides a detailed explanation of all technical concepts needed to understand the semantic modeling concepts within consensus-based vocabulary provided by CIM. The combination of both reports should give most power application developers a sufficient technical background for developing CIM-compliant applications and services. For those interested in developing custom data converters between CIM and other power system model formats, the EPRI CIM primer [3] provides a highly detailed discussion of the process of deriving CIM profiles, data profiles, and database schema (which were introduced conceptually in [1]).

[1] https://sparxsystems.com/products/ea/downloads.html

# 14.0 References

[1] A. A. Anderson, E. G. Stephan and T. E. McDermott, "Enabling Data Exchange and Data Integration with the Common Information Model," Pacific Northwest National Laboratory, PNNL-32679, Richland, WA, 2022.

[2] CIM Users Group, "Current CIM Model Drafts," December 2021. [Online]. Available: https://cimug.ucaiug.org/CIM%20Model%20Releases/Forms/AllItems.aspx.

[3] S. Simmons, "Common Information Model Primer 7th Edition," Electric Power Research Institute, Palo Alto, CA, 2021.

[4] T. E. McDermott, "CIMHub," Jan 2023. [Online]. Available: https://github.com/GRIDAPPSD/CIMHub.

[5] T. McDermott, "Common Information Model diagrams," Jan 2023. [Online]. Available: https://cimhub.readthedocs.io/en/latest/CDPSM.html.

[6] Institute of Electrical and Electronics Engineers, "IEEE Std. 1547-2018: IEEE standard for interconnection and interoperability of distributed energy resources with associated electric power systems interfaces," [Online]. Available: https://ieeexplore.ieee.org/document/8332112.

[7] Institute of Electrical and Electronics Engineers, "IEEE 1547.1-2020: IEEE standard conformance test procedures for equipment interconnecting distributed energy resources with electric power systems and associated interfaces," [Online]. Available: https://ieeexplore.ieee.org/document/9097534.

# Pacific Northwest National Laboratory

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99354
1-888-375-PNNL (7665)

***www.pnnl.gov***